HOW TO PROVIDE MONITORING FACILITIES TO SERVICES WHEN THEY ARE DEPLOYED IN THE CLOUD?

Mohamed Mohamed, Djamel Belaïd and Samir Tata
Institut TELECOM, TELECOM SudParis, UMR CNRS Samovar, Evry, France

Keywords: Cloud Computing, Monitoring, Scalability, Service Containers.

Abstract: Cloud computing is a new maturing model providing efficient solutions in IT domain involving provisionning of virtualized ressources. Meanwhile, monitoring issue remains an active field of research. In this paper, we introduce a new scalable micro-container that enables different monitoring modes. Unlike the existing initiatives in this field, we propose a framework that automatically adds monitoring capabilities to a given service and encapsulate it in a scalable micro-container.

1 INTRODUCTION

In the last decade, Cloud Computing is gaining more and more attention in the IT (Information Technologies) domain. Different levels of enterprises, are trying to benefit the most from this revolutionary technology. However, a consensus on definition of Cloud Computing is not yet obtained. The National Institute of Standards and Technologies (NIST, 2011) defines Cloud Computing as ”a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal effort or service provider interaction”. Provisioning such a huge number of heterogeneous resources needs an efficient management to fit costumers requirements. Many projects are addressing issues of monitoring in the Cloud (e.g. RESERVOIR (Metsch et al., 2010), mOSAIC (Rak et al., 2011)) at different levels of granularities.

In the state of the art, there are different models of monitoring; we are interested in monitoring by polling and monitoring by subscription. First, monitoring by polling is the simplest way of monitoring. In fact, in this model, a service can express its need to know the state of a property provided by another service. To make this possible, the interested service can generally interact with a specific interface that provides a getter of the needed property.

Second, the monitoring by subscription in which there are two modes: 1) the subscription on Interval: it implies that the monitored service broadcasts the state of its properties periodically to the subscribed services and 2) the subscription OnChange: it implies that the monitored service have to notify the subscribed services whenever its properties change.

Almost all of the proposed solutions for monitoring are shallow in the way of collecting data from its producers and the way of using this data. Moreover, in these solutions, to monitor a service in the Cloud, this service must be designed to be monitorable and it must offer an interface to answer monitoring queries. However, some services may not offer any interface to answer such type of queries. These services cannot be monitored by any of the existing approaches.

In order to overtake this limit, we aim to render services monitorable. Our work will ease the task of services' developers, so they can focus in the functional properties of services and leave to our mechanisms the non functional aspects of monitoring.

In our previous work (Yangui et al., 2011), we presented a scalable micro-container that enables deploying and executing services in the Cloud with a proven efficiency and minimal resources consumption. And due the need of monitoring, we aim at adding monitoring facilities to services deployed on top of our micro-container. We started by designing a framework that allows to encapsulate services in a composite adding monitoring interfaces to original ones. We integrated this framework with the micro-container to offer monitoring services using this latter.

In this paper, we will describe how can our new micro-container supports scalability and enables monitoring capabilities in Cloud environments.

The remainder of this paper is structured as follow: the second section will describe briefly our previous work including the architecture of the micro-container and the component model we used to represent services. Section 3 describes the monitoring framework, different transformations we added to render a service monitorable and the new architecture of the micro-container that supports monitoring. Section 4 describes implementation aspects. Section 5 presents related works. We end the paper with conclusions and our future work.

2 BACKGROUND

In this section, we will present the component model we use in our work. Then, we will define the different types of monitoring that we are interested in. Finally, we will describe the micro-container that we use to deploy components in the cloud.

In our work, we are interested in monitoring of component-based applications in the Cloud. The key element for these applications is the component which is a unit of composition with contractually specified interfaces and explicit context dependencies only (Szyperski, 2002). Figure 1 shows main characteristics of a component that provides a service through an interface and may require a service from other components through a reference. The component also may expose properties through which it can be configured. In addition, the component also specifies its dependency on a certain property. This required property, which appears at the bottom of the component, will be satisfied if we can link this component with another component that offers the requested property, thus, solving the dependency.

Components can be combined together in an assembly of components to build complex applications as represented in Figure 1.

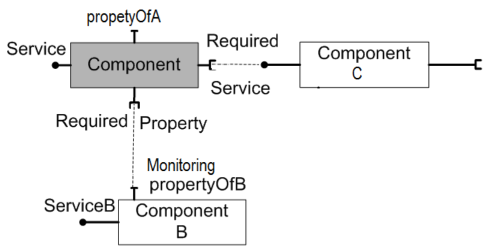

Figure 1: Component-based Application.

2.1 Monitoring

Monitoring process consists in informing the interested component about the changes of required properties or notifying it on a regular way or for each property variation. We consider two types of monitoring: monitoring by polling or by subscription.

Polling is the simpler way of monitoring, as it allows the observer to request the current state of a property whenever there is a need. Subscription allows an observing component to be notified about changes of monitored properties. There are two modes of monitoring by subscription: 1) subscription *on change* which specifies that the subscribed component is notified every time the value of the property changes; 2) subscription *on interval* which specifies that the subscribed component is to be notified after a specified time interval. For notification on change, a component must precise the starting time and the duration of notifications. For notification on interval it must specify the notification interval value. It may also specify the starting time and the duration of notifications. The component A must also implement a notification callback through which it will receive appropriate notification messages.

2.2 Scalable Micro-container

In (Yangui et al., 2011), we introduced a scalable and platform independent micro-container that enables services deployment and execution in the Cloud.

For optimality and performance constraints, features of our micro-container are as minimal as possible. After studying the features provided by the container architectures of Axis2 (Perera et al., 2006), Tomcat 6 (Langlet, 2008) and WSCRA (Dhesiaseelan and Ragunathan, 2004), we drew up a list of basic features that should satisfy our micro-container which directly reflects the different modules that make up its architecture. These basic modules ensure the minimal main process of our micro-container (component hosting, interaction with clients, etc.).

The platform contains a processing module to ensure minimal micro-containers generation process (WSDL Parser, Compiler, etc.) and a set of generic elements on the submission and treatment of non-functional features to be included in the micro-container (HTTP, RMI or other generic Communicators, service administration tools, service billing, etc.). The global architecture of the performed system is composed of several modules detailed in Figure 2: 1) a deployment platform for addressing the component code and a deployment descriptor for the generation of a corresponding micro-container within the component to deploy, 2) micro-container to host the deployed component and handle various clients requests, and 3) thin clients to invoke services offered by components via micro-containers.

The deployment framework is responsible of

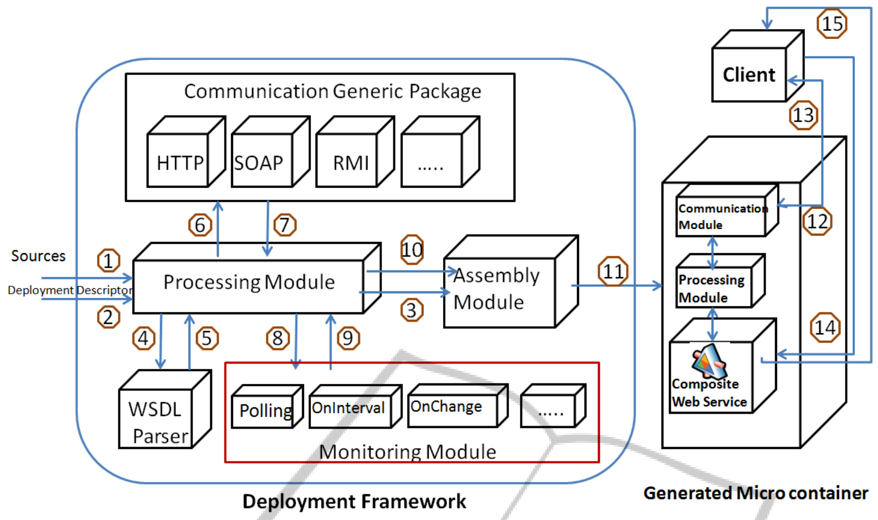

Figure 2: Extension of the Micro container architecture with monitoring.

the generation of the micro-container.To generate a micro-container with a component hosted in, one must provide the component's source and a deployment descriptor which describe how to assemble and deploy the micro-container into a Cloud environment.

The micro-container is responsible of managing the communication with the client, holding the component and processing all the incoming or outgoing messages of the micro-container. It is composed only of the necessary modules for the offered services of the hosted component.

The architecture of the generated micro-container shows three main modules: 1) a Communication module to establish communication and to support connection protocols, 2) a Processing module to process ingoing and outgoing data in the server (packing and unpacking data), and 3) a Service module to store and invoke the requested service.

To add monitoring capabilities to this micro-container, we use the component model we described earlier to represent components and their offered services. Since some properties can be defined as non monitorable, we integrated mechanisms that allow us to transform these properties to monitorable ones.

In the next section, we describe the different transformations to add monitoring facilities to services using our micro-container. Then, we present the resulting extended micro-container architecture.

3 MONITORING FRAMEWORK

Our objective is that given a component without monitoring facilities that we would like to transform in order to be aware in its properties changes. A transformation may apply to this component by encapsulating it in a new composite delivering the same services of the original component and enhancing it with non functional services of monitoring.

Assuming that these components are not monitorable by default, we need to make them monitorable by transformation. We are interested in monitoring by polling and monitoring by subscription.

A transformation of a component is applied dynamically when generating the associated micro-container and is carried out by some predefined components of our framework. For different types of transformation, the framework defines different components. In the next subsections, we introduce features of the monitoring mechanisms and their transformation processes.

3.1 Generic Proxy Service

The transformations that render a component monitorable uses a Generic Proxy Component that implements the Generic Proxy Interface presented in Figure 3 and can be applied to any component.

```
public interface GenericProxy {
 Property[] getProperties();
 Object getPropertyValue(String propertyName);
 void setPropertyValue(String propertyName,
                          Object propertyValue);
 Object invoke(String methodName,
                          Object[] params);}
```

Figure 3: Description of the Generic Proxy interface.

We have defined a general purpose interface GenericProxy that provides four generic methods.

Each implementation of this interface is associated with a component for which the first method getProperties() returns the list of the properties of the component, the getPropertyValue() returns the value of a property, the setPropertyValue() changes the value of a property and the invoke() method invokes a given method on the associated component and returns the result.

3.2 Monitoring Transformations

In (Belaïd et al., 2010), we have presented a monitoring approach to allow a component to be aware of required properties changes. We have considered monitoring by polling and monitoring by subscription.

3.2.1 Monitoring by Polling

A component may express its need to monitor by polling a required property provided by another component. The monitoring by polling of a property can be made by calling its getter. However, the component that wishes to monitor a property of another component does not know a priori the type of this component. To complete the monitoring of any component from the name and type of a property, the interested component uses an appropriate interface that provides the method getPropertyValue(propertyName) to request the current state of a property (Figure 4).

However, the component to monitor may not define its offered properties as monitorable by polling resources despite the request. So, we need to transform the component to make its properties to be monitorable by offering an appropriate interface of monitoring. This can be done dynamically by our framework by encapsulating the component with the predefined GenericProxy component as defined above. The two components are combined together in a single *composite* that offers the services of the original component as well as services of the GenericProxy component. The component can be then monitored using the getPropertyValue() method provided by the composite. The framework then replaces the original service with the newly created composite in the micro-container.

3.2.2 Monitoring by Subscription

For the monitoring by subscription we encapsulate the original component with the two predefined components: GenericProxy and MonitoringBySubscription. For the monitoring with notification mode on interval, as shown in the Figure 5, each time the MonitoringBySubscription component have to notify the subscriber (the component A), it gets (or monitor by polling) the

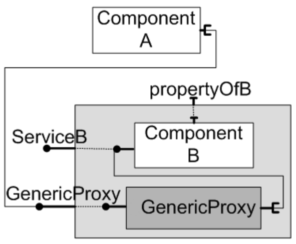

Figure 4: Monitoring by polling.

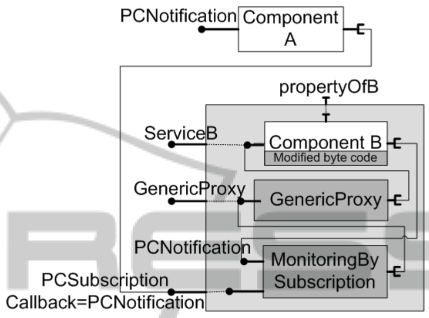

Figure 5: Transformation for monitoring by subscription.

value of the required property of the component B via the GenericProxy component.

When the notification mode is on change for a required property of B (Figure 5), the MonitoringBySubscription component offers a (callback) service of notification PCNotification to the component B so that it can be notified of the changes of a required property and in turn inform all the subscribers of this change. To allow the component B to notify the MonitoringBySubscription for the change of its properties, our framework adds the needed instructions in the bytecode of the component B when generating the associated micro-container.

3.3 Monitoring within Micro-container

In order to integrate these transformations with our micro-container, we added the new monitoring module to the architecture described in section 2.

The extended architecture adds monitoring capabilities to the scalable micro-container. In fact, as described in section 2, the same steps of generating the micro-container are followed. The processing module follows actions from 1 to 7 as in the Figure 2, then it instantiates the chosen monitoring module and applies the needed transformations to the service (Figure 2 Actions 8 and 9) before sending the new resulting code to the assembly module (Figure 2 action 10). The latter generates the new micro-container integrating monitoring capabilities.

The client can interact with the micro-container either to invoke the contained service (Figure 2 actions 12 and 13), or to request monitoring information (Figure 2 action 14). It can also send subscription requests to receive notification on change or on interval.

In order to prove the efficiency of our work, in the next section we will describe the implementation of our scalable micro-container renforced with monitoring capabilities.

4 IMPLEMENTATION

To validate our work, we took the choice to exploit Java Web services with WSDL 2.0 description. A Web service is an elementary service-based application. We have developed a minimal Java deployment framework which allows developers to deploy a Java Web service on a micro-container before deploying both of them in the Cloud. We have also developed Java clients which invoke services in the micro-container and display the result. Furthermore, we defined a module that contains generic communication packages supporting different communication protocols that can support a service. Later, during the deployment, only necessary communication protocol is encapsulated in the micro-container.

The last phase was implementing a prototype of the monitoring framework as services that offer the transformation mechanisms to the applications. To allow a component to notify the MonitoringBySubscription for the change of its properties, we need to inject the notification code in the byte-code of the component implementation (class). For this required dynamic manipulation at byte code level we used the open source software JAVA programming ASSISTant (Javassist) library (JAVA programming Assistant, 2010). Javassist is a class library for editing Java byte codes; it enables Java programs to define a new class and to modify a class file when the Java Virtual Machine (JVM) loads it. The implementation of the GenericProxy component is based on the Java reflection API which provides classes and interfaces for obtaining reflective information about classes and objects.

5 RELATED WORK

The mOSAIC framework offers a Monitoring/Warning system that monitors applications' components and Cloud resources (Rak et al., 2011). From their point of view, this system should realize the following tasks: monitor Cloud resources, monitor applications' components and discover warning conditions. The proposed framework contains four basic elements: 1) monitoring event buses that collects monitoring events from resources, 2) connectors related to the event buses to enable the interception of monitoring events by suitable components, 3) connectors receiving the events from applications to event buses, and 4) monitoring/Warning component.

In (Ferretti et al., 2010), authors proposed a design of a middleware that supports SLA-driven resource provisioning. This middleware supports monitoring of virtual machine provisioning to honor the service layer agreements (SLA). A load balancer is used to forward requests to their suitable execution environment and to monitor the efficiency of the platform by calculating the average response time. The load balancer analyses this time and can take decisions to add or to retrieve resources in the environment.

Authors in (Huang and Wang, 2010) proposed an approach to monitor resources in the Cloud using an hybrid model combining the push model and the pull one. In these models, there are three basic components, the Producer, the Consumer and the Directory services. In the Push model, whenever the producer detects that the status of a resource changed, he sends information to the consumer. Otherwise, in the Pull model, it's the consumer who ask the producer periodically about the resources status. However, these two models have advantages and weakness, the authors are proposing an hybrid model that can switch the suitable model to the user's requirements. The user can define his tolerance to the status inaccuracy between the producer and the consumer. Switch this value, an algorithm can swing between pull and push models.

In (Brandt et al., 2009), authors proposed a tool for monitoring Cloud resources enhancing high-performance computing in Cloud computing environments. This tool can extract the application and resources state, and based on that state can assign new resources or retrieve unused ones during the runtime of the application or for the next usage. The data is collected from the resources using data collectors able to collect information and save it in a distributed data base. Then, a statistical analysis is necessary to take decisions to keep or to reconfigure the resources' assignment for the application.

In their work (Katsaros et al., 2011) proposed an architectural approach spanning over virtualization and physical levels to collect monitoring data. They combined many existing open source solutions to get one holistic application that cover different layers. They use collectors to extract data from different layers and externalize it to the upper layer using an external data collector. Moreover, a monitoring manager serves as the orchestrator of the whole monitoring process by controlling it and providing the needed interfaces to add or to consume monitoring information. In this approach, an aggregator is responsible of aggregating and storing the collected data.

Almost all of these approaches expect monitoring just for monitorable components and don't suppose the case where components are not designed to be monitored. Moreover, in these works, the monitoring system can form a bottleneck in the application since they use just one component as a channel for sending monitoring information (buse, channel, integrator, etc). In contrast, in our approach, we provide transformations to apply on components to render them monitorable even if they where not designed with monitoring facilities. Furthermore, in our work, we are not limited to one channel since we propose a flexible solution to use one channel for all micro-containers or to use one channel per micro-container.

6 CONCLUSIONS

In this paper, we addressed the issue of monitoring in Cloud environments. In this direction, we proposed the architecture and the implementation of a scalable micro-container that enables monitoring capabilities avoiding bottlenecks problems. Our solution, provides needed mechanisms to monitor services. In fact, we provided a framework that allows to add non functional services of monitoring to the functional interfaces of components. The framework that we proposed, chooses the needed components to generate the micro-container that enables a scalable execution of a service with monitoring capabilities. In our future work, we will start by experimenting our architecture in a Cloud environment using OpenNebula IaaS manager to see the overhead of monitoring mechanisms on our micro-container. Indeed we want to prove, by experiments, the efficiency of our approach even with complex applications. Then, we aim at adding the needed mechanisms for monitoring capabilities in micro-containers at their runtime and not only at deployment time.

REFERENCES

Belaïd, D., Ben Lahmar, I., and Mukhtar, H. (Dec 2010). A Framework for Monitoring and Reconfiguration of Components Using Dynamic Transformation. *International Journal On Advances in Software*, vol 3 no 3&4:pages 371–384.

Brandt, J., Gentile, A., Mayo, J., Pebay, P., Roe, D., Thompson, D., and Wong, M. (2009). Resource monitoring and management with ovis to enable hpc in cloud computing environments. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8.

Dhesiaseelan, A. and Ragunathan, A. (2004). Web services container reference architecture (wscra). In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 806–807.

Ferretti, S., Ghini, V., Panzieri, F., Pellegrini, M., and Turrini, E. (2010). Qos #150;aware clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 321–328.

Huang, H. and Wang, L. (2010). P amp;p: A combined push-pull model for resource monitoring in cloud computing environment. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 260–267.

JAVA programming Assistant (2010). http://www.csg.is.titech.ac.jp/*sim*chiba/javassist/.

Katsaros, G., Gallizo, G., Kübert, R., Wang, T., Fitó, J. O., and Henriksson, D. (2011). A multi-level architecture for collecting and managing monitoring information in cloud environments. In *CLOSER*, pages 232–239.

Langlet, E. (2008). *Apache Tomcat 6 Guide d'administration du serveur Java EE sous Windows et Linux*. ENI.

Metsch, T., Edmons, A., and Bayon, V. (2010). Using cloud standards for interoperability of cloud frameworks. Technical report, A technical Reservoi report.

NIST (2011). Final version of nist cloud computing definition published. http://www.nist.gov/itl/csd/cloud-102511.cfm.

Perera, S., Herath, C., Ekanayake, J., Chinthaka, E., Ranabahu, A., Jayasinghe, D., Weerawarana, S., and Daniels, G. (2006). Axis2, middleware for next generation web services. In *Web Services, 2006. ICWS '06. International Conference on*, pages 833–840.

Rak, M., Venticinque, S., Mahr, T., Echevarria, G., and Esnal, G. (2011). Cloud application monitoring: The mosaic approach. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 758–763.

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley/ACM Press, Boston, MA, USA, 2nd edition.

Yangui, S., Mohamed, M., Tata, S., and Moalla, S. (2011). Scalable service containers. In Lambrinoudakis, C., Rizomiliotis, P., and Wlodarczyk, T. W., editors, *CloudCom*, pages 348–356. IEEE.