International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:10, 2012

An Event based approach to Extract the Run Time Execution Path of BPEL Process for Monitoring QoS in the Cloud

Rima Grati, Khouloud Boukadi, and Hanene Ben-Abdallah

***Abstract*—**Due to the dynamic nature of the Cloud, continuous monitoring of QoS requirements is necessary to manage the Cloud computing environment. The process of QoS monitoring and SLA violation detection consists of: collecting low and high level information pertinent to the service, analyzing the collected information, and taking corrective actions when SLA violations are detected. In this paper, we detail the architecture and the implementation of the first step of this process. More specifically, we propose an event-based approach to obtain run time information of services developed as BPEL processes. By catching particular events (i.e., the low level information), our approach recognizes the run-time execution path of a monitored service and uses the BPEL execution patterns to compute QoS of the composite service (i.e., the high level information).

***Keywords*—**Monitoring of Web service composition, Cloud environment, Run-time extraction of execution path of BPEL.

I. Introduction

CLOUD computing is increasingly being promoted as the next-generation of paradigms for hosting and delivering services over the Internet [1]. In this paradigm, services can be provided at different layers: Software (Software as a service: SaaS), Platform (Platform as a Service: PaaS) and Infrastructure (Infrastructure as a Service: IaaS). In fact, Cloud computing provides users with services to access software, data and/or hardware without the need to understand any underlying complexity. In particular, in recent years, SaaS implementations have become an increasingly popular a way to let both users manage typical day-to-day tasks and enterprises make money by arranging an ongoing software licensing agreement with different businesses.

Despite these advantages, given the complexity of the Cloud environment, service failures are quite likely and are the norm rather than the exception. Consequently, Quality of Service (QoS) degradations may frequently occur at all layers. When dealing with SaaS applications, QoS monitoring is essential for two reasons: on the one hand, to provide Cloud usage that is “acceptable” to the various clients, and on the other hand, to spare Cloud providers penalties for not offering services at a certain level of QoS.

To monitor the QoS of services, most works in the literature require modification of either the server or the client implementation code [2, 4, 7]. However, to provide for independence of any Cloud provider/environment, monitoring should be performed in a non-intrusive way, *i.e.*, without modifying the implementation of the deployed Cloud services. Furthermore, to the best of our knowledge, there is a lack of approaches dealing with monitoring service composition in a *SaaS* Cloud environment, the focus of this paper.

In our previous work, we proposed a framework for QoS Monitoring and Detection of SLA Violations (QMoDeSV) [1]. This framework provides for the monitoring of composite services deployed on the Cloud. It is designed to handle the complete Web service composition management lifecycle in the Cloud environment (*i.e.*, composite Web service deployment, resource allocation, monitoring of QoS and SLA violation detection). In addition, QMoDeSV proposes a non-intrusive, modular approach for monitoring QoS attributes: QoS pertinent information is collected by “watching” locally each service component. Then, based on the composition pattern of the composite service, the overall QoS information is computed. This information is used by a separate module in the QMoDeSV framework to look for potential violations of SLA pre-agreed upon QoS attributes. The findings of this module can be very helpful for service providers, who can then take corrective actions to improve their services.

In this paper, we will put the focus on the RTP Extractor module which extracts the run time execution path of the BPEL process. The execution path is then used to calculate the QoS of a composite service in the Cloud according to its patterns. These QoS calculated can be compared to agreed-upon SLA requirements to detect SLA violations.

The remainder of this paper is organized as follows: Section II overviews works related to monitoring Web services. Section III briefly reviews the architecture of our QMoDeSV framework and details the architecture and the implementation of the RTP Extractor Module. Section IV illustrates the functionality of the RTP Extractor through an example of a banking business process. Section V summarizes the presented work and highlights its future directions.

Rima Grati is with the Faculty of Economics and Management of Sfax, Route de l’Aéroport Km 4 Sfax 3018 (corresponding author to provide phone: 00216 27 525 020; e-mail: rima.grati@gmail.com).

Khouloud Boukadi is with the Faculty of Economics and Management of Sfax, Route de l’Aéroport Km 4 Sfax 3018(e-mail: khouloud.boukadi@fsegs.rnu.tn).

Hanene Ben-Abdallah is with the Faculty of Economics and Management of Sfax, Route de l’Aéroport Km 4 Sfax 3018 (e-mail: hanene.BenAbdallah@fsegs.rnu.tn).

II. Related Work

Most works in the literature are centered on the Web service monitoring [2, 3, 4, 5]; less effort has been invested in Cloud monitoring [6, 7, 8, 9, 10].

A. Web Service Monitoring

Thio et al. [2] propose a QoS monitoring framework for Web service based applications. It extends the SOAP implementation API, both for the client and the server, to measure and log QoS parameter values. This enables the user to perform automated performance measurements. An experiment is described running more than 200 services requests per day during 6 days and measuring the response time. The approach depends on the used SOAP implementation, and the required QoS monitoring extensions have to be deployed into the SOAP implementation used by the provider. This solution modifies the SOAP implementation.

Ben-Halima et al. [3] propose a QoS-Oriented Self-Healing middleware (QOSH) for Web service monitoring. QOSH monitors response time parameters of Web services. It is based on the interception of SOAP headers. Since this approach enriches SOAP messages with QoS information. QOSH modifies both the client and the server implementation to allow QoS parameter evaluation.

Haiteng et al. [4] propose a solution to the problem of monitoring Web services described as BPEL processes. The solution introduces Monitor Broker into traditional Web services architecture to access Web service runtime state information and calculate the QoS values. Monitor Broker architecture use Aspect Oriented Programming (AOP) that allows for a clear separation of the service business logic from the monitoring functionality. The initial implementation and experiment with a travel reservation service example shows that this approach is feasible and the monitoring cost is affordable.

Sun et al. [5] propose a monitoring approach based on AOP. Their goal is to check business process conformance with the requirements that are expressed using WS-Policy. The properties (e.g., temporal or reliability) of a Web service are described as Extended Message Sequence Graph (EMSG) and Message Event Transferring Graph (METG). A runtime monitoring framework is then used to monitor the corresponding properties that are then analyzed and checked against the METG graphs. This work is also based on AOP approach.

B. Cloud Service Monitoring

Shao et al. [6] propose a Runtime Model for Cloud Monitoring (RMCM). RMCM uses interceptors (as filters in Apache Tomcat and handlers in Axis) for service monitoring. It collects all Cloud layer performance parameters. In the SaaS layer, RMCM monitors applications while taking into account their required constraints and design models. To do so, it converts the constraints to a corresponding instrumented code and deploys the resulting code at the appropriate location of the monitored applications. Thus, it modifies the source code of the applications.

Cao et al. [7] propose a monitoring architecture for Cloud computing. It describes a QoS model that collects QoS parameter values such as response time, cost, availability, reliability and reputation. Their architecture is interesting, but has not been implemented yet.

Clayman et al. [8] present Lattice framework for Cloud service monitoring in the RESERVOIR EU project. It is capable of monitoring physical resources, virtual machines and customized applications embedded with probes. Compared to our approach, the Lattice framework doesn't consider the detection of SLA violations to avoid SLA penalties.

Rak et al. [9] propose Cloud application monitoring using the mOSAIC approach. In a first step, the authors describe the development of customized applications using mOSAIC API to be deployed on Cloud environments. For these applications, they propose in a second step some monitoring techniques. Their interest is only to gather information that can be used to perform manual or automatic load-balancing, increase/decrease the number of virtual machines or calculate the total cost of application execution. Their approach does not consider the detection of SLA violations to avoid SLA penalty cost and moreover, it is not generic since it monitors only applications developed using the mOSAIC API.

Mdhaffar et al. [10] propose an approach called AOP4CSM (Aspect-Oriented Programming For Cloud Service Monitoring) which is based on aspect-oriented programming and monitors quality-of service parameters of the Software-as-a-Service layer. The use of AOP4CSM is exemplified in the context of fault tolerance. Its installation does not need any access to the source code of the service and the client. It has been implemented within Axis (both Axis1 and Axis2).

In summary, we note that several monitoring works used the AOP approach. These works treated either the Web service monitoring or the Cloud monitoring. In our research, we chose the event-based approach since it provides more efficient results. In fact, the AOP approach is based on the concept of proxy. The class of BPEL engine that will make an appropriate treatment according to the BPEL process instance will be enclosed by a proxy object that can perform processing before or after the invocation of the method in the BPEL engine class. As shown in Fig. 1, when a method (modeled by the orange bubble) is called, the proxy object notifies that the method was called and may indicate the invocation time (which is slightly greater than the real invocation time). However, in the event-based approach, the class of the BPEL engine is responsible for launching event (with information about the type of processing performed by the BPEL engine in real time as well as the execution time). Hence, the event based approach is slightly better in point of view precision and performance than the AOP approach.

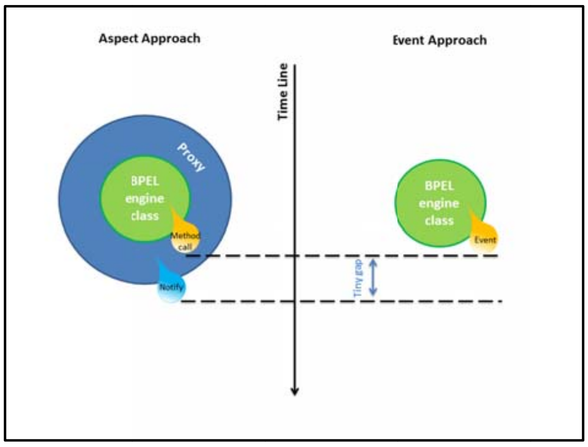


Fig. 1 Comparison between the event-driven approach and the AOP approach

Note that, to the best of our knowledge, none of the discussed approaches deals with monitoring Web service composition in the Cloud using an event driven approach.

III. The QMoDeSV: The RTP- Extractor Module

This section presents the QMoDeSV framework which provides for the monitoring of composite services deployed on the Cloud. As shown in Fig. 2, the QMoDeSV framework covers the design phase (the Extractor Module) and the deployment/run-time phase of a service. In the design phase, the Extractor Module determines the design execution patterns which influence the QoS attributes of the composite service. In the run-time phase, the QMoDeSV framework deploys in parallel five modules: the RTP Extractor, the QoSCalculator, the Local Host Monitor, the Lo2Hi QoS Convertor, and the QoS Detector Violation. Once a Web service is executed, the run time modules run in parallel with the BPEL instance to detect possible SLA violations.

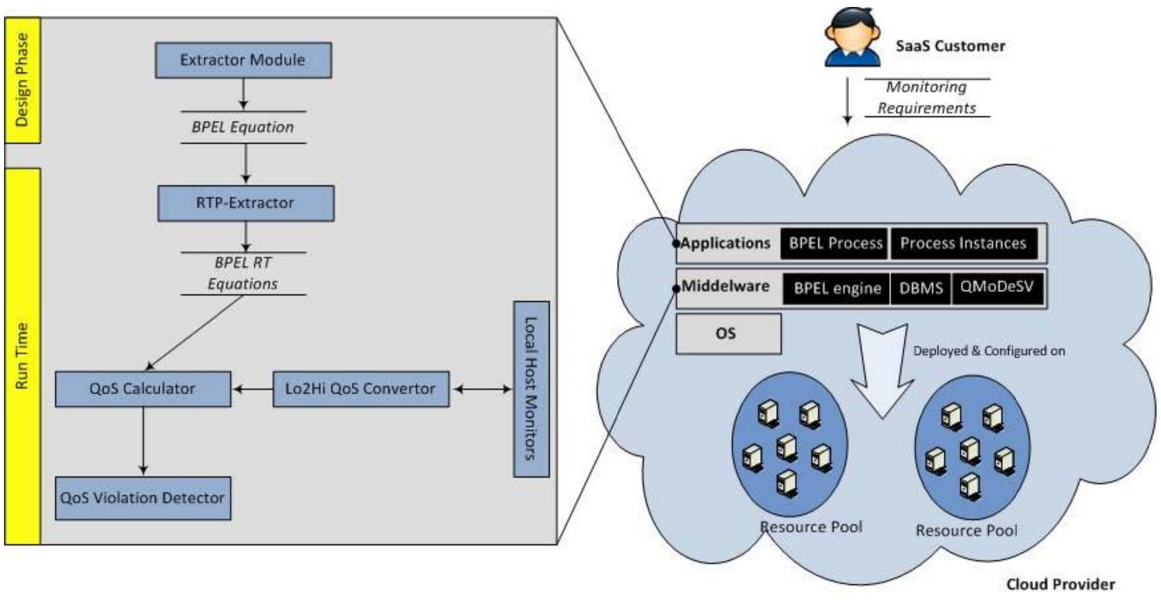


Fig. 2 The QMoDeSV framework overview

QMoDeSV proposes a non-intrusive, modular approach for monitoring QoS attributes: QoS pertinent information is collected by “watching” locally each service component; this is the role of the RTP Extractor. Then, based on the composition pattern of the composite service, the overall QoS information is computed; this is the role of the QoSCalculator. This information is used by the QoS Detector Violation module to look for potential violations of SLA pre-agreed upon QoS attributes. The findings of this module can be very helpful for service providers, who can then take corrective actions to improve their services.

In our previous work, we presented the design phase module of our framework; for more details the reader can refer to [1]. The remainder of this paper describes the RTP Extractor and we will detail its architecture as well as its implementation. We illustrate its functionality though an example and in the case of response time.

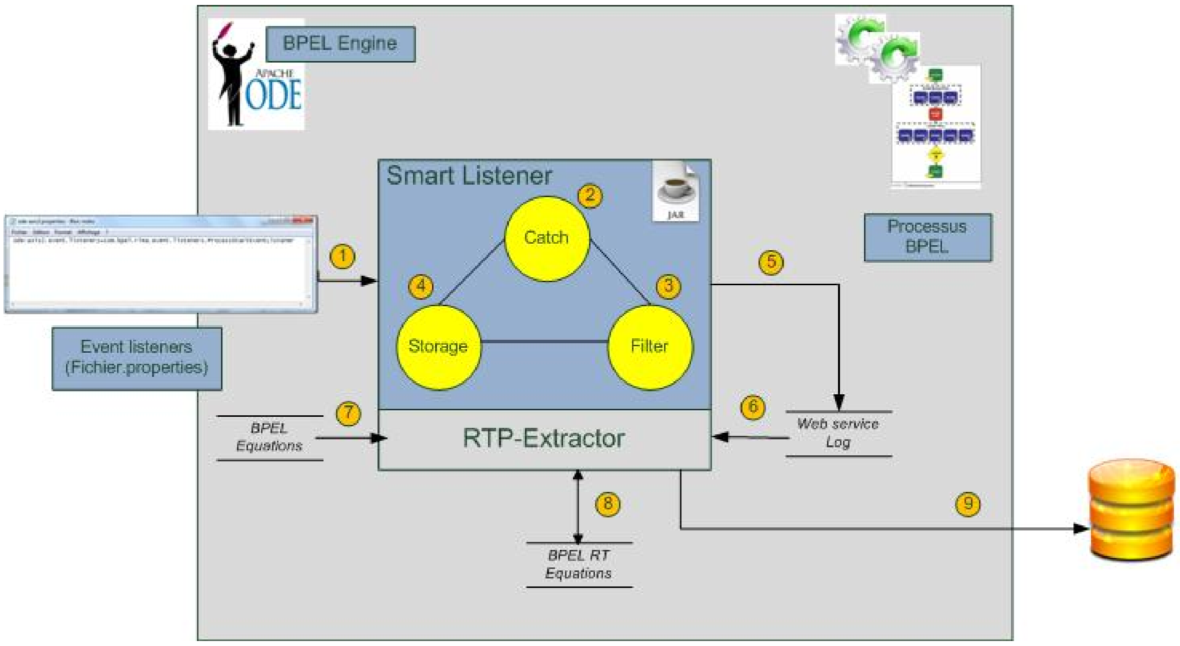


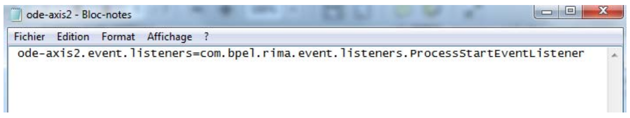
Fig. 3 Conceptual architecture of the RTP-Extractor Module

The role of the RTP-Extractor Module is to detect the services invoked in run time as well as the patterns used by these services. In addition, it extracts the execution path of the BPEL process and the execution time of each elementary service. As shown in Fig. 3, this module contains a sub-module called Smart Listener. This latter has three components namely the Catch component, the Filter component and the Storage component. We will explain later the role of these components.

Once the BPEL process starts its execution, the BPEL engine will look for the listener connected to it (Fig. 3, step 1). In our case we use Apache ODE (Orchestration Director Engine). Apache ODE is an open source and it executes business processes written following the WS-BPEL standard [11].

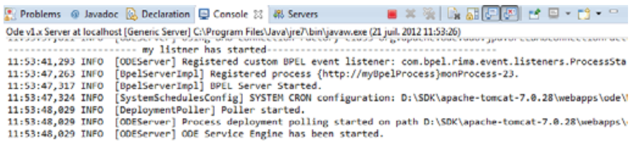
When starting the execution of BPEL process, the BPEL engine will generate progressively events with the execution of BPEL process. The BPEL engine having our Smart listener that has registered its events on startup it will launch the generated events.

Fig. 4 shows the list of the listeners registered for events in the Apache ODE. In our case, we have only one listener (Smart Listener) who is registered from the start of Apache ODE. Fig. 5 illustrates an example of starting the Smart Listener.



```
ode-axis2.event.listeners=com.bpel.rima.event.listeners.ProcessStartEventListener
```

Fig. 4 The list of listeners registered in the Apache ODE



```
Ode v1.x Server at localhost [Generic Server] C:\Program Files\Java\jre7\bin\javaw.exe (21 juil. 2012 11:53:26)
----------------------- my listner has started----------------------------------------
11:53:41,293 INFO  [ODEServer] Registered custom BPEL event listener: com.bpel.rima.event.listeners.ProcessSta
11:53:47,263 INFO  [BpelServerImpl] Registered process {http://myBpelProcess}monProcess-23.
11:53:47,317 INFO  [BpelServerImpl] BPEL Server Started.
11:53:47,324 INFO  [SystemSchedulesConfig] SYSTEM CRON configuration: D:\SDK\apache-tomcat-7.0.28\webapps\ode\
11:53:48,029 INFO  [DeploymentPoller] Poller started.
11:53:48,029 INFO  [ODEServer] Process deployment polling started on path D:\SDK\apache-tomcat-7.0.28\webapps\
11:53:48,029 INFO  [ODEServer] ODE Service Engine has been started.
```

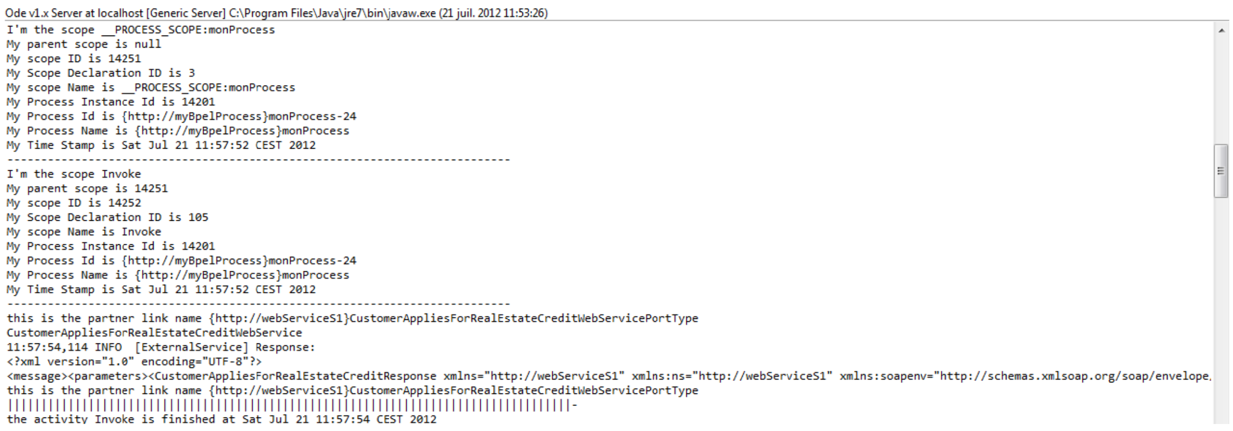
Fig. 5 The start of Smart Listener

This listener is an intelligent listener because it can cover and catch all the information generated by the BPEL process (Fig. 3, step2, the Catch component). In addition, it can retrieve events and filter them according to the needs and accurate information mentioned by the developer (Fig. 3, step3, the Filter component). When the listener detects an event that is part of the needs, it stores it in form of objects in memory; if the event is not needed, than it neglects it (Fig. 3, step 4, the Storage component). The display of the desired/needed events is in the form of a log file (Fig. 3, step 5). Table I shows a list of desired events. Multiple events are generated by the BPEL engine and the Filter allows to distinguish the events described below:

TABLE I
EXAMPLE OF DESIRED EVENTS

Event	Description
ProcessCompletionEvent	The end of the instance of the BPEL process
ScopeStartEvent	Launching the execution of a Scope
ScopeCompletionEvent	End of Scope
ActivityExecEndEvent	The end of the activity (invoke, reply,…)
ProcessMessageExchangeEvent	The instance of the BPEL process received a message from a Partner Link

Fig. 6 shows the results of steps 1, 2, 3 and 4 that run in parallel with the execution of BPEL process.



```
Ode v1.x Server at localhost [Generic Server] C:\Program Files\Java\jre7\bin\javaw.exe (21 juil. 2012 11:53:26)
I'm the scope __PROCESS_SCOPE:monProcess
My parent scope is null
My scope ID is 14251
My Scope Declaration ID is 3
My scope Name is __PROCESS_SCOPE:monProcess
My Process Instance Id is 14201
My Process Id is {http://myBpelProcess}monProcess-24
My Process Name is {http://myBpelProcess}monProcess
My Time Stamp is Sat Jul 21 11:57:52 CEST 2012
---------------------------------------------------------------------------
I'm the scope Invoke
My parent scope is 14251
My scope ID is 14252
My Scope Declaration ID is 105
My scope Name is Invoke
My Process Instance Id is 14201
My Process Id is {http://myBpelProcess}monProcess-24
My Process Name is {http://myBpelProcess}monProcess
My Time Stamp is Sat Jul 21 11:57:52 CEST 2012
---------------------------------------------------------------------------
this is the partner link name {http://webServiceS1}CustomerAppliesForRealEstateCreditWebServicePortType
CustomerAppliesForRealEstateCreditWebService
11:57:54,114 INFO  [ExternalService] Response:
<?xml version="1.0" encoding="UTF-8"?>
<message><parameters><CustomerAppliesForRealEstateCreditResponse xmlns="http://webServiceS1" xmlns:ns="http://webServiceS1" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope,
this is the partner link name {http://webServiceS1}CustomerAppliesForRealEstateCreditWebServicePortType
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||-
the activity Invoke is finished at Sat Jul 21 11:57:54 CEST 2012
```

Fig. 6 Display of Web Service execution monitoring log file

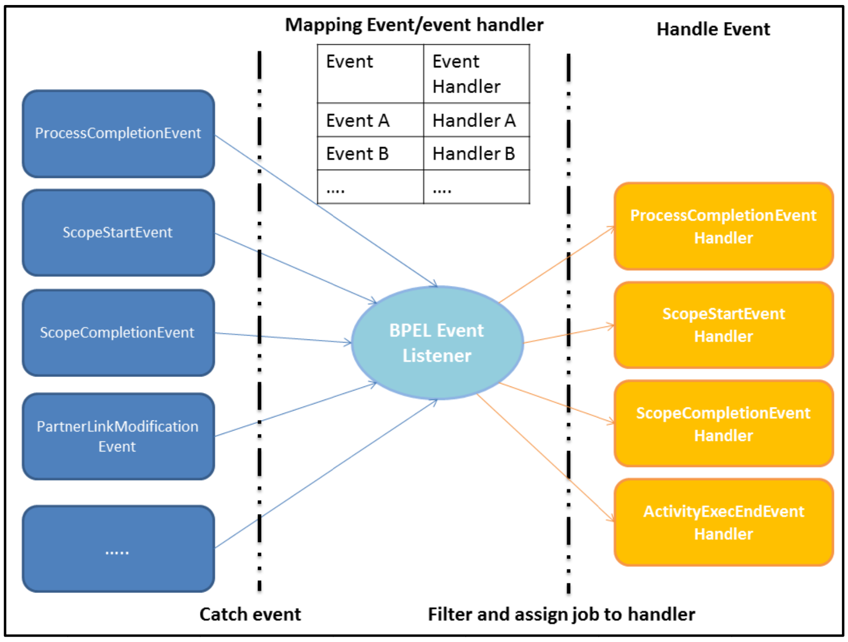


Fig. 7 The architecture of the Smart Listener

To accomplish its mission, the Smart Listener is designed according the architecture illustrated in Fig. 7. Once the Web Service Log is produced by the Smart Listener, it is used by the RTP Extractor Module to determine the appropriate response time equation of the BPEL process. For this, the BPEL equations determined by the Extractor Module (see Fig. 2) are used along with the information in the Web Service Log provided by Smart Listener. These two information pieces are confronted in a mapping phase to obtain the actual BPEL RT equation (Fig. 3, step6, 7 and 8). Once the overall final equation is determined, the relevant information (names of invoked services, used patterns and the response time of each elementary service) are saved (Fig. 3, step 9). This information will be used for monitoring the QoS of the composite Web Service.

IV. Example

To illustrate the functions of our RTP-Extractor module of our QoS monitoring framework, let us consider a simplified business process in a bank designed as a BPMN process (Fig. 8). When a customer applies for a real-estate credit, the customer’s credit rating, the real estate construction documents and the land register record are checked. All these activities are done in parallel; therefore an AND-gateway is used in the model. As the result of those assessments, the application either will be rejected or the contract is to be prepared. The XOR gateway means that only one of the activities ”Reject Application” and ”Prepare Contract” can take place. After the contract has been prepared, the process either can end or the bank might offer additional products: loan protection insurance and residence insurance. Whether loan protection insurance, residence insurance or both are offered, has to be decided case-by-case. The OR-gateway shows that only one of the activities or both of them (in parallel) can take place.

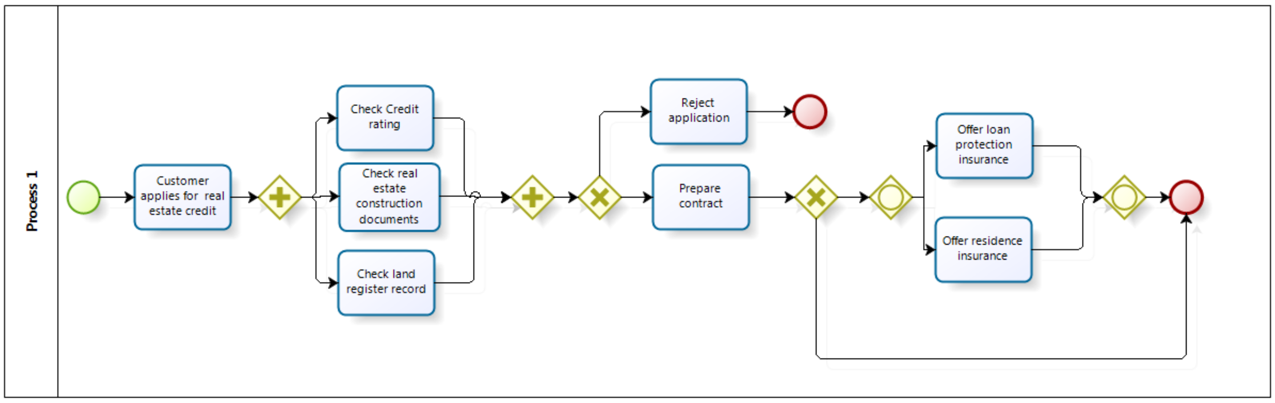

Process 1
Customer applies for real estate credit
Check Credit rating
Check real estate construction documents
Check land register record
Reject application
Prepare contract
Offer loan protection insurance
Offer residence insurance

Fig. 8 BPMN representation of the example

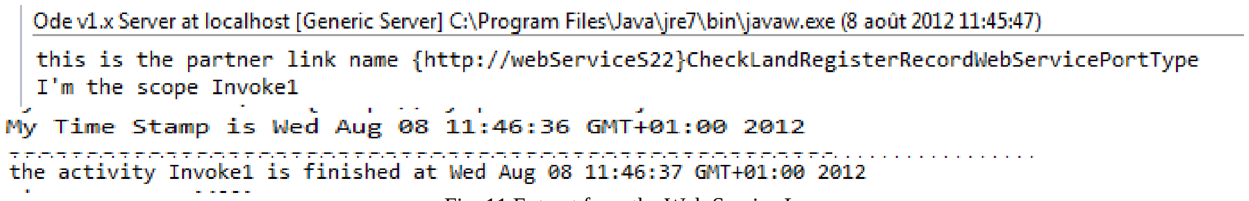
First, we need to implement this BPMN representation to obtain a BPEL process. Secondly, we deployed the resulting BPEL process in Jelastic Cloud [12]. Once deployed, the Extractor Module parsed the BPEL process to obtain automatically a design time equation (the BPEL equation); for more detail on the parsing algorithm, see [1]. For our running example, Fig. 9 represents the BPEL equation which contains the name of the services and the design patterns of the composite service. In the remainder of this section, we adopt the following notation for the various paths in the example:

- S1: Customer applies for real estate credit;
- S21: Check Credit rating;
- S22: Check real estate construction documents;
- S23: Check land register record;
- S31: Reject application;
- S32: Prepare contract;
- S41: Offer loan protection insurance; and
- S42: Offer residence insurance.

Sequence (S1, flow (sequence(S21), sequence (S22), sequence (S23)),If (sequence (S32,if (sequence (flow(sequence(S41),Sequence (S42)))))))

Fig. 9 BPEL Equation obtained during the design phase

Thanks to the Smart Listener (Sub Module of the RTP Extractor Module), we obtain the Web Service Log (Fig. 11). This log contains the names of invoked Web Services in run time and the response time of each service. We consider only the metric of response time for space limitation. From Fig. 10 we conclude that the service CheckLandRegisterRecord is invoked in run time with a response time equal to 1s (My Time Stamp – Finish Time).



```
Ode v1.x Server at localhost [Generic Server] C:\Program Files\Java\jre7\bin\javaw.exe (8 août 2012 11:45:47)
this is the partner link name {http://webServiceS22}CheckLandRegisterRecordWebServicePortType
I'm the scope Invoke1
My Time Stamp is Wed Aug 08 11:46:36 GMT+01:00 2012
the activity Invoke1 is finished at Wed Aug 08 11:46:37 GMT+01:00 2012
```

Fig. 11 Extract from the Web Service Log

Finally, the Web Service Log will be provided to The RT Extractor with the BPEL equation (obtained in the design phase) to obtain the equation in run time (BPEL RT equations) thanks to the mapping phase. Fig. 11 shows the BPEL RT Equation of the running example.

Sequence (S1, flow (sequence (S21), sequence (S22), sequence (S23)), sequence (S31))

Fig. 11 Equation obtained in Run Time

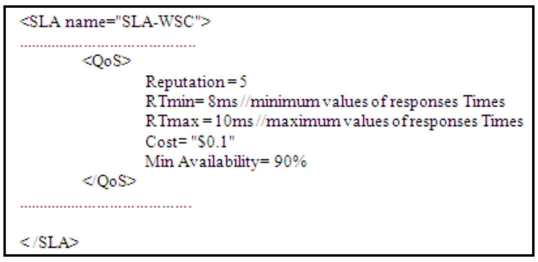
Once the BPEL RT equation is obtained, the QoS Calculator Module computes the response time of the composite web service. This calculation is based on formulas that depend on the used patterns. Table II shows the formulas for calculations according to the used patterns considered in our work.

TABLE II
RESPONSE TIME FOR COMPOSED WEB SERVICES

Patterns / Metrics	Sequence	Flow	Switch	Pick	Loop	Multi choice
Response Time	$\sum_{t=1}^{n} RT(s_t)$	$\max\{RT(s_t)\}$	$\sum_{t=1}^{n} RT(s_t)$	$\sum_{t=1}^{n} RT(s_t)$	n X RT	$\max\{RT(s_t)\}$

For example, for the sequential pattern, the response time is defined as the sum of the response times of the constituent Web services. For the flow pattern (which includes parallel, synchronization and simple merge pattern), the response time is defined as the maximum response time of the constituent Web services.

The values calculated for the composite Web service will be compared with the agreed-upon SLA (see extract of SLA, Fig. 13) to detect any potential violation. Such violation can be signaled to the Cloud provider in order to intervene with the necessary corrective measures. In the running example, the shown execution of our running example respects the agreed-upon SLA.



```
<SLA name="SLA-WSC">
.........................................
        <QoS>
                Reputation = 5
                RTmin= 8ms //minimum values of responses Times
                RTmax = 10ms //maximum values of responses Times
                Cost= "$0.1"
                Min Availability= 90%
        </QoS>
.........................................

< /SLA>
```

Fig. 12 Extract of SLA

V. CONCLUSION

Monitoring Web service composition deployed in the Cloud based on the patterns used in BPEL process remains an open research issue. In this paper we presented in detail a module of our framework QMoDeSV responsible of monitoring and detecting SLA violations in Cloud Computing environment. This framework covers both the design and execution phases of Web services. The module presented in this paper is the RT Extractor Module which is a run time module. It extracts an equation containing the names of invoked services and the actual execution path of BPEL processes forming a composite Web service. To do so, the RT extractor contains a smart listener sub-module composed of three components (Catch, Filter and Storage). The Catch component captures in a non-intruding way all the events produced by the BPEL engine during the execution of a BPEL process. The Filter component filters these events to keep only those events pertinent to the agreed-upon SLA requirements. Finally, the Storage component saves information related to the filtered events in a Web Service Log (names of invoked services and the response time of each elementary service). In its second sub-module, the RT Extractor confronts the information of the saved Web Service Log to the BPEL design-determined equations to determine the actual, run-time equation of the service. This final equation will be used to calculate the QoS of the *composite* service thanks to formulas based on the composition patterns. Finally, the composite QoS can be compared to the agreed-upon SLA to detect any potential violation.

In our future work, we will focus on the LHM (Local Host Monitor) and Lo2Hi modules responsible for managing the mapping of resource metrics gathered from Cloud environment to obtain SLA parameters.

REFERENCES

[1] R. Grati, K. Boukadi and H. Ben-Abdallah “A QoS Monitoring Framework for Composite Web services in the Cloud”, In The Sixth International Conference on Advanced Engineering Computing and Applications in Sciences (Advcomp’12). In press
[2] N. Thio and S. Karunasekera, “Automatic Measurement of a QoS Metric for Web Service Recommendation,” in Proceedings of the Australian conference on Software Engineering (ASWEC’05). IEEE Computer Society, 2005, pp. 202–211.
[3] R. Ben-Halima, K. Drira, and M. Jmaiel, “A QoS-Oriented Reconfigurable Middleware for Self-Healing Web Services,” in Proceedings of the IEEE International Conference on Web Services (ICWS’08). IEEE Computer Society, 2008, pp. 104–111.
[4] Zhang Haiteng, Shao Zhiqing , Zheng Hong “Runtime monitoring Web services implemented in BPEL”, in Proceedings of the IEEE International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE’ 11). IEEE Computer Society, 2011, pp. 228 - 231.
[5] Mingjie Sun; Bixin Li; Pengcheng Zhang “Monitoring BPEL-Based Web Service Composition Using AOP”, in Proceedings of the IEEE International Conference on Computer and Information Science (ICIS’09).IEEE computer Society, 2009, pp. 1172 – 1177.
[6] J. Shao, H. Wei, Q. Wang, and H. Mei, “A Runtime Model Based Monitoring Approach for Cloud,” in Proceedings of 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD 2010), I. C. Society, Ed. Miami, Florida: IEEE Computer Society, 2010, pp. 313–320.
[7] B.-Q. Cao, B. Li, and Q.-M. Xia, “A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture,” in Proceedings of the 1st International Conference on Cloud Computing (CloudCom’09). Berlin, Heidelberg: Springer- Verlag, 2009, pp. 644–649.
[8] S. Clayman, A. Galis, C. Chapman, M. L.R, L. M. Vaquero, K. Nagin, B. Rochwerger, and G. Toffetti. “Monitoring future internet service clouds” In towards the Future Internet - A European Research Perspective book, April 2010.
[9] M. Rak, S. Venticinque, T. a. M andhr, G. Echevarria, and G. Esnal. “Cloud application monitoring: The mosaic approach”. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 758 –763, 29 2011-dec. 1 2011.
[10] Afef Mdhaffar, Riadh Ben Halima, Ernst Juhnke, Mohamed Jmaiel and Bernd Freisleben. In Proceedings of the IEEE International Conference on Computer and Information Technology (CIT’ 11).
[11] Apache ODE http://ode.apache.org/. 2012
[12] Jelastic “http://jelastic.com/.” 2012.