A QoS Monitoring Framework for Composite Web Services in the Cloud

Rima Grati
University of Sfax
Multimedia, InfoRmation system & Advanced Computing Laboratory (Mir@cl)
Tunisia
rima.grati@gmail.com

Khouloud Boukadi
University of Sfax
Multimedia, InfoRmation system & Advanced Computing Laboratory (Mir@cl)
Tunisia
khouloud.boukadi@fsegs.rnu.tn

Hanene Ben-Abdallah
University of Sfax
Multimedia, InfoRmation system & Advanced Computing Laboratory (Mir@cl)
Tunisia
hanene.BenAbdallah@fsegs.rnu.tn

***Abstract*— Due to the dynamic nature of the Cloud, continuous monitoring of QoS requirements is necessary to manage the Cloud computing environment and enforce service level agreements. In this paper, we propose a QoS monitoring framework for composite Web services implemented using the BPEL process and deployed in the Cloud environment. The proposed framework is composed of three basic modules to: collect low and high level information, analyze the collected information, and take corrective actions when SLA violations are detected. This framework provides for a monitoring approach that modifies neither the server nor the client implementation. In addition, its monitoring approach is based on composition patterns to compute elementary QoS metrics for the composed Web service. In this paper, we illustrate our framework for the response time QoS requirement.**

Keywords- Monitoring of Web service composition; Cloud environment; Service Level Agreement ; SLA violation

I. INTRODUCTION

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet. It offers huge opportunities to the IT industry. In addition, it offers two advantages for business owners: it eliminates the requirement to plan ahead for provisioning, and it allows enterprises to start from the small and increase resources only when there is a rise in service demand. Besides these advantages, Cloud computing enables users to utilize services without the need to understand their complexity or acquire the knowledge and expertise to consume them [1]. It provides users with services to access hardware, software, and/or data.

Despite these advantages, business owners require that Cloud providers guarantee a pre-agreed upon set of Quality of Service (QoS) attributes, *e.g.*, response time, availability, security, and reliability. Face to these user requirements, and due to the dynamic nature of the Cloud, continuous monitoring of QoS attributes became mandatory to enforce Service Level Agreements (SLA) [2].

In fact, run-time monitoring has been in demand well before the Cloud. Several monitoring systems, *e.g.*, Ganglia [3], Nagios [4], MonaLisa [5], and GridICE [6] addressed monitoring of large distributed systems. However, these systems did not deal with problems induced by rapidly changing and dynamic infrastructures. This prompted the propositions of some monitoring approaches dealing with applications deployed on the cloud environment as a set of Cloud services [7][8]. Most of these approaches require modification of either the server or the client implementation code. However, to provide for independence of any Cloud provider/environment, monitoring should be performed 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 of the service composition in a Software as a Service (SaaS) cloud environment.

In this paper, we propose a framework for QoS Monitoring and Detection of SLA Violations (QMoDeSV). 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-intervening 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.

The remainder of this paper is organized as follows: Section 2, provides a background of Cloud computing and monitoring, then it overviews related works on monitoring. Section 3 presents our monitoring framework. Section 4 presents our running example. Section 5 summarizes the presented work and highlights some directions for future work.

 ISBN: 978-1-61208-237-0

II. RELATED WORK

A. On SLA and Monitoring

Many definitions are proposed for cloud computing. The generally accepted definition is the one proposed by L. M. Vaquero [10]: "Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs."

This definition clearly emphasizes the role of Service Level Agreement in the context of the Cloud. It requires a Cloud provider to be able to propose and guarantee quality of services for the provided resources. That is, a Cloud provider must be able to both establish contracts, and *continuously monitor and verify the compliance of the offered QoS with the agreed-upon SLAs.*

Once services and business processes become operational, their progress needs to be managed and monitored both to gain a clear view of how services perform within their operational environment, and to take management decisions. *Monitoring* is the procedure of measuring, reporting, and improving the QoS of systems and applications delivered by the service. Monitoring consists also of verifying at run-time that the requirements, specified by the clients and the service providers, are met during execution. The contract signed between the clients and the Cloud provider is called SLA. It includes the non-functional requirements of the service specified as QoS, obligations, service pricing, and penalties in case of agreement violations.

Flexible and reliable management of SLA agreements is of paramount importance for both providers and consumers. On the one hand, prevention of SLA violations avoid penalties that providers have to pay and, on the other hand, based on flexible and timely reactions to possible SLA violations, user interactions with the system can be minimized.

B. Works on monitoring

We classify works pertinent to monitoring into two categories: Web Service Monitoring, and Cloud Service Monitoring.

1) Web Service Monitoring

Rosenberg et al. [11] propose a monitoring approach for Web services. Their approach relies on aspect oriented and object oriented programming techniques and does not require any access to the Java source code of the service implementation. The proposed approach requires information related to the implementation of the monitored Web service (*e.g.*, endpoint and reference to WSDL). It makes use of monitoring tools such as Jpcap to monitor only latency measurement.

Repp et al. [9] present an approach to monitor performance across network layers such as HTTP, TCP, and IP. Their approach aims at monitoring QoS (in terms of network measurements) and detecting SLA violation. In the case of an SLA violation, this approach proposes to reconfigure the system at real time to minimize the substitution cost. For this, it uses the *windump* tool which requires access to the hardware for monitoring. This work monitors only network measurements.

2) Cloud Service Monitoring

Shao et al. [7] 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.

Boniface et al. [8] propose a monitoring module that collects QoS parameters of Cloud Computing. They use a monitoring application component (AC) that must be first described and registered in the application repository. The AC collects QoS parameters at both the application and technical levels. This approach is complicated and hard to install due to the description and registration of AC. Furthermore, their approach remains unevaluated.

To the best of our knowledge, none of the discussed approaches deals with monitoring Web service *composition in the Cloud*. As we describe in the next section, our approach has two additional distinctive features: computing QoS metrics in a modular way based on the patterns used in the composite service deployed in a SaaS Cloud, and collecting information (low level and high level) then comparing these metrics to SLA.

III. THE QMODESV MONITORING FRAMEWORK

The QMoDeSV framework aims at monitoring composite Web services deployed on the Cloud. Its run-time monitor is based on the workflow patterns used in the composition (BPEL process). It is designed to handle the complete Web service composition management lifecycle in the Cloud environment. The service composition lifecycle includes activities such as composite service deployment, resource allocation to the composite service, composite service monitoring, and SLA violation detection.

In our approach, we suppose that the composite Web service (*i.e.*, the BPEL process) is offered through a SaaS provider. the latter should propose the BPEL processes, the BPEL engine responsible for executing the processes instances, the database management system (DBMS) as well as the monitoring framework.

We consider that monitoring begins when the customer places a service composition request through a defined application interface to the Cloud provider. As depicted in Figure 1, the QMoDeSV framework is a two-level framework consisting of a design time module (the Extractor Module) and five run time modules (the RTP Extractor, the QoSCalculator, the Local Host Monitor, the Lo2Hi QoS Convertor, and the QoS Detector Violation). Once the

 ISBN: 978-1-61208-237-0

composite Web service is invoked, the run time modules of QMoDeSV are executed. These modules run in parallel with the BPEL instance in order to detect possible SLA violations.

The remainder of this section describes the role of each module and how it interacts with the other modules.

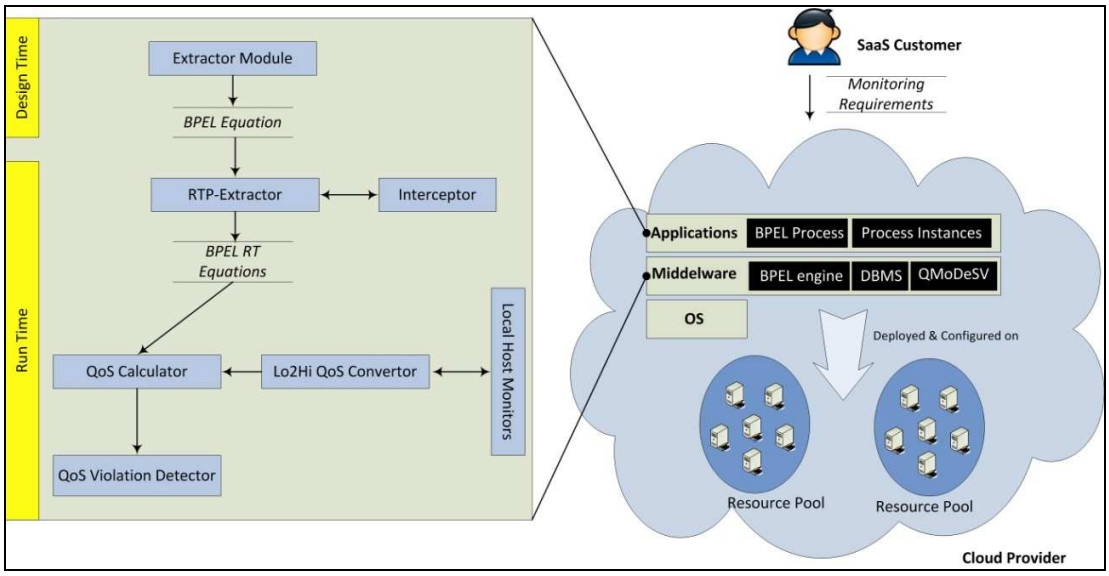

Figure 1. Overview of QMoDeSV architecture and module's interaction

A. *The Extractor Module*

Web services can be composed using different patterns that are based on the usual workflow patterns. Usually, a complex Web service composition combines two or more of these patterns.

Our Extractor module can handle the following common workflow patterns:

- **Sequence pattern**: indicates that the components Web services are executed one after the other.
- **Parallel pattern:** indicates that two or more Web services can be executed in parallel.
- **Synchronization pattern:** indicates that the process will continue after the parallel pattern of the Web service is executed.
- **Exclusive choice pattern:** is a point in the process where a path is chosen from several available paths based on a decision or process data
- **Simple merge pattern:** defines a point in the flow of execution, where two or more alternative branches are merged.
- **Conditional pattern:** indicates that there are multiple services (s_1, s_2, …, s_n) among which only one service can be executed.
- **Synchronizing merge pattern:** marks a point in the process execution, where several branches merge into a single one.
- **Multi-merge pattern:** joins two or more different services without synchronization together.
- **Loop pattern:** indicates that a certain point in the composition block is executed repeatedly.
- **Deferred choice pattern:** describes a point in the composition where some information is used to choose one among several alternative branches. When one branch of the process is enabled, the others should be disabled

The Extractor Module is responsible for analyzing the composite Web service implemented as a BPEL process. It uses the pattern detection algorithm shown in Listing 1 to extract the used patterns from the BPEL process. The output of the Extractor Module is a design time equation containing the name of the components Web services as well as the patterns used for connecting the flows between these components.

 ISBN: 978-1-61208-237-0

Listing 1. The pattern detection algorithm used by the Extractor Module

```
Input = ( BPEL process BP) // the composite Web service implemented using a BPEL process
Output = BPequation    //design time BPEL equation
Func
    GetBPTags (BPEL process BP) return ListOfTags  // this function parse the BPEL
    document and put each tag in a list box
    FilterUsedPatterns (ListOfTags) return ListOfUsedPatterns  // this function removes from
    the list box the set of used patterns
    Get WebServicesNames (ListOfTags) return  ListOfWebServices  // this function retrives
    the names of invoked Web services
    GetBPEquation (ListOfUsedPatterns, ListOfWebServices) return BPequation  // this
    function build the BPEL equation based on the identified pattern and the invoqued Web services


BEGIN
    String patterns[] = {"<sequence", "</sequence", "<receive", "<invoke", "<flow", "</
    flow", "<switch", "</switch", "<while", "</while", "<pick", "</pick", "<link", "</link"};

    for each BPEL process BP do
    {          List<String> ListOfTags  = GetBPTags (BP);
               List<String> ListOfUsedPatterns = FilterUsedPatterns (ListOfTags);
               List<String> ListOfWebServices = Get WebServicesNames (ListOfTags);
               String BPequation=GetBPEquation (ListOfUsedPatterns, ListOfWebServices);
               Print (BPequation);
    }

END
```

B. *The Run Time Extractor Module*

The Run Time Extractor Module (see Fig. 2) "watches" the executed services and refines the equation obtained in the design time into a run time equation. The run time equation represents the execution path of the BPEL process instance. It is derived according to the patterns extracted at the design time. This module intercepts information about the executed service through the Monitoring thread: this latter is the extension of the API apache ODE (Orchestration Director Engine) [12]. The monitoring thread interacts with the BPEL engine to check the process states and informs the Run Time Extractor module to do a comparison between old and new process (see Fig. 2). After that, the Run Time Extractor Module extracts BPEL nodes to establish the execution graph of BPEL. Once the run time extraction path is done, the run time equation is established.

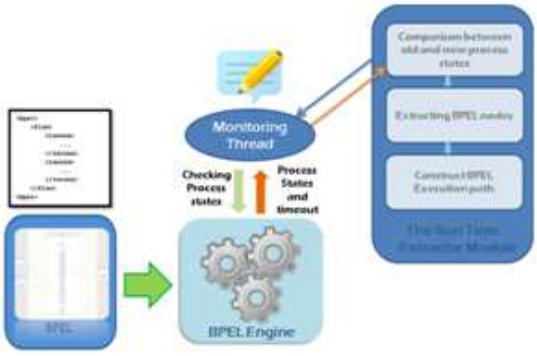

Figure 2. The Run Time Extractor Module

C. *The QoS Calculator Module*

This module computes QoS metrics for the composite Web services. In its computations, it uses the values of the constituent services and the composition pattern.

We illustrate how the QoS calculator functions use the following QoS metrics:

- **Response Time (RT):** the time interval between when a service is invoked and when the service is finished.
- **Service Cost (C):** the price that a service requester has to pay for invoking the service.
- **Throughput (T):** represents the number of Web service requests at a given time period.
- **Reliability (R):** the probability that a request is correctly responded within the expected time.

The overall Web service QoS is derived based on the values collected locally for each constituent service and the composition pattern. For this, we adapt metrics proposed in [13] and [14]. The adapted metrics are instantiated by the QoS calculator based on the composition pattern detected by run time extractor.

Table 1 summarizes the QoS metrics we adapted to account for the composition patterns. To establish these metrics, we noted constituent Web services as s_1, s_2… s_n and the Web service composition that includes these services as $S(s_1, s_2, \ldots, s_n)$. For the conditional pattern, we denote p_i the probability that a service s_i be selected. Finally we denote as $SO(s_i, p_i)$ the selection operation for the conditional patterns, which selects the service s_i with an execution probability p_i.

 ISBN: 978-1-61208-237-0

TABLE I. METRICS FOR COMPOSED WEB SERVICES

Patterns	Response Time	Throughput	Reliability	Cost
Sequence	$\sum_{t=1}^{n} RT(s_t)$	$\frac{1}{\sum_{t=1}^{n} \frac{1}{T(s_t)}}$	$\prod_{t=1}^{n} R(s_t)$	$\sum_{t=1}^{n} C(s_t)$
Parallel Synchronization Simple merge	$max\{RT(s_t)\}$	$\min\{T(s_t)\}$	$\prod_{t=1}^{n} R(s_t)$	$\sum_{t=1}^{n} C(s_t)$
Exclusive choice Deferred choice	$RT(SO(s_t, p_t))$	$T(SO(s_t, p_t))$	$R(SO(s_t, p_t))$	$C(SO(s_t, p_t))$
Multi-choice/ conditional Synchronizing merge	$\max(RT(SO(s_t, p_t)))$	$\min(T(SO(s_t, p_t)))$	$\prod_{t=1}^{n} R(SO(s_t, p_t))$	$\sum_{t=1}^{n} C(SO(s_t, p_t))$
Loop	$n\ X\ RT$	$\frac{1}{\sum_{t=1}^{n} \frac{1}{T}}$	R^n	$n\ X\ C$

For example, when considering the exclusive choice pattern, the response time is calculated by the selection operation, which selects one of the *n* possible Web services. In particular, it is defined as $RT(SO(s_t, p_t))$:. This pattern selects the service s_t with a probability p_t at design time. However, since at run time, the execution path is clear and this metric will be adapted by the QoS calculator into: $\sum_{t=1}^{n} RT(s_t)$.

D. The QoS Violation Detector

The QoS Violation Detector accesses the mapped metrics repository to get the mapped SLA parameters. These parameters are compared with the calculated values obtained from the QoS Calculator. In the case of a violation (none respect of SLA), it dispatches notification messages to the customer/provider to alert about the violation. An example of SLA violation threat can be an indication that the process consumed 5*ns* for a response time while the agreed response time is 3*ns*.

E. The LHM and Lo2Hi QoS Convertor

The Local Host Monitor (LHM) process monitored values and is capable of measuring both hardware and network resources. It can be configured to access different virtual hosts at the same time to collect locally monitored values.

As shown in Figure 1, the Lo2Hi QoS Convertor interacts with two components: the LHM which monitors the resources, and the QoS Calculator which calculates the global obtained metric. Resources are monitored by the Local Host Monitor using arbitrary monitoring tools such as Gmond from Ganglia project [3]. Low level resource metrics include outbytes, inbytes, and packetsize. Based on the predefined mapping rules stored in a database, monitored metrics are periodically mapped to the SLA parameters. These mapping are obtained in a similar way to those in Grids where workflow processes are mapped to a Grid service in order to ensure their quality of service [15].

IV. EXAMPLE

In this section, we illustrate the functioning of the Extractor Module and the Run Time Extractor Module. Our running example deals with the recruitment of an employee, which we modeled in BPMN (Fig 3). We consider a company named AdminCompany and a new employee called Joan. When Joan arrives to AdminCompany, his information should be collected and it is necessary to perform many activities in parallel such as, grant access to company information, sign some legal documents and set up her workstation. After that, the mode of remuneration should be selected either in cash or by check or by bank transfers.

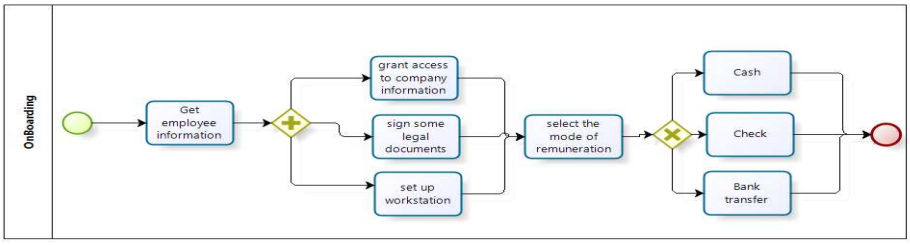

Figure 3. BPMN representation of the example

For space limitation, we consider only the metric of response time (RT). First, the BPEL process corresponding to this example is implemented. Then the Extractor Module parses the BPEL process to extract the design time equation that represents the used patterns (Listing 2).

Listing 2. Equation obtained in Design Time

Sequence (get employee information, **Flow** (grant access to company information, sign some legal documents, set up workstation),
Sequence (select the mode of remuneration), **Switch** (cash, check, bank transfer))

This equation will be refined through the Extractor Run Time Module to obtain a run time equation, corresponding to the set of invoked services as well as the patterns used for connecting the flows between them (Listing 3).

Listing 3. Equation obtained at Run Time

Sequence (get employee information, **Flow** (grant access to company information, sign some legal documents, set up workstation),
Sequence (select the mode of remuneration), **Sequence** (check))

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 (grant access to company information, sign some legal documents, set up workstation).

The values calculated and obtained for the composite Web services will be compared to the agreed SLA

V. CONCLUSION AND FUTURE WORK

Monitoring Web services composition published in Cloud based on the patterns used in BPEL process remains an open research issue in Cloud computing. In this paper, we presented QMoDeSV, a novel architecture for monitoring and detecting SLA violations in Cloud computing environment.

Our framework is designed to handle the complete Web service composition management lifecycle in the Cloud environment and SLA violation detection. In addition, QMoDeSV proposes a non-intervening 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 our framework to detect potential SLA violations. Our framework can be very helpful for service providers, who can then take corrective actions to improve their services and to avoid penalties.

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

REFERENCES

[1] R. Buyya, C. S. Yeo, and S. Venugopa, “Marketoriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities” In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08), pp102-110

[2] A. Al-Flasi and M.A. Serhani, “A Framework for SLA-Based Cloud Services Verification and Composition”, International Conference on Innovations in Information Technology, Abu Dhabi, UAE, April 2011. pp. 363-370

[3] M. L. Massie, B.N. Chun, and D. E. Culler, “The ganglia distributed monitoring system: Design, implementation and experience,” Parallel computing, vol. 30, pp. 200, 206

[4] “Nagios.” http://www.nagios.org/. [retreived: 7,2012]

[5] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, “MonALISA : A distributed monitoring service architecture,” in Proceedings of CHEP03, La Jolla, California, 2003. pp. 214-220

[6] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, and M. C. Vistoli, “GridICE: A monitoring service for grid systems,” Future Gener. Comput. Syst., vol. 21, no. 4, pp. 559–571, 2005

[7] 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.

[8] M. Boniface, B. Nasser, J. Papay, S. C. Phillips, A. Servin, X. Yang, Z. Zlatev, S. V.Gogouvitis, G. Katsaros, K. Konstanteli, G. Kousiouris, A. Menychtas, and D. Kyriazis, “Platformas- a-Service Architecture for Real-Time Quality of Service Management in Clouds,” in Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services (ICIW ’10). Washington, DC, USA: IEEE Computer Society, 2010, pp. 155–160

[9] N. Repp, R. Berbner, O. Heckmann, and R. Steinmetz, “A Cross-Layer Approach to Performance Monitoring of Web Services,” in Proceedings of the Workshop on Emerging Web Services Technology. CEUR-WS, Dec 2006. pp. 140-148

[10] Vaquero L M, Rodero-Merino L, Caceres J , and Lindner M, “A Break in the Clouds, Towards a Cloud Definition”, Computer Communications Review, 2009, Vol. 39, No. 1, pp. 50-55.

[11] F. Rosenberg, C. Platzer, and S. Dustdar, “Bootstrapping Performance and Dependability Attributes of Web Services,” in Proceedings of the IEEE International Conference on Web Services (ICWS’06). IEEE Computer Society, 2006, pp. 205– 212.

[12] Apache ODE http://ode.apache.org/ [retreived: 7, 2012]

[13] H. San-Yih, Wang H, S.Jaideep , and P. Raymond. “A probabilistic QoS Model and computation Framework for Web Services based workflow” In Proc of ER2004, pages 596-609, Sanghai, November 2004. pp. 254-260

[14] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. “QoS Aggregation for Web Service Composition using Workflow Patterns” EDOC ’04 Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International. pp. 52-59

[15] D. Kyriazis, K. Tserpes, A. Menychtas, A. Litke, and T. Varvarigou, An innovative workflow mapping mechanism for grids in the frame of quality of service, Future Generation Computer Systems 24 (6) (2008) pp. 498–511.

 ISBN: 978-1-61208-237-0