Constraints-Based Response Time for Efficient QoS in Web Services Composition

Bassirou Gueye*, Ibrahima Niang*, Bamba Gueye*, Mohamed Ould Deye*, Yahya Slimani**
* *Université Cheikh Anta Diop de Dakar, Senegal*
bassirou.gueye@ucad.edu.sn, iniang@ucad.sn, bamba.gueye@ucad.edu.sn, mohamed.ouldeye@ucad.edu.sn
** *Université de Tunis, Tunisia*
yahya.slimani@fst.rnu.tn

***Abstract*—Web Services Composition (WSC) is a paradigm for enabling application integration within and across organizational boundaries. Nowadays, the Quality of Service (QoS) that WSC should offer becomes a priority for service providers and an exigence for customers. Therefore, the response time of a web services composition is a crucial problem.**

However, previous works proposed only analytical formulas with respect to response times of few *BPEL* constructors. They do not provide any mechanisms or tool in order to analyze and verify the time constraints.

In this paper, we propose a Framework called QoS4WSC that is able to evaluate constraints-based response time in WSC. In order to provide a model for estimating the effective response time of a service composition, we use a parsing file composition technique. Since elementary web services is based on a best effort approach, we propose a new architecture which takes into account the response time of a given service. Our architecture is composed by two components. The first one pre-determines the response time of elementary web services and the second one verifies the QoS constraints of WSC.

***Keywords*-Web Services Composition; Quality of Service; Response Time; BPEL; Parser;**

I. Introduction

The main challenges of Web Services Composition (WSC) in environment like Internet are to ensure high quality execution according to response time. They do not exist tools even fewer models that allow designers to evaluate the efficiency of the proposed QoS. There is a growing interest for QoS verification techniques which enable designers to test and improve WSC time constraints.

In fact, web services are emerging and promising technologies for development, deployment and integration in Internet applications. A major advantage of web services over traditional middleware (CORBA, DCOM and XML-RPC) is the contribution of services interoperability on the Internet.

Firstly, Web services which are based on XML provide an infrastructure for describing web services. In such case, we use the *Web Services Description Language* (WSDL) [1].

Secondly, after the description of the web services, we can use the *Universal Description, Discovery and Integration* [2] in order to publish the elementary web service that has been just created. After the description and the publishing phase, we can invoke the published services. Indeed, we use the *Simple Object Access Protocol SOAP* [3].

Web services are designed to promote SOA (Service Oriented Architecture) [4], integrating highly distributed complex heterogeneous systems, that can cooperate without resorting to a specific and costly integration. It exists different web-based applications that can perform specific tasks.

Nevertheless, with respect to few applications, it is necessary to combine a set of basic web services in order to create a composite web services. Therefore, we built new services that enable to provide more complex requirements. This new composite web service can be viewed as a single web service from the point of view of the customer.

Although it exists few works around service compositions, like the standard *BPEL4WS* [5] (Business Process and Execution Language For Web Services), the management of QoS in service compositions needs new solutions that are either flexible, or reusable, and thus can provide more abstraction.

The characteristics of QoS in web services should respect the agreement between the service provider and the user. This agreement is defined in a SLA (Service Level Agreement) [6], [7] which defines the supply and the demand the of these two entities. The specifications of this document should be verified during the execution of the web service composition through a monitoring mechanism.

The QoS in web service can be monitored by the use of different metrics such as response time, availability, reliability, cost, etc. It is worth noticing that the response time is the most important metric with respect to WSC. An efficient management of the response time induces the availability at any time of the WSC. Indeed, the invocation of a web service can be triggered anywhere on the Internet and the response should be given in a short time interval. In such case, this implies that the service is available. Therefore, in order to take into account the QoS, it is mandatory to develop new tools for its management.

In this paper, we propose **QoS4WSC** (Quality of Service For Web Service Composition) framework that enables to reduce the needed amount of time to get a response during the invocation of a web service composite. Note that, our model fully covers the last version of *WS-BPEL 2.0* which is adopted as OASIS Standard. QoS4WSC allows us to predetermine the response time of elementary web services as well verifies the fixed constraints of a WSC.

The rest of the paper is organized as follows. In section II, we present previous works on WSC. Section III illustrates our QoS4WSC framework that enables to evaluate QoS constraints . The section IV describes the different constructors used by QoS4WSC tool for estimating and verifying the response time of a web service composition. In section V, we validate and evaluate our proposition with respect to previous works. Finally, Section VI concludes this work and investigates some future works.

II. Related work

Web services are an attracting area that interest many researchers and industrial organizations. So, the authors of [8] propose different algorithms in order to aggregate QoS properties for some standardized workflow patterns of web service compositions. These properties include upper and lower bounds of execution time and cost, as well throughput and uptime probability.

Rud et *al.* in [9], have proposed analytical formulas in order to take into account the response time of various BPEL constructors. To achieve these objectives, they use mathematical models by adopting a formalism ratings based on different assumptions. A good overview of these approaches is given in [10].

Haddad et *al.* in [11] have proposed an extension of web service composite model presented by Menasce [12]. Indeed in [12] a composite web service is considered as a set of tasks running in parallel. The authors of [11] argue that the model described by Menasce is acceptable only if all web services participating in the composition can be executed independently. Note that, this assumption is not generally true.

Therefore, with respect to Menasce's work [12], Haddad et *al.* propose analytical formulas for the response time of the following constructors: <sequence>, <switch> and <flow>. It is worth noticing that this paper [11] do not propose a model for WSC with respect to response time.

Haddad et *al.* in [13] improve their work already done in [11]. To overcome the shortcomings noticed in [11], Haddad et *al.* propose new assumptions such as the number of basic web services invoked can be variable, as well the response time of web services follow exponential and heavy-tailed model [13]. The choice of these mathematical models is motivated by the fact that the authors of [14] have shown that the response time of basic services can be modeled by such a distribution.

III. QoS4WSC: Constraint-Based response time for WSC

A. Background on BPEL constructors

The BPEL process is formed by constructors (simple or structured), linked by different workflow. Based on a survey of BPEL 2.0 constructors specification, we describe some of them, which are called "key constructors". It is worth noticing that a "simple" constructor should be use itself, whereas a "structured" constructor can call other constructors that can be structured or simple.

For instance among "key constructors" we can cite : <receive>, <invoke>, <reply>, <wait>, <onAlarm> and <repeatEvery>. In fact, these key constructors enable to define a waiting time and/or processing time during the execution of an elementary service or a WSC.

The following constructors are defined as structured : <flow>, <sequence>, <scope>, <If> with its sub-elements <elseif> and <else>, <repeatUntil>, <while>, <forEach>, <pick> with its sub-elements <onMessage> and <onAlarm>, <eventHandlers> with its sub-elements <onEvent> and <onAlarm> and the constructor <repeatEvery> which is optional.

B. Overview of the QoS4WSC architecture

Since previous works have proposed, for few BPEL constructors [5], analytical formulas in order to take into account the response time, we propose here a new framework that enables to estimate the response time as well to reduce it according to WSC. Nevertheless, offering a short execution time for WSC is not simple.

Indeed, elementary web services, as described by WSDL, are conceptually limited to relatively simple features that are reported as a collection of operations. Moreover, existing public directories have not yet integrated this response time criterion in the representation of the services that they do provide. In such case, most basic web services do not explicitly expose their QoS.

Following this lack of QoS, it is mandatory to built modules that provide a preliminary indication of the response time for basic services. Therefore, the time that one can wait for a given WSC is known in advance. Our proposed QoS4WSC middleware is illustrated in Figure 1 and it is formed by two main modules. The goal of the first module, called "Module 1" (Figure 1), is to estimate the response time of web services. The second one, called "Module 2" (Figure 1) verifies whether the QoS constraints specified, for instance in a given SLA, are achieved.

The different steps, (1, ..., 8), as labelled in Figure 1 enable to estimate the response time of an elementary web services. During each step we have the following tasks: (1) the provider (eg., engineer in Figure 1) of the WSC connects to a directory in order to seek the potential services that can participate in a composition; once a desired service is found, he retrieves the address of the WSDL interface file that owns this service. It should be noted that the web service will be invoked from this address. Afterwards, during the step 2, the provider gives the *URL* of the WSDL file to the Module 1.

Following that, the module 1 (Figure 1) estimates the response time of the selected web service. The goal of this processing is to generate automatically a set of class called "servicename", "servicenameLocator", "servicenameSoap",

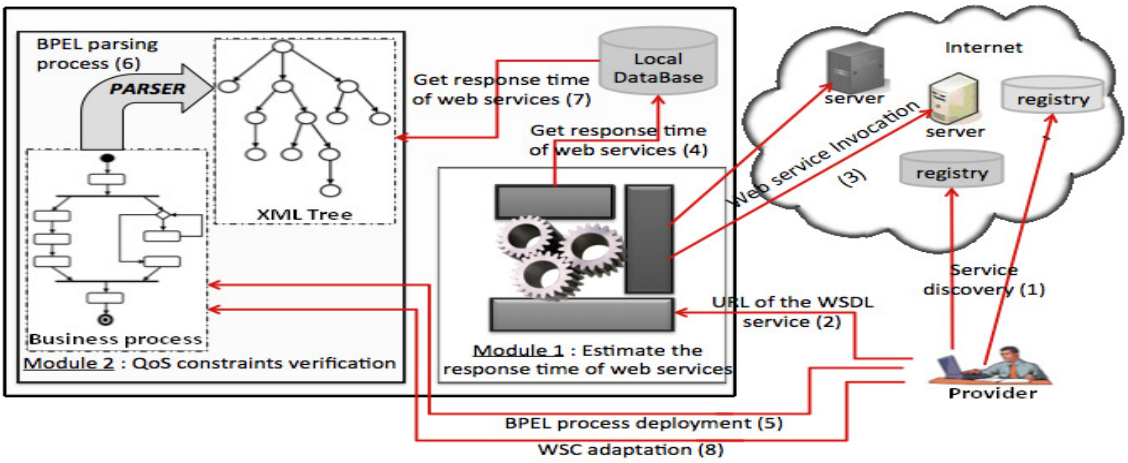

Figure 1. QoS4WSC middleware.

"servicenameSoapProxy" and "servicenameSoapStub" from the URL of the WSDL file obtained previously. These five classes are used in order to estimate the response time of a given elementary web service.

In fact, a main class will be created by considering the previous five classes generated before. In order to obtain optimal results of response time, we submit multiple requests to the server that hosts the web service. Therefore, for each request req_i, we obtain a response time T_i. The average response time of a web service ws_i, $1 < i < k$, is obtained as follows : $T[ws_i] = \sum_{i=1}^{k}(\frac{T_i}{req_i})$

It is worth noticing that, the estimated response time for each web service is saved at the local database of the QoS4WSC middleware (Figure 1). In such case, this value can be reused if the web service is re-invoked shortly.

Afterwards, during step 5, the provider of the WSC should rank the different web services following their precedence and then sends this new WSC to the verification module (Module 2 in Figure 1) in order to verify if the constraints in term of response time is respected. In order to verify, if the time constraints are respected, we consider the parsing file composition technique (step 6) described in more details in Section IV. Put simply, the main goal of the module 2 is to estimate the response time of web service composition as well to verify the QoS constraints by retrieving the response time of elementary web service from the local database (step 7). Furthermore, we should adapt the WSC if the QoS constraints are violated (step 8). In so doing, we can change either the elementary web services that form the WSC, or change the fixed constraints.

IV. Using QoS4WSC for the verification of QoS constraints

In contrast to previous works [9], [11], [13], we propose an automatic tool that verifies QoS constraints for WSC. Our approach is based on file composition parsing. Figure 2 depicts our proposed algorithm in order to parse a BPEL file. Note that the BPEL file is a XML file. We recall that the BPEL file contains the execution order of the web service composition. In fact, the QoS verification module (Figure 1) receives as input a fixed response time as QoS constraint and a BPEL process that describes the WSC (Figure 2). Inside the QoS verification module, the XML tree of the process is created (Figure 2).

A XML file appears as an upside-down tree: if the XML tree is well generated, it has a root that has branches (<partnerLinks> and <sequence>) as illustrated in Figure 2. The <partnerLinks> contains the list of partners (web services) that will participate in the WSC, whereas <sequence> defines the execution order of the WSC as it was specified in the BPEL file. Nevertheless, we should verify if <partnerLinks> and <sequence> nodes are well defined. In such case, the response time counter, identified by the label "ResponseTime" in Figure 2, is initialized to zero. Otherwise, the parsing is stopped (Figure 2) and an error exception is sent.

If the constructors located in the <sequence> node match one of the key constructors defined in Section III-A, we call the related manager in order to estimate the corresponding amount of time. It should be noted that a manager is used in order to estimate the response time of a given elementary service. Furthermore, a manager is related to a key constructor, or a set of key constructors. The set of managers that are used in our QoS4WSC middleware are listed in Section IV-A to IV-H. Following a given manager, if the response time of an elementary web service is violated, the parsing is stopped (Figure 2) and a time-constraint violation is sent to the provider of the web service.

In the following, we describe the managers used in the algorithm illustrated in Figure 2. Also, we adopt the following formalism:

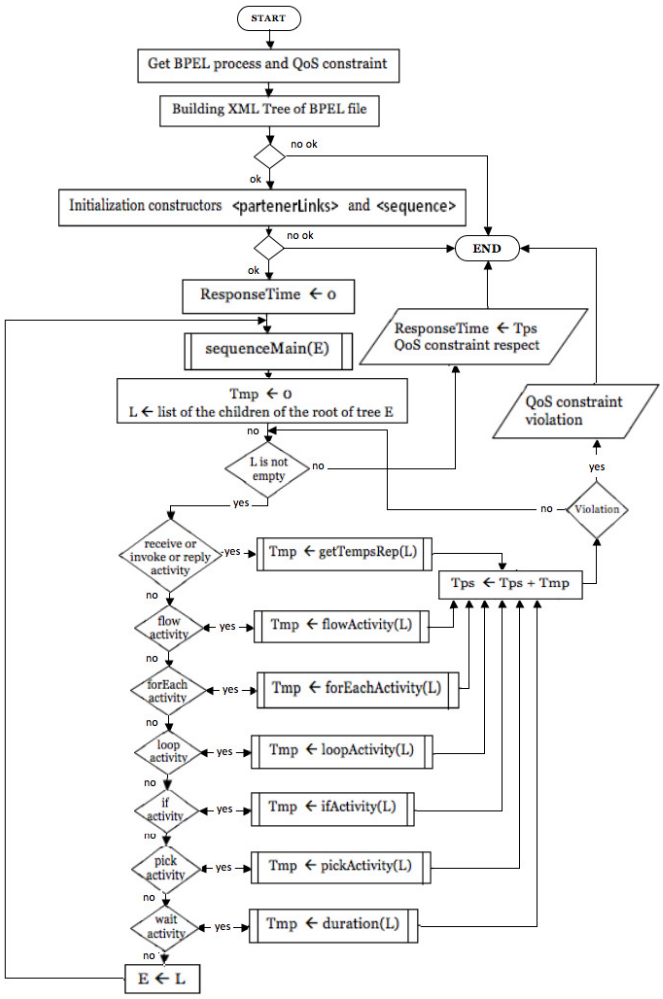

Figure 2. Algorithm for parsing a BPEL process.

$\sum_{i=1}^{n} P[c_i] = 1$; with probability $p[c_i]$ of entering in the branch c_i

T[a]: defines the response time of the activity *a*

T_{wait}: defines the waiting time

T_{body}: defines the amount of time needed to execute one iteration in a given loop

k: defines the number of iterations of a given loop

T_{scopeX}: execution time of scope *X* of one activity

$T_{repeatEvery}$: execution time of one activity *repeatEvery*

A. "getResponseTime" *manager*

This manager makes the correspondence between two attributes (`partnerLink` defined by the caller constructor and `name` defined by the `partnerLink` constructor) to know the invoked web service. Another matching is done between the `name` attribute and `partnerLinkType` in order to know the `operation` attribute of the service. This allow to access to the database in order to retrieve the response time of a given web service.

B. "flowActivity" *manager*

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 2), is a <flow> constructor. The procedure enables to run simultaneously a set of activities. The response time $T[a]$ of this procedure is: $T[a] = \max(T[a_1], ..., T[a_n])$.

Furthermore, during the parsing of the BPEL file, if a key constructor like <receive>, or <invoke>, or <reply> is found we call the ***getResponseTime*** manager. Otherwise, if the <sequence> constructor that specifies a sequential execution is found, we call the ***sequenceActivity*** manager.

C. "forEachActivity" *manager*

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 2), is a <forEach> constructor. In such case, we should perform all activities located in the sub-constructor <scope> exactly k times, where k means the number of iterations specified in the loop . The response time of this activity is given by $T[a] = k \times T_{body}$.

In order to determine k, we fetch the "`parallel`" attribute value. If the value of the parallel attribute is set to "no", then the number of iterations ranges from *<startCounterValue>* to *<finalCounterValue>* parameters which are defined in the "forEach" constructor. Otherwise, If parallel attribute is set to "yes", then k is equals to 1 and all iterations should be execute in parallel. To determine T_{body}, we call the ***sequenceActivity*** manager that receives as input argument the <scope> sub-constructor.

D. "loopActivity" *manager*

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 2), is a <while> or <repeatUntil> constructor. It should be noted that for both constructors, the number of iterations is not known in advance. Therefore, previous works like [15], [16] have proposed to give as response time the overall execution time of the loop. They do not take into account the number of iterations inside the loop. This approach presents several drawbacks in the sense that the response time can be overestimated or underestimated.

To overcome the limitations of previous works, we propose a new approach in order to estimate the response time. The response time is obtained by $T[a] = k * T_{body}$.

In our approach the value of k is related to the amount of time specified in the QoS constraints. k's value is not fixed in contrast to previous works like [17], [18]. The values that k can take depend on the remaining execution time of the WSC. More the remaining execution time, following QoS constraints, is high and more the value of k is important. In such case, we should estimate the value of k in order to estimate the response time.

E. "ifActivity" *manager*

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 2), is a <if> constructor. Note that a "ifActivity" manager can have the following branches <elseif> or/and <else>. Therefore, the response time is equal to $T[a] = \sum_{i=1}^{n}(T[a_i] * p[c_i])$.

If this constructor presents several branches the response time can be estimated as $T[a] = \max(T[a_1], ..., T[a_n])$ where $T[a_i]$ means the elapsed time during the block i.

F. "pickActivity" *manager*

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 2), is a <pick> constructor. The goal is to wait the first message defined by the <onMessage> constructor or a signal marking the end of a timer defined in <onAlarm> constructor. The <pick> constructor is formed by several branches. Each branch refers to an event that can occur.

Therefore, the response time can be estimated as follows $T[a] = \sum_{i=1}^{n}(p[c_i] * (T_{wait} + T[a_i]))$

Since, it is not possible to known in advance when an event will be trigged, the response time is obtained by But we do not know when an event is going to be trigged. For this case, the response time will be using this equation $T[a] = T_{wait} + \max(T[a_1], ..., T[a_n])$

G. "duration" *manager*

This manager is called either by a <wait> constructor, or a <onAlarm> constructor.

The attribute `for` specifies the delay before the awakening of the process (in the case of <wait>), or the triggering of an alarm (in the case of <onAlarm>).

The date and the deadline are specified by assigning a value to the `until` attribute.

H. "sequenceActivity" *manager*

This manager is called by other managers each time that a set of activities should be run sequentially.

The goal of <eventHandlers> constructor is to wait the first event <onEvent> or a signal marking the end of a timer defined in <onAlarm>. If a given event does not occur until the expiration of the deadline, then the associated activities with respect to this <onAlarm> will be executed.

In this case, if the optional constructor <repeatEvery> is defined, then the procedure will be repeated until the parent scope of the <eventHandlers> activity remains active. The response time expression is given by $T[a] = \sum_{i=1}^{n}(p[c_i] * (T_{wait} + k * (T_{scopeAlarm} + T_{repeatEvery})))$ where k is equal to $[\frac{T_{scopeParent} - T_{wait}}{T_{scopeAlarm} + T_{repeatEvery}}]$.

If the "repeatEvery" is undefined (*i.e*, $T_{repeatEvery} = 0$) then $k = 1$.

Note that "eventHandlers" activities are invoked in parallel with other activities during the processing phase. It's the reason why we do not take into account this constructor during the computation of the response time. In fact, the execution of this constructor is embedded inside other key constructors.

V. Implementation and evaluation

We implemented our framework by using the programming language Java (JDK 1.6) and the Eclipse IDE (version 3.4.2). To make this practical environment, we used the plugins `WTP` (Web Tools Platform) version 3.2.0, `EMF` (Eclipse Modeling Framework), `GEF` (Graphical Editing Framework), `GMF` (Graphical Modeling Framework) and BPEL Visual Designer. We considered the following tools like Tomcat (version 6.0.20), Axis2, PostgreSQL (version 8.4), Apache ODE (Orchestration Director Engine) with 1.3.4 version, and JDOM parser (version 1.6.1) which is an open source Java API whose purpose to manipulate an XML document.

To evaluate our proposal, we consider the BPEL process of a travel plan which is described in Figure 3. We use this example in order to evaluate and compare our proposal with respect to related work. We use as QoS metric the response time of a web service composition.

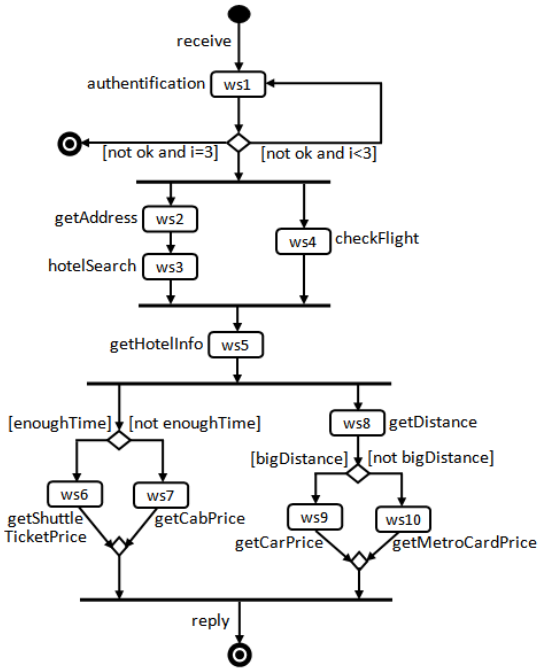

Figure 3. Travel Plan process.

A. Discussion

The techniques proposed by Haddad et `al.` in [11], [13] do not allow to estimate the response time of the composition

depicted in Figure 3. In fact, they do not take into account the existence of *loop* constructor. Therefore, we argue that their proposed techniques in [11], [13] work only if the WSC is formed by the following constructors: *sequence*, *flow* and *switch*. However, if a composition consider only these three constructors, the estimated response time will be done manually. Nevertheless, for complex WSC it is not possible to do it manually.

Rud et al. [9] proposed to estimate the response time of this composition (Figure 3) by using a manual approach due to the lack of models and tools. In so doing, they aggregate the different activities manually. For a complex WSC, this solution is not appropriate due to high response time.

With our QoS4WSC approach, we can automatically find in a few seconds the response time of a given WSC. We performed our tests with a computer with a Core Duo 2.16 GHz frequency and 2 GB of RAM. With this example of composition, despite all of parsing details we obtained a response time of 6 seconds.

Note that our proposed middleware is generic in the sense that it can handle any type of composition, and regardless of its complexity and the number used constructors.

VI. Conclusions

Nowadays, it is mandatory to take into account QoS constraints in WSC. Therefore, in order to achieve a high quality delivery of service composition, we proposed QoS4WSC which is a middleware for QoS verification constraints. The main goal of QoS4WSC is to allow WSC's providers to evaluate the response time of their composition. In such case, we can achieve better efficiency with respect to the amount of time that is need to retrieve the results of a given WSC. In addition, the proposed tools can help designers or suppliers in decision-making when such agreements are established. Furthermore, QoS4WSC allow to these actors to respect the contracted SLA (e.g, reduce or avoid QoS constraints violation).

As future works, our framework can be improved by developing a plug-in that can inherit all features of our QoS constraints solution. The objective is to integrate this plug-in in the palette to create a BPEL process. On the other hand, the introduction of formal semantic in the life cycle of web services composite can be considered.

References

[1] R. Chinnici, J.-J. Moreau, S. Weerawarana, and R. Arthur, "Web services description language (wsdl) version 2.0," W3C Recommendation 26, June 2007.

[2] L. Clement, Systinet, A. Hately, C. Von Riegen, and T. Rogers, "Computer associates. uddi version 3.0.2, oasis specification," October 2004.

[3] M. Gudgin, M. Hadley, J.-J. Moreau, and H. Frystyk Nielsen, "Simple object access protocol (soap) v 1.2. w3c," July 2001.

[4] F. Cubera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services : An introduction to soap, wsdl, and uddi," vol. 6, no 2, pp. 86–93, July 2006.

[5] "Web services business process execution language version 2.0," OASIS standard, April 2007.

[6] L. Jie Jin, V. Machiraju, and A. Sahai, "Analysis of service-level agreement for web services," HPL-2002-180, Tech. Rep., 2002.

[7] A. Daniel Menascé, "Mapping service-level agreements in distributed applications," *IEEE Internet Computing*, pp. 100–102., September-October 2004.

[8] M. C. Jaeger, G. R. Goldmann, and G. Muhl, "Qos aggregation for web service composition using workflow patterns," *Proc. Eighth IEEE International Enterprise Distributed Object Computing Conference*, pp. 149–159., 2004.

[9] D. Rud, M. Kunz, A. Schmietendorf, and R. Dumke, "Performance analysis in ws-bpelbased infrastructures," 2007.

[10] F. Van Breugel and M. Koshkina, "Models and verification of bpel," September 2006.

[11] S. Haddad, L. Mokdad, and S. Youcef, "Response-time analysis of composite web services," in *In Proceedings, IEEE Computer Society*, 23-25 July 2008.

[12] A. Daniel Menascé, "Response-time analysis of composite web services," vol. 8, no 1, pp. 90–92, 2004.

[13] S. Haddad, L. Mokdad, and S. Youcef, "Response time of bpel4ws constructors," in *Proceedings of the 15th IEEE Symposium on Computers and Communications (ISCC'10)*, 2010, pp. 695–700.

[14] U. Vallamsetty, K. Kant, and P. Mohapatra, *Characterization of e-commerce traffic*. Electronic Commerce Research, 2003, vol. 3, no 2.

[15] L. Zeng, B. Bennatallah, H. Anne Ngu, H. Chang, M. Dumas, and J. Kalagnanam, *QoS-aware middleware for web services composition*. IEEE Transactions on Software Engineering, 2004.

[16] G. Canfora, M. Di Penta, R. Esposito, and L. Millani, "A framework for qos-aware binding and re-binding of composite web services," *Journal of Systems and Software, 81(10) :1754-1769*, pp. 414–417, 2008.

[17] J. Cardoso, A. Sheth, J. Miller, J. Arnord, and K. Kochut, "Modeling quality of service for workflows and web service processes," *Web Semantics Journal : Science, Services and Agents on the WWW Journal 1 (3)*, pp. 281–308, 2004.

[18] F. Baligand, N. Rivierre, and T. Ledoux, "A declarative approach for qos- aware web service compositions," vol. 4749 of LNCS, pp. 422–428, 2007.