

QoS4WSC: A Framework for Web Services Composition based on QoS constraints

Bassirou Gueye, Ibrahima Niang, Bamba Gueye and Mohamed Ould Deye

University Cheikh Anta Diop de Dakar
Faculty of Science and Technology
Department of Mathematics and Computer Science
Dakar - SENEGAL

bassirou.gueye@ucad.edu.sn, iniang@ucad.sn, bamba.gueye@ucad.edu.sn, mohamed.ouldeye@ucad.edu.sn

Abstract: Web Services Composition (WSC) is a paradigm for enabling application integration within and across organizational boundaries. Nowadays, the main challenges of WSC in environment like Internet are to ensure high quality execution. Former works on service composition have enabled the elaboration of the *BPEL4WS* standard. Nevertheless, the management of the Quality of Service (QoS) of service composition, generally defined in Service Level Agreement (SLA), was not taken into account. Therefore, it is mandatory to build flexible solutions, reusable, and offering a suitable level of abstraction.

We propose a new framework, called QoS4WSC, that evaluates constraints-based response time in WSC. In order to provide a model which can estimate the effective response time of a service composition, we use a parsing file composition technique. Since elementary web services are based on a best effort approach, we propose an architecture which takes into account the response time of a each 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 related to the WSC.

Keywords: Web Services Composition - BPEL - Quality of Service - SLA.

I. Introduction

Web services are a tremendous technology that ensure interoperability between applications in the Internet [1]. This new concept leads to a development of new paradigms that enable an interaction between different applications. In fact, web services are the latest distributed technology as well the most suitable technology for the realization of SOA (Service Oriented Architecture) [3, 4]. They become the commonly used technology for integrating applications and information systems. This trend is expected to be unabated. Web services provide the technological foundation for achieving interoperability between applications by using different software platforms, operating systems, and programming languages. They are built on XML which is de facto the standard for data-level integration.

From the technological perspective, web services can be seen as a distributed architecture which started with DCE (Distributed Computing Environment), RPC (Remote Procedure Call), and messaging systems. Thereafter, distributed ob-

jects and ORBs (Object Request Brokers), such as CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), and RMI (Remote Method Invocation), have emerged.

Similarly, web services have been proposed, and they were the first distributed technology that were supported by all major software vendors. Therefore, web services promote an universal interoperability between applications running on disparate platforms.

Web services are based on the following fundamental specifications: (i) WSDL *Web Services Description Language* [5] which is the description of the web services; (ii) UDDI *Universal Description, Discovery and Integration* [6] which enables to publish the services; SOAP *Simple Object Access Protocol* [7] which is a protocol of communication that is used in order to invoke the web services. It is worth noting that WSDL, UDDI and SOAP are XML-based which prones web services protocol messages and descriptions to become human readable. Web services are designed to promote SOA, integrating highly distributed complex heterogeneous systems, that can cooperate without considering specific and costly integration. It exists different web-based applications that can perform specific tasks.

According to few applications, it is necessary to combine a set of basic web services in order to create a composite web services. However, the task of composing web services is more complex due to the autonomy and heterogeneity of services, as well the dynamic nature of the composition [8].

Indeed, a composition is not simply a collection of any web services but a group whose tasks are ordered according to the relationships between web services. In addition, web services are usually provided by different organizations, and their executions are not related to context. Since each organization has its own rules of work, web services should be treated as strictly autonomous units.

The QoS (Quality of Service) examines the quality of the relationship that exists between a service and its customers is crucial. Although it exists significant works around services composition which have enabled the development of the standard *BPEL4WS* [9] (Business Process and Execution Language For Web Services), the management of QoS in services composition needs new solutions that are either flexi-

ble, or reusable, To this purpose, we can provide more abstraction levels. Indeed, the expressiveness of BPEL is only the functional aspects of the compositions.

According to SOA context, where the end users are unfamiliar, QoS issues are also important as much as the assembly of functional services. In this respect, it is mandatory to ensure that a given BPEL process will be executed with efficiency after its construction. QoS characteristics in web services should respect the agreement between the service provider and the user. This agreement is defined in a SLA (Service Level Agreement) [10, 12] which defines the supply and the demand of these two entities.

The QoS in web service can be monitored by using different non functional metrics such as response time, availability, cost, etc [12, 13]. 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 web service composition. 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. It is worth noting that, the degradation of the quality of service can lead to serious problems with a loss of customers that promotes a reduction economic gain. In fact, if a service runs during a long time or is not available quite often (failure), it will be not re-used by customers.

Similarly, it is important for the QoS-aware composition to be fast. Hence, for interactive systems, a long execution delay may be unacceptable from a customer point of view. For example, a user of a booking ticket system would not like to wait during a long time when the system is seeking candidate services that offer low cost flight tickets. A fast composition is also required to re-plan a service composition during the execution. In fact, the actual QoS deviates from the estimated one, and hence, this could cause constraint violation, or simply because some services might not be available. In this case, the time composition influences the overall services response time, thus it should be kept as low as possible.

In addition to our early work [2], this paper provides a enhanced description of the execution process of *BPEL* constructors. For instance, an algorithm which describes a thorough operating of the “*sequenceActivity*” constructor is depicted. Afterwards, we give an overview of the different activities that are involved in a life cycle of a service composition. Therefore, the proposed framework **QoS4WSC** (Quality of Service For Web Service Composition) enables to take into account QoS concerns and it is composed by two modules. The first module makes possible a predetermination of the response time of elementaries web services whereas the second module provides a mechanism for checking the QoS constraints of services composition. In order to provide a model for estimating effectively the response time of a service composition, we use the BPEL process parsing technique.

In this respect, QoS4WSC is able to estimate the amount of time that is need for executing any service composition that is constituted by a set of constructors. Consequently, the service providers compose their services by taking into account

the execution time of these constructors, and thus reduce the time for getting a response during the invocation of a web service composite. It should be noted that our model fully covers the last version of *WS-BPEL 2.0* which is adopted as *OASIS* Standard.

The rest of the paper is organized as follows. In section II, we present an overview on previous works on WSC. Next, Section III is devoted to introduce the architecture of QoS4WSC. Section IV presents how our QoS4WSC framework enables to evaluate QoS constraints, as well illustrates the different constructors used by QoS4WSC in order to estimate and verify the response time of a service compositions. Section V shows the impact of QoS4WSC in the life cycle of a web services composition. In section VI, we validate and evaluate our proposition with respect to former works. Finally, Section VII concludes the paper and outlines our future work.

II. Background and related work

In this section we give a brief overview on web services composition, and illustrate the incentives for managing QoS. Afterwards, we survey the former works that propose some solutions in order to tend towards efficient WSC.

A. Overview on web services composition

Recently, WSC has received much interest for supporting Business-To-Business (B2B) and Enterprise Application Integration (EAI). Many industrial standard specifications have been proposed in recent years, such as *BPEL4WS* (Business Process Execution Language for Web Services) [9], *BPML* (Business Process Modeling Language) [14], *XLANG* (XML business process Language) [15], *WSFL* (Web Service Flow Language) [16], *WSCL* (Web Service Conversation Language) [17], and *WSCI* (Web Services Choreography Interface) [18] in order to compose web services. Afterwards, we give a definition of the previously proposed specifications.

BPEL (Business Process Execution Language), also called either *Web Services BPEL* (*WS-BPEL*), or *BPEL4WS*, is a language used for composition, orchestration, and coordination of web services. It provides a rich vocabulary for expressing the behavior of business processes. On the other hand, *BPML* is a meta-language for modeling business processes and provides an abstract execution model for describing collaborations and transactions. It defines a formal model for expressing abstract and executable processes.

XLANG, which is proposed by Microsoft, is an extension of *WSDL* (Web Service Definition Language). It provides a model of services orchestration as well as collaboration contracts between orchestrations.

Similarly, *WSFL* is an XML language for the description of web services compositions as part of a business process definition. It is designed by IBM to be part of the web service technology framework, relies and complements existing specifications like *SOAP*, *WSDL*, *XMLP* and *UDDI*.

WSCL provides a XML schema for defining legal sequences of documents that web services can exchange. *WSCL* makes possible to define the abstract interfaces of web services and may be used in conjunction with other service description languages like *WSDL*.

Finally, WSCI is a language that returns the exchanged flows messages by web services in the context of a process. It describes the collective message exchange among interacting web services, providing a global and message-oriented view of a process involving multiple web services.

The general adoption of business process automation requires to propose standardized and specialized language in order to compose services into business processes. In such case, one provides the ability to express business processes in a standardized way by using a commonly accepted language. To this purpose, our proposition is based on BPEL in view of this language represents the evolution of the former composition languages like BPML, WSFL, WSCL, etc. Furthermore, since 2007 BPEL becomes the standard for web service composition. It is worth noting that BPEL is a convergence of two early workflow languages: IBM WSFL [16] and Microsoft XLANG [15]. It combines both approaches and provides language for the formal specification of business processes and business interaction protocols.

In addition, BPEL is the most comprehensive standard for describing business processes that can exist. It is the most sustained in industry as well as the most accepted by developers. Besides, we noticed a lot of works based on this language such as *METEOR-S* [19], *AO4BPEL* [20], *DYNAMO* [21], *MASC* [22], etc. In this sense, it is a language that is largely deployed in industries.

On the other hand, we note several technologies implemented with BPEL akin to *Microsoft BizTalk Server*, *IBM BPWS4J*, *Oracle BPEL Server*, *Apache ODE*.

BPEL provides a set of constructors allowing to define composite processes from the operations required by the WSDL partners' files. During the BPEL process each element or constructor is considered like an activity which can be described as a primitive or a structured activity. The main difference between both constructors is due to the fact a primitive constructor should be used itself whereas a structured constructor is able to call other constructors that are either structured or primitive [9].

The set of primitive and structured activities are illustrated in figure 1. Notwithstanding a detailed description, of the set of activities enumerated in figure 1, is given in Section IV-B.



















Activities	Forms	Activities	Forms	Activities	Forms
receive		wait		forEach	
invoke		terminate		flow	
reply		sequence		pick	
assign		if		event handler	
throw		while		fault handler	
scope		repeatUntil		compensate	

Figure 1: Activities of BPEL process.

B. Quality of service management

With the emergence of services in software architectures, it appears a new relational model between Internet users. This is due to the fact that a lot of services are provided by providers and hence used by customers. With a lack of coupling, the Service Oriented Architecture (SOA) simplifies the services process discovery, but also their utilization. According to SOA, customers may easily substitute one service for another according to their convenience.

Moreover, with respect to the proliferation of services that have quite often the same functionality, we attend to a competition between providers, in the sense that each one seeks to provide the best service in term of response time, cost, and availability.

In addition, due to the ubiquitous nature of Internet users can invoke services from different networking devices (e.g., mobile phone, laptop, tablet computer, etc.) that can have different bandwidth requirements. Therefore, service providers should have a big concern about the QoS requirements according to these heterogeneous equipments. In this context, the notion of quality of service becomes a crucial issue. Likewise, this notion refers to multiple properties such as response time, availability or cost.

In this sense, the authors of [23] proposed to introduce a framework where the service registries as well as services contribute to an automation of service discovery, and hence, workload is distributed more efficiently. This is achieved by developing a Linked Data compliant Web services framework which communicates with semi-centralised registries that compute themselves their suitability for a given request. Note that, all communications among different framework components use RDF-based message protocols including Input/Output service. Consequently, the offered framework in [23] aims to optimize the load balancing as well the performance by dynamically assembling services at run-time in a massively distributed Web environment.

On the other hand, Tripathy et al. illustrated in [24] a framework for monitoring the compliance of a set of pre-specified requirements of a SBS (Web Service Based Systems). These set of requirements may include behavioral properties of SBS and/or assumptions that can be specified by service providers in terms of events. It is worth noticing that these events are extracted from SBS at run-time. To achieve this, a Monitor Specification Language (MSL) has been developed to specify the properties of the system that should be monitored during the runtime. Note that, according to the proposed framework [24], the SBS runs quite independently the monitoring functionality in a non-intrusive manner.

According to SOA context QoS guarantee is also fundamental as much as the concern of assembling functional services. For example, a service allowing to place orders for purchase/sale of shares on the stock exchange would also be useless if he altered the orders given by customers that if their performances was poor. As business processing, QoS characteristics of web services should meet the supply and demand.

In contrast to the well-established standards in the functional domain of services such as SOAP or WSDL, it does not exist mechanisms which are formally recognized by the community of services with respect to the specification and pro-

cessing of QoS of web services. To overcome these limitations, several works like *WS-Agreement* [25], *IBM WSLA* [26], *Slang* [27] have been proposed. The main goal is to establish agreement between client and service providers through the use of service level agreement (SLA) [11].

A SLA [10, 11] is a part of an agreement where the level of service given by a provider is formally defined. Quite often, the term SLA refers also to the service delivery time or its performance. For instance, Internet Service Providers will commonly include a SLA with respect to the accord that they establish with customers. To this purpose, they define several service levels that will be sold in plain language terms.

Additionally, the SLA has technical specifications like mean time between failures (MTBF), mean time to repair or recovery (MTTR), various data rates, etc [11], that enable to achieve the agreement established between actors during SLA negotiations.

In order to respect the SLA during the execution of services, a monitoring mechanism should be used for checking the QoS constraints. During the execution process, if a violation happens with respect to the negotiated SLA then corrective mechanism should be applied.

C. QoS constraints in web services composition

Web services are an attracting area that interest many researchers and industrial organizations [1].

The authors of [28] proposed different algorithms in order to aggregate QoS properties for some standardized workflow patterns of the web service compositions. These properties include upper and lower bounds of execution time and cost, as well throughput and uptime probability.

In [29], an algorithm which determines the QoS of a web service composition, by aggregating the QoS dimensions of the individual services, is proposed. This algorithm is based on a collection of workflow patterns defined in [30]. The QoS parameters include an upper and lower bounds of execution time as well as throughput.

Furthermore, in order to improve the availability of web services, Cotroneo *et al.* [29] proposed a new architecture which enhances the service availability of premium users groups.

On the other hand, Bhatti *et al.* [31] studied the end-to-end response time for composites web services by taking into account Internet overhead factor in their execution model. Similarly, in METEOR-S project [19], it is proposed to take in consideration the principal idea from workflow QoS and transpose it to web services technologies. In this sense, the authors of this project suggest some QoS constraints that were implemented into METEOR-S workflow management systems [19].

Rud *et al.* in [33] described analytical formulas that take into account the response time of different BPEL constructors. To achieve these objectives, they used mathematical models by adopting a formalism ratings based on different assumptions. A good overview of these approaches is given in [34].

Haddad *et al.* [35] present an extension of web service composite model illustrated previously in [36]. Indeed, in [36] a composite web service is considered as a set of tasks running in parallel.

The authors of [35] argue that the model described by Menasce *et al.* [36] 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 this former work [36], Haddad *et al.* propose analytical formulas according to the response time for the following constructors: <sequence>, <switch> and <flow>. It is worth noticing that the authors of [35] do not propose a model that can estimate the response time of WSC.

Notwithstanding, Haddad *et al.* [37] improved their early work [35]. To this purpose, in order to overcome the shortcomings noticed in [35], they considered 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 [37]. The choice of these mathematical models is motivated by the fact that the authors of [38] showed that the response time of basic services can be modeled by such mathematical distributions.

In a nutshell, these works [33, 35, 37] that are more related to our work, proposed only analytical formulas for few BPEL constructors for managing the response time of web services composition. Basically, they do not offer tools that are able to verify the QoS parameters of WSC.

III. QoS4WSC architecture design

Since former works have just proposed analytical formulas for few BPEL constructors [9] in order to take into account the response time, we promote 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 an easy task.

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 build 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 2 and it is formed by two main modules. The goal of the first module, called “*Module 1*” (Figure 2), is to estimate the response time of web services. The second one, called “*Module 2*” (Figure 2) verifies whether the QoS constraints specified, for instance in a given SLA, are achieved.

The different steps, (1, ..., 8), as labelled in Figure 2 enable to estimate the response time of an elementary web services and to verify the QoS constraints of WSC. During each step we have the following tasks: (step 1) the provider (e.g., engineer in Figure 2) 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

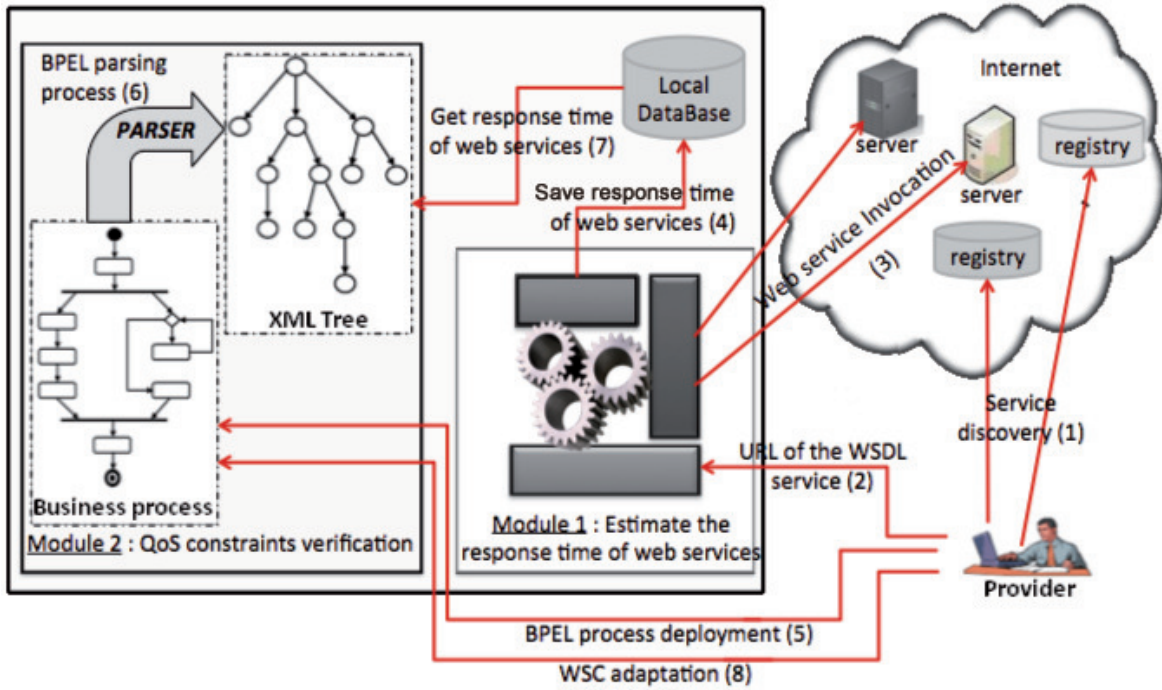


Figure. 2: Our QoS4WSC framework.

the WSDL file to the Module 1. Following that, the module 1 estimates the response time of the selected web service (step 3). The goal of this processing is to generate automatically a set of class called “servicename”, “servicenameLocator”, “servicenameSoap”, “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:

$$T[ws_i] = \sum_{i=1}^k \left(\frac{T_i}{req_i} \right) \quad (1)$$

It is worth noting that, the estimated response time for each web service is saved at the local database of the QoS4WSC framework (step 4). In such case, this value can be reused if the web service is re-invoked shortly.

Thereafter, 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 Module 2 (in Figure 2) 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-B.

Put simply, the main goal of the module 2 is to estimate the response time of web service composition as well as 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 this respect, we can change

either the elementary web services that form the WSC, or change the fixed constraints.

IV. Towards constraints-based web services composition

This section illustrates the set of constructors that are considered in the BPEL language. Furthermore, we describe the manager activities that enable to estimate the response time of each constructor and to verify whether the QoS constraints of a WSC are met.

A. Overview on BPEL constructors

The BPEL process is formed by a set of constructors linked by different workflows. These constructors can be splitted into two groups. The first group is constituted by constructors that are characterized as primitive, whereas the second one is formed by constructors that are called structured [9]. By definition, a constructor is depicted as structured whether it is able to call other constructors (primitive or structured) during the BPEL process. In contrast, a primitive constructor is devoted to do only a given task without having the possibility to call other constructors.

Fundamentally, the following set of constructors <receive>, <invoke>, <reply>, <assign>, <validate>, <throw>, <wait>, <terminate>, <empty>, <compensateScope>, <rethrow>, <extensionActivity>, and <compensate> are considered as primitive constructors (activities) [9].

In contrast, <flow>, <sequence>, <scope>, <If>, <elseif>, <else>, <repeatUntil>, <while>, <forEach>, <pick>, <onMessage>, <onAlarm>, <eventHandlers>, <onEvent>, and <repeatEvery> represent the structured constructors [9].

Based on the BPEL 2.0 constructors surveyed in BPEL spec-

ification [9], we consider amongst them a set of constructors, called “key constructors”. The elapsed time during the execution of these key constructors is useful for satisfying the QoS constraints of a given WSC.

In fact, the key constructors enable to define a waiting and/or processing time during the execution of a service composition. For instance, these key constructors are formed by:

- (i) this limited set of primitive constructors: <receive>, <invoke>, <reply>, and <wait>;
- (ii) and all set of structured constructors listed previously.

B. Using QoS4WSC for the verification of QoS constraints

In contrast to previous works [33, 35, 37], we propose an automatic tool that verifies QoS constraints for web service composition. Our approach is based on file composition parsing. Figure 3 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 2) receives as input a fixed response time as QoS constraint and a BPEL process that describes a WSC (Figure 3). Inside the QoS verification module, the XML tree of the process is created (Figure 3).

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 3. The <partnerLinks> contains the list of partners (web services) that will participate in the WSC, whereas <sequence> defines the execution order of the web service composition 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 3, is initialized to zero. Otherwise, the parsing is stopped (Figure 3) and an error exception is sent. If the constructors located in the <sequence> node match one of the key constructors defined in Section IV-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 key constructor.

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-B.1 to IV-B.8. According to a given manager, if the obtained response time is violated, the parsing is stopped (Figure 3) and a time-constraint violation is sent to the provider of the web services.

Afterwards, we describe the managers used in the algorithm illustrated in Figure 3. Therefore, we adopt the following formalism:

$\sum_{i=1}^n P[c_i] = 1$; 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 $repeatEvery$

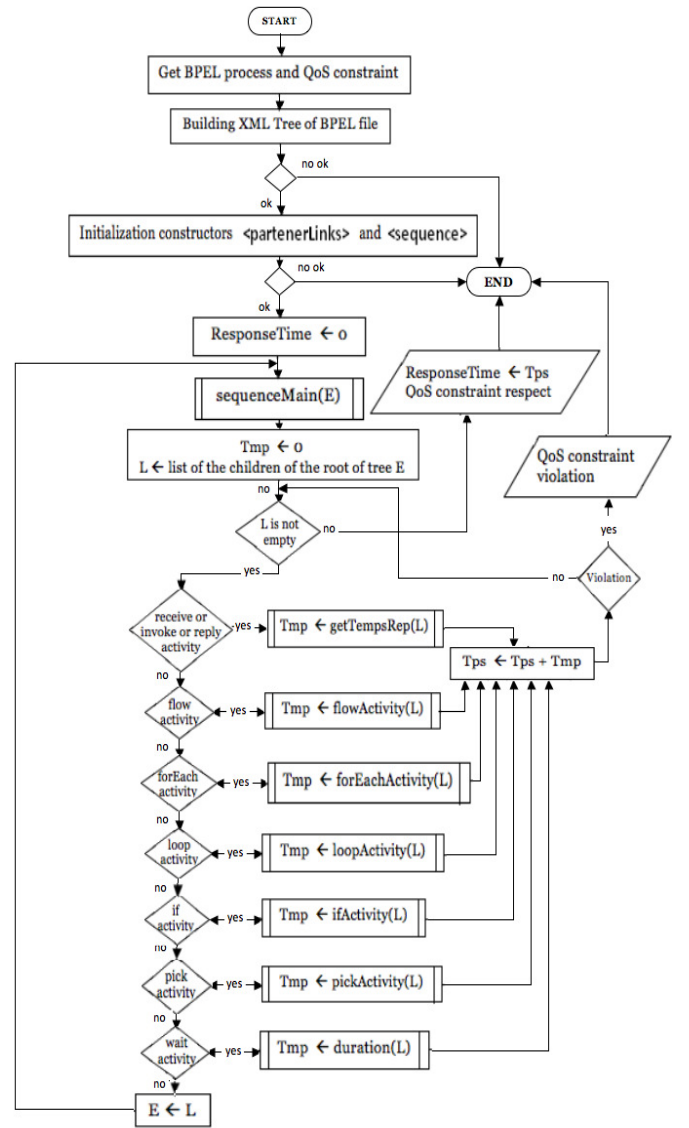


Figure 3: Algorithm for parsing a BPEL process.

1) “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.

2) “flowActivity” manager

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 3), is a <flow> constructor. The procedure enables to run simultaneously a set of activities. The response time $T[a]$ of this procedure is:

$$T[a] = \max(T[a_1], \dots, T[a_n]) \quad (2)$$

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.

3) “forEachActivity” manager

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 3), 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:

$$T[a] = k \times T_{body} \quad (3)$$

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.

4) “loopActivity” manager

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 3), 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 [39, 40] 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:

$$T[a] = k \times T_{body} \quad (4)$$

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 [32, 41].

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.

5) “ifActivity” manager

This manager is invoked if the current element, that we are testing during the execution our algorithm (Figure 3), 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:

$$T[a] = \sum_{i=1}^n (T[a_i] * p[c_i]) \quad (5)$$

If this constructor presents several branches the response time can be estimated as:

$$T[a] = \max(T[a_1], \dots, T[a_n]) \quad (6)$$

where $T[a_i]$ means the elapsed time during the block i .

6) “pickActivity” manager

This manager is invoked if the current element, that we are testing during the execution of our algorithm (Figure 3), 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:

$$T[a] = \sum_{i=1}^n (p[c_i] * (T_{wait} + T[a_i])) \quad (7)$$

Since, it is not possible to know in advance when an event will be triggered, the response time is obtained by:

$$T[a] = T_{wait} + \max(T[a_1], \dots, T[a_n]) \quad (8)$$

7) “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.

8) “sequenceActivity” manager

This manager is called by other managers each time that a set of activities should be run sequentially. The operating of this manager is described by the algorithm defined in Figure 4.

REMARK: 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:

$$T[a] = \sum_{i=1}^n (p[c_i] * (T_{wait} + k * (T_{scopeAlarm} + T_{repeatEvery}))) \quad (9)$$

where k is equal to $\lfloor \frac{T_{scopeParent} - T_{wait}}{T_{scopeAlarm} + T_{repeatEvery}} \rfloor$.

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.

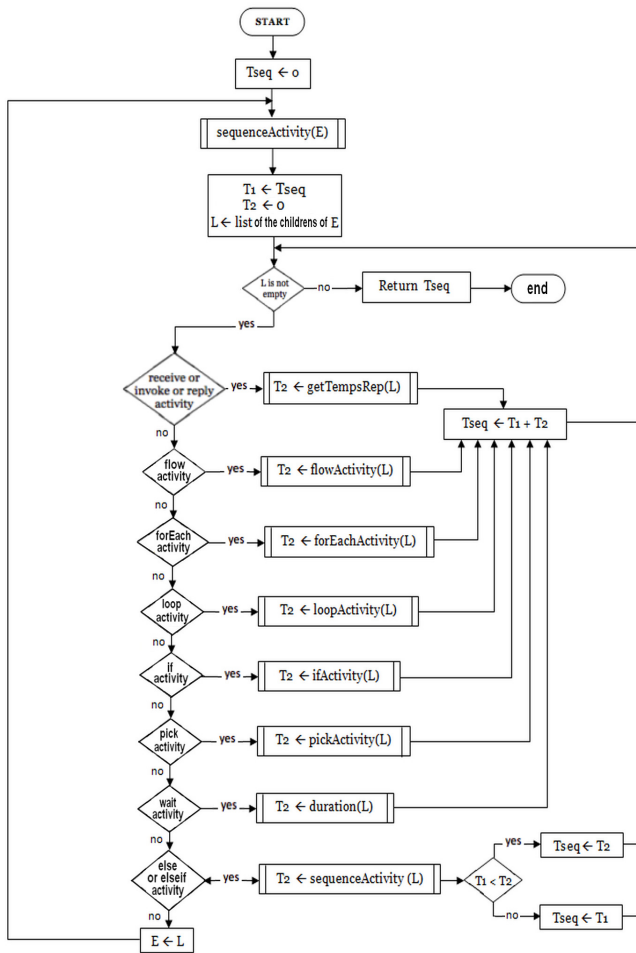


Figure 4: Algorithm for parsing sequenceActivity manager.

V. The impact of QoS4WSC in the life cycle of web services composition

According to [42], the life cycle of web services composition is based on the analysis and design of six activities. Nevertheless, from our point of view two activities, like *Wrapping native services* and *Setting outsourcing agreement*, can be seen as included in the life cycle of an elementary service. However, the last four activities such as *Assembling composite services*, *Execution services*, *Monitoring services*, and *Evolving services* take part according to our definition of a life cycle of web services composition. Therefore, these four activities plus our framework QoS4WSC form the five elements that constitute the life cycle of a WSC (Figure 5).

The various activities and transitions illustrated in Figure 5 are described as follows:

1. “*Assembling composite services*”: This activity enables to describe the overall classes of service that one should compose in order to propose a WSC. This assembly includes an identification phase of the service categories and the specification of their interactions. This activity is triggered by the initial request (1) which is sent either by a user or by a software. It should be noted that this request is transmitted to the web services orchestration engine.
2. “*QoS4WSC*”: The goal of this activity is to seek if the

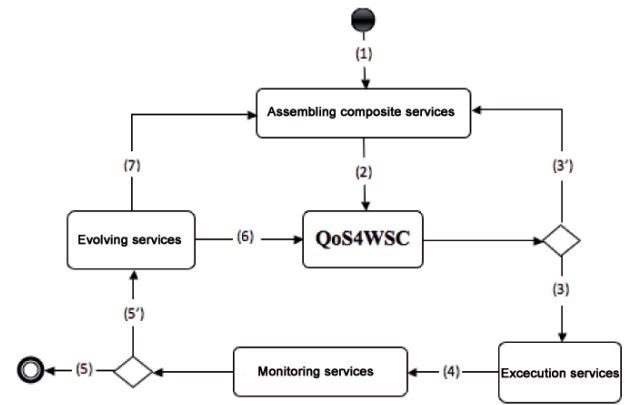


Figure 5: Impact of QoS4WSC in the life cycle of a WSC

composition, as defined in the *Assembling composite services* (2) meets a given QoS constraint. In this respect:

- If the constraint is met, we proceeded to the next execution level of the composition by following the branch labelled 3;
- Otherwise, we return to the previous state (branch 3’) in order to modify the constraint, or to change the service composition.

3. “*Execution services*”: This activity executes the specifications of the composition defined above by checking whether the fixed constraints between each web service are satisfied. Afterwards, the orchestration engine starts to execute the first tasks of the composition defined in *Assembling composite services*.
4. “*Monitoring services*”: This activity supervises the execution of the composition by checking the access to the different services and the exchanged messages. This control is done in order to measure the performance of the called services as well as to manage the execution in case of failures (branch 4). If all services have been called successfully and the composition has been achieved, the composition engine returns the composition result to the user (e.g., human or software) that has sent the original request (branch 5).
5. “*Evolving services*”: This activity enables to evolve the service composition by removing the failed services and then promotes the use of new service. If the “*Evolving services module*” fails, we have the possibility either to readjust the QoS constraints according to a given task (branch 6), or to redefine the service composite (branch 7) by allowing a new service in place of the one which fails before from the “*Assembling composite services*”,

VI. Implementation and results

We implemented our framework with the programming language Java (JDK 1.6) and the Eclipse IDE v.3.4.2. To make this practical environment, we used the plugins WTP (Web Tools Platform) version 3.2, EMF (Eclipse Modeling Frame-

work), GEF (Graphical Editing Framework), GMF (Graphical Modeling Framework) and BPEL Visual Designer.

We considered the following tools such as 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 6. 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.

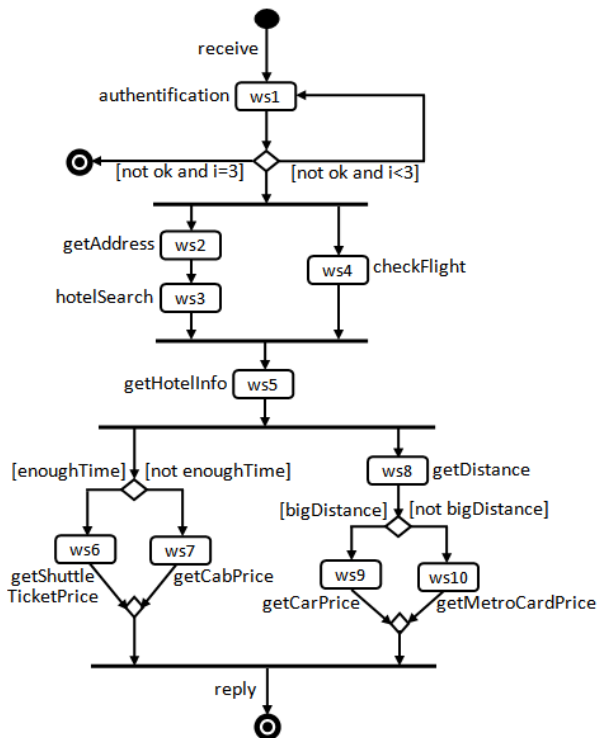


Figure 6: Travel Plan process.

A. Discussion

The techniques proposed by Haddad et al. in [35, 37] do not allow to estimate the response time of the composition depicted in Figure 6. In fact, they do not take into account the existence of *loop* constructor. Therefore, we argue that their proposed techniques in [35, 37] 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 services composition it is not possible to do it manually.

Rud et al. [33] proposed to estimate the response time of this composition (Figure 6) 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.

VII. 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 compositions. 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 the formal semantic in the life cycle of web services composition can be considered.

References

- [1] S. Dustdar and Wolfgang Schreiner, "A survey on web services composition", *Int. J. Web and Grid Services*, Vol. 1, No. 1, 1, pp. 1–30, 2005.
- [2] B. Gueye, I. Niang, B. Gueye, M. O. Deye, Y. Slimani, "Constraints-Based Response Time For Efficient QoS in Web Services Composition", in *Proceedings of the 7th IEEE International Conference on Next Generation Web Services Practices*, pp. 141 – 146, 2011.
- [3] 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.
- [4] Y. Tao, and Z. Yue, L. Kwei-Jay. "Efficient algorithms for Web services selection with end-to-end QoS constraints", *ACM Trans. Web*, Vol. 1, No. 1, Article 6, May 2007, 26 pages.
- [5] R. Chinnici, J.-J. Moreau, S. Weerawarana, and R. Arthur, "Web Services Description Language (WSDL) version 2.0". *W3C Recommendation 26*, June 2007.
- [6] L. Clement, Systinet, A. Hatel, C. Von Riegen and T. Rogers, "Computer Associates. UDDI version 3.0.2, OASIS Specification", October 2004.
- [7] M. Gudgin, M. Hadley, J. Jacques Moreau, and H. Frystyk Nielsen, "Simple Object Access Protocol (SOAP) Version 1.2. W3C". July 2001.

- [8] P. F. Pires and M. Benevides and M. Mattoso. "WEB-TRANSACT: a Framework For Specifying And Coordinating Reliable Web Services Compositions". Technical report, 2002.
- [9] "Web Services Business Process Execution Language (BPEL4WS) version 2.0", OASIS standard, April 2007.
- [10] L. J. Jin, V. Machiraju, and A. Sahai, "Analysis of Service Level Agreement for web services", HPL-2002-180, Tech. Rep., 2002.
- [11] E. Marilly, et al, "SLAs: A Main Challenge for Next Generation Networks", 2nd European Conference on Universal Multiservice Networks", April 2002.
- [12] A. D. Menasce, "Mapping Service Level Agreements (SLA) in distributed applications", IEEE Internet Computing, pages 100–102., September-October 2004.
- [13] M. Godese, U. Bellur, R. Sonar. "Automating QoS Based Service Selection", IEEE International Conference On Web Services, pages 534–541, 2010.
- [14] A. Agarwal, A. Arkin. The BPML specification. BPML Working Draft 0.4, March 2001.
- [15] S. Thatte. XLANG: Web services for business process design. Technical report, Microsoft Corporation, 2001.
- [16] F. Leymann, "Web Services Flow Language (WSFL version 1.0)", IBM Software Group, May 2001.
- [17] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossians, S. Sharma, and S. Williams. "Web Service Conversation Language (WSCL)", 1.0. W3C, March 2002.
- [18] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacs-Nagy, I. Trickovic, and S. Zimek. "Web service choreography interface (WSCl)" 1.0. W3C Note, August 2002.
- [19] R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor. "Constraint driven web service composition in METEOR-S". In IEEE SCC, pages 23-30, 2004.
- [20] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. "Reliable, secure, and transacted web service compositions with AO4BPEL". In Proceedings of the 4th IEEE European Conference on Web Services (ECOWS), December 2006.
- [21] L. Baresi and S. Guinea. "Dynamo and self-healing bpel compositions". In proceedings of the 29th International Conference on Software Engineering (ICSE '07). IEEE Computer Society. pages 69-70, Washington DC, USA, 2007.
- [22] A. Erradi, V. Tasic, and P. Maheshwari. "MASC - netbased middleware for adaptive composite web services". In International Conference on Web Services. IEEE Computer Society, pages 727–734, 2007.
- [23] H.Q Yu, S. Dietze, C. Pedrinaci, and D. Liu. "A linked data compliant framework for dynamic and web-scale consumption of web services". International Journal of Computer Information Systems and Industrial Management Applications, Volume 3, page 796–803, Dynamic Publisher USA, 2011.
- [24] A. K. Tripathy and M. R. Patra. "Service Based System Monitoring Framework". In International Journal of Computer Information Systems and Industrial Management Applications: IJCISIM, Volume 3, Page 924–931, Dynamic Publisher USA, 2011.
- [25] H. Ludwig, A. Dan, and R. K. Cremona. "An architecture and library for creation and monitoring of ws-agreements". Proceedings of the 2nd international conference on Service oriented computing, pages 65-74, New York, NY, USA, 2004. ACM Press.
- [26] A. Keller and H. Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". Journal of Network and Systems Management, Vol. 11, No. 1, pp : 57-81, March 2003.
- [27] D. D. Lamanna, J. Skene, and W. Emmerich. "Slang: A language for defining service level agreements". In 9th IEEE Workshop on Future Trends in Computing Systems, pages 100-106, San Juan, Puerto Rico, 2003. IEEE Computer Society Press.
- [28] M. C. Jaeger, G. R. Goldmann, and G. Muhl, "QoS aggregation for web service composition using workflow patterns", in Proceedings Eighth IEEE International Enterprise Distributed Object Computing Conference, pp. 149–159, 2004.
- [29] G. Kiczales, "Aspect Oriented Programming", ACM Computer. Surv., pp. 154, 1996.
- [30] V. Der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workow patterns", Technical report FIT-TR-2002-2, Faculty of IT, Queensland University of Technology, July 2002.
- [31] M. U. Bhatti, S. Youcef, L. Mokdad and V. Menfort, "Simulation-based Response Time Analysis of composite Web Services", In Proceedings 10th IEEE international Multitopic conference, pp. 1–7, juin 2006.
- [32] 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 World Wide Web Journal 1 (3), pp. 281–308, 2004.
- [33] D. Rud, M. Kunz, A. Schmietendorf and R. Dumke, "Performance Analysis in WS-BPEL Based Infrastructures". In 23rd UK Performance Engineering Workshop, 2007.
- [34] F. V. Breugel and M. Koshkina, "Models and verification of BPEL", September 2006.

- [35] S. Haddad, L. Mokdad, and S. Youcef, "Response-Time analysis of composite web services", In Proceedings , Communication Systems, Networks and Digital Signal Processing (CSNDSP'2008), IEEE Computer Society, 23–25 July 2008, Austria.
- [36] A. D. Menasce, "Response-Time analysis of composite web services", IEEE Internet computing, vol. 8, No. 1, pages 90–92, 2004.
- [37] 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)*, pages 695-700, June 2010.
- [38] U. Vallamsetty, K. Kant, and P. Mohapatra, "Characterization of ecommerce traffic", Electronic Commerce Research, vol. 3, no. 1-2, pp. 167–192, 2003.
- [39] L. Zeng, B. Bennatallah, H. Anne Ngu, H. Chang, M. Dumas, and J. Kalagnanam, H. Chang, "QoS-aware middleware for web services composition". IEEE Transactions on Software Engineering vol. 30 (5), pages 311–327, 2004.
- [40] 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, vol. 81(10), pp. 414–417, 2008.
- [41] F. Baligand, N. Rivierre, and T. Ledoux, "A declarative approach for QoS - aware web service compositions", Fifth International Conference on Service-Oriented Computing (ICSOC), volume 47–49 of LNCS, pages 422–428. Springer, 2007.
- [42] B. Benatallah, M. Dumas, M.C. Fauvet, F.A. Rabhi, Q.Z. Sheng, "Overview of Some Patterns for Architecting and Managing Composite Web Services". ACM SIGecom Exchanges, 2002, vol. 3, No 3, pp.9-16.



Ibrahima NIANG is an Associate Professor in Computer Science at the Faculty of Sciences and Technology of the Université Cheikh Anta Diop de Dakar (UCAD), Senegal. He received the Ph.D. degree in computer science in 2002. His research activities concern QoS, mobility and security in wireless networks and distributed systems (P2PSIP). In recent years, he worked on research and development related to water management using biosensor networks.



Bamba Gueye received the B.Sc. in Computer Science from the Université Cheikh Anta Diop de Dakar (UCAD), Senegal. He received the M.Sc. degree in Networking in 2003 and the Ph.D. degree in computer science in 2006, both from the Université Pierre et Marie Curie (UPMC), France. He is currently an Associate Professor at the Université Cheikh Anta Diop de Dakar.

Between 2007 and 2010, he was Research Assistant with the Université de Liège (ULg, Belgium), Electrical Engineering and Computer Science Department. He was a visiting scientist at Université du Québec à Montréal (UQAM), Canada during the 2011/12 academic year. His current research interests include network virtualization, green cloud computing, wireless networks, and Internet measurements.



Mohamed Ould Deye is an Assistant Professor at the Department of mathematics and computer science at Université Cheikh Anta Diop de Dakar (UCAD), Senegal. Currently, he is preparing a Ph.D. thesis in Computer Science at Tunis El Manar University. His areas of interest are Cloud Computing, Web Services and performance evaluation of distributed systems.

tion of distributed systems.

Author Biographies



Bassirou Gueye received his bachelor in Computer Science in 2007. He received the M. Sc. in Computer Science with a major in distributed systems in 2010 at Université Cheikh Anta Diop de Dakar (UCAD), Senegal. He is currently a PhD student in under joint supervision between Université Cheikh Anta Diop de Dakar (UCAD) and

Université de Reims Champagne-Ardenne (URCA), France. His research topics concern Grid computing, Web services and Peer-to-Peer networks.