Enforcing Quality of Service within Web Services Communities

Mohamed Adel Serhani
Faculty of Information Technology, Al-Ain, UAE
Email: serhanim@uaeu.ac.ae

Abdelghani Benharref
Engineering and Computer Science, Abu Dhabi University, Abu Dhabi, UAE
Email: abdelghani.benharref@adu.ac.ae

***Abstract*—Web services are considered as an attracting distributed approach of application/services integration over the Internet. As the number of Web Services is exponentially growing and expected to do so for the next decade, the need for categorizing and/or classifying Web Services is very crucial for their success and the success of the underlying Service Oriented architecture (SOA). Categorization aims at systematizing Web Services according to their functionalities and their Quality of Service attributes. Communities of Web Services have been used to gather Web Services based on their functionalities. In fact, Web Services in a community can offer similar and/or complementary services. In this paper, we expand Web Services communities' classification by adding a new support layer for Quality of Service classification. This is done through Quality of Services specification, monitoring, and adaptation of Web Services within communities. A Web Service might be admitted to a community thanks to its high Quality of Service or might be ejected from a community due to its low Quality of Service. The focus of this paper is on the design and use of a managerial community to monitor and adapt Quality of Web Services (QoWS) of managerial Web Services for other communities, Web Services providers, and Web Services clients.**

***Index Terms* — Web Services, Communities of Web Services, Quality of Web Services (QoWS), Selection of Web Services, QoWS Monitoring, and QoWS Adaptation.**

I. INTRODUCTION

The phenomenal growth of Internet technologies, largely impacted by the eXtensible Markup Language (XML) and its related technologies is extending the traditional role (client-to-business) of the World Wide Web to a better support of Business-to-Business interactions. The future perspective of the Internet is being driven by Web Services technologies [1].

A Web Service can be defined as an application that exposes its functionality through an interface description and makes it available for use by other programs. Web Services allow computers and devices to automatically interact with each other using the Internet to exchange and gather data. Moreover, on one hand, a composite Web Service can further be created by aggregating a set of Web Services to produce a more complex Web Service with a wide range of functionalities. On the other hand, a set of Web Services can form and operate inside a community.

In the Revised Webster dictionary, a community is defined as "*a body of people having common rights, privileges, or interests, or living in the same place under the same laws and regulations.* On a similar path, a community of Web Services can consist of Web Services offering the same functionalities or sharing similar concerns.

Even with a huge number of related works on Web Services and somehow a reasonable amount on communities of Web Services (e.g. [2], [3], [4]), there is a lack of mechanisms and approaches to establish and enforce inter-community and intra-community rules.

The aim of this paper is, first, to define the rights of a community and participating Web Services, their duties toward peers and clients, and it proposes a novel Quality of Web Services (QoWS) management approach to enforce QoWS-based selection, QoWS monitoring, and QoWS adaptation. These are essential issues to protect a community, its reputation, its interest, and those of each individual Web Services. For example, a Web Service that operates within a community and frequently provides very low quality can affect the reputation of the whole community. In this case, the community should first monitor the QoWS to detect any QoWS violation, and then adapt the violated QoWS so that clients do not notice the QoWS degradation and remain loyal.

Defining and enforcing terms and regulations of/within communities of Web Services raise a set of questions including:

- How members of communities should distribute the load to fairly share benefits and to guarantee a certain QoWS?
- How a managerial community can offer managerial services (e.g. monitoring and adaptation) to other Web Service communities.
- How to define interactions between communities?
- As a member of a community, how to find and select a community to get services from whenever needed?

doi:10.4304/jsw.6.4.554-563

Although this paper does not answer all of these questions, we propose a managerial community of Web Services for management of communities of Web Services. This managerial community is composed by Web Services instrumented with adequate functionalities and services to assess the QoWS of other Web Services and react to QoWS degradations. Such a Web Service is called Managerial Web Service (MWS). QoWS assessment includes test, monitoring, and certification of a Web Service as a partial-requirement to join a community. Moreover, once Web Services are part of a community, the managerial community can monitor, periodically or on request, their behavior and interactions on the fly to detect any potential violation to the terms of their community, which might result in expulsion of the failing Web Service from the community. Moreover, a participating Web Service can make use of the managerial community to show how much it is useful for the community and get some business credit or consideration. Finally, clients of Web Services can use the managerial community to select a community that suits their needs. In fact, many communities are likely to be competing by offering similar services with different conditions. The managerial community can advise a client which community to join based on her/his requirements and the status of the selected community (as known by the managerial community).

The remaining sections of this paper are organized as follows: next section discusses related works. Section 3 presents our managerial community and the main services it provides (monitoring, adaptation of QoWS…) while section 4 discusses the QoWS specification including the description of the QoWS properties, the EFSM specification model of Web Services, and the QoWS-based Web Service selection approach. Section 5 describes the adopted QoWS monitoring scheme, and section 6 details the QoWS adaptation techniques used to adapt the QoWS in three QoWS adaptation situations. A proof of concept summarizing our experience in using the managerial community for selection, monitoring and adaptation of QoWS is presented in section 7. We conclude by conclusion and future work in section 8.

II. Related Work

In general, management of Web Services as well as their QoWS (specification, publication, and discovery) are becoming more and more important as the number of similar, though competing, Web Services available in the Internet proliferates and the need for communities and composition of Web Services increases. Management of QoWS, as an integral part of Web Service management, will play an important role for the success of this paradigm. On one hand, providers of Web Services will have to specify and guarantee QoWS to remain competitive and achieve the highest possible returns on investment from their businesses. On the other hand, clients will have the possibility to look for appropriate Web Services according to their QoWS preferences (e.g., highly available, and respond to client's requests in reasonable time).

As discussed before, works on communities of Web Services are mostly on establishing and building communities rather than managing communities and enforcing appropriate rules. However, there are some works on management of Web Services that are of relevance to this topic. Hereafter is a short list of some works of interest to this paper.

Managing QoWS of Web Services as component of Web Service management was addressed by several research initiatives. In [5], the work introduced Web Services Performance Analysis Centre (sPAC) and shows how customers can verify timeliness of their Web Services semi automatically from the description of workflow of Web Services to reports analysis and results estimation. In [6], the paper identified a set of QoWS metrics in the context of Web Services workflows, and proposes a unified probabilistic model to describe QoWS values of a broader spectrum of atomic and composite Web Services. In [7], the paper proposed a QoWS-aware binding approach based on Genetic Algorithms. The approach included a feature for early run-time re-binding whenever the actual QoWS deviates from initial estimates, or when a service is not available.

In [8], the authors surveyed the key features of Web Services management system (WSMS) and conducted a comparative study on how current research approaches and projects fit in. In [9], the authors proposed a Web Service gateway to monitor and control Web Service access according to SLAs and organizational policies. The authors in [10] presented an implementation to derive on-line monitors for Web Services automatically from SLAs using an Eclipse plug-in.

Several works proposed broker-based architectures for QoWS management ([11], [12], [13]). In these architectures, a broker mediates between clients and providers of Web Services by providing a set of QoWS management operations such as: QoWS verification, QoWS certification, QoWS-based Web Service selection, QoWS negotiation, and QoWS monitoring. However, this model is not scalable considering the number of clients and the number of Web Services that might need to be supported by the broker. Moreover, QoWS properties might be managed differently due to the nature of each property. For example, managing Web Service availability requires a simple invocation of a Web Service by the broker to check if it is responding over a period of time. However, management of Web Service's response time requires that the broker implements or use existing measurement and monitoring techniques to measure the time a client's request is sent and the time its response is received.

Interested in Web Service management, Tosic et al. ([14]) have used the 'class of service' term as a discrete variation of the complete service and QoWS. Authors demonstrated that using classes' specification and management is simpler, faster and incurs less run-time overhead than using custom-made service level agreements (SLAs), client's profiles, or separate Web Services. It is then often easier and faster for a consumer to switch to another class of service within the same Web Service than to search for a replacement Web Service or to renegotiate an SLA. For the sake of the formal

specification of various types of constraints, authors developed the Web Service Offerings Language (WSOL) [15]. It references one or more WSDL files and specifies additional information. A corresponding management infrastructure called the Web Service Offerings Infrastructure (WSOI) was developed to manage monitoring and dynamic manipulation of WSOL service offerings.

With regards to QoWS adaptation, different research works were conducted. We will discuss here only those that are close to our work. In [16], Ming et al. proposed a broker-based architecture for dynamic QoS monitoring and adaptation for composite Web Services. This approach consists of dynamically changing the execution path of a composite Web Service in order to meet QoWS requirements. For achieving this, ordinary Business Process Execution Languages (BPEL) processes were instrumented and enriched to interact with the QoWS-aware broker.

In [17], Brogi et al. proposed a technique to adapt a service in order to suitably overcome both semantic and behaviour mismatches. The proposed technique relies on inspecting service execution traces and generates a service contract tailored to the client needs. Service contracts include a description of the service behaviour expressed by a workflow as well as an ontology-annotated signature.

In [18], Nezhad et al. presented techniques and a tool that provides semi-automated support for identification and resolution of mismatches between service interfaces and protocols, and for generating adapter specification. They implemented the approach in a tool inside IBM WebSphere Integration Developer (WID).

In [19], Chang et al. surveyed representative software adaptation methods, and proposed four types of service variability that are: workflow, composition, interface, and logic variability. They presented practical adaptation methods for resolving the four types of service variability. The proposed adaption methods presented can be implemented in a typical Web Service environment with WSDL, UDDI and BPEL.

In [20], Kongdenfha et al characterized the problem of aligning internal service implementation to a standardized external specification. They proposed an Aspect oriented framework as a solution to provide support for service adaptation. The framework consists of taxonomy of the different possible types of mismatch between external specification and service implementation, a repository of aspect-based templates to automate the task of handling mismatches, and a tool to support template instantiation and their execution together with the service implementation.

III. Overall Architecture

A managerial community of Web Services is a community that is composed of QoWS-management-capable Web Services. Each of these Web Services can assess the QoWS of a Web Service and can passively monitor it while the latter is operating inside a community and/or interacting with peers and/or clients. An exhaustive list of services offered by the managerial community will be discussed in section B.

A. Architecture Description

The core idea in our approach is the managerial community of Web Services. Figure 1 illustrates an environment with one managerial community, two normal competing communities, and a client. Two communities/Web Services are said to be competing if they are offering same functionalities in the same market space. Similarly, two communities/Web Services are said to be complementary if they offer complementary non-competitive services. For example, the Skyteam[1] community consists of few airlines (Air France, KLM, and Delta). While in the same community, those airlines are competing with each other to attract a maximum of passengers. However, Skyteam offers complementary services to the American Hotel and Lodging Associations[2], that is, getting customers into and from the hotel.

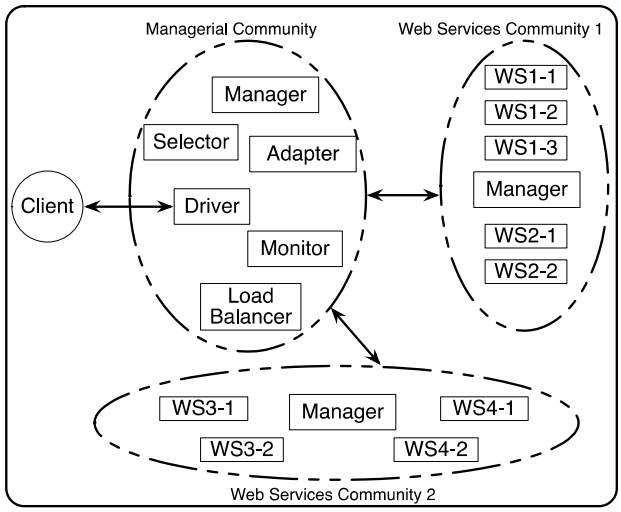

Figure 1. Managerial Community and two Normal Communities

All communities, including the managerial community, are created and maintained by a manager. In Figure 1, Web Services WS1-1, WS1-2, and WS1-3 offer similar services in community 1, these Web Services are competing with Web Services WS3-1 and WS3-2 in community 2. Same for Web Services WS2-1 and WS2-2 in community 1 compared to Web Services WS4-1 and WS4-2 in community 2. In the same figure, Managerial Web Services in the managerial community are all complementary, however, they can be competing as well in the same community and the business model of the community states which one to use and under which circumstances.

Moreover, the managerial community has few specific-purpose Web Services. The selector Web Service helps in selecting an appropriate Web Service based on functional (e.g. operation invocation) and non-functional (QoWS) requirements (e.g. response time, and availability) and based on the load of concerned Web

[1] http://www.skyteam.com/
[2] http://www.ahla.com/

Services as provided by the Load Balancing Web Service. The load balancer collects data from Web Services, the driver, and the monitor to establish a real-time knowledge base containing information about Web Services load. The driver Web Service is the main interface that the managerial community exhibits to clients. Clients interact with this driver for all matters. During interaction between clients and a Web Service (from community 1 and community 2), the monitor checks the correctness of this interaction and reports any anomalies to the driver. A typical flow of events involving different elements of the architecture in Figure 1 is illustrated in Figure 2.

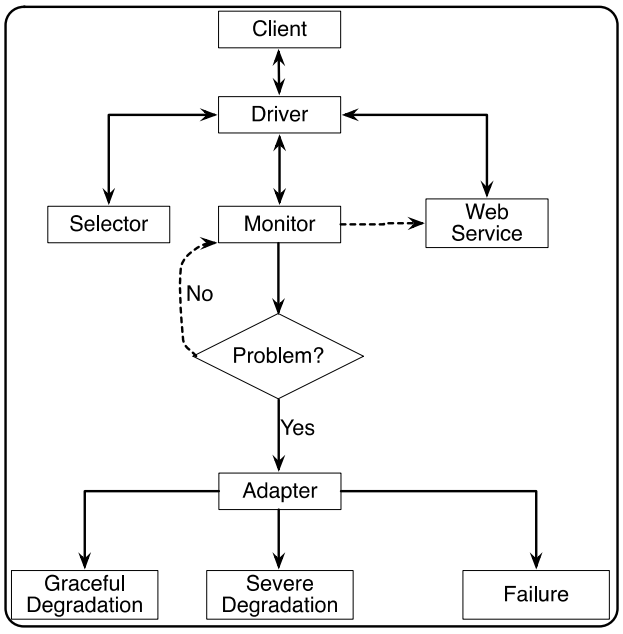

Figure 2. Actions Flow When Using a Managerial Community

The managerial community is intended to providers, clients, and community managers who would like to:

1. Select and use Web Services that fulfills some QoWS requirements,
2. Monitor (online) their interactions with these Web Services, and
3. Adapt the QoWS once the later degrades or is no longer provided. This does not apply to providers of Web Services who would like to test their own Web Services without looking for alternative Web Services.

All the above operations are supported by the managerial community via its driver as depicted in Figure 2. The client submits a "Select" request to the driver specifying the desired functionality and QoWS. The driver forwards the request to the selector who checks for the list of appropriate Web Services within communities being managed by the managerial community and returns the best match. The driver connects to that Web Services and notifies the monitor to start its online checking. During interactions, if the monitor detects a discrepancy of the agreed upon QoWS, it notifies the adapter. Based on the seriousness of the discrepancy, the adapter takes corrective measures.

Graceful degradation occurs when the observed QoWS is slightly less than the expected one. In this case, the adaptation is limited to informing the load balancer about the event so that no future requests can be directed to that Web Service until its QoWS gets back to its normal level. Severe degradation, as a result of a considerable drop of QoWS, requires binding to a similar Web Service for the interaction that showed the degraded QoWS. A failure is observed if the Web Service is completely down or presenting very low QoWS. In this case, the adapter informs the driver to completely switch to a similar Web Service for the actual operation as well as all upcoming operations requested from the failing Web Service. This is a kind of blacklisting until the Web Service gets to its normal status. A Web Service showing frequent severe degradations and/or failure might be blacklisted forever or even banned from the community. Adaptation detail will be discussed in section VI.

The managerial architecture offers in addition to monitoring and adaptation of QoWS, the following services to all partners.

B. Available Services

A MWS offers services to its peers in the managerial community, to Web Services providers, to other Web Services communities, and to clients of Web Services. Although, the main service offered to all of these partners is the verification and monitoring of QoWS, QoWS-based selection, and QoWS adaptation, a MWS offers services to its peers as part of its duties while operating within a community.

1) Services to peers

Cooperation between MWSs is conducted through the MWS-MWS interface. This interface concerns three categories of interactions: negotiation of mutual services, validation/retrieval of information about a given Web Service, and exchange of summary reports and status information.

Negotiation of mutual services: MWSs negotiate the terms and conditions of the services they deliver/receive using SLA. An agreement specifies the kind of services a MWS is willing to provide to other MWSs and the cost of each of these services (if any).

MWS requests delegation: whether serving a community, a Web Service provider, or a client, when a MWS cannot process a request due to lack of expertise or high load, it requests cooperation of other unloaded MWS with appropriate expertise. In a managerial community, a MWS may not have enough knowledge about a specific Web Service when making decisions (e.g. selection of potential Web Services). This is eventually the case for a composite Web Service offering different services and requiring different expertise domains. In this case, a MWS may ask other MWS within its community in order to get information about that Web Service, such as whether its QoWS has been verified and/or monitored before, and if any, what was the outcome/verdict of that process.

Sharing of Web Services' rating information. MWS within the same managerial community may share rating information of Web Services, in very restrained situations, by sending reports to each other periodically or on demand (e.g., list of top qualified Web Services, list of

worst qualified Web Services). These reports are dated and updated by all MWS and made available to other MWS belonging to that managerial community.

Sharing load. MWSs can get help from each other when they receive a large number of requests from clients. Thus, they need to inform each other about their loads.

2) Services to providers of Web Services and other communities

As part of their responsibilities and as stated by their business models, managers of communities of Web Services should (would like to) protect their communities, their members, and their clients. Before adding a Web Service to a community, the manager should make sure the reputation of this potential member is at an acceptable level and will improve the reputation of the community or, at the worst case, will not downgrade it. The reputation of a Web Service is impacted by the QoWS properties presented in section IV.A.

The managerial community can fully verify the QoWS of a Web Service. This verification consists of checking if the QoWS claimed by a Web Service is in fact supported. This requires generation and application of tests cases and/or passive monitoring interactions of that Web Service with clients. Verification of QoWS might be required in two scenarios: 1) when adding a Web Service to a community and 2) when a provider would like to certify the QoWS its Web Service can offer.

3) Services to clients of Web Services

Selecting suitable Web Services with regards to QoWS provision is a determinant factor to ensure customer satisfaction and then loyalty. Different users may have different requirements and preferences with regards to QoWS. For example, a client looking for a Web Service may require minimal reputation while satisfying certain constraints in terms of price and availability; while another client may put more emphasis on the price rather than the reputation; others consider more the availability of a Web Service rather than both previous properties. As the managerial community collects sufficient information about Web Services (with their consent and/or the consent of their community's manager), it can be used for selection of Web Services based on QoWS.

As for monitoring, each monitor, member of the managerial community, is capable of passive monitoring of Web Services using passive testers. This monitoring is of prime importance to assess the QoWS of a Web Service when serving clients and/or operating within a community. Whenever the monitor observes a degradation of the QoWS, it invokes the adapter to take corrective measurement as will be discussed in section VI.

QoWS-based selection and monitoring require a clear and non-ambiguous specification of quality attributes and functional aspects. In the following section, we show how both of these aspects can be described in an Extended Finite State Machine (EFSM).

IV. QoWS Specification

A. QoWS properties

The set of QoWS properties (e.g. response time, cost…) can be very large and depends widely on Web Services and their clients. In this work, we only consider four main properties: availability, reputation, response time, and cost.

- *Response time*: this represents the time needed between issuing a request and getting its response.
- *Cost:* this is the cost charged for using a Web Service. The Web Service cost may be estimated by operation, by volume of exchanged data, and/or a flat rate plan.
- *Availability:* it represents the probability that a Web Service is accessible (available for use) or the percentage of time that the Web Service is operating.
- *Reputation:* this is a measure of Web Service trustworthiness. It depends on clients' experiences in using the Web Service.

B. Specification Model

Among all formal models that have been used to monitor QoWS, we are going to use EFSM [21]. This model needs first to be described in the same description languages used in the SOA paradigm. However, there have been many other models for monitoring functional and non-functional behavior of Web Services for instance FSM [21] and BPEL [22].

EFSM is a richer model than FSM since it allows the expression of data such as variables and parameters. In an EFSM model, input and output events are parameterized and carry data that transitions manipulate in addition to local variables. Figure 3 illustrates an EFSM model described using an XML representation. EFSM has two more attributes than FSM: predicate and assignments. The first attribute indicates a Boolean expression that should evaluate to TRUE in order to fire this transition. The second attribute represents the set of data manipulations to be performed while firing the transition. In addition, we are instrumenting the EFSM to specify the QoWS attributes provided by a Web Service (see profile tag in Figure 3).

```
<efsm name="Name of EFSM/Web Service">
 <state name="State1" initial="YES">
  <transition ID="t1" input="Input1" predicate="true"
    assignments="x:=0;y:=0;z:=0" output="Output1"
    next="State2"/>
    <Profile name="GOLD"> MnPT = NULL MxPT = 10ms
       SC= "$10"
    </Profile>
  </transition>
  <transition ID="t2" input="Input2" predicate="X<3"
    assignments="x:=2;y:=7" output="Output2" next="State3"/>
    <Profile name="SILVER"> MnPT = 10ms MxPT = 30ms
      SC= "$5"
    </Profile>
  </transition>
 </state>
</efsm>
```

Figure 3. XML representation of EFSM machine

C. QoWS-based Web Service Selection

The selection feature of our architecture is QoWS driven and handled by the QoWS selector. The role of the selector is to retrieve the QoWS requirements from the client's request and match it with the QoWS specification described in the EFSM specification of Web Services (see Figure 3).

V. QoWS Monitoring

The managerial community promotes monitoring of both MWSs and normal Web Services. Each MWS is thoroughly monitored by itself, its peers, clients, providers, and/or monitor(s). These different entities might collect a wide range of statistics that can help in assessing the performance of a specific MWS. These statistics include the number of served requests per period of time, periods of high load, periods of low load, number of delegated requests, and number of queued requests from peers, clients, and/or providers. All gathered information is compiled periodically to assess performance of MWSs and guide in deployment of new MWSs or retrieval/replacement of existing MWSs.

Web Services are monitored whenever serving clients' requests. The monitor Web Service conducts observation of Web Services and MWS. This monitor Web Service is invoked by the community driver or manager. A monitor (also called observer) checks the interactions from/to the MWS Under Observation (MWSUO) or Web Service Under Observation (WSUO) during normal operations for the purpose of detecting misbehaviors. The observer detects a failure by analyzing traces collected from the interactions between a client and a Web Service. It compares this information (e.g. input and output data) carried in each exchanged message with the expected information of the MWSUO or WSUO as described in its EFSM document. A failure is detected once the observer identifies dissimilarities in the collected traces or QoWS and the expected behavior.

A. Communication and monitoring overhead

The communication overhead introduced by the managerial community results from requests delegations, profiles updates and retrieval, and monitoring of Web Services. However, the communication overhead due to types of requests depends mainly on the nature of parameters in a request:

1. If the request parameters are basic data structures (e.g. integer, string, or floats), the monitors will be communicating small messages to each others.
2. If the request parameters are complex objects (e.g. documents, tables, collections of objects, or tables of database), monitors will exchange quite large messages.

Communication overhead introduced by different steps in processing requests by a managerial community is somehow limited and the payoff is considerable.

Monitoring the operations of Web Services requires deep analysis of all exchanged messages. Our architecture proposes an online monitoring system that requires on-the-fly forward of all messages to the monitor. This would introduce a heavy communication overhead if these messages have to be forwarded to a far monitor and requests carry complex data objects. Our architecture proposes to use mobile agent monitors that stand close to the Web Service being monitored (i.e. WSUO). The overhead in this case relates to the cost of moving the mobile observer to the network of the observed Web Service. For large number of requests and/or complex data objects, experience showed [23] that this approach is very efficient as will be discussed hereafter.

The communication overhead introduced by updating and retrieving profiles might be omitted. Indeed, these messages are very small since all attributes in profiles records consist of simple data, many profiles can be retrieved in one request, and compression can be considered.

In the following section, we introduce the adaptation scheme that is triggered once QoWS violation occurred and detected via QoWS monitoring.

VI. QoWS Adaptation

An efficient solution to QoWS adaptation should address the following questions: Who should initiate the QoWS adaptation? When and what are the conditions that trigger the adaptation? Which QoS parameters are the targets for the adaptation? What adaption scheme(s) should be used? Does what the adapted values of QoWS are enough?

The QoWS adaptation process is triggered once QoWS is violated. The role of QoWS adaptation is to maintain, as much as possible, the continuity of provisioning the Web Service when the initially contracted QoWS is no longer guaranteed.

The adaptation scheme we are proposing provides the following features:

- It is QoWS-driven, in a sense that it does not rely only on guaranteeing the functional behavior of a Web Service but in addition guaranteeing the provisioned QoWS.
- Relies on a continuous monitoring of QoWS and trace inspection.
- Adaptation can be done by delegation to services from the same community.
- Web Service execution is modeled as EFSM represented as set of states linked via transitions (to keep track of execution states of a MWS)

An adapter component is responsible for QoWS adaptation and Web Service replacement. The following are the steps the adapter executes once a violation of QoWS occurred in order to guarantee QoWS of provisioned Web Services. Three adaptation solutions are considered depending on the gravity of the degradation:

1. Graceful QoWS degradation: no need for service replacement, the adaptation is performed by

means of reducing the load on the monitored Web Service.

2. Severe QoWS degradation: affected QoWS operations are substituted by operations from other Web Services offering the same functionalities.
3. Service fails to provide QoWS: the QoWS degrades dramatically until it becomes impossible to adapt it; therefore the Web Service is replaced.

Input : Event E ($QoWS_{Ex}$, $QoWS_{Obs}$, OP_j, WS_i) /* the received event from the Monitor */
Data : List WebServices /* List of Web Services in the community $WS_1,...,WS_n$ */
Data : Operation $op_{i,j}$ /* operation j offered by Web Service i */
Data : List $operations_i$ /* List of operations offered by Web Service WS_i: $operations_{i,1},...,operations_{i,m_i}$ */
/* Assumption: $op_{i,j}$ is equivalent to $op_{l,j}$, for all Web Services offering similar functionalities */
Data : List $QoWS_{i,j,k}$ /* List of QoWS offered by $operation_j$ of Web Service WS_i: $QoWS_{i,j,1},...,operations_{i,j,k_{ij}}$ */

1 **if** $(QoWS_{Obs} < QoWS_{Ex})$ // graceful QoWS degradation
2 **then**
3 Notify load balancer not to send any new request until notified
4 **else**
5 **if** $(QoWS_{Obs} <<< QoWS_{Ex})$ // severe QoWS degradation
6 **then**
7 select best match from equivalent WS (for example WS_x) ;
8 switch to $operation_{x,j}$;
9 keep binding to WS_i for other operations
10 **else**
/* if none of previous types, it should be a service replacement */
11 select best match from equivalent WS (for example WS_x) ;
12 switch to WS_x
13 **end**
14 **end**

Figure 4 QoWS Adaptation Algorithm

A. *Graceful QoWS adaptation*

Figure 5 illustrates the main components involved in adapting the QoWS once it degrades gracefully. The sequences of adaptation operations are enumerated. The QoWS monitor triggers an event to the adapter notifying that the QoWS is gracefully degrading (1), once the adapter receives the notification message, it takes the appropriate action and notifies the load balancer to stop forwarding requests to that Web Services (2). The load balancer sends a confirmation message to the adapter (3), and then the adapter notifies the monitor to continue the monitoring of that Web Service.

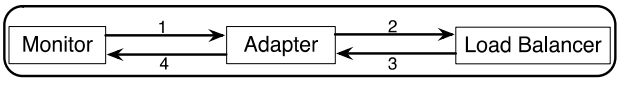

Figure 5 Graceful QoWS Adaptation

B. *Severe QoWS adaptation*

Figure 6 illustrates the main components involved in the adaptation of QoWS once the later degrades severely. The QoWS monitor first triggers an event to the adapter notifying that the QoWS is severely degrading (1), then the adapter checks the QoS contract and compare the contracted QoWS against the observed one (2). Afterwards, the adapter enables the workflow execution engine to call the interface matching component of the adapter (3) to look for operations that substitute the current operation whose QoWS is severely degraded from other services offering the same functionalities (4). Then, the interface matcher provides the workflow execution engine with the list of Web Services offering the same operations (5). Finally, the adapter informs the driver to switch to the same operation of another service (6.1) and the monitor to start monitoring the new Web Service (6.2). In this work, all Web Services offering the same operations are semantically and syntactically equivalent. Therefore, the interface matcher does not look at the syntactical and semantic mismatch, as they are the same.

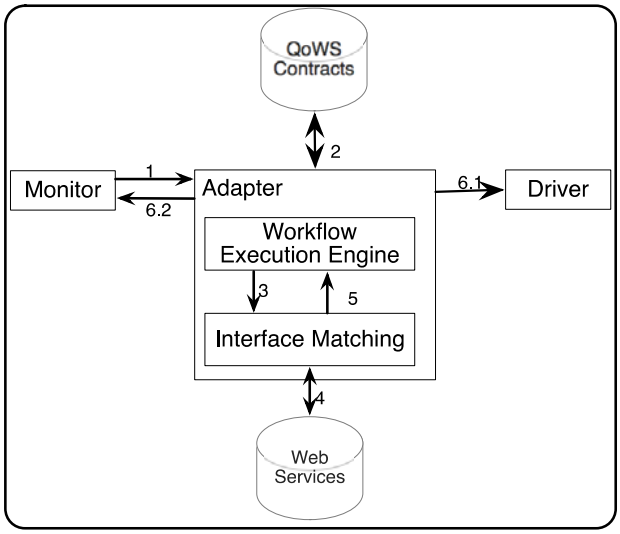

Figure 6 Severe QoWS Adaptation

C. *QoWS adaptation through service replacement*

Figure 7 illustrates the main components involved in adapting the QoWS using Web Service replacement. The monitor triggers an event to the adapter notifying that the Web Service has failed (1), the adapter then checks the QoWS contract and compare the contracted QoWS against the observed once (2), afterward the adapter informs the driver to select an appropriated Web Service replacement (3). The driver communicates with the selector to look for equivalent Web Services (4), the selector search for Web Services providing the same features of the failing Web Services from the Web Service Registry (5) and sends back a list of equivalent Web Services to the driver (6). The driver select among the list the best match Web Service and sends it to the adapter, the later notifies the monitor to start monitoring the equivalent new Web Service.

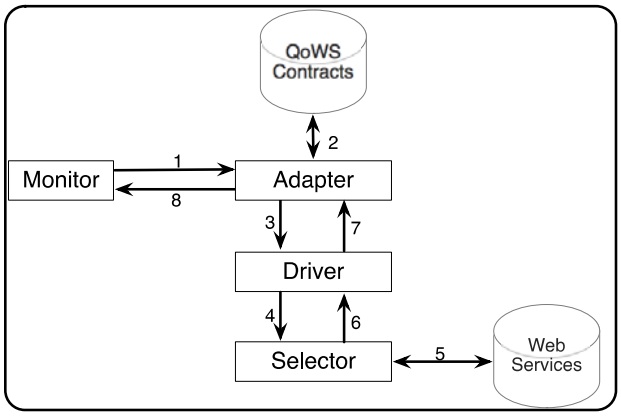

Figure 7 QoWS adaptation through Service replacement

VII. CASE STUDY AND ANALYSIS

To demonstrate the feasibility of our proposed architecture, we have developed a proof of concept (case study) of our proposed architecture. Using this case study, we have conducted a series of experiments in order to evaluate QoWS-aware Web Service monitoring and adaptation schemes supported by our architecture. Scenarios in which all components of the architecture are involved have been considered.

Three adaptation schemes of the adapter were evaluated and applied. In addition, a clients' generator application has been developed so that a large number of requests can be sent to the managerial community of Web Services.

Three scenarios have been conducted to evaluate the adaptation schemes we have proposed. Two QoWS properties namely Response Time (RT) and availability were considered. We have used a number of partner Web Services belonging to the same community.

A. Scenarios

We collected monitoring resulted from the observation of a couple of Web Services' QoWS in two situations: before QoWS adaptation is triggered and after QoWS adaption is triggered on the same Web Services.

1) Before Adaptation

Figure 8 exhibits the result of monitoring the RT over a period of time measured in minutes. We can see that without adaptation, the QoWS contract with regards to RT has been violated frequently. The same applies to the availability of Web Service that degrades and changes very often as shown in Figure 9.

2) After Adaptation

The rest of scenarios are conducted after considering the three QoWS adaptation schemes. Figure 10 shows that the RT has been adapted to a level agreed in the QoWS contract after the severe RT adaptation is applied. By comparing the results above, we can see that our proposed severe adaptation scheme works as expected and the RT is maintained within a range agreed on in the QoWS contract.

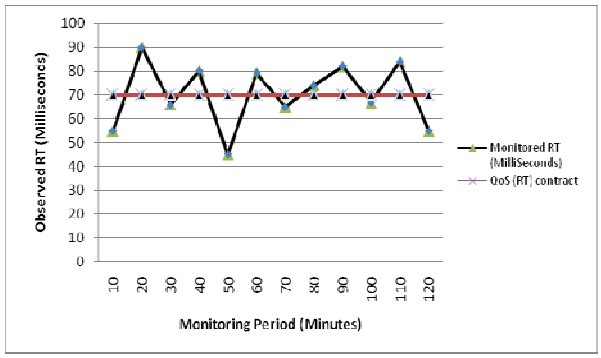

Figure 8. Observed RT before adaptation

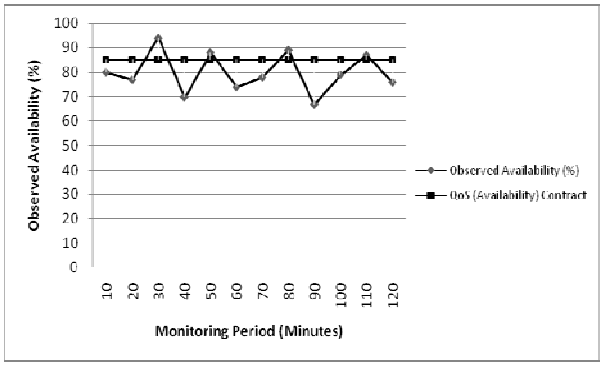

Figure 9. Observed Availability before adaptation

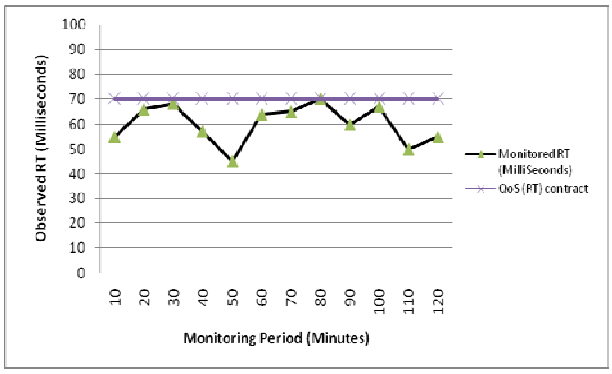

Figure 10. Observed RT after severe QoWS adaptation

Figure 11 presents the result using Web Service availability after we adapt the QoWS via Web Service replacement. By comparing the results before adaptation (Figure 9) Web Service availability increases significantly and stays generally above the pre-agreed QoWS values stated in the QoWS contract.

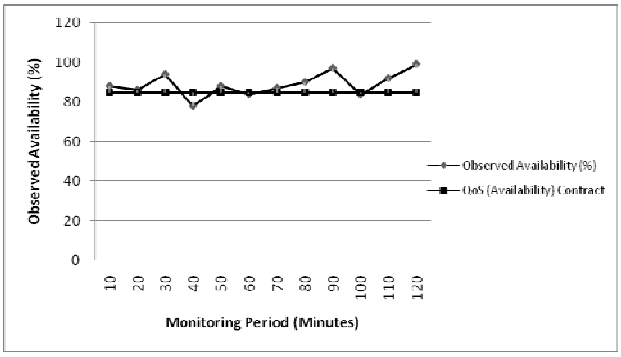

Figure 11. Observed Availability after service replacement adaptation

Figure 12 presents the results of adapting RT using graceful adaption scheme. Compared to the results obtained before adaptation (Figure 8), the RT is improved a bit and maintained more stable and we can now hardly see RT contract violation.

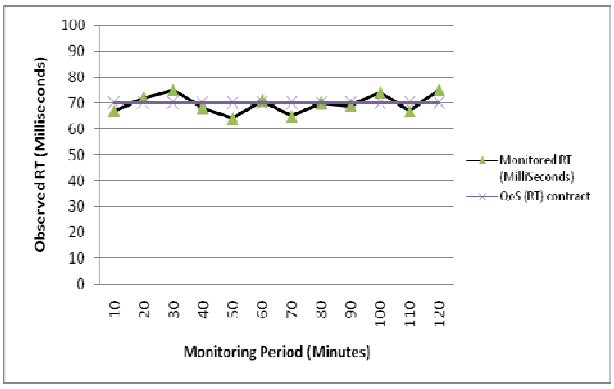

Figure 12. Observed RT after graceful QoWS adaptation

B. *Analysis*

By comparing the results above, we can see that our proposed adaptation architecture works as expected when an appropriate adaptation scheme is selected. Moreover, the QoWS is adapted and maintained to a contractual level.

We can conclude from the preliminary experimentations that it is critical to adapt the QoWS to a certain threshold. In case a service replacement scheme is used, the selection of the best much Web Services is very critical since you cannot assure that what have been chosen yet is the best much selection. The newly selected Web Service, as a replacement of a failing Web Service might fail as well in providing the required QoWS. Also, in the case of graceful degradation, stopping forwards of requests to the failing Web Service does not guarantee it will be back to business as usual quickly. For example, if that Web Service has a big requests buffer, it will take long time to process pending requests, therefore, the QoWS will take time to adapt.

VIII. Conclusion

Nowadays, Web Services providers are trying to maximize their revenues by creating and/or joining appropriate communities. In a community, a Web Service has better visibility and benefits from cooperation of other members when it is overloaded or lacking expertise in a requested domain. Furthermore, owners or members of communities of Web Services would like to protect their communities and its benefits.

In this paper, we proposed a managerial community of Web Services to help in selecting, assessing, monitoring and adapting the quality of service provided by a Web Service. The managerial community is useful before adding a Web Service to a community or to monitor and adapt the QoS of a Web Service operating within a community. Such monitoring gives communities' managers very important and sensitive information about behaviors of different Web Services in their respective communities.

We have proposed and tested three QoWS adaptation schemes of RT and availability properties of Web Services. We currently handle only these two QoWS parameters and we believe they are the most important QoWS parameters that might be subject to degradation. We are planning to extend our work to handle other QoWS and non-functional parameters such as cost and reputation.

As a proof of concept, we implemented a case study and we conducted a series of experiments to evaluate our monitoring and adaptation techniques. The preliminary results are very promising and prove that our monitoring and adaptation approaches perform very well in detecting QoWS violation and adapting their values to match those in the QoWS contract.

In our ongoing and future work, we plan to conduct a complete evaluation and analysis of the architecture. We are working currently on the implementation and tuning of other functionalities. We will eventually try to tackle various business rules in creating and managing communities including the managerial community.

References

[1] W3C, "Web Services Architecture," 2006; http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[2] B. Benatallah, et al., "Facilitating the rapid development and scalable orchestration of composite web services," *Distributed and Parallel Databases*, vol. 17, no. 1, 2005, pp. 5-37.

[3] Z. Maamar, et al., "Web Services Communities-Concepts & Operations," *The 3rd international* conference on Web information systems and technologies, 2007.

[4] B. Medjahed and Y. Atif, "Context-based matching for Web service composition," Distributed and Parallel Databases, vol. 21, no. 1, 2007, pp. 5-37.

[5] H. Song and K. Lee, "sPAC (Web Services Performance Analysis Center): Performance analysis and estimation tool of web services," Lecture notes in computer science, vol. 3649, 2005, pp. 109.

[6] S. Hwang, et al., "A probabilistic approach to modeling and estimating the QoS of web-services-based workflows," Information Sciences, vol. 177, no. 23, 2007, pp. 5484-5503.

[7] G. Canfora, et al., "A framework for QoS-aware binding and re-binding of composite web services," The Journal of Systems & Software, vol. 81, no. 10, 2008, pp. 1754-1769.

[8] Q. Yu, et al., "Deploying and managing Web services: issues, solutions, and directions," The VLDB Journal, vol. 17, no. 3, 2008, pp. 537-572.

[9] B. Overeinder, et al., "Web service access management for integration with agent systems," ACM New York, NY, USA, 2008, pp. 1854-1860.

[10] F. Raimondi and W. Emmerich, "Efficient online monitoring of web-service SLAs," ACM New York, NY, USA, 2008, pp. 170-180.

[11] M.A. Serhani, et al., "VAQoS: architecture for end-to-end QoS management of value added Web services," International Journal of Intelligent Information Technologies, IGI-Global, vol. 2, no. 4, 2006, pp. 37-56.

[12] M.A. Serhani, et al., "CompQoS: Towards an Architecture for QoS composition and monitoring (validation) of composite Web Services," International Conference on Web Technologies, Application, and Services "WTAS", 2006, pp. 78-83.
[13] S. Kalepu, et al., "Verity: a QoS metric for selecting Web services and providers," Fourth International Conference on Web Information Systems Engineering Workshops, IEEE Computer Society, 2004, pp. 131-139.
[14] V. Tosic, et al., "Web Service Offerings Infrastructure (WSOI) - A management infrastructure for XML Web Services," IEEE Symposium Record on Network Operations and Management Symposium, Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States, 2004, pp. 817-830.
[15] V. Tosic, et al., "Web Service Offerings Infrastructure (WSOI)-a management infrastructure for XML Web services," Network Operations and Management Symposium1, IEEE/IFIP, 2004, pp. 817-830.
[16] M. Qiao, et al., "An Architecture for Automatic QoS Adaptation for Composite Web Services," International Journal of Web Services Practices, vol. 4, no. 1, 2009, pp. 18-27.
[17] A. Brogi and R. Popescu, "Service adaptation through trace inspection," International Journal of Business Process Integration and Management, vol. 2, no. Copyright 2008, The Institution of Engineering and Technology, 2007, pp. 9-16.
[18] H.R. Motahari Nezhad, et al., "Semi-automated adaptation of service interactions," 16th International World Wide Web Conference, WWW2007, May 8, 2007 - May 12, 2007, 16th International World Wide Web Conference, WWW2007, Association for Computing Machinery, 2007, pp. 993-1002.
[19] C. Soo Ho, et al., "A comprehensive approach to service adaptation," IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07), 19-20 June 2007, IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07), IEEE, 2007, pp. 191-198.
[20] W. Kongdenfha, et al., "An aspect-oriented framework for service adaptation," 4th International Conference on Service-Oriented Computing, ICSOC 2006, December 4, 2006 - December 7, 2006, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4294 LNCS, Springer Verlag, 2006, pp. 15-26.
[21] R. Dssouli, et al., "Test development for communication protocols: towards automation," Computer Networks, vol. 31, no. 17, 1999, pp. 1835-1872.
[22] T. Andrews, et al., "BPEL4WS Version 1.1 specification," 2003; ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.
[23] A. Benharref, et al., "Efficient Traces' Collection Mechanisms for Passive Testing of Web Services," Information and Software Technology; Elsevier, vol. 51, no. 2, 2009, pp. 362-374.

Mohamed Adel Serhani holds a Ph.D. degree in Computer Engineering from, Concordia University, Canada (August 2006). He is currently an Assistant Professor in the Faculty of Information Technology, U.A.E University, Al Ain, U.A.E. His research area is on web services computing that includes service selection and discovery, service lifecycle, QoS integration in SOA, End-to-End QoS management for web services, QoS and web services composition. He also worked on the application of artificial intelligence techniques mainly fuzzy logic to software engineering, object-oriented metrics, and software quality. He served on several Organizing and Technical Program Committees, including the IEEE DEST, IEEE WiMob, IEEE IWCMC, IIT'09, etc. He also served as guest editor for International Journal of Web Service Practices. He has published one book, around 40 papers in conferences, journals and three book chapters. He has been involved/supervised many projects during the last 6 years on the area of Software Engineering, E-commerce, and Web services.

Abdelghani Benharref received a Ph.D. in Computer Engineering from Concordia University (Canada) in 2007, Master in Network and Telecommnication from Ecole Nationale Superieure d'Informatique et d'Analyse des Systemes (Morocco) in 2000, and bachelor in Computer Science from Cadi Ayyad University (Morocco) in 1998. Since January 2009, he is an Assistant Professor of computer science at Abu Dhabi University, UAE. His interest domains include but not limited to: Web Services, Web Services composition, management of Web Service, QoS of Web Services, software testing, protocl design and validation.