A new Approach for Quality Enforcement in Communities of Web Services

Abdelghani Benharref
College of Engineering and Computer Science
Abu Dhabi University
Abu Dhabi, United Arab Emirates
abdelghani.benharref@adu.ac.ae
M. Adel Serhani, Salah Bouktif
Faculty of Information Technology, United Arab Emirates University, United Arab Emirates
{salahb,serhanim}@uaeu.ac.ae
Jamal Bentahar
Concordia Institute of Information Systems Engineering
Concordia University, Montreal, Canada
bentahar@ciise.concordia.ca

Abstract - **Nowadays, Web Services are considered as de facto and attracting distributed approach of application/services integration over the Internet. Web Services can also operate within communities to improve their visibility and market share. In a community, Web Services usually offer competing and/or complementing services. In this paper, we augment the community approach by defining a specific-purpose community to monitor Web Services operating in any Web Services community. This monitoring community consists of a set of Web Services capable of observing other Web Services. Clients, providers, as well as managers of communities can make use of the monitoring community to check if a Web Service is operating as expected. This paper defines the overall architecture of the monitoring community, the business model behind, different rules and terms to be respected by its members, services it offers to its various classes of customers. The paper also presents promising experimental results using the monitoring community.**

Keywords-component; Web Services communities, monitoring of Web Services communities, Web Services community business rules.

I. INTRODUCTION

Web Services [1] can operate alone or as part of 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; as a community of monks. Hence, a number of animals living in a common home or with some apparent association of interests*". Similarly, a community of Web Services can be composed by Web Services offering related functionalities or sharing similar concerns. Members of a community can compete with each other when offering similar services or can offer complementing services.

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 inter-community and intra-community rules and to enforce them to help sustaining a community in business.

This paper will present a novel framework that would help managers of communities of Web Services keep track of performance of members of their communities as well as enforcing the community rules and terms. This enforcement promotes sustainability of communities by protecting the community, its reputation, its interest, and those of each individual Web Services operating within the community.

The framework is itself a community of Web Services; each of these is a passive monitor. A passive monitor, also known as passive tester ([5], [6], [7], [8]), checks traces exchanged between a Web Service and its client to detect any misbehavior. A passive monitor requires a well-defined description of the behavior of the Web Service and all exchanged traces. The passive monitor can be online or offline: online monitor gets the traces as soon as they happen and analyses them on the fly. However, offline monitoring gets the traces after the interactions are over between a client and a Web Service. In our framework, we put emphasis on online monitoring as it allows real-time detection of misbehaviors. Violations are discovered as soon as they appear which will help the client take judicious and timely decision to continue its interaction with the faulty Web Service or it has to find an alternative Web Service. It will also help managers of communities quickly locate which members are not operating as agreed upon when joining the community.

This work is a major extension of a previous paper where the authors focused on good selection practices when accepting Web Services in a community ([9]). In the present paper, the focus is on real time monitoring of Web Services members of communities. Managers of communities of Web Services would like to make sure that participating Web Services behave according to agreed-on terms and contracts when serving clients or peers inside the same community. In fact, the frameworks supports communities' managers at different operating phases: during creation of a community, when allowing new Web Service to join the community, during operation, in case of receiving complaints from clients, or from other peer Web Services.

During creation or when adding a new Web Service, a community manager might decide to monitor that Web Service during a probation period to make sure it is working properly. Depending on the performance of that Web Service during the probation period, it might be accepted or rejected from the community.

While running the community, the manager can decide to monitor functional and/or non-functional requirements the community is satisfying to check when and where improvements can be made, if possible. Providing a higher

978-0-7695-4462-5/11 $26.00 © 2011 IEEE
DOI 10.1109/SCC.2011.44

Quality of Web Service (QoWS) might give the community competitive advantages over other competing communities.

It might happen that a client of a member of a community complains that the QoWS received is below what he/she agreed on with the community. The manager of the community can initiate a continuous monitoring of the concerned Web Service to investigate the client's allegations.

Members of a community can request services from each other and can balance the load over each other to keep up with the demand during peak periods or when a request cannot be satisfied by a Web Service. In this case, it has to be forwarded to an appropriate Web Service, in the community, who can satisfy it. If a member feels that (some) other members of the community do not cooperate with it as they are supposed to do, the member can request monitoring of those members.

Remaining sections of this paper are organized as follows: next section discusses related works on monitoring of Web Services, their strengths and weaknesses. Section 3 presents the overall architecture of our community along with the main actor's interactions in addition to its business model, terms, and rules. Section 4 details different services offered by the monitoring community and each of its members. Experimentation results and analysis are discussed in section 5. Finally, we conclude the paper and we briefly discuss the ongoing and the future works.

II. Related Work

Web Service Communities (WSCs) are collections of Web Services that are supposed to have similar or related functionalities [10]. These collections are motivated by the ease and the fastness of identifying and discovering Web Services and better market share.

Some works have been done towards addressing the problem of how to discover and quantify Web Services community. For instance in [11], Zhang et al. proposed an approach that forms WSC by grouping closely interactive Web Services which construct Web Service interactive network. They also introduced a technique for composite Web Service discovery.

A consensus of many researchers ([10], [11], [12], [13], [14]) was that the Web Service community concept is basically created in order to sustain high availability of its Web Services. However, the Web Services grouped into a community may have different non-functional requirements. As a result, the mission of a Web Service community management is to ensure high availability without sacrificing a lot of non-functional requirements.

In [10], Subramanian defines high availability of Web Service Community as the ability of a WSC to continue providing services even when master Web Service fails operationally. The author presents a solution to keep the WSC highly available by using a distributed election algorithm that identifies a temporary master Web Service when there is any operational failure in existing master. In order to form high quality WSC, reputation and other selection criteria, have been used for evaluating and ranking Web Service candidates. In their investigation, Elnaffar et al. [12] have stressed the difference between WSC reputations from Web Service providers' perspective and users' perspective. Therefore they proposed WSC architecture based on reputation and defined metrics to evaluate this latter as it is perceived by clients and providers [12]. In order to assess reputation of communities of Web Services, Khosravifar et al. [14] proposed a mechanism based on run-time feedbacks. Evaluation metrics were defined to supervise the logging system in order to verify the validity and soundness of the feedbacks provided by the users and then to assess the reputation of different WSCs.

In [13], Maamar et al. discussed community management, and stated that high-availability of Web Services is based on the traditional replication strategies of Web Services. Although, replacing the latter strategy with using similarly functional Web Services allows avoiding some limitations, it raises other issues. In [13], the authors have examined WSC management issues such as arrival of new Web Services, departure of existing Web Services, identification of Web Services to join composite Web Services, and sanctions on Web Services in case of misbehavior.

For monitoring of WSC, which is the main focus of the present paper, the authors in [15] proposed monitoring capabilities within the Manageable and Adaptive Service Composition middleware. In this middleware, they propose to detect business exceptions and runtime faults. They also provide synchronous and asynchronous monitoring both at the SOAP messaging layer and the process orchestration layer. As another aspect of monitoring, Sun et al. [16] stressed the dynamic aspects of web services and their attributes and proposed a run-time monitoring that analyzes the conformance of a Web Service to the original. A similar aspect of Web Service monitoring was proposed by Wang et al. in [17]. The proposed approach uses a pattern of service constraints that correspond to service requirements and a monitoring model that covers five kinds of system events relevant to client request, service response, application, resource, and management.

The main drawback of works presented above is that they suppose that a WSC will implement as well required management operations. That is, each WSC offers its basic functionalities (hotel, flight, weather…) in addition to management functionalities such as monitoring. Requiring a WSC to offer management functionalities that will be used intermittently is a burden and unwise investment with an eventual low return.

Our framework tries to relief managers of WSC from all monitoring-related issues in such a way that those managers can focus on their primary and core business functionalities rather than management aspects. In fact, our monitoring community offers monitoring services to anyone willing to monitor the behavior of a Web Service.

In the present work, we consider WSCs as a dynamic collection of Web Services that is even more dynamic than its members. A WSC can expand, shrink, and change its QoWS while supporting exchange of requests between members of the same WSC. In this paper, we do not deviate from the original definition of community of people. Moreover, our framework is based on a WSC offering

monitoring services. Using our framework, a member Web Service can monitor another Web Service operating in a community. Each member in our community can be specialized in overseeing one particular feature of Quality of WS of other Web Services and/or responsible of a particular event such as joining a community or ejecting Web Services from a community. Outsourcing monitoring to our monitoring community is straightforward as all members are designed as Web Services and invoking any of these follows the same standardized steps and protocols as invoking any other Web Service.

III. MONITORING COMMUNITY

As discussed earlier, our framework for promoting sustainability of communities of Web Services by enforcing business rules in communities is based on a community of monitors. A community of monitors is a community that is composed of Web Services preserving the ability to check the correctness of interactions between Web Services and their clients. Each of these Monitors Web Services (MWS) can assess functional and non-functional aspects of the behavior of any Web Service Under Monitoring (WSUM) while operating inside a community. An exhaustive list of services offered by the monitoring community will be discussed in section IV.

A. Overall architecture

The core idea in our approach is the monitoring community of Web Services. Figure 1 illustrates an environment with one monitoring community, a client, and two normal competing communities; inside each of them are competing and complementing Web Services. 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 ([18]) community consists of few airlines (Air France, KLM, Delta…). While in the same community, those airlines are competing with each other to attract passengers. However, in exceptional circumstances (aircraft failures, over-booking, code-share flights…), passengers holding Air France tickets can board Delta flights. Share of revenues in these cases is specified in the association terms. Moreover, Skyteam offers complementary services to the American Hotel and Lodging Associations [19], that is, getting customers into and from the hotel. Also, members of AHLA compete with each other for bed occupancy but a hotel can reroute guests to competing AHLA members if it is full or parts of its accommodation have been shut down.

All communities, including the monitoring community, are created and maintained by a manager. In Figure 1, community 1 offers two different services (i.e. functionalities): 1) WS1-1, WS1-2, and WS1-3 compete for the first service and 2) WS2-1 and WS2-2 compete for the other service. Community 2, competing with community 1 offers exactly the same services as community 1: WS3-1 and WS3-2 offer the first service, thus, directly competing with WS1-1, WS1-2, and WS1-3. The second service is delivered by WS4-1 and WS4-2, which are direct competitors of WS2-1 and WS2-2. The monitoring community, however, is offering one service, that is monitoring of Web Services. All Web Services in this monitoring community compete in the same market space while cooperating whenever needed and authorized by the community operating terms and rules.

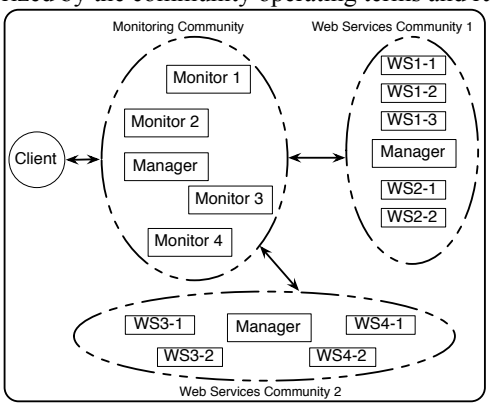

Figure 1. A monitoring Community and Two Normal Communities

When a request is made to the monitoring community, it is received and inspected by the manager. Based on its knowledge on capabilities and resources of each MWS, the manager decides which MWS is likely to handle this request. When a MWS receives a request from the manager or from a peer, it has four options: 1) accept the request and fully take care of it, 2) accept the request and collaborate with a peer to serve it, 3) entirely forward the request to a peer, or 4) reject the request. The monitoring community business model defines conditions and constraints for these four cases as will be discussed in section 3.2.

Figure 2 illustrates a step-by-step scenario using the monitoring community. A client makes a request to the manager (step 1). The manager then selects monitor 2 to take care of the request (step 2). Being little bit overloaded, monitor 2 decides to ask for collaboration from monitor 3 (step 3). Monitor 3 is handling other requests and cannot collaborate; so, it delegates the request to monitor 1 (step 4). In step 5, monitor 5 confirms its will to collaborate with monitor 2 for the sake of this request. Only at this stage, monitor 2 can officially inform the manager that it will satisfy this request (step 6). The manager notifies the client (step 7) so that the later can forward all exchanged traces with the Web Service to be monitored (i.e. the WSUO) to monitor 2 (step 8). All these traces are forwarded to monitor 1 (step 9). The monitoring process culminates at the end of the monitoring process or when a fault is detected. Monitor 1 reports to monitor 2 (step 10), which notifies the manager (step 11). The manager then sends a monitoring report to the client (step 12).

B. Business model

Value proposition: the monitoring community offers a unique service for monitoring of Web Services participating in communities. Managers of communities, providers of Web Services, and their clients can make use of a set of collaborating monitors of Web Services without having to

care about how this might be done and what skills are needed to handle the monitoring activities. In fact, MWS have all the logic to check traces as far as they are provided with the traces and a description of the expected behaviour of the Web Service (see section IV.E). As an aggregation of specialised Web Services for online monitoring of Web Services, a monitoring community offers a unique and single interface for exceptional and trusted monitoring service.

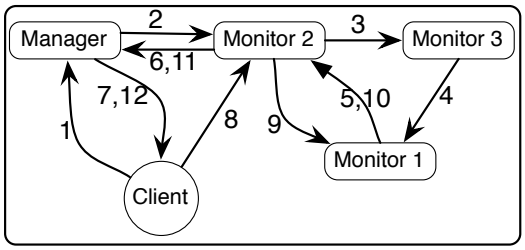

Figure 2. Monitoring Community Utilization Procedure

Revenue model: sharing revenue between the monitoring community manager and providers of MWS is quite similar to those of normal (non-monitoring) communities. Clients of the monitoring community might be charged by transactions or might have a flat monthly or yearly subscription. The monitoring community might get into market by charging fees for transactions based on the monitoring period and the complexity of the behavior of the WSUO. As the monitoring community gets mature with loyal customers, the subscription model will become more advantegeous for all parties. Whatever the model used, we propose the folloiwng schema to share the community benefits:

- For earned revenue from pay-per-transaction model, the community (i.e. its manager) gets a commission of 30% of the transaction revenue when a monitoring request has been delegated to a MWS and has been fulfilled by the same MWS. The remaining 70% goes to that MWS. If two or more MWS have collaborated in processing the request, the community gets a commission of 20% and 80% is shared by MWS who have been involved in that request depending on how much work has been carried by each MWS.
- For earned revenue from subscription, the community divides revenue by usage for each subscriber (based on monitoring time and/or complexity of checked traces). For each usage, the dividend is based on the same schema as in the pay-per-transaction shares (30-70 or 20-80).

Competitive advantages: operating inside a monitoring community gives more visibility and credibility than operating as a single MWS. It's easier for customers to trust and then use a service from a community than from a single MWS especially when the MWS is new in business. On one hand, joining a community gives the newly operating MWS more credibility. On the other hand, a newly admitted MWS with proven reputation will benefit the community and all its members.

Monitoring community terms and rules: the monitoring community is governed by a set of rules to protect the community and each of the participating MWS.

A MWS in the monitoring community should adhere to community rules and provide all appropriate services to community manager (request assignment), peers (collaboration), or clients (accept traces). A detailed list of these services and supporting interfaces that MWS have to provide is discussed in section 4. Rules also define in which situations a MWS can/has to collaborate with peers.

Expansion: A new MWS will not be admitted in the community unless the manager has enough confidence that this expansion will enhance the reputation of the community or at least will not degrade it. This confidence can be gained from many sources: monitoring this MWS over a period of time, reputation of its provider, information from other ranking/monitoring sources, etc. Admission into a monitoring community can be initiated by the manager of the community or the provider of the MWS. In both cases, the MWS will be under probation for a certain period of time and/or certain number of usages. During that period, the MWS might not get paid for the monitoring it's doing and another MWS, member of the community, will monitor the behavior of the MWS under probation. A final decision regarding permanent expansion is taken at the end of the probation period to accept this MWS or to take it out from the monitoring community.

Shrinkage: the monitoring community can shrink as it can expand. Shrinkage happens in two cases: 1) leave and 2) eject. In the first case, a MWS decides by itself to leave the monitoring community. It has to agree with the manager on time to leave and inform peers in the community so that no further request will be delegated to the leaving MWS. The second case is usually a consequence of not abiding by different community's rules where the manager of the community can kick a MWS out from the community. For example, a MWS with low QoWS might be ejected from the community if it does not improve its QoWS after consecutive warnings from the community manager.

Collaboration: a MWS can request help from peers in the community if it feels it cannot handle complete monitoring of functional and non-functional behaviour of a Web Service. For example, it can decide to monitor the functional interactions and ask a peer to check for non-functional aspects. Both of them will monitor the WSUO but the first MWS (who got the request from the manager) will report the verdict of the monitoring. This delegation is optional as far as a MWS can serve the request at acceptable QoWS. Otherwise, it is compulsory for that MWS to request help from others or even to completely delegate that request as this QoWS degradation will affect the whole community.

Delegation: a MWS can decide to completely delegate a monitoring request just assigned by the manager to another peer. The later will perform the whole monitoring activities and will report the final verdict directly to the manager without going through the first MWS who initially got the request from the manager.

The following section defines in details all services a monitoring community and its members have to offer and corresponding application programming interfaces.

IV. MONITORING COMMUNITY SERVICES

The manager of the community and all participating MWS provide a set of services to their customers. The following paragraphs detail the basic services these should provide with descriptions of corresponding interfaces.

A. Services to members of the monitoring community

Each MWS offers two basic services to its peers in the same monitoring community namely: collaboration and delegation. The first one is required so peers can request help from a MWS during the monitoring while the second is required to forward a request to another MWS. Two appropriate interfaces are implemented by all MWS:

receiveCollabortaion(MWS, WS, partialWS Description, startDate, endDate): where MWS is the monitor requesting collaboration, WS is the Web Service to monitor, and partialWSDescription is a description of the behaviour that this MWS should observe. This is a partial description of the behaviour as other parts will be monitored by the originating MWS. Many models can be used to describe the behaviour of a Web Service. Section IV.E lists some of these model and gives an example of a desription as an Extended Finite State Machine (EFSM). The monitoring starts at startDate and ends at endDate or whenever an error is detected, a verdict report is sent back to the invoking MWS.

receiveDelegation(MWS, WS, WSDescription, startDate, endDate): where MWS and WS are as discussed above in the collaboration. WSDescription, however, is a full description of the behaviour of the WS. This usually happens when a MWS cannot handle a monitoring request (new request, collaboration request, or delegation). It then delegates the request to another MWS.

B. Services to providers of Web Services

All interactions of the monitoring community to/from clients go through the manager except forwarding traces, which goes directly from a client of the monitoring community to the appropriate MWS (step 8 in Figure 2). The manager accepts requests for monitoring and returns back (later) the monitoring result/verdict.

Providers of Web Services can use the monitoring community to certify the QoWS of their Web Services. Such certification gives them more credibility and/or a competitive advantage in front of their competitors as some clients might request their service providers to be certified. Certification can also be requested after a client submits a complaint that the received QoWS from a Web Service is below what has been agreed upon with the provider. The manager offers a dedicated interface for certification requests: certifyWebService(WS, WSDescription). This interface returns a certificate document if the Web Service passes the monitoring process, otherwise, the provider is requested to review the QoWS described in the WSDescription and/or improve the performance of involved servers.

C. Services to clients of Web Services

Web Service can request services from the monitoring community to check interactions with a Web Service. This request can serve two complimentary purposes: 1) check if the paid for QoWS is delivered and 2) have a valid and supporting document of QoWS violations, to be used as part of an eventual complaint. The manager offers the checkInteractions interface as follows: checkInteractions(WS, WSDescritpion, startDate, endDate). This interface is similar to certifyWebService except that the later returns a certificate while checkInteractions returns the result of the monitoring with a list of violations if any.

D. Services to community managers

The monitoring community can be of great help to communities' managers willing to build solid communities of high quality Web Services. In fact, a manager can ask the monitoring community to provide a comprehensive report on the sustainability of a new community composed of a list of Web Services using the assessCommunity(WSs, WSsDesrciption, communityDescription). WSs is the set of potential Web Services to form the new community and WSsDescription is a list of their descriptions. The communityDescription defines the QoWS to be provided by the new community. This interface provides the manager of the new community with good insights either this list of Web Services can cooperate to provide the QoWS defined in communityDescription or adjustment will have to be performed on Web Services and/or desired QoWS of the community.

For a community already in business, its manager can make use of the monitoring community during probation periods, after complaints about a member Web Service, certification of a Web Service, or to check and/or certify an upgrade of the QoWS of the community.

E. QoWS description

For a MWS to be able to check the correctness of an interaction, it should be provided with the data of this interaction as well as a concise description of the expected behavior of the WSUM. The following two paragraphs define the QoWS properties we are using so far and how they can be unambiguously specified.

1) QoWS properties: the set of QoWS properties (e.g. response time, availability, reputation…) can be very large and depends widely on Web Services and their clients. In this work, we consider two main properties: response time and availability. Other properties can be added as far as we can precisely define their objectiveness and thresholds.

- *Response time*: this represents the time needed by a Web Service between issuing a request and getting its response.

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.

2) *Specification Model:* Among all formal models that have been used to monitor QoWS, we are going to use EFSM [20]. However, there have been many other models for monitoring functional and non-functional behavior of Web Services for instance Finite State Machine (FSM) [20]. However, an FSM model does not support data which is built into the EFSM model. Figure 3 illustrates an

eXtensible Markup Language (XML) representation of an EFSM model. Transitions of an EFSM model have two more attributes than FSM: predicate and assignments. The first attribute is an assertion that should evaluate to TRUE in order to fire the 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"> MinRT = NULL MaxRT = 10ms
       availability= "95%"
    </Profile> </transition>
  <transition ID="t2" input="Input2" predicate="X<3"
    assignments="x:=2;y:=7" output="Output2" next="State3"/>
    <Profile name="SILVER"> MinRT = 10ms MaxRT = 30ms
      Availability= "75%"
    </Profile>  </transition> </state>
</efsm>
```

Figure 3. EFSM specification as an XML representation

V. CASE STUDY

For the purpose of proving the feasibility of our proposed approach, we have developed a case study in which we validate monitoring features of both functional and non-functional requirements of Web Services belonging to communities by the monitoring community. As shown in Figure 1, we have created two communities of Web services. The first one is a community of flight reservation Web Services and the second is a community of Web Services for hotel reservation. Using this case study, we have conducted a series of experiments in order to evaluate detection capabilities of the monitoring community of QoWS violations.

Three well-designed scenarios are proposed to handle monitoring of Web Service client, providers, and other monitors (peers) within communities. In these scenarios, both functional and non-functional have been monitored based on response time and availability (section IV.E.1).

A. Test-bed Configuration

The test-bed, based on the configuration of Figure 1, consists of:

• Two QoWS-aware communities (hotel reservation and flight reservation), each with its own manager.

• One monitoring community with 4 monitors. Each monitor is specialized in monitoring a certain functional and/or non-functional properties.

• Varying number of Web Services for flight reservation and hotel reservation supporting varying classes of QoWS. Table 1 presents a description of each Web Service and the main operations it provides. MWS and Web Services have been deployed on different hosts.

• A multi-threaded Java client that generates requests to the managerial community. The client application allows the specification of the following:

- Test setup: period of test, requests distributions, etc.
- Locations of MWSs that will receive the requests generated by the client application.

TABLE I WEB SERVICES OPERATIONS

Web Services	Description	Main operations
Flight Reservation (FR) Web Service.	Provides information about flights, make reservation, and payment.	*get_flights_info(), get_flight_detail(), make_fllight_reservation(), update_flight_reservation(), cancel_flight_reservation() pay_flight(),…*
Hotel Reservation (HR) Web Service.	Implemented a set of operations to get information about hotel availability, make reservation, and payment.	*get_hotel_info(), get_hotel_availabilityl(), make_hotel_reservation(), update_hotel_reservation(), cancel_hotel_reservation() pay_hotel(),…*

B. Experiments

We conducted experiments using the above configuration and we collected monitoring resulted from the observation of response time and availability properties of the above Web Services' using different scenarios: 1) monitoring Web Services before their probation, 2) after receiving complaints, and 3) conducting periodical monitoring. In those scenarios, we injected QoWS violations and checked to what extent these will be detected, and evaluated the introduced monitoring overhead.

1) *Scenario #1: Before Probation*. In this scenario, the Web service is continuously monitored during a probation period and extensive test scenarios are applied for both functional and non-functional behavior.

2) *Scenario #2: After Complaints*. In this scenario, test cases are mainly tailored to the type of complaints received and stressed around these. For example, if complaints are mostly related to the performance of Web Services then the monitoring test cases focus on monitoring the performance of Web Services.

3) *Scenario #3: Periodical Monitoring*. In this case, monitoring scenarios are conducted periodically to ensure that the Web Service is behaving properly in terms of functional and non-functional behavior.

C. Discussion

Figure 4 illustrates monitoring the average availability of Web Services within both communities as described above. The monitoring is triggered in three situations as shown in the figure. During probation period, Web Services average availability is relatively high and it decreases slightly while the number of request increases. However, for periodical monitoring, the average availability of Web services fluctuate and remain acceptable then it decreases significantly once the number of requests exceeds 300 requests. Once a monitoring is triggered at the reception of complaints mainly from clients, the average availability of Web Services is maintained high up to 200 requests and

starts decreasing significantly while the number of request decreases.

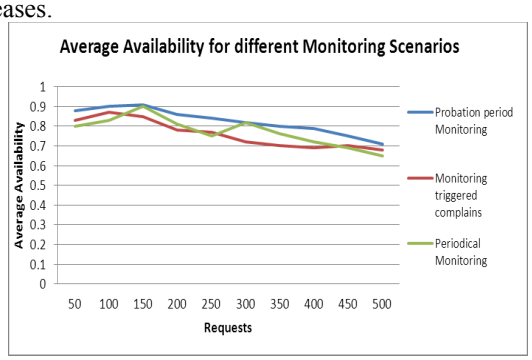

Figure 4. Monitoring Service availability using different monitoring scenarios

Figure 5 illustrates monitoring the average response time of Web services within both communities. The average response time of Web services is relatively small up to a number of requests (around 250 requests). The average response time of Web services increase steadily while the number of requests increases. In the monitoring scenarios, monitoring during probation shows better response time of Web services compared to the two other monitoring schemes. However, periodical monitoring shows little bit higher response time of Web services and monitoring triggered by complaints remains in the middle between the two other monitoring schemes.

Figure 6 illustrates measures of the network load induced by the monitoring activities for the three monitoring schemes. As expected, the periodical monitoring clearly consumes more network resources than the other two monitoring schemes since it is usually conducted for an increased period of time. However, the monitoring triggered by complaints generates the least overhead as it is conducted only when monitoring violations are detected, therefore, it is likely to generate less load. The load generated throughout monitoring during probation is slightly more than the load generated in complaints/-triggered monitoring. This is explained by the extra monitoring activities that is conducted in this situation to insure that the Web Service deserves being part of the community by respecting the community requirements in term of guarantying functional and non-functional properties of provided Web services.

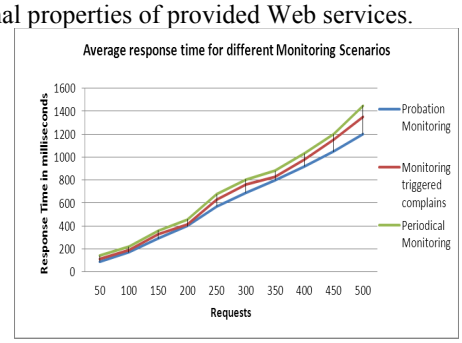

Figure 5. Monitoring Service RT using different monitoring scenarios

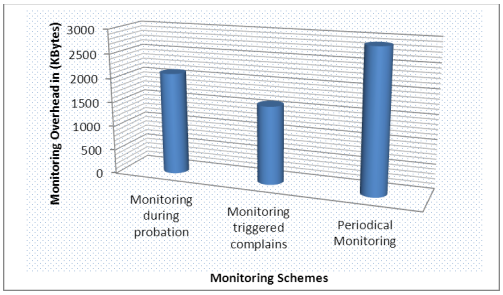

Figure 6. Network load measures for the different monitoring scenarios

Table 2 reports some of the scenarios of monitoring functional behavior of some Web Services within the two communities that have been executed with their observed results.

TABLE II SOME EXECUTED MONITORING SCENARIOS AND VERDICTS

Operation	Scenario	Verdict	Comments
get_flights_info()	Valid (user request is consistent)	Pass	User provides correct info in the query
get_hotel_availabilityl()	Valid	Pass	
update_hotel_reservation()	Invalid (incomplete update data)	Fail	The observer detects wrong update info
cancel_hotel_reservation()	Invalid (fees is applied)	Fail	The observer detects invalid payment info
make_fllight_reservation()	Valid	Pass	
make_hotel_reservation()	Valid	Pass	
update_flight_reservation()	Invalid (possible)	Fail	Service is not available for the specified user's data
pay_hotel()	Valid (payment information required)	Pass	The observer detects wrong payment info
cancel_flight_reservation()	Valid	Pass	Cancelation clause is respected
get_hotel_info()	Valid	Pass	
get_flight_detail()	Valid	Pass	

In summary, the monitoring community has been able to detect all QoWS violations. Moreover, cooperation between participating monitors helped in keeping up with the high number of requests that resulted in a high number of requests to be checked by monitors for correctness. On the negative side, there is a network overhead due to this monitoring. However, the payoff is worth this overhead as the monitoring community gives a community manager an effective online mechanism to assess the performance of Web Services in a community. The online property allows failure detection and identification as soon as problems occur so that service adaptation or even service replacement can be initiated.

VI. CONCLUSION

Nowadays, Web Services are seen as a de facto paradigm for Business-to-Business interactions, which will empower interactions with different services. As the number of Web Services is getting very high and covering almost all online services, the new trend is for Web Services is to get into communities rather than operating alone. Web Services can participate in communities where other Web Services are offering similar or related services.

High quality communities improve visibility and reputations of participating Web Services, and high quality Web Services make their communities of high reputation. Such sustainable communities would like to make sure their high-reputation is being protected and none of their members is damaging it. To make sure all members are adhering to various rules and terms while operating inside the community, community managers can monitor members in specific circumstances.

This paper proposed a new framework that levitates all monitoring-related activities off the shoulders of communities' managers. In fact, the monitoring framework offers a community of Web Services, which are passive observers capable of observing any Web Service operating in any community. We presented and discussed different components of the framework, the business model of the monitoring community, services that members should provide to peers and external clients, and application programming interfaces to acquire these services. We also developed a proof of concepts and used it in observing Web Services members of communities during probation, after complaints, and periodical check out. Preliminary results are very promising since they showed that the monitoring community detects violations, online as far as they appear, in interactions between a Web Service and clients. The overhead of the monitoring process has also been studied.

As future work, we are extending the framework to cover more QoWS properties and looking into methods for traces collection to reduce the monitoring overhead. We are also exploring different adaptation options whenever a violation is detected to decide when it's time advise the Web Service to improve its QoWS and when it is time to look for another Web Service and/or another community.

REFERENCES

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

[2] B. Benatallah, M. Dumas, and Q. Sheng, "Facilitating the rapid development and scalable orchestration of composite web services," Distributed and Parallel Databases, vol. 17, pp. 5-37, 2005.

[3] Z. Maamar, M. Lahkim, D. Benslimane, P. Thiran, and S. Sattanathan, "Web Services Communities-Concepts & Operations", The 3rd international conference on Web information systems and technologies. Barcelona, 2007.

[4] B. Medjahed and Y. Atif, "Context-based matching for Web service composition," Distributed and Parallel Databases, vol. 21, pp. 5-37, 2007.

[5] M. Diaz, G. Juanole, and J. P. Courtiat, "Observer-a concept for formal on-line validation of distributed systems," IEEE Transactions on Software Engineering, vol. 20, pp. 900-13, 1994.

[6] D. Lee, C. Dongluo, H. Ruibing, R. E. Miller, W. Jianping, and Y. Xia, "Network protocol system monitoring-a formal approach with passive testing," IEEE/ACM Transactions on Networking, vol. 14, pp. 424-37, 2006.

[7] E. Bayse, A. Cavalli, M. Nunez, and F. Zaidi, "A passive testing approach based on invariants: application to the WAP," Computer Networks, vol. 48, pp. 247-66, 2005.

[8] B. T. Ladani, B. Alcalde, and A. Cavalli, "Passive testing - a constrained invariant checking approach", 17th International Conference on Testing of communicating systems (TestCom), Lecture Notes in Computer Science (LNCS), pub: Springer Verlag. Montreal, Que., Canada, pp. 9-22, 2005.

[9] A. Benharref, M. A. Serhani, S. Bouktif, and J. Bentahar, "A Managerial Community of Web Services for Management of Communities of Web Services", 10th international conference on New Technologies of Distributed Systems. Tozeur, Tunis, pp. 97-104, 2010.

[10] S. Subramanian, "Highly-available web service community", 6th International Conference on Information Technology: New Generations, ITNG 2009, April 27, 2009 - April 29, 2009, ITNG 2009 - 6th International Conference on Information Technology: New Generations, pub: IEEE Computer Society. Las Vegas, NV, United states, pp. 296-301, 2009.

[11] Z. Xizhe, Y. Ying, Z. Mingwei, and Z. Bin, "A composite web services discovery technique based on community mining", 2009 IEEE Asia-Pacific Services Computing Conference, pub: IEEE. Piscataway, NJ, USA, pp. 445-50, 2009.

[12] S. Elnaffar, Z. Maamar, H. Yahyaoui, J. Bentahar, and P. Thiran, "Reputation of communities of Web services - Preliminary investigation", 22nd International Conference on Advanced Information Networking and Applications Workshops/Symposia, pub: Inst. of Elec. and Elect. Eng. Inc. Gino-wan, Okinawa, Japan, pp. 1603-1608, 2008.

[13] Z. Maamar, Q. Z. Sheng, and D. Benslimane, "Sustaining Web services high-availability using communities", 2008 3rd International Conference on Availability, Reliability and Security (ARES '08), , pub: IEEE. Piscataway, NJ, USA, pp. 834-41, 2008.

[14] B. Khosravifar, J. Bentahar, P. Thiran, A. Moazin, and A. Guiot, "An approach to incentive-based reputation for communities of Web services", 2009 IEEE International Conference on Web Services (ICWS), 6-10 July 2009, 2009 IEEE International Conference on Web Services (ICWS), pub: IEEE. Piscataway, NJ, USA, pp. 303-10, 2009.

[15] A. Chourmouziadis and G. Pavlou, "Web services monitoring: An initial case study on the tools perspective", NOMS 2008 - IEEE/IFIP Network Operations and Management Symposium, pub: Inst. of Elec. and Elec. Eng. Computer Society. Salvador - Bahia, Brazil, pp. 827-830, 2008.

[16] S. Mingjie, L. Bixin, and Z. Pengcheng, "Monitoring BPEL-based Web service composition using AOP", 2009 8th IEEE/ACIS International Conference on Computer and Information Science (ICIS), , pub: IEEE. Piscataway, NJ, USA, pp. 1172-7, 2009.

[17] W. Qianxiang, S. Jin, D. Fang, L. Yonggang, L. Min, H. Jun, and M. Hong, "An online monitoring approach for Web service requirements," IEEE Transactions on Services Computing, vol. 2, pp. 338-51, 2009.

[18] SkyTeam, "SkyTeam", at www.skyteam.com, Visited on 2010.

[19] AHLA, "American Hotel and Lodging Association", at www.skyteam.com, Visited on 2010.

[20] R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary, and C. Bourhfir, "Test development for communication protocols: towards automation," Computer Networks, vol. 31, pp. 1835-72, 1999.

[21] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "BPEL4WS Version 1.1 specification", at ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, Visited on 2009.