Towards the Testing of Composed Web Services in 3rd Generation Networks

Abdelghani Benharref, Rachida Dssouli, Roch Glitho, Mohamed Adel Serhani

Concordia University
1455 de Maisonneuve West Bd, Montreal, Quebec
H3G 1M8, Canada
{abdel,m_serhan}@ece.concordia.ca,
{dssouli,glitho}ciise.concordia.ca,

Abstract. With the proliferation of web services in business and as the number of web services is increasing, it is anticipated that a single web service will become insufficient to handle multitude, heterogeneous, and complex functions. Hence, web service composition will be used to create new value added services with a wide range of functionalities. Management of a composed web service is a complex issue compared to the management of a non-composed (basic) web service. In this paper, we propose a multi-observer architecture for detecting and locating faults in composed web services. It makes use of a network of observers that cooperate together to observe a composed web service. An observation strategy based on a set of heuristics is presented to reduce the number of web services to be observed. Observers are developed as mobile agent observers to help reducing the load introduced by the observation. Algorithms for fault detection, notification, and collaboration between observers are described. Finally, the architecture is illustrated through a case study for observing a composed teleconferencing web services in a 3G network. Different components of the architecture are developed. The network load introduced by the observation is measured and the fault detection capabilities of the architecture are discussed.

1 Introduction

Web services offer a set of mechanisms for program-to-program interactions over the Internet [1]. They make use of a multitude of emerging standard protocols, such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI).

Managing web services is critical because they are being deployed actually in heterogeneous environments and used in a wide range of applications especially in 3G networks. In 3G networks, they are being used for engineering Value Added Services (VAS). VAS are telecommunication standards that add value to those services already available on the network. Another use is digital imaging where their use is being standardized [2]. Their use in telecommunications networks is being standardized by the Open Mobile Alliance (OMA) [3].

Testing web services in open environments with multitude of participants is a hot issue. Testing can be active or passive. In active testing, fault detection is usually based on test cases that are applied to the web service under test. Passive testing, known also as *passive observation,* is based on traces collection and traces analysis.

A new kind of web services is known as "composed web services". A composed web service is any web service that makes use of a set of available web services to provide a different, more complex, service. Web services composition is generating considerable interest in recent years ([4], [5], [6]). It has a considerable potential of reducing development time and effort for new applications by reusing already available web services. The composed web service is also known as the final web service and a service participating in a composition as a basic web service.

Currently, there are standards or languages that help building composed web services such as: WSFL [7], DAML-S [8], and BPEL [9]. These languages make the web services composition process easier by providing concepts to represent partners and orchestrate their interactions. BPEL, which represents the merging of IBM's WSFL and Microsoft's XLANG, is gaining a lot of interest and is positioned to become the primer standard for Web service composition. This is the main reason for which BPEL is used in our work and then will be considered in the remaining parts of this paper.

Observation of composed web services is more complex than observation of basic web services. For instance, a fault occurring in the composed web service can originate in one of the basic web services and propagate to another basic web service. Furthermore, some faults may occur due to the composition itself, these faults are known as feature interaction.

Tracking a fault into its originating web service, will require the passive observation of all, or a subset of, the basic web services. As in distributed systems [10], this observation requires a network of observers rather than a single observer.

In this paper, we propose a novel architecture for online fault management of composed web services by observing their basic web services as well as the composed web service. The architecture is rooted in passive observation. The observers are model-based and are designed and implemented as mobile agents. The architecture makes available a web service observer that can be invoked and mobile observers that are sent back following an invocation.

The remainder sections of the paper are organized as follows: section 2 presents briefly web services composition and involved technologies followed by related works on management of composed web services. Section 3 and 4 discuss respectively the requirements and the fault model of the new architecture for observation of composed web services. Section 5 introduces different components of the multi-observer based architecture. It also discusses the limitations of the architecture in terms of necessary resources and network load. In section 6, we illustrate the observation procedures through a case study where a conferencing composed web service is observed. Finally, we provide a conclusion that summarizes the paper and discusses items for future work.

2 Related work

Management of composed web services is a key issue for their success. Nowadays, this management is vendor-dependent and too much coupled to the application servers on which the composed web services are deployed. Few companies provide limited management features embedded within their platforms. The BPEL process manager [11], ActiveBPEL [12] and Process eXecution Engine [13] provide web-based consoles to deploy/undeploy services and manage their instances. These tools can only be used by the service provider. They manage the composed web services as if it was a basic web service, that is, without taking into consideration the management of basic web services participating in the composition. Moreover, since the tools managing basic web services are also vendor-dependent, exchange of information between different tools managing different entities is not straightforward.

Most of research activities on management of composed web services are actually on non-functional aspects of composed web services such as Quality of Service (QoS) ([14], [15], [16], [17]). For functional aspects, the authors in [18] propose the publication of some testing scripts in the registries. These scripts can be used by entities to test the correctness of desired web services. This approach requires active testers and not transparent to concerned web services.

The web services-based architecture for management of web services presented in [19] is limited to the observation of basic (non-composed) web services. It does not offer mechanisms to observe composed web services. The observation starts by invoking the web service observer. The latter generates a mobile agent and sends it to the hosting platform. The mobile observer checks all the traffic between the client and the observed web service and reports misbehaviors.

In this paper, we extend this architecture to observe composed web services while respecting its initial properties of transparency and availability. The observation is transparent since it does not invoke the web service for the sake of testing. The architecture is also available to all involved parties including the web service provider and the requestor since the architecture is based on web services.

3 Requirements

As stated above, observation of composed web services is based on the observation of the final web service and the participating basic web services. A set of information/resources is required for the sake of this observation. First of all, the web service observer must have access to the choreography document describing this composition. Another issue to solve toward making this observation possible is how to get the exact locations of the participating web services. Once this list of locations is known, models of the web services in this list (FSM or FSM annotated with timing properties) and WSDL documents should also be handed to the observers before the observation.

Discussions of possible mechanisms to satisfy these requirements are presented in section 5.4.

4 Fault model

Fault detection is based on the information contained within the available resources. The models of these resources can be grouped in two groups: statefull (FSM/Annotated FSM, BPEL) and stateless (WSDL). The observers use this information to detect the following classes of faults:

From BPEL:

- **Ordering Faults (OF)**: this fault occurs when the order of invocations of different participating web services is not respected. The "activities" section of the BPEL document describes the rules and order in which participating web services must be invoked. It is in fact a violation of an orchestration scenario that is a global property. This fault only can be detected by a global observer.

From FSM/FSM with timing annotations:

- **Input Fault (IF)**: an input fault occurs if a requestor invokes an operation unavailable from the actual state of the web service. Unlike OF, this is a local property.
- **Timing Constraints Fault (TCF)**: when monitoring the response time of web services, observers can measure the response time of web services and compare it to the threshold described in the model.

From WSDL:

- **Input Type Fault (ITF)**: an input type fault is observed when a method is invoked with a wrong number and/or wrong types of parameters with regards to its signature published in the WSDL.
- **Output Type Fault (OTF)**: this fault occurs if the type of the returned result is different from the type expected in the WSDL document.

5 Multi-observer architecture

In this section, we will present different functional entities of the architecture and their interactions. We will then detail required steps for observation, starting from the invocation of the web service observer and ending with the result of the observation. Some steps of this procedure will smoothly change depending on the participation of different involved web services as will be presented latter.

5.1 Overall architecture

The multi-observer architecture is illustrated in Figure 1 where the global observer and the local observers cooperate for fault management of composed web services. Local observers check exchanged messages observed at their points of observation and send these messages to the global observer. Whenever a local observer detects a fault, it informs the global observer then location and isolation procedures take place.

Due to the position of observation points, fault location at this level is restricted to the location of a faulty web service, not the exact component within this web service.

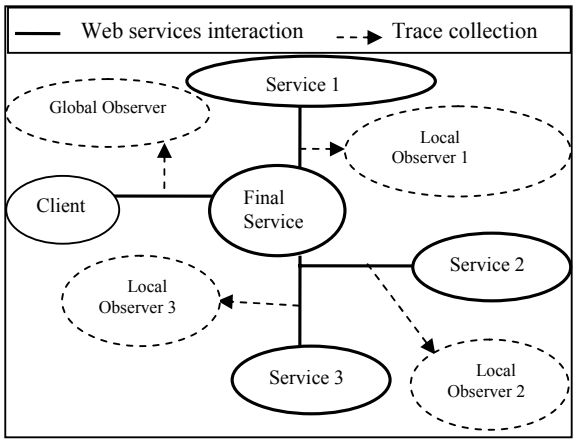


Figure 1 Multi-observer architecture

In the case where a basic web service is itself a composed web service, the same architecture applies for its observation. That is, the web services participating in its composition will be observed also. This gives the architecture a tree structure and makes it very flexible in the observation of composed web services.

One of the design keys to be studied is the number of observers. In some cases, observing all participating web services in a composition is nothing but costly and useless. A full observation can dump the observers and the network with redundant information. Observing, for example, just the main web services that represent the core of the composition can be enough from a fault detection point of view. Potential suggestions and hints to select the web services to observe are discussed in section 5.3.

For each web service in this list, a mobile agent will passively observe its behavior. Two problems to solve: who hosts the mobile agent and how to provide it with exchanged messages? These problems are implementation-related issues and will be discussed in section 6.3.

5.2 Procedure

Observation is performed in two main steps. The first step consists of the configuration of the observation components, and the second step is fault management.

Observation is initiated by invocation of the web service observer. This is done by the entity willing to observe, which can be the provider of the composed web service, its client, one of the basic web services, or a mandated third party. After a successful invocation, the web service observer generates a set of mobile agents and sends them to the location(s) specified during invocation. Once the mobile agents reach their

target locations, one of them becomes the global observer; other mobile agents are local observers. The local observers must inform the global observer of their locations and information about particular basic web services they are observing. At that point, all the components of the architecture are ready to start observation at the time specified during the invocation.

After deployment and configuration of all the observers, they start traces analysis at the time specified during the invocation. This observation will end at the time specified also during invocation. Whenever misbehavior is observed, local observers report to the global observer who reports to the web service observer.

5.3 Optimization

In this section, we discuss a set of suggestions and potential criteria that can be considered to build the list of web services to be observed.

For the selection of web services to observe, an important criterion is the number of interactions between the final web service and a basic web service. If a web service has few published interfaces and is invoked few times while others are invoked very often, observing the latter web services can be more appropriate. Another criterion is the complexity of a basic web service, from its FSM model: more a model of a web service is complex (number of states, number of transitions, etc.), more is the necessity for its observation.

Statistics on previous detected faults is another criterion. If faults occurred in a web service a certain number of times, a periodic observation of this web service can be a wise decision.

Selection of web services to observe might be implied by preferences of the final web service provider. These preferences depend on the importance a basic web service is playing in the composition or the tolerance of the final web service to some specific faults generated by some specific basic web services.

5.4 Participation of web services in their self observation

The information required for observation (section 3) can be gathered through participation of involved web services providers: the provider of the composed web service, the providers of basic web services or from both. We designate these types of participation, respectively, as *final web service provider's participation*, *basic web services providers' participation* or *hybrid participation*.

Final web service provider participation. In this participation, the final web service provider supplies all the required information and resources necessary for the observation. This includes the BPEL description, WSDL documents, models of the web services (basic and final), and the list of nodes to host the mobile agents observers.

This kind of participation is completely transparent to basic web services and their providers. Additionally, due to the cloning nature of mobile agents, the web service observer sends only one mobile agent to the final web service provider's side instead of a separate mobile agent for each web service to be observed. This mobile agent

will clone itself once it gets into its location. Doing so reduces significantly the traffic generated by moving mobile agents. If n is the number of web services to be observed, the complexity of the introduced load decreases from $\Theta(n)$ to $\Theta(1)$.

The load that will be introduced by the cooperation of observers to detect and locate a fault is limited to in-site load, that is, within the provider's domain since all observers are located there. The complexity of this load can be considered as $\Theta(1)$. Synchronization of observers is also easier than if observers were scattered between many sites.

The major weakness of the participation of one side is that all information, resources and observation activities will be within one web service provider.

Basic web services providers' participation. Unlike the centralized participation, the basic web services' participation requires the participation of the providers of all web services that have to be observed, including the final web service. Each web service provider supplies the WSDL document and the model of its web service and hosts the associated mobile observer. In addition, the final web service provides the BPEL document.

The network load is the major weakness of this type of participation. First, a mobile agent is generated and sent to each web service in the list of web services to be observed. The complexity of the load here is $\Theta(n)$. The cooperation of the observers introduces also another $\Theta(n)$ network load since observers are in different locations.

Hybrid participation. The hybrid participation is a compromise between the two kinds of participation presented above. The participation is neither completely distributed nor centered. The final web service provider supplies a portion of the required information and resources while a subset of the list of web services to be observed supplies the remaining portions.

This can be a possible alternative when the final web service can not provide all the information and resources and only a subset of basic web services' providers are willing to participate in the observation. Those basic web services providers' who accept to participate in the observation will supply the information related to their web services and host the associated mobile observers. The final web service provider's furnishes information for other basic web services.

The configuration of the hybrid participation ranges between the centralized configuration and the distributed information, depending on how many basic web services providers' are participating in the observation and how much. Thus, the complexity of the load generated by moving the mobile observers ranges from $\Theta(1)$ to $\Theta(n)$ and for the cooperation of observers from in-site load to $\Theta(n)$. In the average, these complexities are around $\Theta(\log n)$.

In the following section, algorithms implemented by observers are presented and discussed.

5.5 Algorithms

Passive observation is performed in two steps: 1) passive homing and 2) fault detection [20]. The homing procedure is required to bring the observer to the same state as the observed web service. It is needed if the observation starts while interaction between entities has already started. When the observation starts at the

same time as the interaction between observed entities, the homing sequence is empty.

For fault detection, every observed event in traces (request or response) is checked against the expected behavior in the corresponding model. We must note here that there are some cases where the observer can not decide if a response is expected or not. This is due to the fact that when the observation starts, it may miss some previous requests and the homing procedure might not give indication on requests not yet served. This is mainly the case for asynchronous invocations.

Each time a fault is detected by a local observer, a notification is sent to the global observer. Notifications must be purged before their correlation. This is done through two methods: purgeFinalNotification implemented by the global observer and purgeLocalNotification implemented by local observers. The main purging role is the ability of a receiver (client, final web service or basic web service) to detect a faulty received request or response. When a local observer detects an output fault, it notifies the global observer. It waits then for the reaction of the invoked web service. If the response of the latter contains a fault indication (in the SOAP message), the local observer informs the global observer. Otherwise, it sends a second notification to the global observer requesting fault location.

A faulty output generated by a web service will be detected by its associated observer. It will also be detected as an input fault by the observer of the receiving web service. Both observers will generate fault notification. The two notifications must be correlated since they refer to the same fault.

After receiving a notification from a local observer, the global observer associates it, if possible, to a previous fault or notification and updates the fault records accordingly. It waits then for a second notification for a specific period of time before starting the correlation. This starts by checking the fault records for previously detected fault. If the same fault has been detected before, the list of suspected web services is updated with the faulty web service(s) in the fault record. The list of suspects is then augmented by all basic web services invoked before the notification. This list is derived from the “activities” section of the BPEL document. Traces observed by the local observers of the web services in this list are checked to find the faulty web service. This process is repeated until a faulty web service is identified, remaining web services are not observed, or no decision can be made due to a lack of information on behaviors.

In the next section, we illustrate the applicability of the architecture through a motivating example of a composed web service for conferencing. The detailed requirements and steps for observation are depicted all along this example.

6 Case study

In this section, we present our experiments using the multi-observer architecture to observe a composed web service. We introduce the context of utilization of the composed web service and its participating basic web services. We show a situation where the observation of basic web services gives more insights for fault

identification. We present implementations of different components of the architecture, and discuss results with analysis.

6.1 Context

For the end of year meetings, a general manager has to meet with managers from different departments (Sales and R&D for example). Managers are located in different locations and due to their time tables cannot meet in a single meeting room. A practical option is to perform these meetings in a series of teleconferences. Only mangers are concerned and only those of them that are in their offices can join a conference. This is implied by security issues since confidential information will be exchanged during the meetings and communication between different locations is secured (VPN for example). At the end of each meeting, meetings' reports must be printed and distributed among all participating managers.

The manager decides to use a "Conferencing Web Service" (CWS), a composed web service, who performs all of the required tasks. In fact, it allows creation of a conference, add and remove users depending on their locations and profiles. At the end of each meeting, the CWS submits the produced reports for printing. Once printed and finalized, the paper version is distributed to appropriate locations.

6.2 Web services

To perform the tasks presented above, the CWS is a composition of the following basic web services:

- Presence WS: this web service contains information on users' profiles (name, address, location, status, position, availability).
- Sensors: this web service detects the physical presence of users.
- Call Control: this web service creates and manages a multiparty conference (initiates the conference, adds/removes users, and ends conferences).
- Printing: at some points during the conferences or later on, managers may want to print documents (meeting reports …). The printing web service will print these documents and keeps them for shipping.
- Shipping: documents printed during and after the conference should be distributed among users located in different locations. The CWS informs the shipping web service of the location of the documents to be shipped and their final destinations.

Figure 2 shows the composed CWS and its interactions with the basic web services.

6.3 Implementation issues

All web services, including the web service observer, are implemented in BEA WebLogic. In fact, CWS is implemented in BEA even if it has a BPEL description. This is due to some limitations of the BPEL language and the available (non commercial) application servers. Implementing the CWS in BEA does not affect the

observation process since the latter deals only with the exchanged SOAP messages which are independent from the adopted platform.

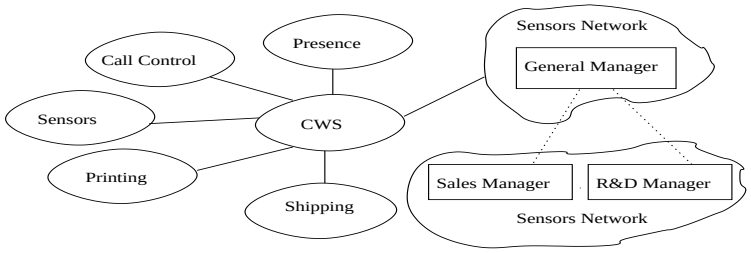


Figure 2 Composed/composing web services

To host the mobile observers, a mobile agent platform should be available. In this case study, we use JADE [21], an open source platform easy to configure and deploy. All nodes willing to host mobile observers must download and configure the JADE libraries. The configuration of jade consists of adding the path of different libraries to the "path" system environment variable.

For trace collection, in this case study, we make use of the SOAP Handlers available within the BEA platform. A SOAP Handler, a special java class, intercepts a request or a response to/from a web service before it gets to the core web service or the client respectively, and can perform operations on it. In our case, the SOAP handler sends each event (request or response) in a UDP Datagram to the concerned mobile observer. The date of occurrence of the event is also sent in this datagram so that the observer can compute the response time. To be able to detect lost UDP datagrams, a sequence number field is used. When a mobile observer detects a lost Datagram (wrong sequence number), it suspends the fault detection and re-perform the homing procedure. It restarts the fault detection once this procedure is achieved correctly. Since the behavior of SOAP handlers within all observed web services is similar, a unique generic SOAP Handler is developed and then distributed to all providers.

6.4 Single observation

When using the single-observer architecture initially presented in [19], the observer will check only the traffic between the manager and the CWS. Figure 3 shows the overall configuration and the information (traces) available to the observer where it is not aware of the interactions (request/response pairs) between CWS and basic web services. By doing so, if the CWS fails to provide the requested service or if the QoS degrades, the observer cannot designate the faulty web service. For example, if the "Sensors" web service (basic WS) fails to check the actual physical location of a manager, the CWS can not create a conference. From the observer's point of view (and then the manager's point of view), the CWS failed to create the conference. No more indication on the failure is available. Figure 4 shows a typical observation scenario from invocation of the observer (WSO) to the delivery of the verdict of

observation. In this scenario, traces are collected through a participation of the web service's provider.

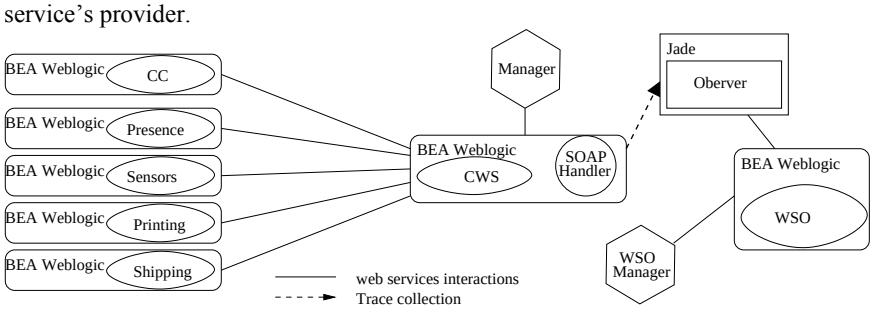


Figure 3 Single-observer configuration

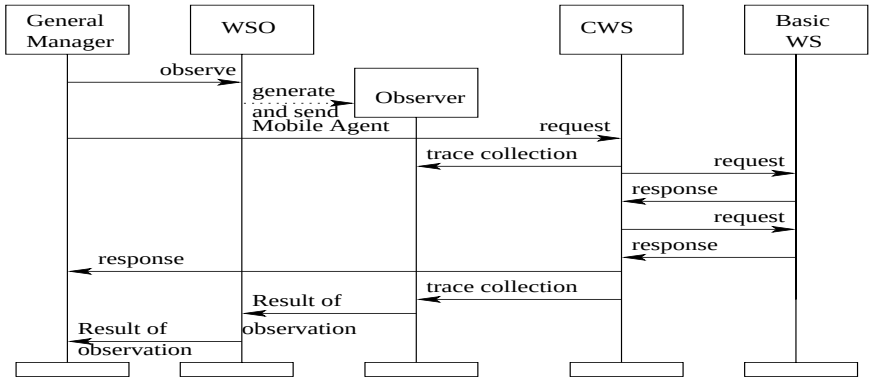


Figure 4 Single-observer scenario

As will be illustrated in the following subsections, the multi-observer architecture gives more information in case of misbehaviors. This capability is made possible by using a network of observers rather than a single observer. Whenever an abnormal event occurs, cooperation between observers is initiated to track the faulty web service.

6.5 Multi-observer observation procedure

The general manager is highly concerned about the environment in which meetings will be carried out using CWS. He decides to make use of the passive observer available as a web service (WSO) to observe the behavior of the CWS. In addition to the observation of the CWS, the manager needs to assure that all the steps are performed according to the agreed on contract and QoS. All the providers accept to participate in the observation. The provider of the CWS will host all the mobile observers. It will also provide the BPEL and WSDLs documents, and the FSM models of each of the basic web services.

Once deployed and configured, mobile observers start by performing the homing procedure. When this procedure is carried out correctly, fault detection starts. Each local observer is listening to a UDP port to receive events from SOAP handlers. The global observer is listening to two different UDP ports: one to receive events (request or response) from local observers and another port to receive information on detected faults by the local observers. Each event from a client to its web service is sent by the SOAP handler to the attached local observer. The latter forwards this event to the global observer and checks the validity of this event with regards to the model of the observed web service. If a fault is detected, the local observer notifies the global observer through a UDP datagram. The global observer tries to associate the new received fault with a previous fault. If the correlation fails, the global observer notifies the final service provider, otherwise, the fault is logged and fault detection continues.
For the purpose of this case study, we developed a graphical client allowing the user to select one of the operations to invoke and provide valid or invalid parameters. Figure 5 shows the overall configuration of interacting web services, mobile observers and communication between these entities.

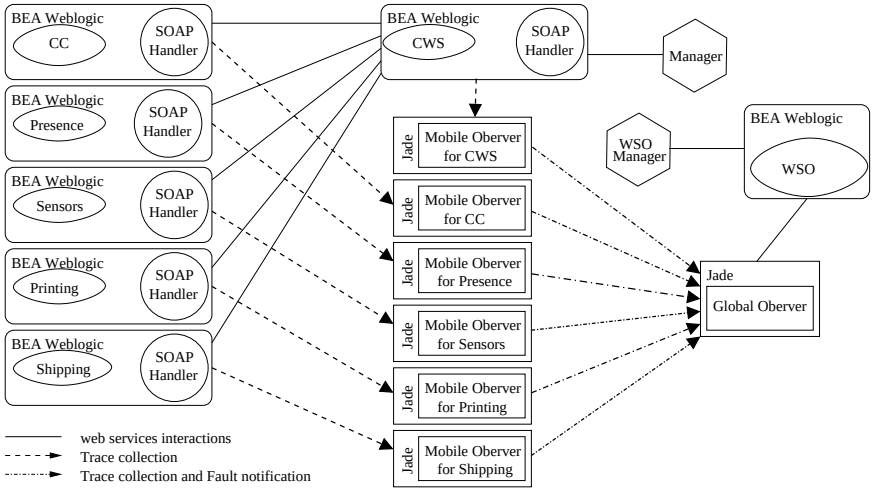


Figure 5 Multi-observer configuration

The observation procedure of CWS is performed following the steps illustrated in Figure 6. To keep the figure simple, just one web service handler and one web service client are depicted in the figure.

6.6 Optimization

The main web service in the composition of the CWS is the Call Control web service. For this reason, we decide to observe it. Moreover, security of communication during conferences is of prime importance. As requested by the general manager, only managers that are in their offices should participate in a conference. So, the

observation of the Presence and the Sensors web services is required. The Printing and Shipping web services are the only web services that deal with documents, so we decide to observe only one of them, the Printing web service. We assume that in case of a misbehavior during printing and shipping procedures, if the fault is not detected at the Printing web service by its attached observer, the fault is then within the Shipping web service.

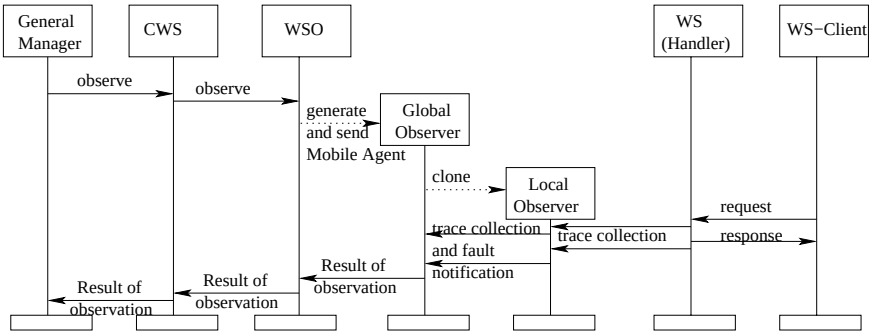


Figure 6 Multi-observer scenario

6.7 Results and analysis

Network load. The network load introduced by the observation is classified into two classes: 1) load due to the deployment of mobile agents and 2) load due to the trace collection process.

Deployment load. Since all observation activities is taking place within the final service provider's side, only one mobile agent is generated by the web service observer and sent to the hosting platform. The size of the traffic to move a mobile agent from the web service observer to the final web service provider is around 600 Kilobytes (600 Kb). This size is smaller than the size of the mobile agent that was initially used in [19]. The new mobile observer offers in addition to the fault detection capabilities, correlation procedures. This includes the ability of a local observer to send an event to the global event, and the global observer to process a received fault and correlate it with previous faults. This reduction is made possible by reducing the size of required libraries and tuning the used data structures.

Trace collection load. Generally, for each interaction between a web service and its client, 2 UDP datagrams are generated: a first datagram from the SOAP handler to a local observer, and a second datagram from this local observer to the global observer. Whenever a fault is detected in a local observer, a third datagram is sent (fault notification). The average size of a datagram is 150 bytes. So, each response/request pair introduces 4 datagrams if everything goes fine, 5 datagrams if one of the events is faulty, or 6 datagrams if both are faulty. We suppose that faults will not occur often, and then few fault notifications will be generated. This assumption is realistic since

all web services are supposed to undergo an acceptable active testing process. The trace collection load then is reduced to the forward of events, that is, 4 datagrams for a request/response pair. This represents a load of 600 bytes.

Executed scenarios. The client application offers, through its graphical interface, the possibility to invoke any operation from those offered by the CWS. For each operation, the client decides between a valid and invalid invocation. This selection is imposed by the FSM-based observers, which are unable to process the parameters of the invoked operation to decide between valid and invalid parameters. For all operations, the web service should return the output "true" if the operation is valid and "false" if the operation is invalid, otherwise a fault occurred.

To illustrate the detection capabilities of our architecture, we injected faults to the web services and or in the network and monitored the behaviour of the observers. Most of the injected faults have been detected by the observers. The global observer was also able to link related notifications that are originated by the same faulty event. From the BPEL document, the global observer builds the list of partners and the order in which they are invoked. Correlation is based on this information and the event sent within the fault notification message.

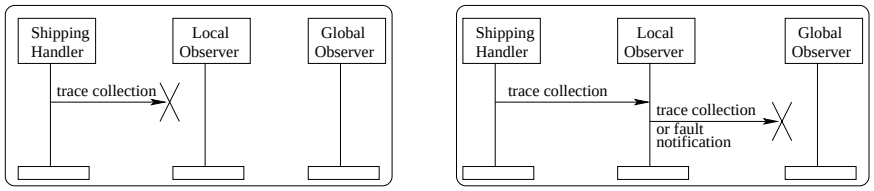
A fault that cannot be detected occurs when the last event in a communication between a web service and its client is lost. As discussed before, traces are sent as UDP packets. To be able to detect lost packets and recover the observation, a sequence number attribute is used. An observer detects a lost packet if the sequence number of the following received packet is different than expected. When a lost packet carries the last event in a communication, observers will not be able to detect this incident since no future packets will arrive. Table 1 shows brief descriptions of some of the executed scenarios and the reactions of observers (both local and global) to the fault.

Target web service	Fault description	Comments
CWS	Submit a printDocument request before creating a conference	Fault detected by local and global observer
Call Control	Add a user before creating a conference	Fault detected by local and global observer
Shipping	A trace collection event (shipDocument response) from a handler to the local observer is lost (Figure 7.a)	Neither the local observer nor the global observer will detect the fault.
Shipping	A trace collection event (shipDocument response) or a fault notification from a local observer to the global observer is lost (Figure 7.b)	The global observer will not be able to detect the fault or process the notification (correlation)

Table 1 Some of the executed scenarios

7 Conclusion and future work

As web services, both basic and composed, are rapidly emerging as a new concept for business-to-business interactions, their management becomes a critical requirement for their success. Management of composed web services is more complex than the management of basic web services. This complexity is implied by the fact that a composed web service aggregates a set of basic web services to provide a different, more complex service. In fact, in addition to the management of the composed web service in its own, management of basic web services must be performed accordingly and all management entities should share management information.



a. trace event does not reach the local observer

b. trace event or fault notification does not reach the Global Observer

Figure 7 Scenarios of non-detected faults

In passive observation, the single observation of a composed web service does not give insights on the behaviors of the basic web services. Many events observed between a final web service and its client can not be studied and explained without information on the exchanged events between the final web service and its basic web services. Thus, observation of all basic web services or at least a subset of these web services is needed.

In this paper, we presented a multi-observer architecture for the observation of composed web services. The architecture proposes to observe the final web service and a set of basic web services. Heuristics to select the basic web services to be observed are also discussed. To reduce the network load generated by the observation, the architecture considers mobile agent observers. We discussed also the network load in terms of mathematical complexity for each type of participation of web services: final web service provider's participation, basic web services providers' participation or hybrid participation

As a proof of concept, we developed a set of basic web services and a composed web service for conferencing management. We also evaluated the network load introduced by the observation and the fault detection capabilities of different observers.

Future work includes the consideration of an Extended Finite State Machine based observers. This is a main issue in web services interactions where data flow is important and fundamental.

References

[1] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
[2] http://www.i3a.org/i_cpxe.html
[3] http://openmobilealliance.org
[4] B. Benatallah, M. Dumas, Q. Z. heng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web services. *In Proc. of ICDE'02, IEEE Computer society, pages 297-308, and Jose, 2002*.
[5] Rachid Hamadi, Boualem Benatallah. A Petri Net-based Model for Web services Composition. *ADC 2003: 191-200.*
[6] S. Narayanan, and McIlraith, S.Simulation, verification and automated composition of web services. *In Proceedings of the World Wide Web Conference,* 2002.
[7] F. Leymann, Web service flow language (WSFL) 1.0. Available online at http://www-4.ibm.com/software/ solutions/webservices/pdf/WSFL.pdf 2001.
[8] A. Ankolekar, M. Burstein, J.R. Hobbs, O. Lassila, D. McDermott, D. Martin, S.A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara, "DAML-S: Web Service Description for the Semantic Web," *Proc. First Int'l Semantic Web Conf. (ISWC 02), 2002.*
[9] BPEL4WS Version 1.1 specification, May 2003 ftp://www6.software.ibm.com/software/ developer/library/ws-bpel.pdf
[10] S. Ghosh and A. Mathur. Issues in testing distributed component-based systems. 1st ICSE Workshop on Testing Distributed Component-Based Systems. May 1999.
[11] www.oracle.com
[12] http://www.activebpel.org
[13] http://www.fivesight.com/pxe.shtml
[14] A. Mani and A. Nagarajan, "Understanding quality of service for web services", January 2002. IBM paper: http://www-106.ibm.com/developerworks/library/ws-quality.html
[15] M.A. Serhani, R.Dssouli, A. Hafid, H. Sahraoui "A QoS broker based architecture for efficient web services selection" IEEE international conference on web services, July 2005, Orlando Florida, USA.
[16] Hongan Chen; Tao Yu; Kwei-Jay Lin, "QCWS: an implementation of QoS-capable multimedia web services", Proceedings of the Fifth International Symposium on multimedia software engineering, 2003.
[17] M.A. Serhani, R.Dssouli, H. Sahraoui, A. Benharef, E. Badidi "QoS Integration in Value Added Web Services" In second international conference on Innovations in Information Technology (IIT05) Dubai, U.A.E, 26-28 September 2005.
[18] Tsai, W.T.; Chen, Y.; Paul, R.; Liao, N.; Huang, H.; "Cooperative and Group Testing in Verification of Dynamic Composite Web Services" Computer Software and Applications Conference, 2004. Proceedings of the 28th Annual International, Volume 2, 2004 Page(s):170 - 173 vol.2
[19] A Benharref, R. Glitho and R. Dssouli, Mobile Agents for Testing Web Services in Next Generation Networks, 2nd International Workshop on Mobility Aware Technologies and Applications, (MATA 2005), Montreal , Canada, October 2005
[20] D. Lee et al. Passive Testing and Applications to Network Management. *Proceedings of IEEE International Conference on Network Protocols*, pages 113-122, October 1997.
[21] http://jade.tilab.com