Timed Extended Invariants for the Passive Testing of Web Services *

Gerardo Morales, Stephane Maag, Ana Cavalli
Télécom Sud-Paris, CNRS/IMR5157
9, rue Charles Fourier 91000, Evry, France
{gerardo.morales, stephane.maag, ana.cavalli}@it-sudparis.eu

Wissam Mallouli, Edgardo Montes de Oca, Bachar Wehbi
Montimage EURL
39, rue Bobillot 75013, Paris, France
{wissam.mallouli, edgardo.montesdeoca,bachar.wehbi}@montimage.com

Abstract

The service-oriented approach is becoming more and more popular to integrate highly heterogeneous systems. Web services are the natural evolution of conventional middleware technologies to support Web-based and enterprise-level integration. Formal testing of such Web-based technology is a key point to guarantee its reliability. In this paper, we choose a non-intrusive approach based on monitoring to propose a conformance passive testing methodology to check that a composed Web service respects its functional requirements. This methodology is based on a set of formal invariants representing properties to be tested including data and time constraints. Passive testing of an industrial system (that uses a composition of Web services) is briefly presented to demonstrate the effectiveness of the proposed approach.

1 Introduction

A Web Service is defined by the World Wide Web Consortium W3C as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [8]). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

*The research leading to these results has received funding in part from the French ANR Webmov Project (http://webmov.lri.fr/) and the European Community's Seventh Framework Program (FP 7/2007-203) under grant agreement n^0 215995 (http://www.shields-project.eu/)

The composition of Web services consists of combining existing Web services to define higher-level functionalities. The Business Process Execution Language (BPEL) [4] is emerging as the standard composition language for specifying business process behavior based on Web services. A BPEL process implements one Web service by specifying its interactions with other Web services (called partners).

As the Web service implementations grow in size and complexity, the necessity for testing their reliability and level of security is becoming more and more crucial. The Web Services technology, which is based on the SOA (Service Oriented Architecture), is hard to test because it relies on distributed systems that complicate the runtime behavior.

In this distributed context, formal active testing [9] can be rather difficult to achieve since it requires to control remote deployed Web systems. Indeed, active testing usually relies on the comparison between the behavior of an implementation and its formal specification by checking whether they are equivalent. Test sequences are commonly automatically or semi-automatically generated from formal models that represent test criteria, hypothesis and test objectives. These sequences (with an executable format) are performed by designing a testing architecture and establishing Points of Control and Observation (PCO, execution interfaces) that may disturb the natural behavior of already deployed Web services.

To avoid this problem, there is an interest in applying the passive testing approach for the conformance checking of Web services. Passive testing consists in observing the exchange of messages (inputs and outputs) of a Web service implementation under test (IUT). The term *passive* means that the tests do not disturb the natural operation of the IUT. Instead, the exchanged messages will be recorded in a trace that will be inspected later on against the properties derived from the Web service by system experts.

In the work we present here, we extend the concept of invariants already defined in [1] to take data and time constraints into consideration. Then we define a methodology with a new algorithm for formal passive testing based on these extended invariants. The main contributions of this paper are the following:

- The definition of timed extended invariants that allow to formally specify functional properties that the Web service has to respect. These properties are related to the exchanged packets between Web services and may deal with the data portion of these packets and/or with time constraints.
- The proposition of conformance testing approach for Web services using passive testing approach. This approach is particularly well adapted when it is difficult to control remote partners and permits to avoid disturbing the natural running of already deployed Web services.
- Evaluation of the proposed methodology and tool TIPS (Testing Invariants for Protocols and Services) on the Travel Reservation System composed Web service. This tool performs a dynamic packet inspection to analyze the execution traces to check the formal extended invariants.

The rest of the paper is organized as follows. Section 3 presents the formal concept of timed extended invariants. In section 4, we present our passive testing methodology as well as its related tool and algorithm. Section 5 presents our experiment on a Travel Reservation System that uses a Web service composition orchestrated by a BPEL engine. We apply on it our passive testing tool and analyze the obtained results. Section 6 concludes this paper and presents some future perspectives.

2 Related work

Many recent research work are devoted to model and test machine-to-machine communications through Web services. Besides, big efforts are currently dedicated to provide standards for Web services interoperability testing [7], while very few research tackle their passive conformance testing applying formal methods. We mention some of them in the following.

[2] proposes to extend the architecture used to deploy Web services (SOA) with an "observer element". The authors use the model FSM+ (FSM that can take into account some timed constraints) that are applied as the reference behavior to set verdicts over the tests. While timing aspects are studied in this work, data portions and SOA architecture used to deploy the Web services are not processed.

[5] presents the TGSE tool, based on the methodology presented in [12]. TGSE allows the runtime coverage of transitions based on guard conditions (by examining variable values). In this approach, the authors model an execution trace as a test purpose (i.e. also referred as a TEFSM) by giving real values to each input message. After modeling the composed Web service specification and the trace as TEFSMs and declaring their synchronization rules, they use TGSE to verify this system. If the TGSE output is a sequence, it means that this trace is a valid trace of the interactions among the systems. However, this method requires a formal specification of the Web service composition which is often unavailable (or hard to provide) in such Web systems.

The methodologies and tools that are presented in our work to perform passive testing are based on a formal invariant approach [1] applied to the SOA architecture of Web service deployment.

3 Timed extended invariants

3.1 Preliminaries

Invariants are properties that the implementation under test is expected to satisfy. They are used to express constraints over exchanged messages between the system entities (in this case, distributed Web services). Basically, they allow expressing that the occurrence of an event must be necessarily preceded or followed by a sequence of events. In [1], a first introduction to the invariants was provided. These invariants are described according to a finite state machine (FSM) based specification of the system under test.

This model is not well adapted to express complex functional behavior of Web services that may involve data exchange with time constraints. To solve this problem, we propose to extend the formalization of invariants by adding these two aspects (data and time), and rely thus on a specification of the IUT described using a Timed Extended Finite State Machine (TEFSM) model ([3]).

Notice that only the system inputs and outputs can be observable in the collected system traces. These actions represent the events that we take into account in the timed extended invariants. Based on the definition of a TEFSM, we enhance the formalization of the messages collected within a trace and the events described within the invariants.

In the following definitions a captured packet is a network packet exchanged between the BPEL engine (the composed Web service orchestrator) and its Web partners. This packet is captured by a sniffer that adds some extra data (for example the timestamp of when it was captured), i.e. captured packet = network packet + extra data.

Definition 1 *(Collected trace) A collected trace is a set of ordered captured packets.*

- *A trace $T = \bigcup_{i=1}^{n} p_i$ where n is the number of captured packets, p_1 is the first captured packet in the trace and p_n the last one.*
- *Each packet p_i has a rank r_i that corresponds to its position in the trace T.*
- *$\forall p_i \in T$, $p_i = \bigcup_{j=1}^{m_i} f_{i,j}$ where $f_{i,j}$ is a field of the packet p_i and m_i is the number of fields of the packet p_i. Each field $f_{i,j}$ of the packet p_i has a value $v_{i,j}$.*
- *$\forall p_i \in T, \exists f_{i,j} \in p_i$ / $f_{i,j} = t_i$ where t_i is the timestamp when p_i was captured.*
- *$\forall r_i, r_j$ where r_i is rank of p_i and r_j is rank of p_j, if $r_i > r_j$ then $t_i > t_j$*

Definition 2 *(Value function ϕ) Let T be a collected trace of n packets, F the set of fields of all the packets p_i of the trace T, V the domain of values and P the set of packets. $V = \mathbb{R} \cup S \cup NULL$ where S is a finite set of strings. We define the function: ϕ: $P \times F \rightarrow V$ as the function allowing to provide the value of a field in a specific packet of the trace T:*

- *$\phi(p_i, f_{m,n}) = v_{i,n}$ if $f_{m,n} \in p_i$ and*
- *$\phi(p_i, f_{m,n}) = NULL$ if $f_{m,n} \notin p_i$*

3.2 Definition of timed extended invariants

An invariant is an If-Then property. It allows expressing the desired behavior of a communicating system and describes the correct order of messages collected in the trace with potential time constraints. If specific conditions on the exchanged messages hold, then the occurrence of a set of events must happen. An event is a set of conditions on some field values of exchanged packets.

Definition 3 *(Conditions) Conditions are predicates on packets' fields values. Let p_i and $p_{i\prime}$ be two captured packets, V be the domain of values, $f_{i,j}$ be a field of the packet p_i, $f_{i\prime,j\prime}$ be a field of $p_{i\prime}$ and $x \in V$. Two types of conditions can be defined: c_s (simple condition) and c_c (complex condition)*

$$c_s ::= \phi(p_i, f_{i,j}) \text{ op } x$$
We say that the packet p_i satisfies c_s iff $v_{i,j}$ op x is true.

$$c_c ::= \phi(p_i, f_{i,j}) \text{ op } \phi(p_{i\prime}, f_{i\prime,j\prime})$$
We say that packet p_i satisfies c_c iff $v_{i,j}$ op $v_{i\prime,j\prime}$ is true.

op is an element of $O_T = O_{\mathbb{R}} \cup O_S$ where $O_{\mathbb{R}}$={$\leq$, $\geq$, =, $\neq$, $\in$} and O_S={contain, contain not}. The operator must respect these rules:

- *If $v_{i,j} \in \mathbb{R}$ then op $\in O_{\mathbb{R}}$.*
- *If $v_{i,j} \in S$ then op $\in O_T$ and:*
  - *the operators that are in O_S are used when a field's value has a string type.*

Notice that $v_{i,j}$, $v_{i\prime,j\prime}$ and x must be part of the same domain.

Definition 4 *(Basic event) A timed extended event e_j is a set of conditions on relevant fields of captured packets. $e_j = \bigcup_{k=1}^{m_j} c_{j,k}$, m_j being the number of conditions.*

Definition 5 *(Event satisfaction) Let p_i be a packet and e_j an event with m_j conditions and $c_{j,k}$ the k^{th} condition of e_j*

A packet p_i satisfies an event e_j iff
$\forall\, k \in [1, m_j]$, $c_{j,k}$ is true

Definition 6 *(Abstention of having an event)*
If e is an event, then $\neg e$ is also an event. $\neg e$ is satisfied if no packet that satisfies the event e occurs in the collected trace.

Definition 7 *(Successive Events) Let $n \in \mathbb{N} \cup \{-1\}$, $t \in \mathbb{R}^{+*} \cup \{-1\}$ and e_1 and e_2 be two basic events. $(e_1; e_2)_{n,t}$ is a complex event denoted also by Follows((e_1, e_2),n,t). It is composed of two basic events.*
$[p_1, p_2]$ satisfies $(e_1; e_2)_{n,t}$ $\Leftrightarrow$

- *p_1 satisfies e_1 and*
- *p_2 satisfies e_2 and*
- *time(p_1) < time(p_2) < time(p_1)+t if ($t \neq -1$) and*
- *rank(p_1) < rank(p_2) < rank(p_1)+n if ($n \neq -1$).*

In other words, $[p_1, p_2]$ satisfies $(e_1; e_2)_{n,t}$ iff p_2 follows p_1 and they are separated by at most n packets and t units of time.

Definition 8 *(complex events: AND) Let $n \in \mathbb{N} \cup \{-1\}$, $t \in \mathbb{R}^{+*} \cup \{-1\}$ and e_1 and e_2 two basic events. $(e_1 \wedge e_2)_{n,t}$ is a complex event denoted also AND($(e_1$,$e_2)$,n,t). It is composed of two basic events.*
$[p_1, p_2]$ satisfies $(e_1 \wedge e_2)_{n,t}$ $\Leftrightarrow$
$[p_1, p_2]$ satisfies $(e_1; e_2)_{n,t}$ or (p_1, p_2) satisfies $(e_2; e_1)_{n,t}$

Intuitively, p_1 and p_2 satisfy $(e_1 \wedge e_2)_{n,t}$ iff p_2 and p_1 are separated by at most n packets and t units of time.

Notice that in this definition the value of n or t can be -1, this is to "disable" the constraint on this variable (n or t). This is highlighted later in the definition 10

Definition 9 *(complex events: OR) Let* $n \in \mathbb{N} \cup \{-1\}$, $t \in \mathbb{R}^{+*} \cup \{-1\}$ *and* e_1 *and* e_2 *two basic events.* $(e_1 \vee e_2)_{n,t}$ *is a complex event denoted also OR((*e_1*,*e_2*),n,t), It is composed of two basic events.*
p_1 *satisfies* $(e_1 \vee e_2)_{n,t} \Leftrightarrow p_1$ *satisfies* e_1 *or* p_1 *satisfies* e_2

Definition 10 *(Timed Extended Invariant) Let* $''When'' \in \{BEFORE, AFTER\}$, $n \in \mathbb{N} \cup \{-1\}$, $t \in \mathbb{R}^{+*} \cup \{-1\}$ *and* e_1 *and* e_2 *two events (basic or not). A timed extended invariant is an IF-THEN expression that allows expressing a property regarding the messages exchanged in a captured trace* $P = \{p_1, ..., p_{m'}\}$*. It has the following syntax:*

$$e_1 \xrightarrow{When,n,t} e_2$$

This property expresses that if the event e_1 *is satisfied (by one or several packets* p_i*,* $i \in \{1, ..., m\}$*), then event* e_2 *must be satisfied (by another set of packets* p_j*,* $j \in \{1, ..., m\}$*) before or after (depending on the* $When$ *value) at most* n *packets (if* $n \neq -1$*) and* t *units of time (if* $t \neq -1$*).*

It is important to highlight that in the invariant it is possible to disable n and/or t by setting their values to -1.

If ($When = AFTER$), we say that the invariant is a *simple invariant*. If ($When = BEFORE$), we have an *obligation invariant*.

Example:

According to this last definition, invariants can express complex properties including data and time constraints. For instance in a HttpPost restful web service, we can express that if an OK message (the code 200) is received, then a POST request must have been sent before. This is described by the following obligation invariant:

$$e_1 = \{\phi(p_1, http.response.code) = 200\ \}$$
$$\xrightarrow{BEFORE,-1,6}$$
$$e_2 = \{\phi(p_2), http.request.method) = POST,$$
$$\phi(p_2, tcp.srcport) = \phi(p_1, tcp.dtsport),$$
$$\phi(p_2, tcp.dtsport) = \phi(p_1, srcport)\}$$

Notice that in this invariant, we do not take into account the number of packets that are between the OK and the POST messages, on the other hand, we specify that the delay of time between the messages must be at most 6 seconds which is the timeout value for this service.

4 Conformance passive testing approach for Web services

4.1 Methodology overview

Conformance testing in the passive testing approach aims to test the correctness of a Web service implementation through a set of properties (invariants) and monitored traces (extracted from the running Web service through probes called also Points of Observation (PO)). The conformance passive testing procedure follows these four steps:

- Step 1: Properties formulation. The relevant Web service properties to be tested are provided by the standards or by the Web service experts. These properties express the main requirements related to the communication between the Web partners in the context of Web services.
- Step 2: Properties as invariants. Properties have to be formulated by means of invariants that express local properties; i.e. related to a local entity. Moreover, the properties could be formally verified on the formal system specification (as a timed automata) if it is available, ensuring that they are correct vis-à-vis the requirements.
- Step 3: Extraction of execution traces. In order to obtain such traces, different PO are to be set up by means of a network sniffer. The captured traces can be stored in an XML format.
- Step 4: Test of the invariants on the traces. The traces are processed in order to obtain information concerning particular events as well as relevant data (e.g. source and destination address, origin of data to initialize a variable, etc.). During this process, the test of the expected properties is performed and a verdict is emitted (Pass, Fail or Inconclusive). An inconclusive verdict may be obtained if the trace does not contain enough information to allow a Pass or Fail verdict.

4.2 Passive testing tool

Based on the methodology of our work, we developed the TIPS tool [6] that performs automated analysis of the captured traces to determine if the given timed extended invariants are satisfied or not. The TIPS tool aims to passively test a deployed communicating system under test to verify if it respects a set of properties. In the case of TIPS, invariants describe the correct order of exchanged messages among system entities with conditions on communicated data and time. Passive testing consists of observing input and output events of the system implementation in run-time and detecting potential misbehavior or errors.

Figure 1 illustrates the modules of the TIPS tool. It has three different inputs:

1) SUT Information on the Web service under test that is being observed. This information represents data of interest (for example protocol packets field names) that are relevant to the automated analysis of the captured traces. 2) The invariants defined in XML format. The non-respect of an invariant may imply an error in the Web services.3) The

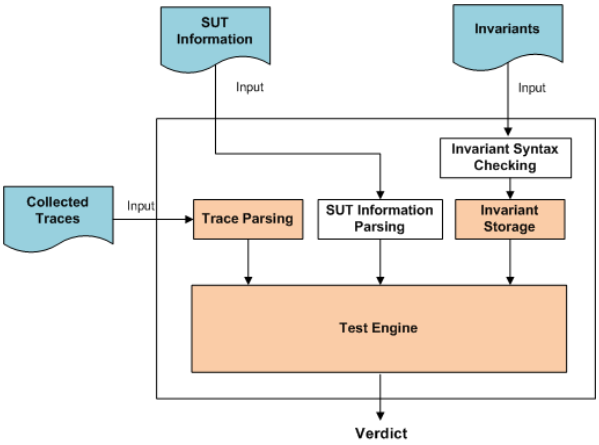

Figure 1. TIPS architecture for Web service checking

collected traces represented in XML format (captured using Wireshark[1] for instance).

In order to use the TIPS tool, the first step consists in defining the invariants and the data of interest. This can be done by an expert of the Web service under test. The invariants are then verified with respect to their XML schema.

The next step consists in capturing the communication traces using a sniffer (we used *Wireshark*) and analyzing them using the TIPS tool.

In the case of Web services, what we analyze are the SOAP messages. A SOAP request is an XML-based Remote Procedure Call (RPC) sent using the HTTP transport protocol. The payload of the SOAP message is an XML document that specifies the call being made and the parameters being passed.

Within the body of a SOAP message, XML namespaces are used to qualify element and attribute names within the parts of the document. Element names can be global (referenced throughout the SOAP message) or local. The local element names are provided by namespaces and are used in the particular part of the message where they are located. Thus, SOAP messages use namespaces to qualify element names in the separate parts of a message. Namespaces also identify the SOAP envelope version and encoding style.

4.3 Invariants storage

Within an invariant, the <*if*> tag identifies the triggering events. We call this part of the invariant the *Trigger Context*. The events that need to be verified on the trace are found in the <*then*> tag. We call this part of the invariant the *Verdict Clause*.

[1]http://www.wireshark.org/

The invariants are written in XML format and are represented in an *If-Then* structure. Basically, if in the trace we find the events of the *Trigger Context*, then we need to verify that the events of the *Verdict Clause* are part of the trace as well.

The invariant in Figure 2 expresses that if a system receives a reply message, then this means that it wa sent a request message 10 seconds before. In this invariant we have the *Trigger Context* from lines 5 to 15, this means that if in the collected trace, we find a message respecting the event described in the <*if*> context (lines 6 to 14), the process of verifying the invariant over the trace is triggered. The *Verdict Clause* (lines 18 to 33) contains the events that must be satisfied in the tracee once the verification process is triggered.

```
1.  ...
2.
3.  <invariant name="INVARIANT 1">
4.  <!- Trigger Context start here ->
5.  <if>
6.  <event reference= “true” id= “001”>
7.  <condition>
8.  <variable>Type</variable>
9.  <operation>=</operation>
10.  <value>REPLY</value>
11.  </condition>
12.
13.  ...
14.  </event>
15.  </if>
16.  <!- Trigger Context end here ->
17.  <!- Verdict Clause start here ->
18.  <then type="BEFORE" max_skip="-1"
 max_time="10">
19.  <event id= “032”>
20.  <condition>
21.  <variable>Type</variable>
22.  <operation>=</operation>
23.  <variable>REQ</value>
24.  </condition>
25.  <condition>
26.  <variable>Source</variable>
27.  <operation>=</operation>
28.  <variable type=="e001">Destination</value>
29.  </condition>
30.
31.  ...
32.  </event>
33.  </then>
34.  <!- Verdict Clause end here ->
35.  </invariant>
36.
37.  ...
```

Figure 2. XML format for defining an invariant

4.4 Passive testing algorithm

This section presents the algorithm that allows the deduction of a verdict by analyzing the system trace with respect to a set of predefined obligation invariants. The obtained verdict for an invariant can be either: Pass, Fail or Inconclusive meaning respectively that all events were satisfied, that at least one event was not satisfied or that it is

not possible to give a verdict due to the lack of information in the trace.

The algorithm used by TIPS analyzes the traces in, at most, time complexity of $O(N^2)$. This implies that the the time required to analyze a trace is in the order of N^2 x I x T where N = the number of packets in the trace, I = the number of invariants and T = the average time spent in analyzing an invariant on a packet. This can be reduced to $Nlog(N)$ x I x T if we store information of each condition for each packet in a hash table. This will be done for the future version of the tool that will thus be able to efficiently perform on-line invariant analysis.

The behavior of the obligation algorithm is illustrated (as an abstraction) in the flowchart of Figure 3 where we consider the trace $T = \{p_1 ... p_k\}$. The pointers Current packet (CrPkt) and Reference packet (RefPkt) are used to traverse T using the function *stepback* (see Def 11). Note that since the obligation invariant uses the logic "if a happens then b should have happened before a", the trace is covered backwards starting from p_k and finishing in p_1.

Definition 11 *(Stepback function) Using the elements of Definition 1, let r_i and r_h be the ranks of the packets p_i and p_h respectively in the trace T. We define the stepback function: $\sigma(p_i) = p_h$ where $r_h = r_i - 1$*

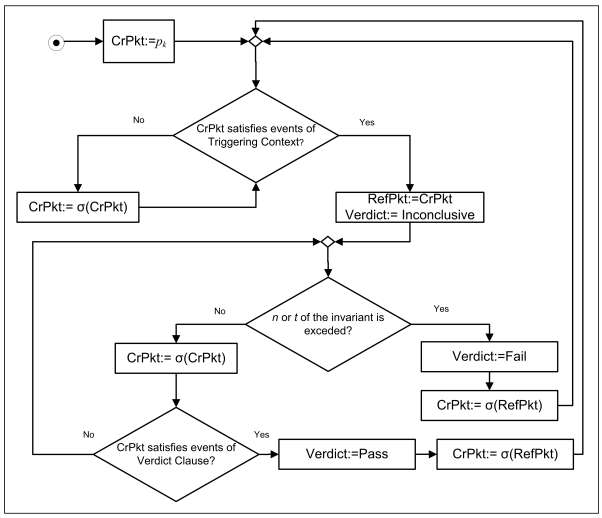

Figure 3. Obligation algorithm flowchart

The algorithm traverses the trace backwards until the triggering context events of the invariant are satisfied by a packet. In this case, TIPS continues traversing (backwards) the trace to verify the satisfaction of the verdict clause of the invariant. If this is the case, TIPS issues a Pass verdict for this invariant otherwise, a Fail or Inconclusive verdict is issued.

The algorithm that allows checking simple invariants on a captured trace is a variant of this algorithm. For simple invariants, we begin the parsing process from the beginning of the trace and move forward until the end of the trace.

5 Case Study

5.1 Travel Reservation Service description

The Travel Reservation Service (TRS) is an example of real-life service for travel organization provided within the Netbeans IDE platform [10]. It acts as a logical aggregator of three other Web services: Airline Reservation Service (ARS), Hotel Reservation Service (HRS) and Vehicle Reservation Service (VRS).

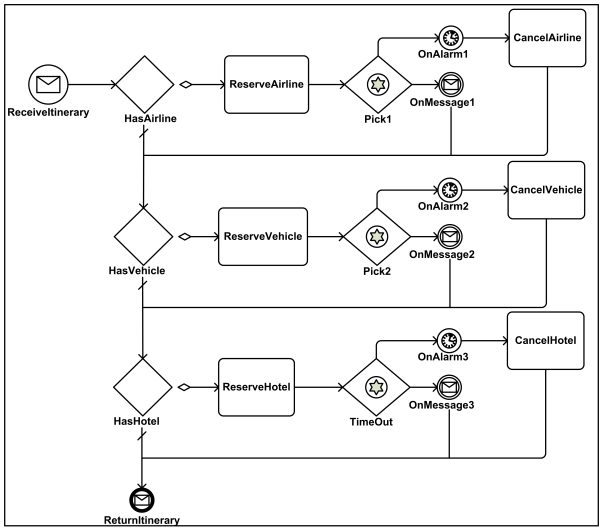

Figure 4. Behavior of the TRS described in BPMN

The process assumes that an External Partner initiates the process by sending a message that contains a partial travel itinerary document. The client's travel itinerary may have: no pre-existing reservations, or a combination of pre-existing airline, vehicle and/or hotel reservations.

The TRS examines the incoming client itinerary and processes it for completion. If the client itinerary does not contain a pre-existing airline reservation, the TRS passes the itinerary to the ARS in order to make an airline reservation. The ARS passes back the modified itinerary to the TRS. The TRS conducts similar logic for both vehicle and hotel reservations. In each case it will delegate the actual provisioning of the reservation to the VRS and HRS. Finally, the TRS passes the completed itinerary back to the original client, completing the process.

A function of TRS called *Build Itinerary* allows the External Partner to an itinerary for his traveling needs. This

partner initiates the process by sending a message that contains a partial travel itinerary document. The Travel system receives the itinerary request and contacts its service partners (ARS, VRS and HRS) if needed. After the reservations are done, the system sends back the completed itinerary to the External Partner.

The behavior of the TRS and the communications with the partner Web services are described using the Business Process Modeling Notation (BPMN) [11] in Figure 4.

5.2 Test Objectives

Starting from the TRS requirements, a set of 12 test objectives (TO) has been defined. These objectives are related to the order of the Web partners' interactions or to data or time constraints. Here we present a selection of 2 test objectives:

- TO1. A partial itinerary is sent to the TRS. It does not contain any previous reservation (no airline, no vehicle and no hotel). The TRS system contacts the ARS partner, then the VRS partner, then the HRS partner, before sending back the complete itinerary.
- TO11. The TRS does not receive any response from the HRS within 20 seconds, a request for canceling the reservation is sent to this partner. No hotel item is included in the completed reservation.

Taking as an example the TO11, it can be represented as a timed extended invariant format presented in the Section 3 as follows:

$$((e_0; \neg e_1)_{-1,20}) \xrightarrow{After,-1,1} e_2$$

- e_0: the TRS sends a reservation request to the HRS. For the sake of simplicity we will write TRS-IP and HRS-IP instead of the real ip of the machine where the TRS and HRS are running. Therefore e_0= {ϕ(p_0, ip.src)= TRS-IP, ϕ(p_0, ip.dst)= HRS-IP, ϕ(p_0, data) $contain$ "<itinerary>" }
- e_1: the answer to the request is described by e_0. Therefore e_1= {ϕ(p_1, ip.src)= HRS-IP, ϕ(p_1, ip.dst)= TRS-IP, ϕ(p_1, data) $contain$ "<hotel>"}.
- e_2:TRS sends the cancelation of the reservation to the HRS. Therefore e_2= {ϕ(p_2, ip.src)= TRS-IP, ϕ(p_2, ip.dst)= HRS-IP, ϕ(p_2, data) $contain$ "<cancel>"}

5.3 Trace capture

The traces of the composed Web service have been captured through the PO installed on the TRS server to capture the communications between the BPEL engine considered as a black box and its external Web service partners (HRS, VRS and ARS). Wireshark was used to capture and save the traces in XML format. The main advantage of installing the PO on the server runnin the orchestration (TRS) is that it allows observing all exchanges between the BPEL engine and its Web service partners.

5.4 Results and analysis

Applying TIPS to the traces was fast (less than 1 sec for a trace of 1000 packets). The obtained verdicts were Pass for some invariants and Inconclusive for the others. Some of the invariants were never tested since they were not found in the collected traces. Indeed, the collection of a trace from a running system should be long enough to cover all the expected Web service scenarios. Otherwise, a simulation of these scenarios must be performed. For example, to be able to test the 11th test objective, the hotel reservation service must not answer the itinerary request (or answer it 20 seconds after it receives the request). This kind of situation is not very frequent in a real system deployment and to test it (i.e test the invariant describing this test), we would need to wait for a sufficiently long time or we can simulate the non answer by shutting down the hotel Web service. The complexity of TIPS ($O(N^2)$) is illustrated in Figure 5 as a function depending on the length of the trace.

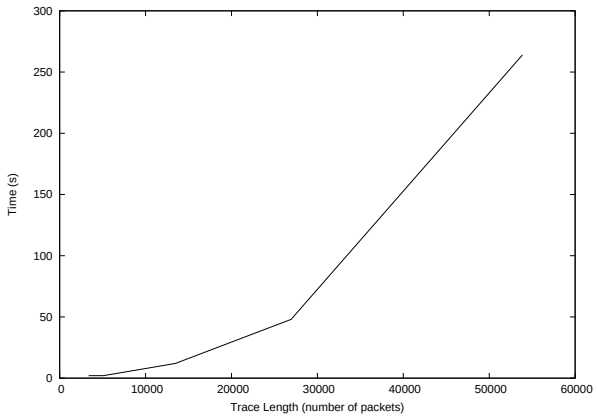

Figure 5. Processing time as a function of the length of the trace

During this experiment we finally tested the invariants at least once. The obtained verdicts were all PASSED which demonstrate the correctness of the Web service.

To prove the efficiency of our TIPS passive testing tool, we manually edited the file containing traces of Travel Reservation Service. We have, for instance, deleted a packet that violates the invariant number 1. The verdict given then by the tester was FAIL: the violation of the invariant has

been detected and the tool provided the packet that violates it which indicates the correctness of TIPS.

6 Conclusion

This paper proposed a new passive conformance test approach for Web services. Besides, it illustrates that passive testing is relevant when Web services are already deployed or provided as back box implementations.

We have also provided a new approach for testing the collaboration between different systems, focusing in particular on the communication between SOA partners. We have proposed and evaluated an extensible and more flexible approach were properties called timed extended invariants are checked on a collected trace.

Our approach is flexible enough to detect new errors, such as functional errors and security flaws. The invariants can be updated to include new elements, for instance, feedback resulting from experimentations, and can also be combined with active testing to deduce a verdict after a Web service stimulation.

Finally the methodologies and tools were applied to a real world case study to demonstrate the effectiveness of this approach.

Currently, TIPS uses a collected trace to perform the testing, however, we are working towards improving this tool to be able to analyze packets in real time in order to detect functional/security violations on the fly.

References

[1] E. Bayse, A. Cavalli, M. Nunez, and F. Zaïdi. A passive Testing Approach Based on Invariants: Application to the WAP. *Comput. Netw. ISDN Syst.*, 48(2):247–266, 2005.

[2] A. Benharref, R. Dssouli, M. A. Serhani, and R. Glitho. Efficient traces' collection mechanisms for passive testing of web services. *Inf. Softw. Technol.*, 51(2):362–374, 2009.

[3] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 237–267. Springer, 2004.

[4] Business Process Execution Language V2.0, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[5] T.-D. Cao, P. Félix, R. Castanet, and I. Berrada. Testing web services composition using the tgse tool. *IEEE Congress on Services*, 0:187–194, 2009.

[6] A. R. Cavalli, E. Montes De Oca, W. Mallouli, and M. Lallali. Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints. In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 315–318, Washington, DC, USA, 2008. IEEE Computer Society.

[7] W.-I. Consortium. http://www.ws-i.org/. 2009.

[8] G. M. E. Christensen, F. Curbera and S. Weerawarana, 15 March, 2001. Web Services Description Language (WSDL) 1.1.http://www.w3.org/TR/wsdl.

[9] M. Lallali, F. Zaidi, A. Cavalli, and I. Hwang. Automatic Timed Test Case Generation for Web Services Composition. In *ECOWS '08: Proceedings of the 2008 Sixth European Conference on Web Services*, pages 53–62, Washington, DC, USA, 2008. IEEE Computer Society.

[10] NetBeans Framework. http://www.netbeans.org/. 2009.

[11] OASIS. http://www.bpmn.org/. 2009.

[12] Y. Yan and P. Dague. Monitoring and Diagnosing Orchestrated Web Service Processes. In *Proceedings of the 2007 IEEE International Conference on Web Services, Salt Lake City, Utah, USA*, 2007.