Testing of Web services: Tools and Experiences

Dung Cao, Richard Castanet, Patrick Félix, Gerardo Morales

Testing of Web services: Tools and Experiments

Tien-Dung Cao
Department of Computer Science,
School of Engineering - Tan Tao University,
Tan Duc E-city, Duc Hoa District,
Long An, Vietnam
Email: dung.cao@ttu.edu.vn

Richard Castanet and Patrick Felix
LaBRI - CNRS - UMR 5800
University of Bordeaux
351 cours de la libération,
33405 Talence cedex, France.
Email: {castanet,felix}@labri.fr

Gerardo Morales
Montimage
39 rue Bobillot,
75013 Paris, France.
Email: gerardo.morales@montimage.com

***Abstract*—Web services are the element based in SOA (i.e., Service Oriented Architecture) applications that are commonly used in software systems. However the validation problem on quality of web services via testing has to be improved. This paper presents two tools for conformance testing of web services. One tool for unit testing that is implemented by an on-line approach. This tool can be used to test a web service-based and/or an orchestration by simulating its partners. The other focuses on verification of a timed trace with respect to a set of constraints. Specially, we can use this tool to verify on-line or off-line a timed trace. Finally, we show our experiments on a real-life case study, Product Retriever, by combining the two tools.**

***Keywords*-Web services, Conformance testing, Active/Passive testing.**

I. Introduction

Currently, web services are having an increasingly important role in the construction of computer applications used remotely by many users. It is not possible for a Web service provider, to make available a new Web service or to make a radical change from an old Web service without first checking this service. As the software system, a web service (WS) can be tested by traditional software testing techniques such as: conformance testing, performance testing, availability testing, robustness testing, etc. Web services are applications that allow the formation of SOA (Service-Oriented Architectures) applications which are built by composing other web services. There are two types of composition:

- **Orchestration**: Describes the internal logic of a single component by specifying the control flow and data flow dependencies of this element. In an orchestration, there is only one main process (named conductor) that controls its partner services using a central architecture.
- **Choreography**: Describes a collaboration of services in a composite service. In this context, there is no main process and all participants involved in the composition and interact with each other.

Two conventional approaches may be used to test a web service:

- **Assets Test**: In a web service composition, a service with a poor quality may affect other partner agencies and their composition. We will apply a technique for testing actively to find the error of each service in isolation without the interaction of its partners, however only by using simulation of the partners. It is called unit testing. We focus here on conformance testing of web service composition. In addition, time constraints must be taken into account in our approach because Web services are real time applications.
- **Reliability**: This is an approach that is applied to the running services to verify specific properties. Find all the faults of a service by testing is impossible. However monitoring after the publication of the service is a practice that may detect malfunctions. The passive test is a technical monitoring of certain properties of the services: The execution traces are collected and analyzed according the properties to be verified and a verdict (true / false) is given.

Some other research works have been published about Web Services testing and experimental tools have been developed. However, these works often do not take in account time constraints or bringing together active and passive testing. Dranidis et al [1], present a monitoring of the execution but does not present a tool to verify a trace of execution of the web services. Baresi et al [2] use a behavioral characteristics composition expressed in BPEL. Bartolini et al [3] present a methodology based on WSDL and a tool for active testing.

This paper is organized as follows: we present, in section II, the active testing with the WSOTF tool and in section III the passive testing method with the RV4WS tool. Section IV shows how the testing techniques are applied on a real example. Finally, the last part of this paper presents conclusions and perspectives in Section V.

II. Active testing

A. Online testing

The test activities are test case generation, test execution and verdict assignment. The problem of the test case generation approach is the explosion of state because we must

pass all states of formal model while generating a set of test cases. On-line testing [4], [5] is an approach in which test case generation and test execution are run in parallel. This approach solves the problem of off-line approach by randomly selecting the test case. However this approach does not guarantee that all test cases are found. From an initial state, only a single primitive test (input event) is generated from the model at a time which is then immediately executed on the system under test (SUT). The output produced by the SUT, as well as its time of occurrence, is checked against the specification. At a given time, a new primitive test is produced based on the values of previous events or random selection. If there is some acceptable options, and so on forth until we arrive at a final state. Figure 1 shows an illustration of this approach. We used this approach for unit testing of a web service [6] based on a formal model, named TEFSM (Timed Extended Finite State Machine) that is defined by Lallali et al [7] to model a web service composition.

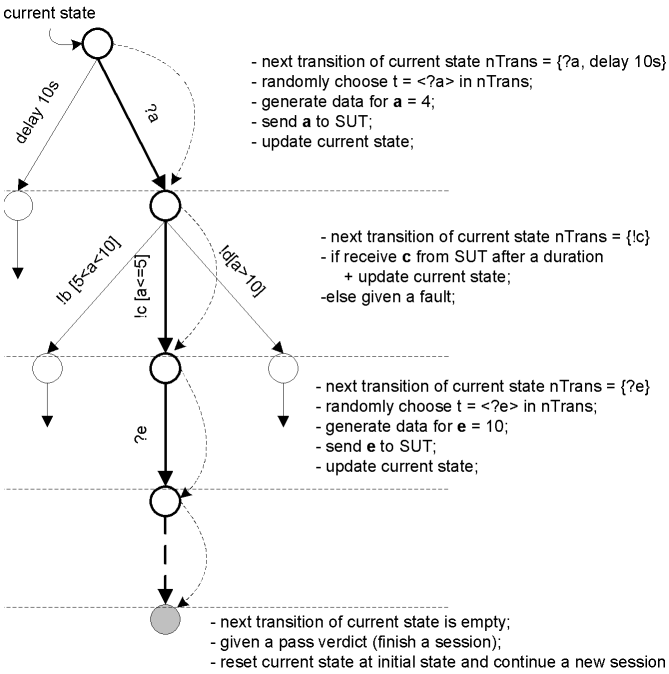


Figure 1. Online testing approach illustration

B. WSOTF Tool

WSOTF (Web Service, On-line Testing Framework) is developed based on on-line algorithm [6]. We can use this tool to test a web service based (WSDL specification) or a web service orchestration. It is implemented by Java and its detailed architecture is shown in Figure 2, which consists of two parts: the controller and the adapter. The controller is composed of five main components: a loader, an interface to process data, a data function library, an interface to send/receive the messages to/from SUT and an executer.

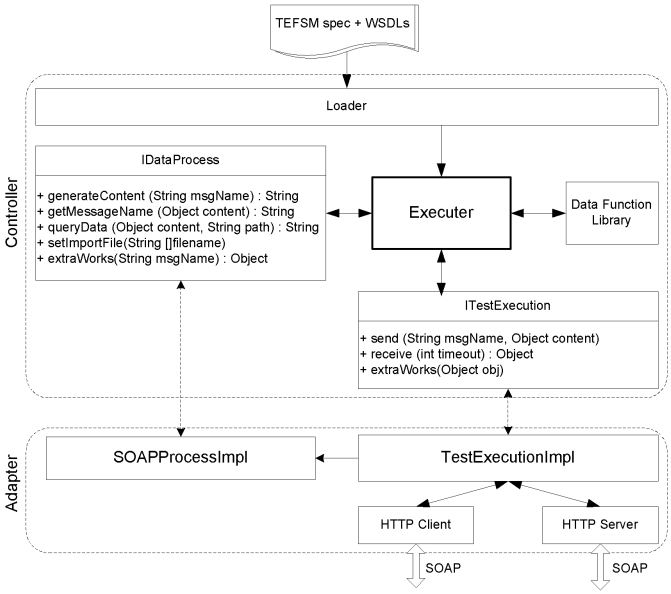


Figure 2. Architecture of the WSOTF engine

1) **Loader:** loads the input format and analyses it;
2) **DataProcess Interface:** used to generate the data content, get the name of the message and query the value of the message. To implement this interface, we reuse the code of SoapUI [8] to generate a SOAP format. The data for each field of a SOAP message is randomly generated or a default value is used. This depends on the configuration file;
3) **Data function library:** defines a list of functions that are used to update the variables or evaluate a boolean expression. In the current version, only variables with data types int, boolean or string are supported to control the internal behavior;
4) **Executer:** implements on-line testing algorithm to generate a test case, controls test execution and assigns the verdict. It uses the data process interface and the data function library to generate SOAP message. It updates the value of variables and evaluates the constraints;
5) **Test Execution Interface:** used to send and receive messages to/from SUT. To implement it, we used a HTTP client to invoke the request into SUT (the client request or partner callback in the case of asynchronous services, the result is returned on a different port) and HTTP server to receive and return the message from SUT on the same port. When test execution receives a SOAP message from SUT, it puts this message in a queue to wait for a processing of the executer. It receives directly SOAP message from the executer and sends it to SUT.

The Adapter plays the role of Tester (i.e., sends/receives request and response) and also simulates all partners in the

case we test a web service orchestration. Its input format is a XML file which represents a TEFSM specification. This format consists of a partner section that declares the partners name and location of WSDL specification, variables list (variable types are: int, $boolean$ or message type of SOAP that is defined in WSDL), local clock, initial state, and a list of transitions. Each transition consists of seven fields: source state, target state, event name, guard on variable, guard on clock, data update function and local clock to be reset. In WSOTF, we use the form *?pl.pt.op.msg* to represent a format of an input action that means the reception of the message (*msg*) for the operator (*op*) of the portTyte *pt* from the partner (*pl*) and the form *!pl.pt.op.msg* represents a format of an output action (resp. the emission of the message (*msg*) for the operator (*op*) of the portTyte *pt* to the partner (*pl*)). The result of WSOTF (traces including interval time between two actions and its correspondent verdict) is saved in a xml file. This tool enables the declaration of the value of some fields of SOAP message in an enumeration file that can be used to test by purpose (fix the condition for each branch), correlation data (current version does not support the correlation data functions) or to debug. At each execution time, WSOTF requests a number N (integer) and it repeats N times to generate N traces if there is no error and the corresponding verdict is $Pass$. WSOTF will stop immediately if it finds an error (message receive incorrect, or timeout).

III. Passive testing

Passive testing is a method that collects the traces and analyzes it to conform (i.e., with respect) to a set of constraints (i.e., rule) [9], [10], [11]. In this section, we introduce a formal syntax to define a constraint and a trace collection mechanism for web services. We can understand a rule on a natural language as follow: if a message $M1$ occurs then a message $M2$ (or a suite of message $SM2$) must occur before/after $M1$ for a period of time. A generic temporal logic (like LTL) is usually used to define the constraints on the order of messages for model checking engine. Modeling constraint is required to specify permissions and prohibitions. However, the following problems must be considered when we define a set of constraints to verify a real-time system:

- Time constraints: The passive testing is the verification of messages order (before/after) in a sequence. When a message has occurred and we are waiting the next message. This message may appear after a long duration where our system does not satisfy or does not appear. We cannot know that a message appears or not if we do not have a deadline because our system is running. Take for example a time constraint to verify a successful login if we send a $loginResquest$, we must receive a $loginResponse$ within 10 seconds.
- Condition on message content: In case we do not want to verify all message in a sequence, we only verify some messages which its content satisfies some conditions. For instance, we are only interested in the messages that are sent/received to/from the machine A. This information is identified by message content (SourceIp=A or DestIp=A).
- Data correlation: A observable trace may be mixed by many traces or many sessions that are executed in parallel and we need to apply our constraints on the messages that belong to a trace or a session. In this case, firstly we must find the messages that have a correlation by its data values, then apply our constraints on these messages. For example, there are many sessions that are executed in parallel and each message has a $sessionId$ field. Afterwards, we need to group the messages belonging to a session by using $sessionId$ field of the messages before applying our rules to check the correctness. This is called data correlation.
- Combination of conditions: Sometime, to express a constraint, we must use the operations AND, OR, NOT to combine the conditions. For example, to express a security whenever we modify the database, before needing a login and this session still validate (i.e., do not logout).

A. Rule definition

In our works, we consider each message as an atomic action. We use one or several messages to define a formula by using the operations AND, OR, NOT. During the formula definition, the constraint on message parameters value may be considered. Finally, from these formulas, the rule is defined in two parts: supposition (or condition) and context. The set of data correlations are included as option.

Definition 1: (Atomic action): We define an atomic action as one of following actions: an input message, an output message. Formally:

$$AA \ := \ Event(Const)$$

where:

- $Event$ represents an input/output message name;
- $Const \ := \ P \ \approx \ V | Const \wedge Const | Const \vee Const$ where:
  - P are the parameters. These parameters represent the relevant fields in the message.
  - V are the possible parameters values.
  - $\approx \in \{=, \neq, <, >, \leq, \geq\}$.

Definition 2: (Formula): A formula is defined recursively as following:

$$F \ := \ start(A) \mid done(A) \mid \neg F \mid F \wedge F \mid F \vee F \mid O^{d \in [m,n]} F$$

where:

- A is the atomic action.

- $start(A)$: A is being started.
- $done(A)$: A has been finished.
- $O^{d\in[m,n]}F$: F was true d units of time ago if $m > n$, F will be true d units of time if $m < n$, where m, n are two natural numbers.

Definition 3: (Data correlation): A data correlation is a set of parameters that have the same data type where each different parameter represents a relevant field in a different message and the operator = (equal) is used to compare a parameter with others. A data correlation is considered as a property on data.

Definition 4: (Rule with data correlation): Let α and β be formula, CS is a set of data correlations based on α and β (CS is defined based on the messages of α and β). A rule with data correlation is defined as: $\mathcal{R}(\alpha|\beta)/CS$ (CS is an optional part) where $\mathcal{R} \in \{\mathcal{P}$: permission; $\mathcal{F}$: Forbidden;$\}$. The constraint $\mathcal{P}(\alpha|\beta)$ (resp. $\mathcal{F}$) means that it is permitted (resp. prohibited) to have α true when context β holds within the conditions of CS.

Example 1: We only allow a new account on the services if we have had successfully login within maximum one day ago and have not logged out.

$$\mathcal{P}(start(createAccountReq)|O^{d\in[1,0]D}$$
$$done(loginRes[sessionId \neq null]) \wedge\neg done(logoutReq))$$

In case we want to indicate the messages belonging to a session by using sessionId.

$$\mathcal{P}(start(createAccountReq)|O^{d\in[1,0]D}$$
$$done(loginRes[sessionId \neq null])\wedge\neg done(logoutReq))/$$
$$\{\{createAccountReq.sessionId,$$
$$loginRes.sessionId, logoutReq.sessionId\}\}$$

Semantics: A model of rules corresponds to a pair $r = (P_r, C_r)$ where:

- P_r is a total function that associates every integer x with a propositional formula.
- C_r is a total function that associates every integer x with a pairs (α, d) where α is a formula and d is a positive integer.

Intuitively, $\forall x$, $p \in P_r(x)$ means that proposition p is true at time x. Then (α, d) $\in C_r(x)$ means that context of formula α holds (evaluate true) at time t where

- $t \in [x, x + d]$ if we focus on future time.
- $t \in [x - d, x]$ if we focus on past time.

B. RV4WS: A tool for passive testing

RV4WS (Runtime Verification for Web services) is implemented to verify a web service at runtime based on a set of constraints that are declared by the defined syntax in section III-A. This tool receives a sequence of messages (message content and its occurrence time) via a TCP/IP port, then verifies the correctness of this sequence. The detail of architecture is shown in Figure 3.

One of the most interesting components in this architecture is the checking engine component that implemented

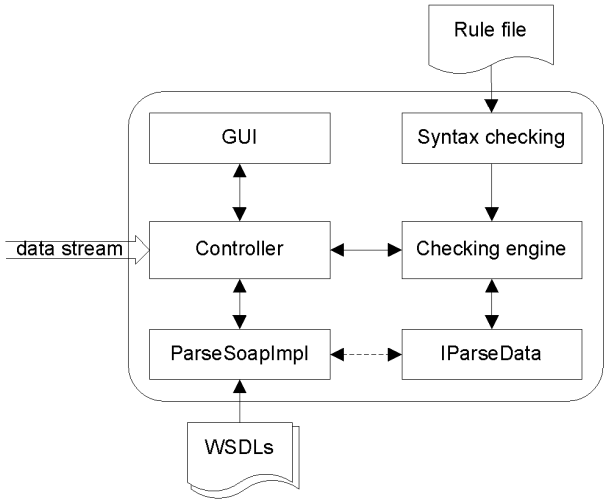


Figure 3. Architecture of the RV4WS tool

the runtime verification algorithm [12]. The engine allows us to verify each incoming message without any constraint of order dependencies, so we can apply this approach to both of on-line and off-line testing. Also, this algorithm verifies the validation of current message without using any storage memory. In order to use this engine for the other systems, there is a difference between the systems is the data structure of input/output messages, we define an interface (i.e., $IParseData$, shown in Figure 4) as an adapter to parse the incoming data of RV4WS. The methods in $IParseData$ are for gathering information from incoming message. $getMesssageName()$ returns the message name from its content and $queryData()$ allows us to query a data value from a field of message content. In each concrete case, we will implement this interface. For example, in the case of Web services, its implementation is the class $ParseSoapImpl$. This engine has been designed as a Java library and is controlled by a component called Controller, which received a data stream coming from TCP/IP port.

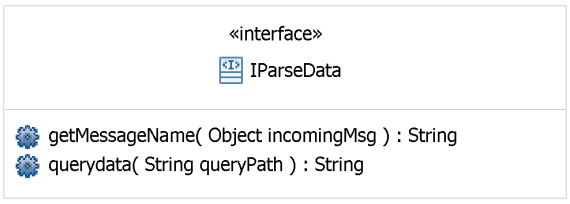


Figure 4. ParseData Interface of RV4WS

The input format for this tool is a XML file that has been defined in Figure 5. A rule with a $true$ verdict represents a permission and a $false$ verdict represents a prohibition. A context of rule will be expressed as an expression with three operators AND, OR and NOT. Each data correlation is defined as a property with some query expressions from the

different SOAP messages. In the case of web services, we have developed a Graphic User Interface (GUI) that allows us to easily define a set of rules from WSDL files.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
      <rule applyProperty="true" id="1" name="" verdict="true">
      <if>
            <message> requestRequest[]</message>
      </if>
      <then time_max="1" time_min="0" time_type="m" type="after">
            <context>
                  <expression> xLoanConfirmRequest[]</expression>
            </context>
      </then>
      <properties>
            <property name="correlation_Id" type="int">
                  <query>requestRequest.requestInfo/id</query>
                  <query>xLoanConfirmRequest.confirmIn/id</query>
            </property>
      </properties>
      </rule>
</rules>
```

Figure 5. Rule format example

If a rule is found to be not satisfying, the checking algorithm returns a fail verdict. This rule may be not applied to the current message. To know which rule has failed at an arrival message, we have also presented a Graphic User Interface (GUI) that is used to visualize some statistical properties, calculated at any moment of testing process. Whenever a rule is activated, this means that its conditions have been satisfied and a statistical property such as type counter will be used to compute the percentage of unsatisfying time when applying the rule on the input data stream. If the rule was satisfied, we need to know the time duration from the activating moment to its context's holding moment. We have three statistical properties about time (time-min, time-max and time-average) for each rule.

IV. Case Study

A. Product Retriever

In this section, we present a real-life case study, named Product Retriever [13], of WebMov project. This case study is a BPEL process that allows users to automate part of the purchasing process. It enables you to retrieve one searched product sold by a preauthorize provider. The search is limited by specifying a budget range and one or more keywords characterizing the product. The searched product is done through the operation $getProduct$ and the parameter $RequestProductType$ that is composed of information about the user (firstname, lastname and department) and searched product (keyword, max price, category). This process has 4 partner services, named $CurrencyExchange$, $AmazonFR$, $AmazonUK$ and $PurchaseService$ that are developed by Montimage[1] and available at http://80.14.167.59:11404/servicename, and its overview behavior, illustrated in Figure 6, is described by the following:

[1]http://www.montimage.com/

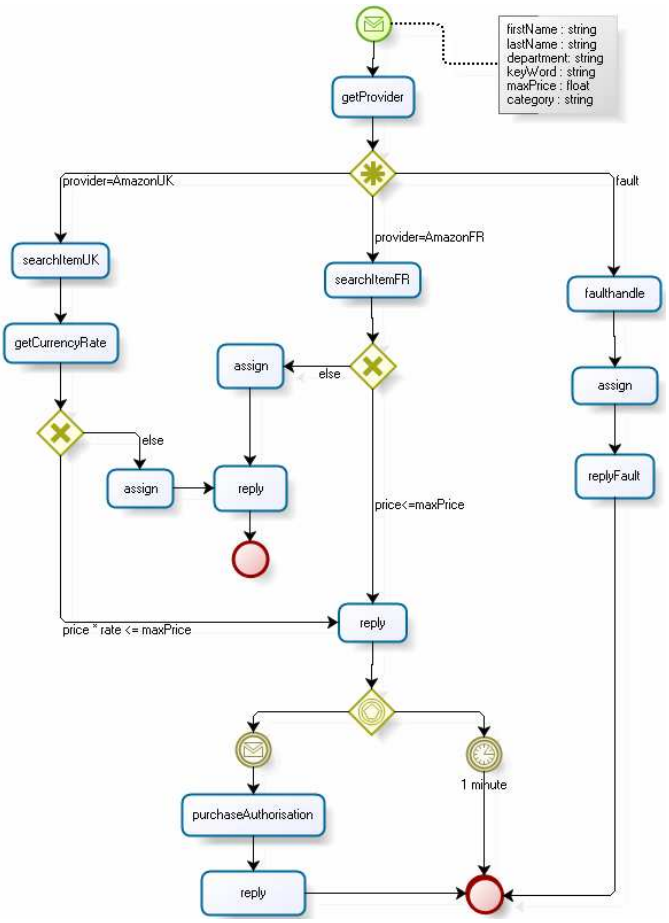


Figure 6. ProductRetriever - BPMN specification

1) Receives a message from the client with the product and keywords of the characteristics of the product.
2) Contacts the $PurchaseService$ partner to obtain the list of authorized providers for that product. In a case where there is no authorized provider, an announcement will be sent to the client by a fault message response.
3) Depending on the authorized provider result, the process contacts either the AmazonFR or the AmazonUK service to search a product that matches the price limit by Euro and the keywords.
4) Sends back to the client the product information and the name of the provider where the product was found and a link to where it can be ordered. If a matching product is not found, a response with unsatisfying product will be sent back to the client.
5) After receiving the product information, the client can send an authorization request to confirm the purchase of the product within a certain duration (i.e., one minute) of time.

The Product Retriever service is built in Netbeans 6.5.1 and deployed by a Sun-Bpel-engine within a Glassfish 2.1 web server.

B. Experiment results

1) Combination of two tools: This section describes how to combine our two tools to test a web service. We use our active testing tool (i.e., WSOTF) to test the Product Retriever (PR) service. While WSOTF is executing a test, the communicating between PR service and the WSOTF tool are configured via the proxy that collect the input/output messages to build a trace. Then the correctness of the trace is checked, in parallel with WSOTF, by the RV4WS tool. Figure 7 shows a testbed architecture that we combine our two tools. The test results will be analyzed by each tool in the next section.

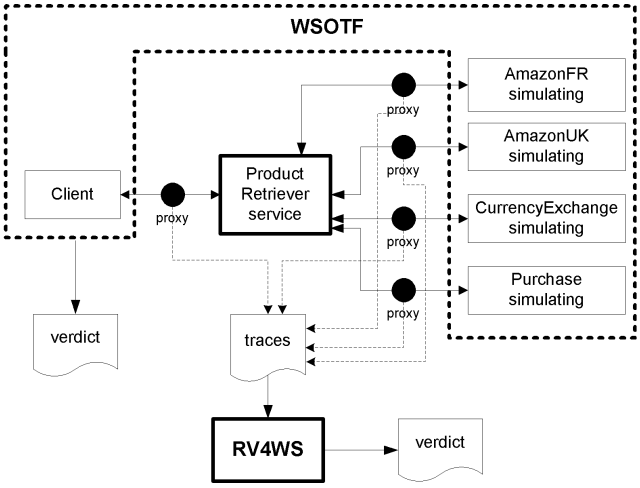


Figure 7. Testbed architecture of two tools

2) Modeling the Product Retriever orchestration by TEFSM: Lallali et al [7] define many rules to translate a BPEL specification into a TEFSM. They use a local clock and a global clock is optional to verify the time constraints. The value of input/output message will be handled by a variable. Here, we use these rules to model the Product Retriever orchestration by a TEFSM. But the assign activity is not considered because we are only interested in the time constraints and the communicating activities. Figure 8 presents a graph of ProductRetriever's TEFSM.

3) Test results and analysis of WSOTF tool: In this section, we introduce the test results of Product Retriever service on WSOTF tool. Because this case study has the data correlation when the client sends the message getAuthorisationRequest (firstName, lastName and userId). Our tool currently does not support this problem when it generates the SOAP messages. However we can declare an enumeration for each field of SOAP messages in an enum file. To limit the repeat of coincident cases, we declared a list of fix values

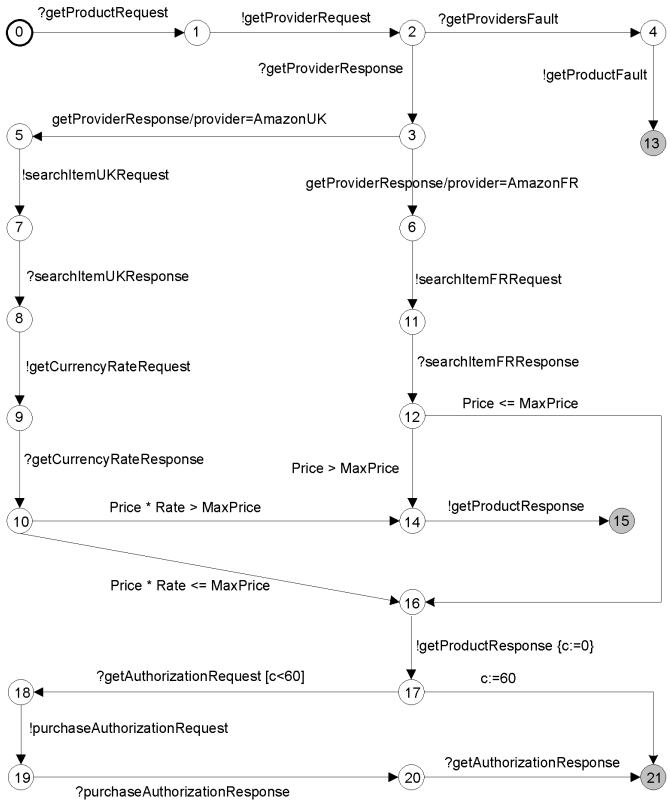


Figure 8. TEFSM of ProductRetriever

for the condition fields, for example: provider=AmazonFR; AmazonUK or maxPrice=1000;900;800 etc. Table I resumes the test results of Product Retriever service using WSOTF. In these test results, we found a fault that is not admitted in the specification when the product price by Euro of the partner is greater than the maxPrice.

4) Testing analysis by RV4WS: We can define many test purposes to verify the interaction order with partner services. Here we introduce three test purposes:

1) During the execution of service, if the client receives a $ProductFault$ message, so before the Purchase service must return a $ProviderFault$ message. The time constraint of this test purpose is less important, so we define the maximum interval time between two messages as 10 seconds.
$$\mathcal{P}(start(ProductFault)|O^{d\in[10,0]s} done(ProviderFault))$$
2) If the Purchase service introduces the provider service AmazonUK then the orchestration must contact the CurrencyExchange service within maximum of 10 seconds.
$$\mathcal{P}(start(getProviderResponse[provider = AmazonUK])|O^{d\in[0,10]s} done(getCurrencyRateRequest))$$
3) When the client sends an authorization request mes-

	Traces	Verdict
1	?getProductRequest(maxPrice=1000) → !getProviderRequest() → ?getProviderResponse(provider=AmazonFR) → !searchItemFRRequest() →?searchItemFRResponse(price=500) → !getProductResponse(price=500) → delay=60	Pass
2	?getProductRequest() → !getProviderRequest() → ?getProviderFault() → !getProductFault()	Pass
3	?getProductRequest(maxPrice=800) → !getProviderRequest() → ?getProviderResponse(provider=AmazonUK) → !searchItemUKRequest() →?searchItemUKResponse(price=500) → !getCurrencyRateRequest() → ?getCurrencyRateResponse(rate=1.5) → !getProductResponse(price=750) → delay=60	Pass
4	?getProductRequest(maxPrice=1000) → !getProviderRequest() → ?getProviderResponse(provider=AmazonUK) → !searchItemUKRequest() →?searchItemUKResponse(price=500) → !getCurrencyRateRequest() → ?getCurrencyRateResponse(rate=2) → !getProductResponse(price=1000) → ?getAuthorizationRequest() → !purchaseAuthorizationRequest() → ?purchaseAuthorizationResponse() → !getAuthorizationResponse()	Pass
5	?getProductRequest(maxPrice=1000) → !getProviderRequest() → ?getProviderResponse(provider=AmazonUK) → !searchItemUKRequest() →?searchItemUKResponse(price=500) → !getCurrencyRateRequest() → ?getCurrencyRateResponse(rate=2.5) → FAULT	Fail

Table I
TEST RESULTS OF PRODUCTRETRIEVER BY WSOTF TOOL

sage to confirm the purchase of a product, so it must receive a product response message with the *EmptyResponseProduct* field be null within maximum one minute ago. In this rule, the data correlation is used by $userId$.

$$\mathcal{P}(start(getAuthorizationRequest)|O^{d\in[1,0]m} done(getProductResponse [EmptyResponseProduct = null]))/ (getAuthorizationRequest.userid, getProductResponse.userid)$$

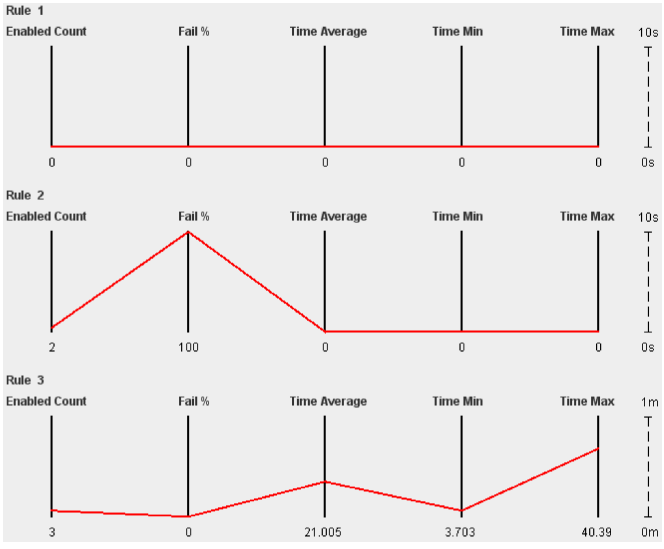
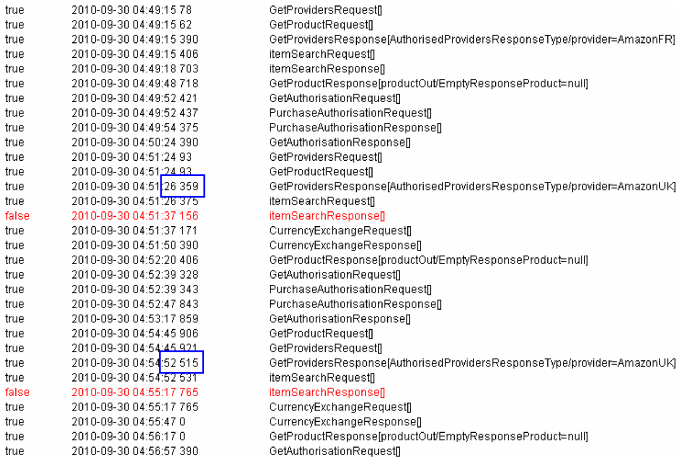


Figure 9. Checking analysis of Product Retriever

Figure 9 presents the checking analysis of the Product Retriever. This figure indicates: 1) the fault messages that are defined in rule 1 do not occur. 2) the *getProviderResponse* message with *provider=AmazonUK* appeared two times, but the tool did not found a message *getCurrencyRateRequest* within 10 seconds from the occurrence time of the message *getProviderResponse*. See figure 10, we found the interval time between them is 11 seconds for the first case and 25 seconds for the second case. 3) the message *getAuthorizationRequest* appeared three times. Before that, the *getProductResponse* message also appeared with the field *EmptyResponseProduct* is empty and the interval time between them is less than one minute. Figure 10 returns the $false$ verdict when the *itemSearchResponse* arrives because at the occurrence time of *itemSearchResponse*, the time constraint for the second rule (i.e., 10 seconds) does not satify.

V. CONCLUSIONS

In this paper, we introduce two testing approaches and two correspondent tools to test web services. One tool focuses on unit testing of an orchestration of web service composition by applying on-line approach. The other tool focuses on passive testing by verifying a timed trace with respect to a set of constraints. Finally, by combining two tools to test a case study, we have shown an experiment on how to use our tools to test a web service. This experiment indicates that the active testing is not enough to find the bugs. We need to combine two approaches (active and passive approach) to check in parallel the conformance of a SUT



true	2010-09-30 04:49:15 78	GetProvidersRequest[]
true	2010-09-30 04:49:15 62	GetProductRequest[]
true	2010-09-30 04:49:15 390	GetProvidersResponse[AuthorisedProvidersResponseType/provider=AmazonFR]
true	2010-09-30 04:49:15 406	itemSearchRequest[]
true	2010-09-30 04:49:18 703	itemSearchResponse[]
true	2010-09-30 04:49:48 718	GetProductResponse[productOut/EmptyResponseProduct=null]
true	2010-09-30 04:49:52 421	GetAuthorisationRequest[]
true	2010-09-30 04:49:52 437	PurchaseAuthorisationRequest[]
true	2010-09-30 04:49:54 375	PurchaseAuthorisationResponse[]
true	2010-09-30 04:50:24 390	GetAuthorisationResponse[]
true	2010-09-30 04:51:24 93	GetProvidersRequest[]
true	2010-09-30 04:51:24 93	GetProductRequest[]
true	2010-09-30 04:51:26 359	GetProvidersResponse[AuthorisedProvidersResponseType/provider=AmazonUK]
true	2010-09-30 04:51:26 375	itemSearchRequest[]
false	2010-09-30 04:51:37 156	itemSearchResponse[]
true	2010-09-30 04:51:37 171	CurrencyExchangeRequest[]
true	2010-09-30 04:51:50 390	CurrencyExchangeResponse[]
true	2010-09-30 04:52:20 406	GetProductResponse[productOut/EmptyResponseProduct=null]
true	2010-09-30 04:52:39 328	GetAuthorisationRequest[]
true	2010-09-30 04:52:39 343	PurchaseAuthorisationRequest[]
true	2010-09-30 04:52:47 843	PurchaseAuthorisationResponse[]
true	2010-09-30 04:53:17 859	GetAuthorisationResponse[]
true	2010-09-30 04:54:45 906	GetProductRequest[]
true	2010-09-30 04:54:45 921	GetProvidersRequest[]
true	2010-09-30 04:54:52 515	GetProvidersResponse[AuthorisedProvidersResponseType/provider=AmazonUK]
true	2010-09-30 04:54:52 531	itemSearchRequest[]
false	2010-09-30 04:55:17 765	itemSearchResponse[]
true	2010-09-30 04:55:17 765	CurrencyExchangeRequest[]
true	2010-09-30 04:55:47 0	CurrencyExchangeResponse[]
true	2010-09-30 04:56:17 0	GetProductResponse[productOut/EmptyResponseProduct=null]
true	2010-09-30 04:56:57 390	GetAuthorisationRequest[]

Figure 10. Trace collection of Product Retriever

with its specification and some global properties of system at runtime.

The first perspective is to apply the tools on a larger number of Web services. We must also develop the advantages of combining the assets test and liabilities test for a better overall quality of the test. On the other hand, significant problems in the field of Web services concern the security. The choice of language Nomad [14] will help to develop a test methodology for security. A final perspective is the extension of the methods and tools for the cloud computing.

ACKNOWLEDGMENT

This research is supported by French National Agency of Research within the WebMov project http://webmov.lri.fr

We also would like to thank Nguyen Thi Kim Dung, a master student of PUF (Pole Universitaire Français) in Ho Chi Minh city, who help us to develop the RV4WS tool in the context of her internship in LaBRI.

REFERENCES

[1] D. Dranidis, E. Ramollari, and D. Kourtesis, "Run-time verification of behavioural conformance for conversational web services," in *2009 Seventh IEEE European Conference on Web Services*, Eindhoven, The Netherlands, Nov 9 - 11 2009, pp. 139 – 147.

[2] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + Astro: An integrated approach for BPEL monitoring," in *2009 IEEE International Conference on Web Service*, Los Angeles, CA, USA, July 6-10 2009, pp. 230 – 237.

[3] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "WS-TAXI: A WSDL-based testing tool for web services," in *2009 International Conference on Software Testing Verification and Validation*. Denver, Colorado - USA: IEEE Computer Society, April 01- 04 2009, pp. 326–335.

[4] G. J. Tretmans and H. Brinksma, "TorX: Automated model-based testing," in *First European Conference on Model-Driven Software Engineering*, Nuremberg, Germany, Dec. 11-12 2003, pp. 31–43.

[5] M. Mikucionis, K. G. Larsen, and B. Nielsen, "T-UPPAAL: Online model-based testing of real-time systems," in *19th IEEE international conference on Automated software engineering*, Linz, Austria, Sept 24 2004, pp. 396–397.

[6] T.-D. Cao, P. Felix, R. Castanet, and I. Berrada, "Online testing framework for web services," in *Third International Conference on Software Testing, Verification and Validation*, Paris, France., April 6 - 9, 2010, pp. 363 – 372.

[7] M. Lallali, F. Zaidi, and A. Cavalli, "Timed modeling of web services composition for automatic testing," in *IEEE International Conference on Signal-Image Technologies and Internet-Based System*, Shangai, China, Dec. 16-19, 2007, pp. 417–426.

[8] Eviware, http://www.eviware.com/.

[9] C. Andrés, M. G. Merayo, and M. Núnez, "Formal correctness of a passive testing approach for timed systems," in *IEEE International Conference on Software Testing, Verification, and Validation Workshops*, Denver, Colorado , USA, April 01- 04 2009, pp. 67–76.

[10] 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. 247–266, 2005.

[11] A. Cavalli, A. Benameur, W. Mallouli, and K. Li, "A passive testing approach for security checking and its pratical usage for web services monitoring," in *NOTERE 2009*, Montreal, Canada, 2009.

[12] T.-D. Cao, T.-T. Phan-Quang, P. Felix, and R. Castanet, "Automated runtime verification for web services," in *2010 The IEEE International Conference on Web Services.*, Miami, Florida, USA, July 5-10, 2010, pp. 76–82.

[13] Montimage, "Webmov case studies: definition of functional requirements and test purposes," WebMov, Tech. Rep. WEBMOV-FC-D5.1/T5.1, 2009.

[14] F. Cuppens, N. Cuppens-Boulahia, and T. Sans, "Nomad: A security model with non atomic actions and deadlines," in *18th IEEE workshop on Computer Security Foundations*. IEEE Computer Society, 2005, pp. 186–196.