WebMov: A dedicated framework for the modelling and testing of Web Services composition

Ana Cavalli[1], Tien-Dung Cao[2], Wissam Mallouli[3], Eliane Martins[4],
Andrey Sadovykh[5], Sebastien Salva[6], Fatiha Zaidi,[7,8]
[1] TELECOM SudParis - CNRS SAMOVAR, F-91011 Evry, France.
[2] LaBRI - CNRS - UMR 5800, Univ. of Bordeaux 1
[3] Montimage, 39 rue bobillot, 75013 Paris
[4] Unicamp, Institute of Computing (IC), Brasil
[5] SOFTEAM, 21avenue Victor Hugo, 75016 Paris
[6] LIMOS,Univ. of Auvergne, Aubière, France
[7] Univ. Paris-Sud, LRI, UMR 8623, Orsay F-91405;[8] CNRS, Orsay, F-91405.

***Abstract*—This paper presents a methodology and a set of tools for the modelling, validation and testing of Web service composition, conceived and developed within the French national project WebMov. This methodology includes several modelling techniques, based mainly on some variations of Timed Extended Finite State Machines (TEFSM) formalism, which provide a formal model of the BPEL description of Web services composition. These models are used as a reference for the application of different test generation and passive testing techniques for conformance and robustness checking. The whole WebMov methodology is integrated within a dedicated framework, composed by a set of tools that implement the model representation, the test generation and passive testing algorithms. This framework also permits the interaction of these tools to achieve specific modelling and testing activities in a complementary way. A case study based on a real service, a Travel Reservation Web Service, is presented as well as the results of the application of the proposed WebMov methodology and tools.**

I. INTRODUCTION

SOA (Service Oriented Architecture) and Web services are gaining industry wide-acceptance and usage. They are very popular because they offer complete interoperability between systems. This popularity implies that Web service implementations grow in size and complexity, the necessity for testing their reliability is becoming more and more crucial. Besides knowing whether a Web service satisfies its functional requirements, it is also important for a user to know whether each service behaves appropriately when interoperating with other services. New services can be constructed by composition of services. This composition mechanism is called orchestration. To achieve compositions a language, the WS-BPEL (BPEL for short), has emerged as a standard. In this paper, we present the framework that has been developed within the national project WebMov[1] to contribute to the modeling and testing of Web services orchestration. The objectives of this platform are manifold. We define a methodology based on a high level abstraction view and a SOA based logical architecture for the design and composition of Web services. Based on the BPEL model derived from this architecture, we derive formal models from which test cases are generated. The latter allow us to apply model-based testing techniques and to reason with a non ambiguous model to automate the test cases generation. These latter have to take into account the specific features of BPEL, all its constructs, and its data part described by XML schemas. We also propose in this framework a monitoring approach based on passive testing technique which permits to test the deployed Web services without interacting with it. To complement the conformance testing, we also first test the Web services robustness. We address this issue by a black box stateless Web services testing method that sends to the implementation unexpected events and then a fault injection method. This one, by aiming to provoke errors and failures, permits to assess error recovery and fault tolerant mechanisms. We have developed a tools chain covering the different steps previously mentioned and hence the different steps of services life cycle (modelling, orchestration, test generation and execution) as well as service operation by monitoring the services.

[1]Research supported by the French National Agency of Research, http://webmov.lri.fr

The paper is structured as follows. Next Section discusses related work. Section III describes the tools chain developped inside the WebMov project. In Section IV we present the logical architecture and sketch one formal model used by the testing generation method. Section V lay out our model-based testing approach for conformance and robustness testing. Section VI describes the passive testing technique and also fault injection technique. Finally, Section VII presents the experimentals results and Section VIII ends with conclusions and perspectives.

II. Related Work

It exists several tools that permit to perform unit testing of Web services composition. We can cite SOAPUI or BPELUNIT [1]. However, test cases are mainly generated using empirical approaches, and without automation. The tests are handcrafted. By surveying the literature, we found approaches that only address the test of services by focusing on the services signatures, i.e. their WSDL descriptions [2]. A Web service, especially composite service, described in BPEL, provides also a behavioural description. To test the behaviour, several works have been proposed that are based on classical code coverage criteria and different techniques have been used, white-box testing [3]–[5], model-checking [6], [7]. Within the WebMov project, we propose efficient algorithms to perform model-based testing that take into account both level of Web services description, i.e. the signature level and behaviour level. By the use of formal approach, we can generate automatically test cases to establish the conformance of the implementation. We also support robustness testing and provide an integrated framework that cover the life cycle of composite Web services. To our knowledge, it do not exist such an initiative in this domain.

III. Description of the tools chain

A. Tools chain Overview

The main objective of the WebMov project is to study and develop different methodologies and conceive associated software tools that allow formally designing, testing and validating Web Services composition. These tools are integrated in a dedicated framework (see Figure 1) that implements these methods and algorithms and allows the interactions between different tools in order to achieve specific tasks.

In the tools chain, *Modelio* can be considered as the starting block of WebMov methodology. This tool allows to automatically generate a composed Web service based on its logical architecture and on UML profile. Three types of files are generated: (1) the WSDL interfaces of the main service and its Web partners, (2) the BPEL process file describing the composition of the Web services (3) and the data part described in XSD format.

These generated files can be used for two different purposes. The first purpose is the classical one and corresponds to the model driven engineering (MDE) that aims at supporting a quick implementation and deployment of designed Web services. In this case, the deployed Web service is considered as the system under test and the other WebMov tools will rely on other inputs (e.g. the Web service requirements document) to be able to generate tests and check the correctness of the implemented Web service.

The second usage of the output files of *Modelio* are destined to testing purposes. Indeed, based on the WSDL description of each Web service, we can use the *WS-AT* tool to perform a unit test of individual Web services. *WS-AT* gives a testing verdict on the conformance of the Web service to its description provided in the WSDL and also checks its robustness. It checks the operations existence, the exceptions management, the type of returned values and also checks the Web service robustness by observing whether each operation does not crash or hang by calling it with the hazard “using unusual values”.

The WSDL files combined with the BPEL file can be considered as inputs for the *BPEL2IF* tool. This tool considers the BPEL as the specification model and generates a formal model of the composed Web service as an IF timed automaton [8]. IF formalism is well adapted to describe communicating systems like Web services and is composed of active processes instances running in parallel and interacting asynchronously through shared variables and signals via channels or by direct addressing. Based on the formal specification of the Web service under test (e.g. the IF formal specification), a set of test generation tools is conceived and implemented during the WebMov project. These tools relie on different models (IF for *TestGen-IF*, EIOTA for *TGSE* and STS for *STG*) and allow to automatically generate abstract test cases. These tests are translated to concrete XML tests that can be interpreted by the activeBPEL tool.

The execution of the generated test cases can be performed by using *activeBPEL* tool that allows to emulate the composed Web service and its partners. *ActiveBPEL* is a BPEL engine that allows the managing of the emission and reception of SOAP messages and the addressing of Web services functionalities as described in the orchestration of the BPEL process. *ActiveBPEL* can be easily combined with the *BPELUnit* framework that allows unit testing of Web services. In the BPELUnit framework, the composed Web service under test is isolated from its real environment. It is deployed in an application server (e.g. Tomcat) while the client and the service partners are emulated respectively by BPELUnit as a customer track and partners tracks. When running tests, all tracks are run simultaneously as independent services.

Some fault injection strategies can be applied on the Web service to test its robustness. The *WSInject* tool permits to inject both communication and interface faults and can be used to perturb communication between a client application and a single Web service or between services from a global composition.

ActiveBPEL and *WS-Inject* tools interact with Web service under test and stimulate it to analyze its reaction. *TIPS* allows to passively testing the deployed composed Web service under test (SUT) without disturbing its natural run-time (non intrusive testing technique). It allows to check if a set of properties called invariants are respected on a collected trace of the Web service (the capture of the trace can be performed by a classical sniffer e.g. Wireshark). The invariants are a set of behavioral constraints with time constraints that need to

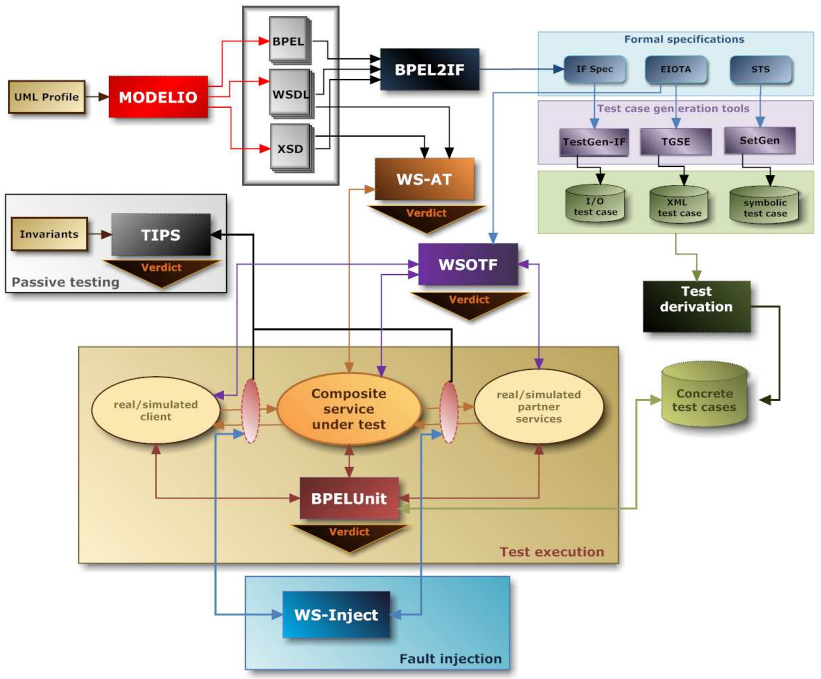

Figure 1. WebMov tools chain.

be respected by the system under test. This may include the sequence order of exchanged messages, the communicated data, etc. This passive technique can be combined with the active testing and fault injection techniques in order to emit a verdict about the correctness of the deployed Web services.

IV. Logical architecture and formal model

A. *The logical Architecture*

The Logical Architecture is the starting point of our approach. This one was developed by SOFTEAM [9] in the frame of Enterprise Architecture modeling approach instrumenting architecture frameworks such as TOGAF [10] and PRAXEME [11]. The Logical Architecture is dedicated to fill the gap between the business models and the technical design models by providing a simple and flexible language for describing the service oriented architectures. Indeed, following the architecture frameworks, on the business model level, the goals, requirements, business semantics and organization are described. On the technical design model level, UML models [12] for specific platforms are elaborated. In order to ensure the alignment between business and IT systems, the service layer models are elaborated, which allows business domain specialist and IT specialists to efficiently collaborate. The Logical Architecture defines the service layer of the IT system: service components, services, exchanged messages, components assembly and finally the components behavior with extended BPMN [13] models. Service components represent the main functional building blocks of the system. They are linked to each other through required and provided interfaces services. The services provide operations parameterized with message types. The message types define the exchanged data. The behavior of the service components is expressed in BPMN, which is close both to the business and IT specialists. In the context of the project, the Logical Architecture was extended with specific model transformations targeting the Web Service platform. With these transformations, the message descriptions are translated into XML Schemas and WSDL, while the behavioral descriptions in BPMN are translated into BPEL. In addition, specific modeling language extensions were implemented in order to introduce the test purposes at the SOA modeling stage. This is achieved by specific annotations, which are attached to model elements. This allows generation of skeletons for test purposes specifications.

B. *Formal Model*

Several formal models are proposed in this paper and are associated to a testing method. As said in section III, test generation tools have been developed in our project. Each one relies on a different test approach and also uses different formal models such as *IF* for *TestGen_IF*, *EIOTA* for *TGSE* and *WSOTF*, and *STS* for *SetGen*. One formal model, the TEFSM [14] (i.e., Timed Extended Finite State Machine) can cover two timed models (*IF* and *EIOTA*), of our formal model based approaches. This formalism is closely related to Timed Automata [15] and permits to carry out timing constraints, clocks, state invariants on clocks, properties on transitions and data variables. The tool

BPELtoIF is able to generate from the BPEL the formal model. We have also defined in an process algebraic way the transformation of BPEL in STS [19]

V. Model-based testing

Model-based testing of Web service consists in deriving a suite of test cases from a model representing its behaviour. Such a model can be generated from an informal specification of the service and designed by software engineers through the use of diagram manipulation tools. Moreover, Web services can express time constraints; consequently, engineers developing these services are confronted with functional requirements with time constraints. One of the main objectives of the WebMov project is the definition of model-based testing methods to test web services composition.

A. Conformance Testing

For Web services composition, we can distinguish two types of testing: unit and integration testing [2, 3]. Unit testing is used to find bugs on an isolated (separated from partners) composite Web service. Integration testing is used to test this composite service in combination with its partners. In addition, conformance testing establishes that the implementation respects its specification. We know that testing conceptually consists of three activities: test case generation, test case execution and verdict assignment. Depending the test approach, these activities can be applied in parallel or in sequential. We present here two test approaches: $offline$ testing and *online* testing.

1) Offline testing: The first approach is based on the unit testing of an implementation of a Web service composition, based on a black box approach [16]. This last means that a composite Web service is tested without any information on its internal structure. In this approach, system requirements are specified as test purposes, which will be transformed on tests to be executed against the service implementation. The specification of the Web service composition is done using BPEL, which permits to describe the service taking into account the data and flow requirements, and the WSDL descriptions. To facilitate the test generation a formal model of BPEL is proposed. With this aim, it is proposed a BPEL transformation into an Intermediate Format (IF) language which is associated to an efficient open-source simulator. Using this simulator, we can explore, through exhaustive simulation, the state space of the model and generate test cases. Moreover, this language is based on communicating timed automata extended with variables, which can handle the BPEL constructs. As we pointed out before, the proposed black box approach is based on test purposes and a conformance relation. To guide the test generation, we developed a test generation algorithm with (timed) test purposes. The conformance relation is defined as a timed-traces-inclusion relation. We introduced, in the IF simulator, a Hit-or-Jump exploration strategy [17] to generate the test cases in a timely fashion. The proposed methods are implemented by a set of tools, one is devoted to the BPEL transformation approach, and the other one to implement the test generation algorithm by extending the IF toolset [8].

The second approach is based on an existent tool (i.e., TGSE). As previously, we focus on unit testing of web service composition, based on black-box approach and system requirements are also specified as test purposes [18]. But in this approach, both the specification (i.e. BPEL and WSDL) and the test purpose are translated into an EIOTA. Secondly, these EIOTAs are modeled as a communicating system (CS) by declaring a set of rules to synchronize the actions of test purpose and the actions of its specification. Afterwards, using TGSE to simulate this CS, we have an output XML file if a trace that satisfies the test purpose is found. Finally, this trace is run by BPELUnit against the orchestration implementation to give the verdict. This approach complements the work on timing constraints of the first approach by solving constraints related to time and giving intervals for such constraints. Such intervals can be used by the first approach to assign values to clocks.

Finally an approach based on Symbolic Transition Systems, which support data computations and exchanges, while avoiding the state explosion problem, has been developed [19]. The proposed methodology is as follows: the orchestration specification is first translated into a formal model, namely a Symbolic Transition System (STS). In a second step, a Symbolic Execution Tree (SET) is computed from this STS. It supports the retrieval of the STS execution semantics, without producing a state explosion in the presence of unbounded data types, as used in full-fledged BPEL. Given a coverage criterion, from the SET a set of execution paths is generated, and are finally run by a test oracle against the orchestration implementation. Compared to the previous approaches, this approach is more efficient to handle the XML data part and can circumvent the state space explosion.

2) Online testing: Online testing is an approach that combines test generation and execution: only a single test primitive (input event or time delay) is generated from the model at a time which is then immediately executed on the service under test (SUT). Then the output produced by the SUT as well as its time of occurrence is checked against the specification [20]. At a time, a new test primitive is produced based on values of previous events or random selection if there is some acceptable options and so forth until we arrive a final state. It means that the complete test scenario (test case suite and data) is built during test execution. This approach does not use test purposes. In WebMov, this latter is also applied for unit testing of web service composition [21].

B. Robustness Testing

Web services are distributed in nature, and can be used by different and heterogeneous client applications. So, Web service trustability is ensured in condition that they also behave correctly despite the receipt of unspecified events, called *hazards*. In other words, Web services must be robust. The Webmov project tackles to Web service robustness by proposing two testing techniques, a black box stateless Web service testing method and a fault injection method.

Web services dip into a peculiar environment, spawned by SOAP. This one modifies and reduces the message observability on account of message serialization and of SOAP processors (first Web server component which filters out the SOAP requests and may produce responses instead of Web services). So, in a preliminary study, we analyzed the Web service observability in the presence of hazards to determine those which can really reveal robustness issues. In [22], we concluded that only the hazards ”Replacing /Adding operation names” and ”Using unusual values” are relevant to stateless and statefull Web service testing. Any other hazard is blocked by SOAP processsors. From this analysis, we propose the two following approaches:

- black box Web service robustness testing: This method, implemented in the *WSAT* tool (Web Service Automatic Testing, see section III), generates and executes test cases automatically from WSDL descriptions, in order to test both the *Existence of all service operations* and the *robustness of all Web service operations*. This method is particularly interesting for testing stateless Web services or BPEL process partners.
  For operation existence, each Operation is called with random parameter values respecting the WSDL file. An operation exists if it returns either a response type as described in the WSDL file or a SOAP fault whose the cause is different from ”Client” or and ”the endpoint reference not found”. (This first cause means the operation is called with bad parameter types. The second cause means that the operation name does not exist).
  Then, the operation robustness is tested by observing whether each operation does not crash or hang by calling it with the hazard ”using unusual values”. This latter is modeled, for each value type, by an XML set composed of values used in software testing which are assumed to have a high bug-revealing rate when used as inputs [23]. An operation is robust if either it returns a ”classical” response (as described in WSDL file), or a SOAP fault whose the cause is equal to ”Remote-Exception”. We consider that the Web service is not robust if no response is observed or if another kind of SOAP fault, constructed by the SOAP processor, is received. The test case schema is illustrated in figure 2. ”pass” means that the operation exists and is robust, ”inconclusive” means that the operation exists only and the ”fail” verdict is given when both the operation existence and robustness are not satisfied.
- fault injection: This method is based on the injection of faults in SOAP messages to test the stateless Web service and BPEL process robustness. This method has been implemented in the *WSInject* tool. In summary, fault injection is modeled by dedicated rules allowing to delete messages, to augment the response delay, to duplicate messages or to corrupt them. A Web service is robust if despite fault injections, it is still available. The test case execution is founded on a passing testing architecture which is described below.

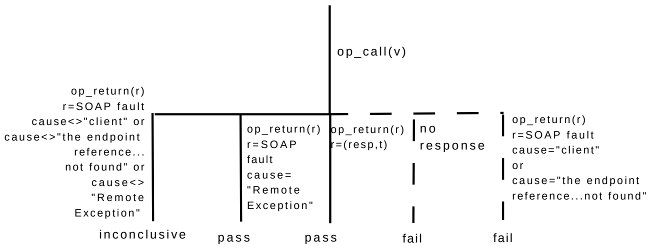
op_call(v)
op_return(r) r=SOAP fault cause<>"client" or cause<>"the endpoint reference... not found" or cause<> "Remote Exception"
op_return(r) r=SOAP fault cause= "Remote Exception"
op_return(r) r=(resp,t)
no response
op_return(r) r=SOAP fault cause="client" or cause="the endpoint reference...not found"
inconclusive
pass
pass
fail
fail

Figure 2. Test case schema for testing the operation existence

VI. PASSIVE TESTING AND FAULT INJECTION TESTING

A. Passive testing

Passive testing [24] consists in observing during run time the exchange of messages (input and output events) between distributed Web services that collaborate to achieve a meaningful business goal. The term passive means that the tests do not disturb the natural operation of Web services (partners) under test. The record of the observed events is called a trace. This trace will be compared to a set of properties derived from the Web services composition specification. The passive testing techniques are applied, in particular, because they are non-intrusive whereas the active testing techniques need to stimulate the system under test and may cause its crush. In addition, passive testing can be applied on a system in its real context (with real users).

In the context of WebMov, TIPS tool [25] (Test of Invariants for Protocols and Services) is conceived and adapted to fit the Web services requirements. This tool aims at passively testing a deployed communicating Web sevice to verify if it respects a set of properties called invariants [24]. These invariants describe the correct order of exchanged messages among Web system entities with conditions on communicated data and time.

In order to use the TIPS, the first step consists in defining the invariants. This can be done by an expert of service under test that understands in details the studied service composition. The next step consists in capturing the communication traces using Wireshark and analysing them using the TIPS tool. In the case of violated invariants, involved packets are provided to help developers to make a diagnosis of the error causes. Can be removed.

B. *Fault injection testing technique*

Fault injection is a powerful testing technique which consists to introduce deliberate errors in a system and observe its behavior. This technique is usually used to assess error recovery and fault tolerant mechanisms, to perform some dependability measures such as availability, integrity and performance or simply to understand the effects of real faults.

In the case of Web services, faults can be injected at both interface and communication levels. Interface faults affect operations input/output parameters and other SOAP message fields by corrupting data or assigning invalid parameter values. On the other hand, communication faults consider SOAP messages as black boxes. Besides corrupting carried data, SOAP messages are replicated, deleted or delayed.

WSInject [26] is a Web service fault injector able to inject both interface and communication faults. In the context of WebMov, this tool is used to study Web services robustness.

WSInject is a script driven tool. Users can specify the type of faults and the injection conditions using a simple and powerful script grammar. During the experiments, the injector intercepts all the SOAP messages exchanged between service partners. Then, according to the injection script, it will inject errors only on messages fitting the injection conditions. Faults can be injected between a client application and its Web service or between service partners within the same composition.

VII. Experimentations and results

In this section, we present the experimentations performed within the WebMov project on a real-life case study: a travel reservation service Web service (TRS). Other experimentations have been processed on other case studies but for sake of space, we will only present those that dealt with TRS. The main objectives of these experimentations are to evaluate the methods and tools developped within the WebMov tool chain in order to demonstrate their effectiveness and their scalability. For this reason, the choice of the case study was very important since we wanted to apply our solution to a complete Web services context that is able to identify realistic requirement and test purposes.

A. *Case study: a Travel Reservation Service*

The Travel Reservation Service (TRS) is an example of real-life service for travel organization provided within the Netbeans IDE 6.5.1 platform [27]. It acts as a logical aggregator of three other Web services: Airline Reservation Service (ARS), Hotel Reservation Service (HRS) and Vehicle Reservation Service (VRS).

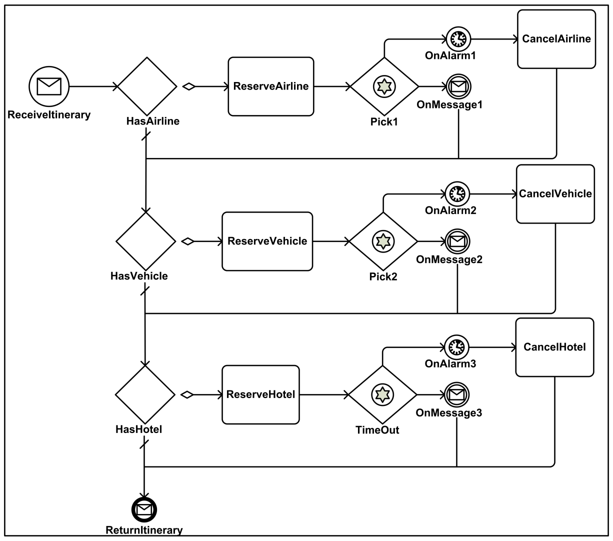

Figure 3. Behavior of the TRS described in BPMN

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 add the 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.

The behavior of the TRS and the communications with the partner Web services are described using the Business Process Modeling Notation (BPMN) [13] notation in Figure 3.

B. *Experimentations and results*

Several tools have been applied on the TRS in order to check its conformance and detect potential errors or misconfigurations in its implementation. A formal specification of the TRS Web service has been derived from its BPEL and WSDL files using BPEL2IF tool. The generated specification has 31 processes, 109 states and 135 transitions (almost 3 states per IF process). These metrics demonstrate that we are dealing with a large system composed of many processes dealing with small tasks. This IF specification is used as a first input to TestGen-IF tool in order to automatically generate a set of test cases.

Similarly, and starting from the TRS requirements [28], 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 airline reservation service partner, then the VRS partner, then the hotel reservation service 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.

These test objectives have been formalized and used in different testing tools in order to automatically generate test cases. The following table presents the number of generated test cases and their execution results. Notice that the number of test cases changes from a tool to another depending on the length of each test case (a test case can cover several test purposes in the same time) and on the data used in each test case (we can have many test cases for the test scenario using different values for the same variable). Let us note that the implementation that has been tested is the BPEL provided by netbeans and the formal models have been produced from the BPEL generated from the logical architecture, i.e. by the Modelio tool.

Table I
ACTIVE TESTING RESULTS OF TRS.

Tool	Test Cases	Pass	Fail
TestGen-IF	11	10	1
WSOTF	20	20	0
WS-AT	200	200	0

Notice that the fail verdict obtained for TestGen-IF tool is due to the application of the test cases to a mutant of TRS Web service where the messages correlation were removed. Indeed, we intentionally introduced an error in the Web service and the execution (using ActiveBPEL engine) of test cases generated using TestGen-IF tool was able to detect the inserted error. As future work, we intend to generate more mutants of TRS Web service and test them using different tool in the WebMov integrated framework.

The passive testing technique has been also applied on the TRS Web service using TIPS tool. Twelve invariants have been designed based on the TRS test objectives. The traces of the composed Web service have been captured running the implementation as a black-box through the PO installed in the TRS server to capture the communications with the external systems (HRS, VRS and ARS). Wireshark was used to save the traces in XML format. The main advantage of using the selected PO is that it allows observing all exchanges between the BPEL engine and its Web partners.

Applying TIPS to the traces was rapid (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 be sure to cover all the expected Web service scenarios. Otherwise, a simulation of these scenarios must proceed. For instance, 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.

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. And to prove the efficiency of TIPS 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 by the tester was FAIL: the violation of the invariant has been detected (and provides the packet characteristics that violate the invariant) which indicates the correctness of TIPS.

We also applied fault injection to the TRS application. For the moment, the fault model applied was SOAP message corruption. Faults were injected in integer fields of messages between the BPEL client and its partner services. In other words, integer fields, like dates, for example, were substituted by wrong values. In all cases the BPEL process had a normal end. This indicates a lack of robustness, since the application accepts any values as valid. For example, dates like Null/Maxint/Minint were accepted as valid reservation dates.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper we presented the research work and the experimental results of the project WebMov, whose main objective is to study different methodologies and to develop a set of tools covering all the phases of the development and validation cycle of Web Services composition. As presented in the paper, different formal techniques have been developed for web services modelling, in particular, those based on different variants of Timed Extended Finite State Machines. This allowed taking into account timed features, which are crucial for characterising web service behaviour. Another contribution is the design and development of different test generation and passive testing techniques. These techniques have been combined with fault injection in order to perform conformance and robustness testing. The proposed techniques for modelling and test have been integrated in a dedicated framework, in which the tools implement these methods and algorithms allowing the interaction between different tools to perform specific modelling or testing activities. All the techniques have being applied to several case studies. In this paper, we presented one of the most representatives, the Travel Reservation Service (TRS), which is a real-life service provided in Netbeans IDE 6.5.1 platform [27]. The experimentation results show that these techniques are complementary and contribute to increase the detection of faults and robustness of web service compositions. As future work, we plan to continue our work on service composition and testing, in particular we plan to work on the design of new composition algorithms in order to take into account the evolution of services. This implies enhancing the formal service description to include user requests, information provided by the context, as well as the evolution of service composition environments.

Acknowledgment

We would like to thank the members of the project that have contributed to its success. We thank A. Abherve, L. Bentakouk, F. Bessayah, M. Lallali, E. Montes de Oca, G. Morales, I. Rabhi.

References

[1] P. Mayer, "Design and implementation of a framework for testing bpel compositions," Ph.D. dissertation, Leibnitz University, Germany, 2006.

[2] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "Towards Automated WSDL-Based Testing of Web Services," in *Proc. of ICSOC*, ser. LNCS, vol. 5364, 2008.

[3] C. Bartolini, A. Bertolino, E. Marchetti, and I. Parissis, *Architecting Dependable Systems*, ser. LNCS, 2008, vol. 5135, ch. Data Flow-Based Validation of Web Services Compositions: Perspective and Examples.

[4] L. Mei, W. Chan, and T. Tse, "Data Flow Testing of Service-Oriented Workflow Applications," in *Proc. of ICSE*, 2008.

[5] Z. Li, W. Sun, B. Jiang, and X. Zhang, "BPEL4WS Unit Testing: Framework and Implementation," in *Proc. of ICWS*, 2005.

[6] Y. Zheng, J. Zhou, and P. Krause, "An Automatic Test Case Generation Framework for Web Services," *Journal of Software*, vol. 2, no. 3, pp. 64–77, 2007.

[7] J. García-Fanjul, J. Tuya, and C. de la Riva, "Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN," in *Proc. of WS-MaTe*, 2006.

[8] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," in *SFM*, ser. Lecture Notes in Computer Science, M. Bernardo and F. Corradini, Eds., vol. 3185. Springer, 2004, pp. 237–267.

[9] SOFTEAM, *Pratical Guide to the Logical Architecture*. [Online]. Available: http://www.modeliosoft.com/component/docman/doc_download/3-entreprise\-architecture\-pratical\-guide\-to\-logical\-architecture.html

[10] "The open group architecture framework," http://www.togaf.org.

[11] "PRAXEME methodology," http://www.praxeme.org.

[12] OMG, *Unified Modeling Language*, http://www.uml.org.

[13] OASIS, "http://www.bpmn.org/," 2009.

[14] W. Mallouli, M. Lallali, G. Morales, and A. R. Cavalli, "Modeling and Testing Secure Web-Based Systems: Application to an Industrial Case Study," in *Proc. of SITIS*, Bali, Indonesia, November 30 - December 03 2008.

[15] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[16] M. Lallali, F. Zaidi, A. Cavalli, and I. Hwang, "Automatic Timed Test Case Generation for Web Services Composition," in *Proc. of ECOWS*. Dublin: IEEE, 2008, pp. 53–62.

[17] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zadi, "Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services," in *Proc. of FORTE*, Beijing, China, october 1999, pp. 41–56.

[18] T.-D. Cao, P. Flix, R. Castanet, and I. Berrada, "Testing web services composition using the TGSE tool," *Services, IEEE Congress on*, vol. 0, pp. 187–194, 2009.

[19] P. Poizat, L. Bentakouk, and F. Zaïdi, "A Formal Framework for Service Orchestration Testing based on Symbolic Transition Systems," in *Proc. of TESTCOM*, 2009.

[20] M. Mikucionis, K. G. Larsen, and B. Nielsen, "Online on-the-fly testing of real-time systems," Basic Research In Computer Science, iesd, rs RS-03-49, dec 2003, 14pp.

[21] T.-D. Cao, P. Flix, R. Castanet, and I. Berrada, "Online testing framework for web services," in *Proc. of ICST*. IEEE Computer Society, April 6-9, 2010, to appear.

[22] S. Salva and I. Rabhi, "Statefull web service robustness," in *Proc. of ICIW*. IEEE Computer Society, may 2010.

[23] N. P. Kropp, P. J. Koopman, and D. P. Siewiorek, "Automated robustness testing of off-the-shelf software components," in *Proc. of FTCS '98*. Washington, DC, USA: IEEE Computer Society, 1998, p. 230.

[24] E. Bayse, A. Cavalli, M. Nunez, and F. Zaïdi, "A passive Testing Approach Based on Invariants: Application to the WAP," *Comput. Netw. ISDN Syst.*, vol. 48, no. 2, pp. 247–266, 2005.

[25] 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 *Proc. of DS-RT*. Washington, DC, USA: IEEE, 2008, pp. 315–318.

[26] A. C. F. Bessayah, "A fault injection tool for testing web service compositions," *Technical Report ISPN:10005-LOR, IT/Telecom Sudparis, France.*, 2010.

[27] NetBeans Framework, "http://www.netbeans.org/," 2009.

[28] W. P. Consortium, "D5.1 webmov case studies: definition of functional requirements and test purposes," WebMov, Tech. Rep., 2009.