Modeling and Testing of Web Based Systems

Ana Rosa Cavalli, Mounir Lallali, Stephane Maag, Gerardo Morales, Fatiha Zaidi

Modeling and Testing of Web Based Systems

Ana Cavalli, Mounir Lallali, Stephane Maag, Gerardo Morales, and Fatiha Zaidi

Abstract The success and the massive adoption of Web applications and services are pushing the community to increase and enhance their developments. By that way, the complexity and size of Web based systems are definitely growing. Accordingly, the need for sophisticated and complete methods used to test the reliability and security aspects of Web systems is increasing as well. Quality and relevant test cases development can achieve up to 70% of the total cost of the project when these test cases are hand crafted. Because of this, the industry and the research community are making big efforts to automate test cases generation. That is the reason why the test generator must be supplied with a precise and unambiguous semantic description of the implementation under test (IUT), i.e. a formal model. This chapter presents two methodologies to attain automatic test cases generation: The first one applies extended finite state machines to model Web services composition described in WS-BPEL, while the other one uses UML to model Web applications. Together with the formal models of the web systems, this chapter presents methods for conformance and non-regression test generation.

Ana Cavalli, Mounir Lallali, Stephane Maag and Gerardo Morales
Telecom & Management SudParis - SAMOVAR CNRS UMR 5157, 9 rue Charles Fourrier, F-91011 Evry Cedex.
e-mail: {Ana.Cavalli,Mounir.Lallali,Stephane.Maag,Gerardo.Morales}@it-sudparis.eu

Fatiha Zaidi
LRI ; Univ. Paris-Sud, CNRS
e-mail: Fatiha.Zaidi@lri.fr

Research supported in part by the French National Agency of Research within the WebMov project: http://webmov.lri.fr

1 Introduction

Web-based systems (services and applications) are gaining industry-wide acceptance and usage. They are very popular because they offer complete interoperability between systems. Using Web-based systems is relatively easy and inexpensive for companies and institutions. It makes it easier for them to share their expertise, cooperate by outsourcing tasks between them and making their incompatible software systems interoperable. It hardly matters what operative system, database or other characteristics their systems rely on. The idea behind Web-based systems was to create an interface that describes what a system does. Every operation with its input to the system and output obtained from the system is described. Another system can access the Web-based system, viewing it as a black box, by using internet as the channel of communication. As the Web-based system implementations grow in size and complexity, the necessity for testing their reliability and level of security are becoming more and more crucial.

Besides knowing whether a given Web-based system satisfies its functional requirements, it is also important for a user to know whether the system behaves appropriately when interoperating with others. Some of the questions that need be answered are: (i) does the host of a selected Web-based system behave correctly without any implementation error? (ii) does the Web-based system behave correctly when new services are integrated to the network? (iii) does the selected Web-based system tolerate erroneous behavior of other services with whom it must interoperate? (iv) does the selected Web-based system produce outputs that can cause other components to fail?

Frequently we can find hand crafted methodologies for conformance testing. Although in most of the cases the execution of the test cases is automated, the biggest part of the test cases generation process is hand made. Under these conditions, the production of quality and relevant test cases can be expensive compared with the cost of the global project (30% to 70%). Because of this, the industry and the research community have been working and making efforts to automate test cases generation. So that a machine can generate the test cases and the oracle to assign the verdict for the tests, one needs to feed it with a description of the implementation under test (IUT), as precise and less ambiguous as possible; i.e. a formal description with a precise semantic.

The main contribution of this chapter is the presentation of a methodology to model and test Web-based systems. For the modeling of Web-based systems, we have chosen UML (Unified Modeling Language) [35] to specify web applications. The Web Services Business Process Execution Language (WS-BPEL) [34] has been chosen to specify services. This last language is well adapted for service composition description. For testing purposes, we propose a model-based approach that relies on formal description languages. WS-BPEL descriptions are translated into another formalism, the Time Extended Finite State Machines for Web Services (WS-TEFSM), which is well adapted for the modeling and the testing of Web-based systems. Once the formal model has been designed (in WS-TEFSM or UML), based

on formal testing relations and on a fault-model, we provide test methods and its associated algorithms to generate the test cases. Afterwards, the test cases can be automatically or manually executed on the real implementation, i.e. the deployed Web-based system.

The tests which are presented in this chapter, are conformance and non-regression tests. By conformance testing we mean the assessment that a product conforms to its specification. Test cases are designed to test particular aspects of the Web-based system, which are called test purposes. Non-regression testing consists of testing modified software to detect whether new errors have been introduced by the modifications, and provides confidence that the modifications do not change the system behavior.

The proposed methodology is composed of two approaches, one based on WS-TEFSMs models and the other on UML models. The methodology based on the UML models has been applied to a real case study, an open source e-learning platform dotLRN [13]. Then, test generation methods have been applied to them. Numerous experiments have been performed for conformance and non regression testing, based on automatic test generation but also with hand crafted tests using TCLwebtests.

In this chapter, we do not address the Web semantic. Our work is related to testing and in particular to automation of testing Web-based systems to establish their correctness with respect to their specifications. To perform the conformance and non regressions testing, it exists languages and standards of reference to describe the Web-based systems and from which we generate the tests and execute them on the real implementation. The languages that are used in that chapter are a subset of UML and WS-BPEL. The languages that are used in the domain of semantic Web are very specific and address particular objectives that are not those addressed by this chapter. Nevertheless, we give below some references and information about the semantic Web that can help to understand what this domain covers and to establish the differences with the work performed here.

The semantic Web is the abstract representation of data on the World Wide Web to make it easily processed by machines, offering more effective discovery, automation, integration, and reuse among various applications [54]. It is different from the Web that is essentially syntactic. It aims to facilitate the communication human-machine and machine-to-machine and the automatic data processing. The semantic Web is based on some standards Ontology is the core of the Semantic Web. It consists of a set of concepts, axioms, and relationships that describe a domain of interest (e.g. system model, data model, etc.). Ontology engineering is supported by primary semantic Web standards and languages (e.g. RDF (Resource Definition Framework [49], Web Ontology Language (OWL) [48]).

The semantic Web and Web services are complementary. The former aims at providing a semantic interoperability of content, while Web services aim at giving a syntactic interoperability of data exchanges. In addition, several information needed by the automation of design and implementation of Web services (like description, publication, discovery, selection, execution, composition, monitoring, replacement,

compensation, etc..) are either absent or described to be only used or interpreted by humans. Such information can be provided by the semantic Web in a way that machines can understand and process it.

Several semantic types can be considered in the Web services domain [5]: (i) functional semantics: (the semantics of service signature (inputs/outputs)); (ii) data semantics (annotation of data involved in Web service operation using ontologies); quality of service semantics (the semantics of different quality aspects, e.g. deadlines, cost of service); (iv) execution semantics (these semantics concern message sequence, flows, effects of service invocation, etc); (v) domain semantics: the use of domain-specific semantics can ameliorate the discovery and selection of services.

Some approaches [5] have been developed to introduce semantics to Web services by using: WSDL-S (Web Service Description Language with Semantics) [52], OWL-S language (Ontology Web Language for Services) [51], WSMO (Web Services Modeling Ontology) [53].

As stated before, the semantic Web and our work address different objectives. Indeed, Semantic Web service technologies are developed in order to give richer semantics to services leading to the automation of the Web Service usage process. Nevertheless, there exist some works that try to apply testing techniques to semantic Web services as in [39, 59], and especially in automatic composition of semantic Web Services [43, 60].

The chapter is organized as follows: in section 2, basic concepts of Web-based systems and testing techniques are presented. This section also includes the presentation of related work. Section 3 presents model-based testing techniques starting with a presentation of the modeling languages WS-BPEL, WS-TEFSMs and UML, used to describe the expected requirements or properties of the Web-based systems. In section 4, Web services composition mechanisms and the test generation algorithms to generate tests from WS-TEFSMs and UML models are presented. Section 5 presents the case study, the dotLRN system, and the test generation from an UML model of the system by the application of the methods described in Section 4. Finally, we conclude the chapter and illustrate some perspectives in section 6.

2 Preliminaries

This preliminary section is devoted first to the presentation of basic concepts related to the Web-based systems. Then, the testing vocabulary is detailed especially in focusing on the formal aspects. Finally we present the related works in that domain in order to illustrate the contribution of the Web-based community.

2.1 Definitions

According to W3C definition, a Web service provides a standard means of interoperating between different software applications. Web services deal with making heterogeneous applications interoperate. A standardized way of integrating Web-based applications is ensured by the XML (eXtensible Markup Language) [47], SOAP (Simple Object Access Protocol) [55], WSDL (Web Service Description Language) [7] and UDDI (Universal Description, Discovery and Integration) [33] that are open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is the protocol used to transfer the data, WSDL describes the available services and UDDI is a kind of registry.

Used primarily as a way for businesses to communicate with each other and with clients, Web services allow organizations to communicate data without intimate knowledge of each other system behind the firewall. The combination of services—internal and external to an organization—makes up a service-oriented architecture. The composition of Web services became this last decade a subject of interest as well as for researchers then for industrials. Several proposals of languages to program and/or to specify Web services composition came up and among them, as previously mentioned, the WS-BPEL language. This latter is well tailored to address the composition of Web services and it is also widely used [32]. The composition is also commonly known as *Orchestration* and this is carried on at a single partner level.

While Web services are dedicated to the interactions between different programs, Web applications are commonly defined as a collection of logically connected Web pages managed as a single entity reachable via a Web browser over a network such as the Internet or an Intranet [50]. A Web applications is like a software application implemented in a browser-supported language (such as HTML, JavaScript, Java, etc.) executed through a Web browser.

We can notice that both types of systems, i.e. Web service and Web application rely on their access and availability through Internet and on specific dedicated languages. In our case, we consider the composition of Web services in order to produce more complex ones. The language to describe the workflow of the composition is the WS-BPEL. The Web application language is the HTML and for the modeling we use UML diagrams.

2.2 Testing Techniques

After the system has been implemented, the implementation must be verified to conform to its specification, to ensure that the system will operate correctly. This procedure is known as conformance testing, and can be accomplished by applying a sequence of inputs to the implementation, by means of an external tester, and by verifying if the sequence of outputs is the one specified.

If a test sequence is capable of detecting all erroneous implementations, it is said to provide *full* fault coverage. There are many methods for generating automatically a test sequence to check a given implementation against a specification. Most of the methods used in this chapter have been developed in the framework of protocol engineering, since protocols are normally programs for which precise specifications can be defined. However, these techniques can be adapted and applied successfully to the test of Web services and applications.

2.2.1 Basic concepts

A program specification is typically composed by a control part and a data part. This chapter deals with the control part only; other approaches are oriented to the analysis of control and data dependencies [21]. The control part of a program, which will be referred as program specification, can be modeled as a Finite State Machine (FSM) with a finite set of states $S = \{s_1, s_2, \cdots, s_n\}$, a finite set of inputs $I = \{a_1, a_2, \cdots, a_k\}$, and a finite set of outputs $O = \{x_1, x_2, \cdots, x_m\}$. The next state (σ) and output (φ) are given by a set of mappings $\sigma : S \times I \rightarrow S$ and $\varphi : S \times I \rightarrow O$. The FSM is usually also represented by a direct graph $G = (V, E)$, where the set $V = \{v_1, v_2, \cdots, v_n\}$ of vertices represents the set of states S, and a directed edge represents a transition from one state to another in the FSM. Each edge in G is labeled by an input a_r and a corresponding output x_q. An edge in E from v_i to v_j which has label a_r/x_q means that the FSM, in state s_i, upon receiving input a_r produces output x_q and moves to state s_j. A triplet $(s_i, a_r/x_q, s_j)$ is used in the text to denote a transition.

A FSM is said to be *fully specified* is from each state it has a transition for every input symbol, otherwise the FSM is said to be *partially specified*. If a FSM is partially specified and a non specified input is applied, under the *Completeness Assumption* the FSM will either stay in the same state without any output or signal an error. The initial state of a FSM is the state the FSM enters immediately after power-up.

State s_i is said to be weakly equivalent to state s_j if any specified $input/output$ sequence for s_i is also specified for s_j. If two states are weakly equivalent to each other they are said to be strongly equivalent. A FSM is *deterministic*, if for each state $s_i \in S$, with two associated transitions $(s_i, a_r/x_q, s_j)$ and $(s_i, a_w/x_p, s_k)$ where $a_r \neq a_w$ and $s_j \neq s_k$.

A graph representation of a FSM is depicted in Fig. 1. For the FSM represented, $I = \{a, b\}$ and $O = \{x, y\}$.

2.2.2 Conformance

Since the implementation is tested as a black box (meaning that we do not have any internal views of the system), the strongest conformance relation that can be tested is *trace equivalence*: two FSMs are trace equivalent if the two cannot be distinguished by any sequence of inputs. That is, both implementation and specification

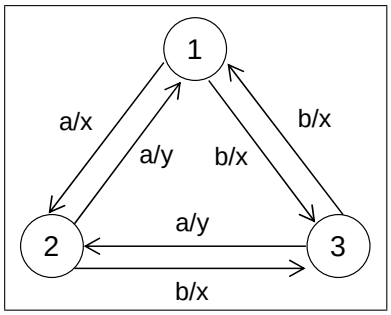

Fig. 1 A Graphical Representation of a FSM

will generate the same outputs (i.e. trace) for all specified input sequences. To prove trace equivalence it suffices to show that (i) there is a set of implementation states $\{p_1, p_2, \cdots, p_n\}$ respectively isomorphic to specification states $\{s_1, s_2, \cdots, s_n\}$, and (ii) every transition in the specification has a corresponding isomorphic transition in the implementation. The Figure 2 illustrates the goal of the conformance testing.

Conformance testing consists in making the implementation under test (IUT) to interact with its environment. This environment is simulated by a tester (cf. Fig. 3) that executes the test cases and stimulates the IUT. The interfaces of the tester are called Points of Control and Observation (PCO).

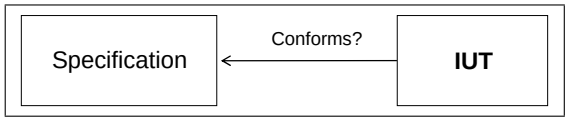

Fig. 2 Conformance Testing Scheme

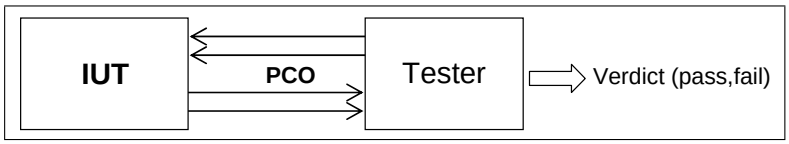

Fig. 3 Role of the Tester

2.2.3 Models of faults

The types of faults detected by methods based on input/output FSMs are output and transfer faults. An output fault occurs when a transition produces an unexpected

output for a given input; that is, a transition specified as $(s_i, a_r/x_q, s_j)$ is implemented as $(s_i, a_r/x_w, s_j)$ where $x_q \neq x_w$. A transfer fault occurs when a transition leaves the implementation in the incorrect state; that is, a transition specified as $(p_i, a_r/x_q, p_j)$ is implemented as $(p_i, a_r/x_q, p_k)$ where p_j (the state that should implements s_j) and p_k are not equivalent. Note that a single transition may incur into an output and transfer fault.

2.2.4 Automatic test Generation

The purpose of test generation is to produce a sequence of inputs (and corresponding outputs), called a test sequence, which can be applied to an implementation to verify that it correctly implements the specification.

There is a number of necessary assumptions that must be made in order to make the experiment possible: (i) the specification FSM is strongly connected, so that all states can be visited; (ii) the specification FSM does not have strongly equivalent states (it is minimal); (iii) there is an upper bound on the number of states in the implementation FSM (otherwise one could always construct a machine which would pass a given test sequence by using as many states as there are transitions in the sequence).

2.3 Related Works

Several techniques to perform testing have been proposed in the literature. We first describe what it exists in the domain of Web applications and we continue with the Web services and particularly with Web services composition. The considered tests are conformance and non regression tests. As mentioned previously, conformance tests establish that the implementation respects its specification. In other words, we interact with the implementation by feeding it with inputs and compare the outputs of the implementation to the expected answers. The relation that exists between the specification and the implementation is an implementation relation. The specification is produced from an informal description of what the system is expected to do, here the specification languages are UML and WS-BPEL. The second kind of test is regression testing, that is the process of validating modified software to detect whether new errors have been introduced into previously tested code, and provide confidence that modifications are correct.

As the use of the Web applications is growing up, the interest on testing these systems and the number of works is increasing as well. Among the recent works it can be quoted [1], work in which the authors survey several analysis modeling methods used in Website verification and testing. In [27], a methodology of Model-Driven Testing for Web application is presented. Moreover, concerning the testing needed to help maintaining the stability of Web applications, numerous works are using the regression testing approach. This testing technique as well as conformance

testing is important to apply to the open source Web applications because of its continuous developing speed and changeable user demands. The work presented in [57] proposes a method based on Slicing to avoid the re-execution of all the regression test cases of a Web application, and selects just the test cases that will interact with the part of the Web application that have suffered a change with the insertion of the new code.

Several tools and methodologies have been developed to achieve an automatic regression testing on Web applications that do not have a formal specification. Among all the tools used to build the regression test cases, the ones that can record the interactions between the user and the Web application during a certain time, or a well defined number of interactions (e.g. to follow a link or to submit a form) are the most popular. Selenium [12], for example is a tool in which the trace of interactions (i.e. the record) is written as HTML tables. However, for our purpose it was needed to obtain a trace flexible enough to be changed and re-used in an easy and fast way, for example a trace written in a scripting language. Tclwebtest [41] is a tool to write tests for Web applications. It provides an API for issuing HTTP requests and processing results. It assumes specific response values, while taking care of the details such as redirects and cookies. It has the basic HTML parsing functionality to provide access to elements of the resulting HTML pages that are needed for testing, mainly links and forms. The execution of a test case written in Tclwebtest will simulate a user that is interacting with the Web application through a Web browser. Using the links and forms it is possible to add, edit or delete data of the Web application by executing the test case script. There exists also a lot of works that have focused their research on functional testing based on a formal model such as [25, 28, 42]. We can also mention some Web application tools (e.g. OpenSTA [37], WAPT [45], SOASTA [44]). The listing of load/performance test tools and Web functional/ regression test is presented in [46].

Regarding Web services, we survey the existing works for Web services composition. As previously stated, in the area of Web services, the design is the most important phase of the orchestration layer. This is where one describes the complete services behavior. The growing use of Web services makes it necessary to ensure that the behavior is correct. The common practice in the area is to generate unit and integration tests with tools such as SOAPUI [14] that are based on empirical approaches. In the last years, the software testing community has started to get involved in the Web services domain. As a consequence, several works have been published to try to bridge the gap between the usual Web testing practices and a formal testing process. The empirical approaches that are still used in the domain give acceptable results but they have become more and more costly and do not allow to cover all the problems raised by such systems. Moreover, when systems become complex, such approaches offer only partial verifications and validations. Consequently, if we want to validate Web systems with an improved test coverage, we need to introduce rigorous methods which nevertheless conform to the economical constraints of the Web services development process. The original nature of Web services requires to first validate them individually and test them when integrated.

Industrial tools exist and are available to perform such kinds of tests. When complex Web services are considered, we also need to deal with the Web services composition or in other words, orchestration.

The current practice deals with the Web Service Description Language WSDL. WSDL is used to describe what a Web service can do, where it resides, and how to invoke it. In our case, we consider WS-BPEL that permits describing the control and data flows; thus, allowing to study the behavioral aspects of the Web service. Several WS-BPEL activities are related to time constraints. So, we need to take into account such constraints for their modeling. Currently, in the literature, we can find several papers dealing with WS-BPEL specification and formal models for Web services. Several models are described to transform the WS-BPEL specifications. In [56] a transformation of WS-BPEL in an annotated deterministic finite state machine (FSM) is proposed. This formalism does not allow to capture the timing aspects of some WS-BPEL activities and does not consider WS-BPEL variables. In [31] another formalism is proposed that deals with variables, the Extended Finite State Machine (EFSM), but no timing constraints are considered. In [23] a formalism taking into account timing constraints is proposed, the Web Service Timed Transition System (WSTTS). However, this formalism uses only clocks but no variables. We can also mention works that use different kinds of formalisms [4, 15, 19, 58]. In [19, 58] Petri nets are used to specify the Web services. This formalism is not well adapted for testing Web services composition because it results in a global description that lacks structure making it impossible to differentiate the service components.

In [4] an interesting approach is presented using RT-UML to model real time aspects of Web service choreographies but this does not consider orchestration and interoperability of services and components. The work presented in [15] is similar to this one: to design a methodology based on WS-BPEL. Formalization of WS-BPEL language semantics is necessary to eliminate ambiguities and make abstract operational specifications executable. Here an Abstract State Machine (ASM) model is used but unlike FSM or WS-TEFSM [26] (Web Services Timed Extended Finite State Machine) models used in this work, it is not directly executable and requires not so evident translations, making it more difficult to develop testing tools. The used Distributed ASM needs also to be extended with timing constraints and variables. Other not yet very advanced approaches exist, such as in [17] where WS-BPEL is translated to Promela which is the input language of the SPIN [20] model checker, allowing test generation. These approaches need to be developed and implemented in tools where scalability and usability can be evaluated.

3 Web-based system modeling

The proposed methodology is based on two main elements: the modeling language and the generation methods. This latter will be detailed in the Section 4. The modeling step provides a precise representation of the system. The description language

used here to describe the composition of Web services is the WS-BPEL language for the reasons that have been mentioned above. In the case of Web applications, the UML notation and in particular some diagrams have been chosen. This notation is well-suited to describe Web applications which are often developed using object concepts. With UML we can represent the navigation by means of a navigation map which provides information about the dynamic content of each Web page. Furthermore, the sequence diagrams that represent the exchange between the elements of the system are good candidate to express the test objectives.

We detailed below the modeling choices of both Web services and Web applications, note that the choices have been made taking into account the adequacy of the language to the system to be described and also for testing purposes. Let us note that both Web-based systems in our methodology need to be formally described for testing generation methods which is one of our problematics. The willing to deal with such an approach is issued from our own experience in industrial projects[1]. We notice that the delays to produce a software or a new release are becoming more and more short. Furthermore in a competitive market the clients request cost reduction in their management and maintenance costs. For all these reasons a more automated technique to generate tests is needed as hand-crafted tests are a time-consuming activity. Hence, as said, a model is the first step and represents the specification of the system inputs and can be handled at a very early stage of the development cycle of a Web-based system, i.e. from the requirement information.

3.1 Modeling of Web Services

To standardize the specification of a Web services composition, IBM and other companies have proposed the WS-BPEL language [34] that became in 2007 an OASIS Standard [32].

WS-BPEL is a coordination and composition language that captures business interactions between Web services. It can also be viewed as a workflow language for Web Services. WS-BPEL provides constructs to describe arbitrarily complex business processes. A WS-BPEL process can interact synchronously or asynchronously with its partners (i.e. client service or services that are invoked by the WS-BPEL process). The building blocks for a WS-BPEL process are the descriptions of the parts participating in the process, in terms of data and activities flow. A basic process in WS-BPEL is defined as a root element, consisting of one or more child elements describing *partners links*, a set of *variables* that records the state of the process, *correlation Sets*, *event handlers*, *fault handlers*, *compensation handlers* and *activities*. This latter defines the logical interactions of a process and its partners. The WS-BPEL activities that can be performed by a business process instance are categorized into basic (e.g. `<wait>`, `<exit>` and `<assign>` activities),communication (e.g. `<receive>`, `<reply>` and `<invoke>` activities) and structured activities (e.g.

[1] PLATONIS (http://www-lor.int-evry.fr/platonis), ASK-IT (http://www.ask-it.org/.) and WebMov (http://webmov.lri.fr)

<sequence>, <flow> and <while> activities). For instance, the <receive> activity waits for a message from a partner, and the <reply> activity sends an answer for a message previously received with a <receive> activity. The <wait> activity waits for a specified amount of time before continuing while the <onAlarm> WS-BPEL construct corresponds to a timer-based alarm.

We present in the next section the timed modeling of the Web Services composition described in WS-BPEL.

3.1.1 WS-BPEL Timed Modeling Approach

A WS-BPEL process component implements a business process (i.e. Web service). The behavior of a Web service is described by sequences of activities, their execution time and their semantic. For instance, the <wait> activity is triggered when the timeout occurs, the <reply> of invocation may be considered as an instantaneous activity, while the <receive> activity may require an arbitrary amount of time. To model this temporal behavior, we propose the WS-TEFSM which extends EFSM with clock variables, state invariants on clocks and priorities on transitions. The time progresses in states, and transitions take no time to be executed. In order to represent the time progress, a local clock is added in the WS-TEFSM. It can be initialized at the beginning of each activity to be executed and can be reset at its end. Moreover, in order to model an absolute time, a global clock, denoted *gc*, is added in the WS-TEFSM. It can be explicitly set to a certain value at the beginning of the WS-BPEL process execution, and is never reset later. In order to control the time progress, we use time invariants in the states.

All WS-BPEL activities are modeled as instantaneous activities except <wait>, <receive>, <empty> (with *duration* attribute) and <onAlarm> that are related explicitly to a time notion. These instantaneous activities are modeled, as instant transitions called *action* transitions. These transitions are equivalent to assign the time invariant $c \leq 0$ (where c is a local clock initialized to *zero*) to the source state of the transition. In that case, the time cannot progress in this source state. Contrarily, the non instantaneous activities are modeled as *delay* transitions. They take certain amount of time represented by time increment in the state and followed by their immediate execution. It is semantically equivalent to add first a clock invariant to the source state of the transition depending on time, and secondly guards on the clock variables of transition guards on WS-BPEL process variables.

For instance, the <wait> activity and the <onAlarm> element are used to represent timeouts. These constructs have two forms. In the first one (with *for* attribute) they are triggered after a duration d. We associate in this case the invariant $c \leq d$ to the source state of the transition. In the second one (with *until* attribute), the transitions are triggered if the current absolute time has the specified value (i.e. deadline dl). We associate the invariant $gc \leq dl$ to the source state of the transition.

A transition priority is used to model interruptions in real-time systems. In this model, we use priority on transitions to model the fault handlers and the termination

of the WS-BPEL process and its sub-activities. Each non atomic activity can be interrupted by adding an urgent transition from each state to a particular *stop* state. All the transitions of each WS-BPEL atomic activity have a highest priority and cannot be blocked or interrupted. A *delay* transition has a lowest priority.

In order to introduce the Timed Extended Finite State Machine with Priorities (WS-TEFSM), we need first to depict the clock valuation concept.

Let V a finite set of data variables. A valuation v over V is a function $v : V \mapsto D_V^{|V|}$ that assigns to each variable $x \in V$ a value in the data variables domain $D_V^{|V|}$. The initial data variables valuation is noted v_0. $v[\overrightarrow{v} := \overrightarrow{x}]$ denotes the data valuation which updates the variables $\overrightarrow{v} = \{v_1, \cdots, v_n\}$ and keeps the rest of variables (i.e. $V \setminus \{v_1, \cdots, v_n\}$) unchanged.

A clock valuation u over the set of clocks C is a function $u : C \mapsto \mathbb{R}_+^{|C|}$ (noted by $u \in \mathbb{R}_+^{|C|}$) that assigns to each clock $c \in C$ a value in $\mathbb{R}_+$. The initial clock valuation u_0 corresponds to the initialization to 0 of all clock variables: $\forall c \in C,\ u_0(c) = 0$.

We introduce in the following the formal definition of the Timed Extended Finite State Machine with Priorities which is used to model the WS-BPEL process.

Definition 1 (WS-TEFSM). A machine WS-TEFSM M is a tuple $M = (Q, \Sigma, V, C, q_0, F, T, Pri, Inv)$ where:

- $Q = q_0, q_1, \cdots, q_n$: a finite set of states;
- $\Sigma = a, b, c, \ldots$: alphabet of the actions including symbols $!m$ (output action) and $?m$ (input action);
- V: finite set of data variables where $\overrightarrow{v} = (v_1, v_2, \cdots, v_m)$;
- C: finite set of clocks where $\overrightarrow{c} = (c_1, c_2, \cdots, c_n)$;
- $q_0 \in Q$: initial state;
- $F \subseteq Q$: finite set of end states;
- $T \subseteq Q \times A \times 2^Q$: transition relation that:
  A: Set of transition actions $\Sigma \times P(V) \wedge \phi(C) \times \mu \times 2^C$ where:

  $P(\overrightarrow{v}) \wedge \phi(\overrightarrow{c})$: guard condition is logical formula on data variables and clocks;
  $\mu(\overrightarrow{v})$: data variables update function;
  2^C: set of clocks to be reset.

- $Pri : T \times D_C^{|C|} \mapsto N_{\geq 0}$ assigns to each transition its priority that respects the clock valuation u;
- $Inv : Q \mapsto \Phi(C)$ assigns a set of time invariants (logical formulas) to the states.

The actions in Σ represent an observable actions. The label $\tau \notin \Sigma$ denotes an *internal* action that is unobservable. We note Σ_τ the set $\Sigma \cup \{\tau\}$.

In the WS-TEFSM model, the states are associated with state invariants that express simple clock conditions. The invariants of a state should be *true* when the

machine is in this state. This machine may remain in a state as long as the clock valuation satisfies the invariant condition of the considered state. We can assign a set of time invariants to one state because it can be the source state of several transitions such as `<if>` and `<pick>` activities. This set associated to a state q is denoted by $Inv(q) = \{e_1, e_2, ...\}$. We will write $u \in Inv(q)$ to denote that the clock valuation u satisfy $e_i \mid \exists\, e_i \in Inv(q)$.

Each transition $t = q_i \xrightarrow{<cond,a,[\overrightarrow{v} := \overrightarrow{x};R]>} q_j$ is annotated with a set of guards on data variables and clocks (e.g. *cond*), actions (e.g. *a*), data variable updates (denoted $\overrightarrow{v} := \overrightarrow{x}$) and a clock set to be reset (e.g. *R*). The transition priority depends of the time and can be dynamically updated with respect to time progress. It assigns a non-negative integer value to each transition priority with respect to a clock valuation. An enabled transition can block another one if it has a higher priority.

The WS-TEFSM semantic are defined as follows to be used in the description of the timed test case generation algorithm (presented in Sect. 4.1.2).

3.1.2 Semantic of WS-TEFSM

The *delay* transition indicates that if the other transitions outgoing from the same source state have a lower priority, then the any action. In other words, the state is not modified, but the machine increments the current value of the clocks d by $u \oplus d$ (that represents a valuation where all clocks have been incremented by the real value d from their value in u). A *delay* transition may affect the priority of other transitions through the function *Pri* and does not block any transition. The priority of the *delay* transition is a constant value *zero*.

The *action* transition having the highest priority, indicates that if the condition *cond* is evaluated to true, then the machine follows the transition by executing the action *a*, changing the current values of the data variables by the action $[\overrightarrow{v} := \overrightarrow{x}]$ (i.e. $v' = v[\overrightarrow{v} := \overrightarrow{x}]$), resetting the subset clocks R (the clock valuation u resets each clock in the set R) and moving to the next state q'. Particularly in the case of the internal action τ, the clocks remain unchanged and consequently $u' = u$.

Let $s, s' \in S$. In the following, we denote the *delay* transition by $s \overset{d}{\Rightarrow} s \oplus d$, and the *action* transition by $s \overset{a}{\Rightarrow} s'$.

Definition 2 (WS-TEFSM Semantic). Let M be a WS-TEFSM $M = (Q, \Sigma, V, C, q_0, F, T, Pri, Inv)$. The WS-TEFSM semantic is defined by a labeled transition system (LTS) $Sem_M = (S, s_0, \Gamma, \Rightarrow)$:

- $S \subseteq Q \times \mathbb{R}_+^{|C|} \times D_V^{|V|}$ is the set of semantic states (q, u, v) where:
  - q is a state of a machine M;
  - u is an assignment (i.e. clock values represented by clock valuation u) that satisfies one invariant of the state q (i.e. $u \in Inv(q)$);

– v is a data values represented by data variable valuation v.

- $s_0 = (q_0, u_0, v_0)$ is the *initial* state;
- $\Gamma = \Sigma_\tau \cup \{d \mid d \in \mathbb{R}_+\}$ is the label set where d corresponds to the elapsed time.
- $\Rightarrow \subseteq S \times \mathbb{R}_+ \times S$ is the transition relation defined by:
  - *action* transition: Let (q,u,v) and (q',u',v') be two states.
    Then $(q,u,v) \overset{a}{\Rightarrow} (q',u',v')$ if $\exists\, t = q \xrightarrow{<cond,a,[\overrightarrow{v}:=\overrightarrow{x};R]>} q' \in T$ such that
    · $u \in cond,\ u' = u[R \mapsto 0],\ u' \in Inv(q'), v' = v[\overrightarrow{v} := \overrightarrow{x}]$
    · $\forall\, t' = q \xrightarrow{<cond',a',[\overrightarrow{v}:=\overrightarrow{x'};R']>} q'' \in T, u \in cond' \Rightarrow$
    $(Pri(t,u) > Pri(t',u)) \vee ((Pri(t,u) = Pri(t',u)) \wedge rand(t,t') = t)$
  - *delay* transition: Then $(q,u,v) \overset{d}{\Rightarrow} (q, u \oplus d, v)$ if
    · $\forall\, 0 \leq d' \leq d,\ u \oplus d' \in Inv(q)$;
    · $\forall\, t = \left(q \xrightarrow{<cond,a,[\overrightarrow{v}:=\overrightarrow{x};R]>} q'\right) \in T,$
    $\forall\, 0 \leq d' \leq d,\ u \oplus d' \in cond \Rightarrow Pri(t, u \oplus d') = 0.$

After defining a WS-TEFSM and its semantic, an overview of the transformation of WS-BPEL into WS-TEFSM and two examples (e.g. activities) are presented.

3.1.3 Overview of WS-BPEL Mapping into WS-TEFSM

A WS-BPEL process always starts with the `<process>` element which contains the workflow definition. It is composed of the following optional children: `<partnerLinks>`, `<partners>`, `<variables>`, `<correlationSets>`, `<faultHandlers>`, `<compensationHandlers>` and `<eventHandlers>`. The execution of sub-activities is carried out in parallel because the fault, the event and the compensation handlers can be carried out independently of the principal `<process>` activity. If this is not the case, these sub-activities are synchronized by the tuple (sending, reception).

A transformation rule is defined for each WS-BPEL construct. The recursive mapping starts by transforming all the WS-BPEL constructs into partial WS-TEFSM. The WS-BPEL process model is based on the asynchronous product of the partial WS-TEFSMs of its sub-activities. This asynchronous product represents the parallel execution of the partial WS-TEFSMs where they are synchronized by the tuple (sending/receiving).

The recursion result is a partial WS-TEFSM *PM* which can be represented as a WS-TEFSM M by renaming the *initial* state q_0 by the *input* state q_{in}, adding the *output* states of Q_{out} to the final states set F and adding the *global* clock gc to the clocks set C. The data variables set V is the union of the transformation of the `<variables>`, `<partners>` and `<partnerLinks>` elements.

$$\begin{aligned} M &= (Q, \Sigma, V, C \cup \{gc\}, q_{in}, F \cup Q_{out}, T, Pri, Inv) \\ PM &= (Q, \Sigma, V, C, q_{in}, Q_{out}, F, T, Pri, Inv) \end{aligned} \tag{1}$$

3.1.4 Examples of WS-BPEL Construct Transformation

The Wait Activity

The `<wait>` activity allows to wait for a given time period (i.e. duration d or until a certain deadline dl). One of the expiration criteria must be specified exactly [34]. The `<wait>` syntax is `<wait for=d>` or `<wait until=dl>`. Let $Pri_w = \{(t_1, \_, l_p)\}$ where l_p is a low priority.

- `<wait for=d>` is modeled as a partial WS-TEFSM PM

$$\begin{aligned} PM &= \{\{q_{in}, q_{out}\}, \emptyset, \emptyset, \{c\}, q_{in}, \{q_{out}\}, \emptyset, \{t_1\}, Pri_w, Inv\} \\ t_1 &= (q_{in}, < c = d, \_, [\_; \{c\}] >, q_{out}) \\ Inv &= \{(q_{in}, c \leq d), (q_{out}, true)\} \end{aligned} \tag{2}$$

- `<wait until=dl>` is modeled as a partial WS-TEFSM PM

$$\begin{aligned} PM &= \{\{q_{in}, q_{out}\}, \emptyset, \emptyset, \emptyset, q_{in}, \{q_{out}\}, \emptyset, \{t_1\}, Pri_w, Inv\} \\ t_1 &= (q_{in}, < gc = dl, \_, \_ >, q_{out}) \\ Inv &= \{(q_{in}, gc \leq dl), (q_{out}, true)\} \end{aligned} \tag{3}$$

The partial WS-TEFSMs of the `<wait for>` and the `<wait until>` activities are illustrated in Fig. 4 and Fig. 5. Note that in these examples, there are not actions.

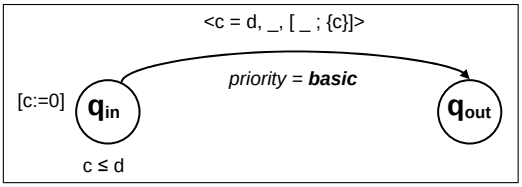

Fig. 4 The partial WS-TEFSM of the `<wait for>` activity

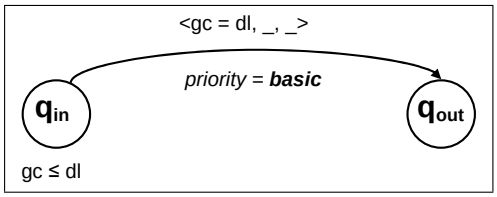

Fig. 5 The partial WS-TEFSM of the `<wait until>` activity

The Wait for Machine with Termination Handling

Each `<wait>`, for instance, can be interrupted and terminated prematurely. To terminate this activity, we add a stop transition t_2 to a `<wait>` machine (described above) which has an urgent priority and can interrupt the transition t_1. Let h_p be a highest priority. *stopProcess* is a global variable of the process that is assigned to *true* by the `<exit>` activity. *stopScope* is a local variable of each scope that is assigned to *true* by the scope `<throw>` activity. These two boolean variables are used to handle the termination. The `<wait>` machine with termination is defined as:

$$\begin{aligned} PM &= \{\{q_{in}, q_{out}, q_{stop}\}, \emptyset, \emptyset, \{c\}, q_{in}, \{q_{out}\}, \emptyset, \{t_1, t_2\}, Pri_w, Inv\} \\ t_1 &= (q_{in}, < c = d, \_, [\_; \{c\}] >, q_{out}) \\ t_2 &= (q_{in}, < stopProcess \vee stopScope, \_, [\_; \{c\}] >, q_{stop}) \\ Pri_w &= \{(t_1, \_, l_p), (t_2, \_, h_p)\} \\ Inv &= \{(q_{in}, c \leq d), (q_{out}, true), (q_{stop}, true)\} \end{aligned} \tag{4}$$

The partial WS-TEFSM of the `<wait for>` with termination is given in Fig. 6.

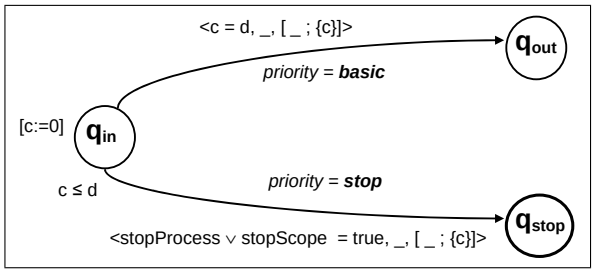

Fig. 6 The partial WS-TEFSM of the `<wait for>` activity with Termination

We can transform all the WS-BPEL constructs (elements and activities) in a similar way. While we have detailed how to model Web services, we tackle in the following the Web applications modeling.

3.2 Modeling of Web Applications

This methodology describes how UML diagrams developed in the analysis phase are used to automatically produce test cases. We will show how to check the graphical user interfaces but also the generated pages content.

3.2.1 UML Overview

The UML language is a standardized visual specification language for object modeling [35]; It is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system. Using UML, no one diagram can capture the different elements of a system in its entirety. Hence, UML 2.0 is made up of thirteen diagrams that can be used to model a system at different points of time in the software life cycle of a system. Nevertheless, in the methodology presented in this chapter only four UML diagrams, which constitute the input data, are needed to be applied for the Web application modeling and testing. The UML diagrams used in this approach are fully described in the next subsection.

3.2.2 Using UML to model Web applications

In order to derive the test cases, it is needed to grasp and describe the system functionality at least in a semi-formal way. A model-based approach of Web applications can be performed using UML techniques and UML notation. A design methodology based on a UML extension for Hypermedia [8] is used. It consists of three main steps that constitute the conceptual model, the navigation map, and the presentation model. The conceptual model is built taking into account the functional requirements captured with use cases [9]. The output of this step is not directly used as an input to the test suite but it is important in order to design the UML diagrams which actually constitute the input data.

From this conceptual model, the navigation space model is constructed, also represented as a static class model. It defines a view on the conceptual model showing which classes may be visited through navigation in the Web application. Finally, a dynamic presentation model is represented by UML sequence diagrams describing the collaborations and behaviors of the navigational objects and access primitives. In order to specify our model, the following diagrams are involved:

- Class Diagram to introduce the main classes of the system;
- Activity Diagram for each actor to display dependencies among the use cases;
- Navigation Map to provide information about the dynamic content of the Web pages;
- Sequence Diagram for each use case describing the main and the alternative scenarios of the use case to represent the dynamic presentation model.

These diagrams are exported in an XMI format (XML Metadata Interchange format) [36] , i.e. an OMG standard for exchanging metadata information via XML. This activity is supported by all the modern CASE tools like ArgoUML [10] and Rational Rose [22]. Afterwards the XMI is parsed and it is produced a program which connects to the Web server and makes requests according to the given scenario in the Sequence Diagram. Finally, the response Web page is examined to verify if it conforms to the specification. Figure 7 presents briefly the different steps of our study.

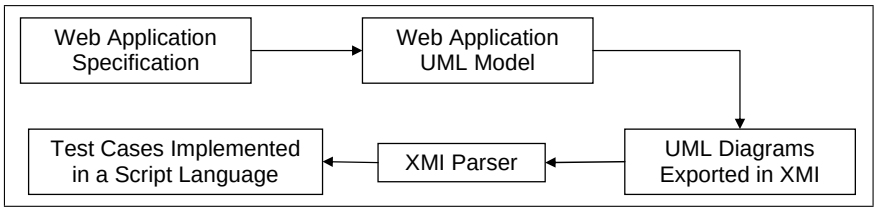

Fig. 7 Representation of the methodology

3.2.3 From Conceptual Model to Navigation Map

In order to specify a Web application, first it is needed to build a conceptual model of the application domain taking into account the functional requirements captured with use cases. Techniques such as finding classes and associations, and defining inheritance structures are performed.

The Navigation Map of a Web application introduced in [3] is used because it provides information about the dynamic content of each Web page which is part of the system as well as the links between the different Web pages. This information is essential during the parsing of the HTML pages. Navigation Map is a Class diagram where each Web page is a class and a link between two pages is an association between the two respective classes. This extension of UML for Web applications introduces a number of tagged values, constraints and stereotypes (such as indexes, guided tours, queries and menus).

3.2.4 Modeling Use Case Dependencies

The use cases of a system are not independent. Apart from *include* and *extend* relationships, there are also sequential dependencies. In order to execute one use case, another should have taken place before. For instance, in any Web application such as an e-mail Web interface or e-learning platform, the user should login before being able to do anything else. Since testing automation procedure is also concerned, it is mandatory to describe somehow these dependencies. This is achieved by introducing an activity diagram where the vertices represent use cases and edges are sequential dependencies between the use cases.

An edge in such a diagram denotes that the use case in the tail has to be executed before the use case in the head. *Fork* and *join* are used when some use cases should be executed independently in order another one to take place. For instance, in an e-learning platform the addition of the subject and the addition of the term are two independent use cases which should be synchronized to allow the testing of the creation of a class.

3.2.5 Sequence Diagram

In UML, a Sequence diagram realizes the interaction of objects via the interchange of messages in time. Similarly, as in activity diagrams the objects are instances of a class described in the Class diagram. Usually the sequence diagrams describe a single scenario. The messages are enumerated in a way that allows to illustrate a number of alternative scenarios in the same diagram. According to this convention, capital letters denote alternatives (error messages). By adopting this tactic it is possible to derive easily the different Message Sequences [2] related to the same use case.

Now that the modeling of Web-based systems has been presented, that is: a proposed WS-TEFSM model for a Web services composition described in WS-BPEL, an overview of the WS-BPEL transformation into WS-TEFSM illustrated by an example and a methodology to model Web applications by using UML, we may present in the following techniques to test these Web-based systems from the formal models.

4 Web-based System Testing

As previously mentioned, the main goal of Web-based systems modeling is to ease their testing. Therefore, this section presents testing techniques devoted to Web services and Web applications. The first method generates timed test cases and uses a timed-traces equivalence as conformance relation between a Web service implementation and its formal specification. The second method illustrates how to generate a test case for a Web application from the obtained UML diagrams.

4.1 Web Services Testing

To define a Web services composition testing methodology, a deterministic WS-TEFSM that models a composite Web service described in WS-BPEL and its semantic (see Sect. 3.1) are considered. First, the concept of timed sequences, timed traces, timed test cases and the conformance relation [16, 24] (between the specification and the implementation) to be used in this testing methodology are defined. Secondly, the timed test cases generation algorithm is detailed. Finally, an example of Web service modeling and testing is detailed.

4.1.1 Conformance Relation and Timed Test Cases

Let $M = (Q, \Sigma, V, C, q_0, F, T, Pri, Inv)$ be a WS-TEFSM and $[M] = (S, s_0, \Gamma, \Rightarrow)$ be the LTS describing the M semantic. Let $s, s_0, s_1, \ldots, s_n \in S$. Let $Seq(\Sigma) = (\Sigma \cup \mathbb{R})^*$

be the set of all finite timed sequences over Σ. A timed sequence $\sigma \in Seq(\Sigma)$ is composed of actions a and non-negative real d where: $s \stackrel{a}{\Rightarrow} s'$ and $s \stackrel{d}{\Rightarrow} s \oplus d$. $\sigma_\varepsilon \in Seq(M)$ is the `<empty>` sequence.

Let $\Sigma' \subseteq \Sigma$ and $\sigma \in Seq(\Sigma)$ a timed sequence. $\pi_{\Sigma'}(\sigma)$ denotes the projection of σ to Σ' obtained by deleting in σ all actions not present in Σ'. $Time(\sigma)$ denotes the sum of all delays in a sequence σ.

Let $\sigma = \sigma_1.\sigma_2 \cdots \sigma_n$ are a timed sequence, $s \stackrel{\sigma}{\Rightarrow}$ is used to denote that there exists s_n and $s \stackrel{\sigma_1}{\Rightarrow} s_1 \stackrel{\sigma_2}{\Rightarrow} s_2 \cdots \stackrel{\sigma_n}{\Rightarrow} s_n$. The observable timed traces of a WS-TEFSM M is defined by:

$$Tr(M) = \{\pi_\Sigma(\sigma) \mid \sigma \in Seq(\Sigma_\tau) \wedge s_0 \stackrel{\sigma}{\Rightarrow}\} \tag{5}$$

Let M_S and M_I two WS-TEFSMs which model respectively the specification of a composite Web service (a WS-BPEL description) and its implementation (a WS-BPEL process instance). *timed-traces equivalence* noted $\cong_{Tr}$ is considered as a *conformance* relation where the time delays is considered to be observable actions. First, the *timed-traces inclusion* relation noted $\preccurlyeq_{Tr}$ is defined. $M_I \preccurlyeq_{Tr} M_S$ requires each observable sequence of M_S to be an observable sequence of M_I:

$$\big(M_I \preccurlyeq_{Tr} M_S\big) \Leftrightarrow \big(Tr(M_S) \subseteq Tr(M_I)\big) \tag{6}$$

The $\preccurlyeq_{Tr}$ conformance relation (timed-trace inclusion) can be extended to $\cong_{Tr}$. This latter requires that each observable sequence of M_I is also an observable sequence of M_S. M_S *conforms* M_I, denoted $M_S \cong_{Tr} M_I$ if $Tr(M_S) = Tr(M_I)$:

$$\big(M_I \cong_{Tr} M_S\big) \Leftrightarrow \big(Tr(M_S) \preccurlyeq_{Tr} Tr(M_I) \wedge Tr(M_I) \preccurlyeq_{Tr} Tr(M_S)\big) \tag{7}$$

A timed test case is a timed trace that validates some timed requirements (e.g. timed test purposes) and generates a *pass* or *fail* verdict: the *pass* verdict if all test purposes are satisfied, the *fail* verdict else if.

Based on the WS-TEFSM semantic, a set of timed test purposes and the Hit-or-Jump exploration strategy [6]—the timed test case generation algorithm—is detailed in the following.

4.1.2 Timed Test Case Generation Algorithm

This algorithm generates timed test cases from a composite Web service specification given in WS-TEFSM and timed test purposes. For test case generation, the Hit-or-Jump strategy (a generalization of the exhaustive search technique and random walks) is adapted to WS-TEFSM model. The generated timed test cases are used to check the conformance of a composite Web service implementation to its

specification. The test cases are a sequence of observable actions which have to be executed according to some time constraints.

Starting from the initial state of the WS-TEFSM and considering a search *depth limit* and a set of timed test purposes to be satisfied (represented by events or timed constraints), a partial search is conducted from the current state s_i of the reachability graph until:

A Hit step. Reached a state s_j (a Hit state) where one or more test purposes are satisfied. Then the sequence from s_i to s_j is concatenated to the test sequence, the test purposes set is updated and the Hit step is repeated from s_j.

A Jump step. Reached a search *depth limit* without satisfying any test purpose. Then one leaf node (i.e. a state s_j) of the partial search tree is selected, the sequence from s_i to s_j is concatenated to the test sequence, the state s_j is moved (a Jump state) and the Hit step is repeated from s_j.

The algorithm terminates when all the test purposes are satisfied or when there are no more transitions to explore. The main interest of this algorithm is that the construction of the complete WS-TEFSM reachability graph is not required. The test case generation algorithm is illustrated in Fig. 8.

4.1.3 Example of Web Service Modeling and Testing

In this section, The PICK Web service is applied to illustrate the WS-TEFSM resulting of the transformation of a WS-BPEL description, and the test cases generating from this WS-TEFSM by using the test generation algorithm detailed in the previous section.

The PICK process receives a loan application document from the user (e.g. the LOAN service). It invokes the asynchronous loan service (i.e. the ASYNCBPELSERVICE service) by sending this document and uses a BPEL `<pick>` activity to receive an asynchronous response from the partner service or to exit after a timeout (e.g. 30 seconds). This partner service sets the credit rating according the loan amount and returns the loan application document to the PICK service. If the loan amount is greater than 10000, it takes about 30 seconds for the partner service to process it and therefore a timeout will be raised. Finally, the PICK service sends the loan application document to its user.

The PICK Web service is illustrated in Fig. 9 (in BPMN Notation [18]). Its WS-BPEL description and the resulting WS-TEFSM (according to the methodology described in Sect. 3.1) are respectively given in Appendix.

We want to test the two `<pick>` activity branches (see Fig. 9). We define, for instance, two test scenarios as following:

INITIAL CONDITION:

- The system is in an initial state $s_0 = (q_0, u_0, v_0)$;
- The Set of timed test purposes to be satisfied is $TP = \{tp_1, tp_2, \ldots, tp_m\}$.
- The timed test sequence *seq* is empty (i.e. $seq = \sigma_\varepsilon$).

TERMINATION:

The algorithm terminates when all the timed test purposes are satisfied, i.e. $TP = \emptyset$;

EXECUTION:

Repeat

① **Hit**: From the current system state s_i, conduct a search by exploring all possible transitions: *action* transition $s_i \stackrel{a}{\Rightarrow} s_{i+1}$ or *delay* transition $s_i \stackrel{d}{\Rightarrow} s_i \oplus d$ until ⓐ or ⓑ :

ⓐ Reach a state s_j such that $s_i \stackrel{\sigma}{\Rightarrow} s_j$ and
forall k such that $s_j \models tp_k : \ TP = TP \setminus \{tp_k\}$ — a Hit. **Then**:
(i) Concatenate the sequence σ from s_i to s_j to the test sequence: $seq = seq.\sigma$;
(ii) Move to ①.

ⓑ Reach a search *depth limit*. **Then** move to ②.

Until ($TP = \emptyset \vee$ "no transition to explore").

IF $TP = \emptyset$ **Then return** *seq* **else** "no test sequence!".

② **Jump**:

(i) A partial searched tree has been constructed, rooted at s_i;
(ii) Examine all the tree leaf nodes, and select one (s_j such that $s_i \stackrel{\sigma}{\Rightarrow} s_j$) uniformly and randomly;
(iii) Concatenate the sequence σ from s_i to s_j to the test sequence: $seq = seq.\sigma$;
(iv) Arrive at s_j — a Jump.
(v) Move to ①.

Fig. 8 Timed Test Case Generation Algorithm

Scenario 1. Receive a ASYNCBPELSERVICE response after 10 seconds and reply this response to the user.

Scenario 2. Exit the BPEL process after waiting a ASYNCBPELSERVICE response 30 seconds.

For the two scenarios, we present below the test purposes used in the timed test case generation algorithm (defined in Sect. 4.1.2).

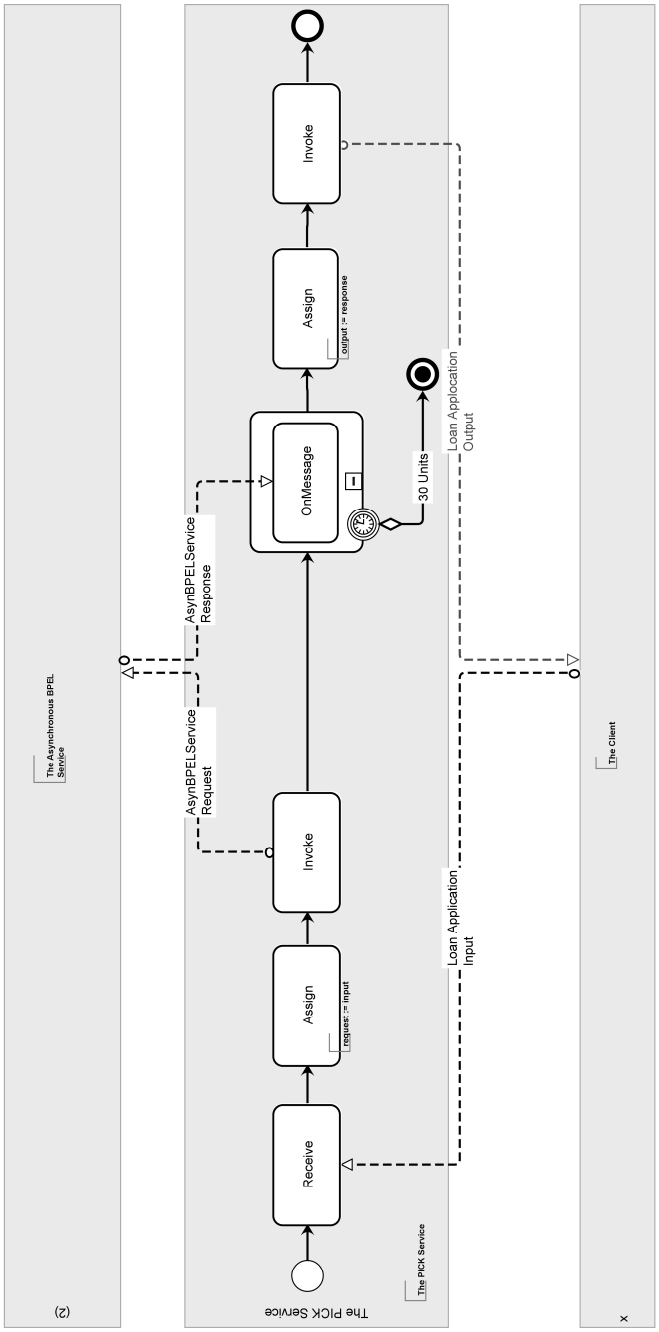

Fig. 9 The BPMN Notation of the PICK Web Service

Test purposes for scenario 1

The PICK service receives the ASYNCBPELSERVICE response (i.e. *response* variable) after 10 seconds. Finally, it sends this loan application document (i.e. *output* variable) to the user [2]. The test purposes for scenario 1 can be formulated as:

$$\begin{aligned} TP_1 &= \{tp_1, tp_2, tp_3\} \\ tp_1 &= \text{“}\textit{action :}\texttt{ AsyncBPELService ?onResult(response)”} \\ tp_2 &= \text{“}\textit{action :}\texttt{ client !onResult(output)”} \\ tp_3 &= \text{“}\textit{clock :}\texttt{ c4 = 10”} \end{aligned} \tag{8}$$

Test purposes for Scenario 2

The PICK service invokes the ASYNCBPELSERVICE service and waits its response for 30 seconds. After, it terminates its activity. The timed test purposes for scenario 2 can be formulated as:

$$\begin{aligned} TP_2 &= \{tp_1, tp_2\} \\ tp_1 &= \text{“}\textit{action :}\texttt{ AsyncBPELService !initiate(request)”} \\ tp_2 &= \text{“}\textit{clock :}\texttt{ c4 = 30”} \end{aligned} \tag{9}$$

The two following timed test cases (presented in Fig. 10) are generated from the WS-TEFSM model of the PICK Web service (see Fig. 19 in Appendix) and the timed test purposes (TP_1 and TP_2).

Timed Test case for Scenario 1

```
1. client ?initiate(input)
2. AsyncBPELService !initiate(request)
3. delay = 10 seconds
4. AsyncBPELService ?onResult(response)
5. client !onResult(output)
```

Timed Test case for Scenario 2

```
1. client ?initiate(input)
2. AsyncBPELService !initiate(request)
3. delays = 30 seconds
```

Fig. 10 The Timed Test Cases for the Two Scenarios

[2] A Web service that is involved in the WS-BPEL process is always modeled as a `<porType>` (i.e. abstract group of operations (noted op) supported by a service). These operations are executed via a `<partnerlink>` (noted by pl) that specifies the communication channel. In the following, the *input* message $pl\ ?op(v)$ denotes the receiving of the message $op(v)$ (constructed from the operation op and the WS-BPEL variable v) via the channel pl. The *output* message is denoted by $pl\ !op(v)$.

After dealing with the test generation for Web services, an approach to generate test cases from a parsed UML model for Web applications is presented.

4.2 Test Generation for Web applications

To automate test generation, the goal is first to parse the UML diagrams obtained from the previous steps. Therefore, based on these diagrams, the necessary requests to the Web application server are generated and then checked if the server's replies are as expected by the previous models. The requests are HTTP requests that simulate a user navigating the site through a Web browser. The possible actions are to fill and submit a form, to click on a link or to navigate to a given URL. Similarly the server's replies are HTTP Responses that can either contain an HTML page, a redirection to another URL or an error message. Assuming the first case, the HTML page of the response has to be parsed to see if its contents are the expected ones. Based on these requirements, it is needed to choose the components that were required to build the test suite.

Since we are dealing with UML, it would be more efficient to choose an object-oriented language as a test scripting language, which also serves to provide easy string handling and a high level of abstraction for network communications. For the reasons mentioned below, the chosen programming language as the most suitable is the Python scripting language. Python is a modern object-oriented language which combines remarkable power with very clear syntax. Its built-in modules provide numerous functions that facilitate string handling and networking.

4.2.1 Parsing and executing the UML

To parse the UML diagrams it is possible either to use the API of a UML tool or to export the diagrams in an XMI format that would allow parsing them using an XML parser. Exporting to XMI was the preferred solution since it does not tie the methodology to a specific tool. Although having the possibility to use any XML parser to parse the XMI, due to the high complexity of the standard it was decided to use a specialized XMI parser.

The used one was the parser included in the System Modeling Workbench (SMW) tool [40]. It is free, open source and also written in Python, making it easier to integrate with the code.

4.2.2 Parsing the HTML pages

Since HTML mixes presentation and content data, the HTML output of the Web application does not allow extracting the wanted information without first looking the implementation details. To avoid this, it is needed to change the page templates

of the Web application in order to provide the data in a more formal way. This is achieved by adding id attributes to the tags we want to query. For example, to the `td` tag that contains the user's name in the user pages will have an attribute `id=username`. By this way it can query any page independently of the implementation of the page layout.

The approaches presented in this section are applied to a real case study illustrated in the following section.

5 Case Study: dotLRN

To exercise the testing generation methods previously presented, we consider a real case study which is an open source e-learning platform (dotLRN) [13]. An open source platform is more demanding of testing aspects to maintain a correct e-learning Web application. Indeed, all the open source software are constantly changing with addition of new features. Therefore new bugs may appear disabling some functionalities. Then we need to re-execute all the previous tests that were generated at the conformance testing phase in order to guarantee the stability of the system. This step of testing is known as non-regression testing. We present the experiments that have been conducted on the e-learning platform. This case study is representative enough to experience the approach. We have made the choice to deal with only one real case study for both types of Web-based systems. This last point is motivated by the growing convergence of these two types of systems. Indeed, we can notice that the industrial market is more and more interested by editing its development as Web services to promote the use of its development. It can also allow to use the Web application in the framework of new development by composition of this application transformed into a web service with other available web services. This aspect particularly holds for open source software that need to be used to get feedback to improve the software.

We present in the following the model-based approach with the UML modeling of dotLRN, the test objectives and how we execute the tests directly on the platform. Afterwards, we also explain how hand tests can be produced for this particular application with TCL script when no formal model is provided. Finally, based on the work described in [29] we explain how we can transform a Web application in a Web service. This latter aspect is essential for certain kind of Web applications whose the model is very tough to obtain (even using UML). Indeed, by migrating them to Web services, the use of testing techniques based on WS-BPEL can be applied. case-study/introduction-application

5.1 The DotLRN framework

DotLRN is a learning management Web application [13]. It is an open source platform for supporting e-learning and digital communities. The tool was originally developed at the Massachusetts Institute of Technology (MIT) as a virtual learning environment and it evolved into a comprehensive platform including not only e-learning support but also generic Web resources.

The platform is based on the OpenACS Web application framework [11], a toolkit for building scalable, community-oriented Web applications. The toolkit structure is highly modular and dotLRN is a set of modules that provide the additional features to deploy an e-learning environment. The OpenACS (and therefore dotLRN) is tightly integrated with a relational database, both PostgreSQL and Oracle are currently supported. The Web server that handles requests at the basic level is AOLServer, the America Online's open source Web server. One of its main features is the integration in its core of a multi-threaded TCL interpreter which provides an effective solution for industrial strength type of services such as those present in large higher educational institutions [38].

As in most open source projects, there is a community around dotLRN/OpenACS involving nearly 11,000 registered users. The community portal is itself based on this platform and coordinates the interaction between developers, users, technical personnel employed by higher education institutions and anybody interested on exchanging ideas, solutions and information about the tool.

Several features make dotLRN an effective and powerful e-learning platform. Its modular structure allows for very fast customization and prototyping of new applications. The user space is organized through a customizable set of portlets, each of them offering access to one of the various services available. The underlying OpenACS toolkit provides an ever increasing set of Web functionality most of them suitable to be adopted by the e-learning platform.

The fact that OpenACS is a community-oriented toolkit has influenced and shaped dotLRN into what it could be called a "communication oriented LMS" (Learning Management System). Most of the current LMS focused at the beginning of their existence on providing content management for teaching staff and learners. DotLRN, on the other hand, was conceived as a platform to facilitate communication among all the different actors in a learning experience.

5.2 Test Generation from the UML model

In the Section 3.2 are described the UML diagrams that are needed to model a Web Application, now it will be illustrated how these UML diagrams are used to model the dotLRN framework and how the test cases can be generated taking into account the Section 4.2.

5.2.1 Modeling Use Case Dependencies

In the Class diagram the use case parameters are also included. The reason is that sometimes it is easier to realize the dependencies between the parameters of the use cases. For instance, in the Figure 11, in order to add a new class, the administrator should provide information about the term (`Term.name`) and the subject of the class (`Subject.name`). As a consequence, there is a dependency between the `Add class`, the `Add Term` and the `Add Subject` use cases.

Finally, in the diagram the use cases are organized in groups according to the objects they are associated with. These objects are instances of the classes in the Class diagram. Figure 11 shows the respective activity diagram for the Administrator. According to this latter, `Add department` should precede `Add subject`. Also, `Add term` and `Add subject` should occur before `Add class`, and `Add user` and `Add class` should take place before the execution of `Assign user to class`. Finally, `Manage User` depends on `Add User` since first the user should be added to the system and then the administrator can edit his profile and modify his permissions.

In the testing phase, before simulating the scenarios in the Sequence diagrams, these activity diagrams should be scanned to obtain the sequence in which the use cases will be tested.

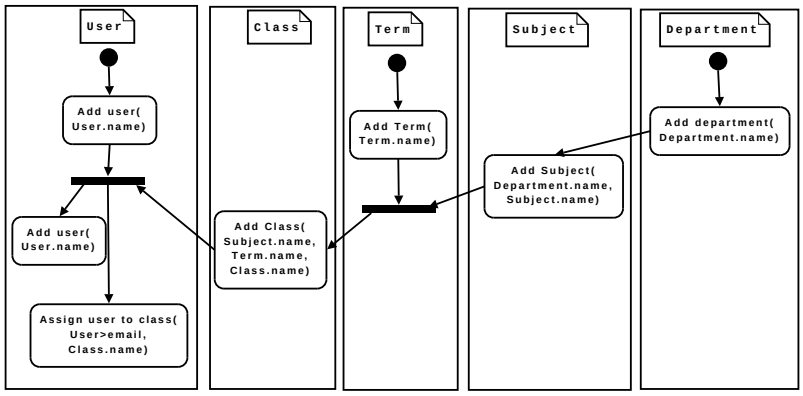

Fig. 11 The class diagram

5.2.2 Sequences diagrams

The Sequence diagrams are also parameterized since input parameters can influence the execution and constitute separate choices [2]. Such a parameter can be the email of a user. Whether this email belongs to a registered user (exists in the database) or belongs to a new user (does not exist in the database) determines what is going to

occur. In the former case the dotLRN page is displayed otherwise a warning appears in the Log In page. During the testing procedure, if there are such branches and parameters then the produced program has to fork to test all the different possibilities. Figure 12 shows the respective sequence diagram for the “Login” use case.

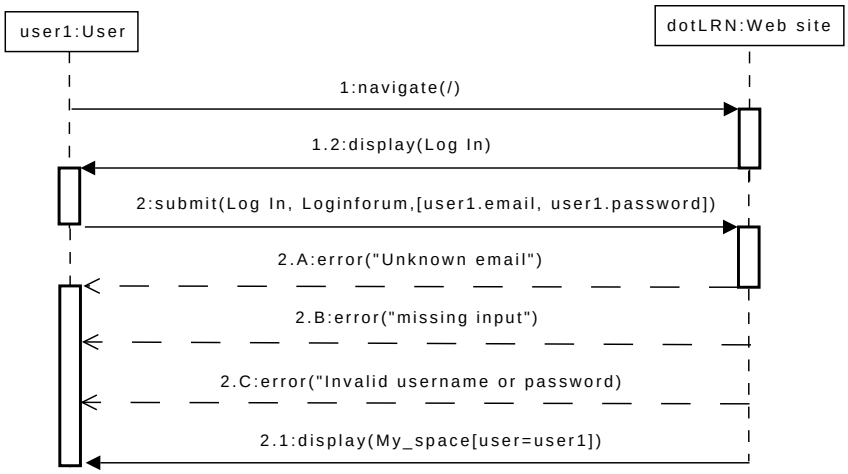

Fig. 12 HTTP sequences

Table 1 summarizes the actions used in the Sequence diagram organized as HTTP requests of the user and possible HTTP responses returned to the user by the server, since the success or the failure of the tests depends upon these requests and the respective responses. The system under test being a Web application, there are three possibilities for the user: either (1) navigates to a URL or (2) requests a Web page through another one (clicks on a link to the wanted page) or (3) submits information by filling an HTML form. System answers either directly by returning the requested page (display) or by giving an error message.

HTTP Request	**HTTP Response**
– navigate(url:String): User makes an HTTP request for an URL.	**– display(page:Webpage):** Web server returns the requested Web.
– link(target:String): User clicks in a HTTP link.	**– display(page:Webpage):** Web server returns the target Web page.
– Sublit(page:WebPage, form:Form,data:list): User submits an HTTP form.	**– display(page:Webpage):** In case of legitimate input the Web server responses with a new Web page. **– error(msg:String):** In case of wrong input the Web server responses with the previous page displaying a warning message.

Table 1 Actions of the Sequence Diagram

5.2.3 dotLRN HTML pages parsing and tests execution

The Figure 13 presents the skeleton of a possible result of the manner to parse the HTML and UML and then execute the UML. In the code we have left some code (mainly some functions) in order to reduce the size and increase clarity.

```
1:  from urllib import urlopen
2:  from smw.io umport XMIStreamer
3:  from smw.metamodel import UML14
4:
5:  class TestSuite:
6:    returned page = None
7:
8:    def validateLink(self,link)
9:     operation = self.getOperation(link)
10:
11:   if operation.name == "navigate":
12:    url = self.getOperationParameters(operation)
13:    fd = urlopen(serverbase + url)
14:    self.returnedPage = fd.read()
15:    fd.close
16:
17:   elseif operation.nale == "display":
18:    params = self.getOperationParameters(operation)
19:    pageTemplate = generatePageTemplate(params)
20:
21:  parser = dotHTMLParser(pageTemplate)
22:  parser.feed(returnedPage)
23:
24:  def execute(self,source):
25:   xmi = XMIStreamer(UML14)
26:   fd = open(source,"r")
27:   model = xmi.loadFromStream(fd)
28:   fd.close()
29:
30:  sequenceDiagram = self.SequenceDiagram(model)
31:  for link in self.getLinks(sequenceDiagram)
32:   self.validateLink(link)
```

Fig. 13 Parsing and execution of UML

Function `execute` (line 24) is the main function of the class that reads the XMI code from the file defined in the variable `source` that is given as a parameter. It then isolates the sequence diagram (for this example we assume that only one exists) and then validates one by one all its messages (links).

All the getX functions (like `getLinks`) consist of navigating through the structure generated by SMW to get a specific data. They are assumed to be defined inside the class. The validation of each link depends on the operation.

If the operation is `navigate` (lines 11-15) then we have to extract the destination URL from the parameters and then get the requested page. We assume that destination is a relative URL, so we use the `serverbase` variable (line 13) to make it absolute. The page is then kept in the `returnedPage` variable to be used by the following commands. In the case of a `display` operation (lines 17-22), we create a template of the page based on the operation parameters and then use the HTML parser to compare the `returnedPage` with the template. The skeleton of the parser is presented in the Figure 14.

```
1:  from HTMLParser import HTMLParser
2:
3:  class dotHTMLParser(HTMLParser)
4:    pageTemplate = None
5:
6:    def init(self,pageTemplate):
7:      self.pageTemplate = pageTemplate
8:
9:    def handle_starttag(self,tag,attrs):
10:     for att in attrs
11:       if "id" in attr:
12:         validateElement(tag,attr)
```

Fig. 14 Parsing the HTML

The `dotHTMLParser` class inherits the `HTMLParser` class and overrides the `handle_starttag` function to search for elements that have an *id* attribute. Every such element will be validated according to the `pageTemplate` that was given during the instantiation. Similarly to the two example operations we can write the code to handle the rest of the supported operations.

5.3 Non-regression testing for Web applications

Besides and as this is above mentioned, the non-regression testing aims to verify if after the insertion of new source code into the application, all the functionalities still run correctly. Hence the first task for testing a Web application is to test if the behavior of the platform is conform to the blue-prints used to build it (i.e. its specification). Because of the lack of formal specification, the documentation of dotLRN and our expertise in OpenACS/dotLRN was used to build an informal specification (i.e. the list of requirements and standards that dotLRN must meet). By using this specification it is possible to interact with the platform and observe if it meets the standards or not. Now, to achieve the stability of the platform during its continuous development by applying a simple testing strategy, i.e. re-test all, requires an unacceptable amount of time and resources. It is needed then to automate the process to re-test the platform to improve the testing efficiency.

5.4 Alternative method for the test generation of Web applications

In order to test a Web application that does not have a formal specification, test cases may be manually developed. In this Section the methodology followed to develop these test cases for dotLRN using the Tclwebtest recorder tool is illustrated. It will also introduce the acs-automated-testing, an OpenACS [11] package for the management of test cases execution and the verdict storage of each test case. Although the phase of the conformance testing was a hand made process, by using the TwtR plug in it is possible to obtain a record or static trace of the interaction of the user with the Web application, this static trace is the base for the development of the non-regression test cases. The process of obtaining a static trace is illustrated in the Figure 15.

The TwtR [41] is a recent tool based in the interaction recording as Selenium, by using this tool it is possible to obtain the trace written in a TCL script language, more specifically in Tclwebtest code. TwtR will produce a static trace that does not contain all the interactions between the user and the Web application, but only the stimulations of the user to the Web application. This static trace is the basis for the part of the functional testing process presented in this article.

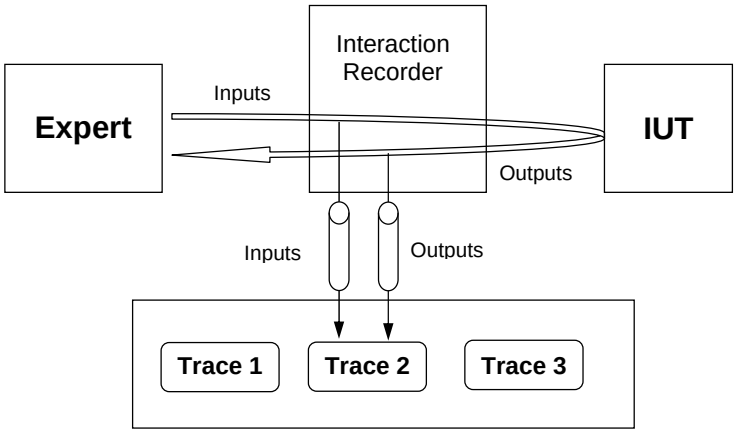

Fig. 15 Generation of the static traces used as a base for further non-regression testing.

To build the tests for the dotLRN features, the information is first extracted from the OpenACS documentation and the dotLRN platform usability. Then the informal specification is built. After this, we interact with dotLRN to test its conformance while recording the static trace using TwtR.

If the result of this conformance test execution verdict is positive, the static trace can be re-used to serve as a basis to build the non-regression test case. This static trace must be modified by inserting variables instead of hard coded values (e.g. the ids and the URLs). Each test case will be finally described in Tclwebtest and composed by a set of concatenated scripts.

The generation of each test case using Tclwebtest is processed by following these steps:

1. Create the script of the test preamble, i.e. a sequence of operations that will lead the system to a state where the test case can be executed. For instance to test the edition of the user's name, first it is needed to corroborate the user existence and that it has already a name assigned, otherwise it is needed to create these elements;
2. Develop the script that will interact with the feature to be tested;
3. Develop the scripts that will analyze the reaction of the Web application to the interaction with the test case. This script will also assign the verdict (pass or fail); basically it will observe if the platform did what it was supposed to.

5.4.1 Example: the addition of a `Question and Answer` from the `faq` package of dotLRN

The trace produced during the addition of a `Question and Answer` (Q&A) of the `FAQ` is illustrated in the Figure 16. In this trace it can be observed that the elements such as the `FAQ` name and the `Q&A` is hard coded. In the first line it can be viewed that the user has followed a link named "`Test faq`" which is the name of the `FAQ` to be tested. After this in the third line the user followed the link to come to the page to create a new `Q&A`. Then from the lines four to seven the form was filled with the question "`What is your name?`" and the answer "`Harry`".

```
1: ::tclwebtest::link follow Test faq ;#
                ~u {http://domain/faq/ad/one-faq?faq\_id=12}}
2: ::tclwebtest::assert text FAQ
3: ::tclwebtest::link follow Create New Q&A ;#
                ~u {http://domain/faq/ad/q-a-add?faq\_id=12}}
4: ::tclwebtest::form find ~n {new\_quest\_answ}}
5: ::tclwebtest::field fill What is your name? ;#
                     ~n {question} ;\# type of field = text}
6: ::tclwebtest::field fill Harry ;#
                   ~n {answer} ;\# type of field = password}
7: ::tclwebtest::form submit
```

Fig. 16 Static trace of the interactions of the user with dotLRN when creating a new Q&A.

In replacing by variables all the texts to be inserted in the HTML forms and text of the links to be followed, the static trace is transformed into a dynamic script that will be the part of the test case for testing the addition of a `Q&A`. It will also be re-used and served as the preamble of other test cases, for example to the test "`Edit a Q&A`". Remember that before editing an item, it is mandatory that this item exists.

It is important to notice that the chunk of code of the Figure 17 will be just the part of the non-regression test case that will interact with the part of dotLRN that allows to add a new Q&A. The entire test case must include the preamble (log in of the user, creation of the FAQ, assign a value to the variables to be used, etc.) and the part of the test case that will analyze if the Q&A was correctly created and assign the verdict.

```
1: tclwebtest::link follow $faq_name
2: tclwebtest::link follow "Create New Q&A"
3: tclwebtest::form find ~n "new_quest_answ"
4: tclwebtest::field find ~n "question"
5: tclwebtest::field fill "$question"
6: tclwebtest::field find ~n "answer"
7: tclwebtest::field fill "$answer"
8: tclwebtest::form submit
9: aa_log "Faq Question Form submitted"
```

Fig. 17 Chunk of the test case that was extracted from the TwtR trace illustrated in the figure 16.

Besides, when a developer adds a new functionality to a system (in this case a Web application), most of the times he manually tests this new functionality to be sure that it works, to then release the new version of the system. However, not only the new functionality should be tested, but all the system functionalities to be sure that the new inserted implementation does not disturb the behavior of the rest of the system.

The OpenAcs framework has among its packages the acs-automated-testing, this package allows to execute the test cases and store the value of the verdict. In this way, it is possible to test dotLRN feature by feature and to store the verdict. The non-regression testing consists in executing the package A version x and then to test the package A version $x+1$. As the verdicts of both tests are stored, it is possible to detect when a functionality that was working fine in the version x does not work anymore in the new version $(x+1)$. By doing this it is possible to maintain the stability of all the features of the version x of the package A.

Next to the illustration of the methodologies to generate test suites for Web services and Web applications, we introduce in the following part of the chapter a method that allows to describe dotLRN in the Service Oriented Architecture (SOA). The goal of this method is to migrate from Web application into Web service. The main goal is of course to apply the methodology presented to testing Web services to dotLRN, and to generalize it to any other Web application.

5.5 Migrating Web Applications Functionalities into Web Services

Service Oriented Architecture (SOA) [30] is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It permits to encapsulate application logic in services with an uniformly defined interface and making these publicly available via discovery mechanisms. Software applications (offered by different providers) can be interconnected using the common SOA infrastructure for obtaining new services and applications. Some systematic approaches for exporting existing software applications towards the new service oriented architectures are proposed in the literature.

In [29], an approach based on wrapping techniques is proposed to migrate functionalities of existing Web applications into Web services. The main goal is to apply modeling and testing techniques used for Web services to Web application functionalities whose the formalization is often difficult to obtain. For instance, the Web application User Interface which is tough to model (even using UML) could be migrated towards a formal Web service and then be tested as mentioned in Sect. 4.1.

This migrating approach uses black-box reverse engineering techniques for modeling the Web application User Interface (i.e. a model of the interactions between a user and the Web application) for each functionality. A wrapper interacting with the Web application is used to transform the User Interface into Web service request/response interface (e.g. WSDL description [7]). This wrapper is constituted by the following four components:

i. Automaton which provides a model of the interactions associated with a given functionality. This model is a set of interaction states, actions and a set of transitions between these states;
ii. Web Application Interaction Executor which executes the interactions of the Automaton;
iii. Automaton Interpreter that coordinates the Web application execution (i.e. the execution of the actions associated to each state of the Automaton);
iv. Web Service Interface Manager that manages external Web service requests and responses.

This approach and a proposed migration platform are used to assist the migration of the dotLRN platform. Functionalities of each dotLRN module (e.g. forums, faqs, calendar, file storage, etc.) can be migrated into a simple or a composite Web service. A selected dotLRN functionality can be captured by an use case. This latter can be decomposed in more elementary use cases which can be wrapped into single Web service. Therefore, the use case can be represented as a WS-BPEL process having as partners single services.

The process of this migration includes the following four steps:

1. Selection of the dotLRN functionality to be turned into a Web service;
2. Reverse engineering of the dotLRN User Interface: identification of execution scenarios and characterization of their states;

3. Design of the interaction model: evaluation of the modeling solutions and specification of the model in WSDL(for single automata) and eventually in WS-BPEL (for composite automata);
4. Wrapper validation and deploy: testing the wrapped Web service and publishing its WSDL description in an application server. The Web services composition testing approach is a used in this step to discover the execution failures of the wrapper Web service (e.g. unexpected output responses), unidentified Web pages, etc.

6 Conclusion

We have presented in this chapter two approaches to test a Web-based system. We have dealt with Web services composition and Web applications. We have proposed a stepwise methodology that consists first to describe formally the system that responds to the requirement information; then from the model we have developed methods to generate the tests and finally we have executed the tests on a real implementation, an e-learning platform.

The modeling step has been performed with two dedicated languages, i.e. WS-BPEL for the Web services composition and UML for Web applications. We have defined rules of mapping from WS-BPEL towards a formal specification, i.e. the Timed Extended Finite State Machines for Web Services (WS-TEFSM). The objective of this mapping is twofold: to give a semantic to the WS-BPEL composition and to dispose of a model from which the tests have been generated. For the Web applications, we have made the choice of UML as modeling language. This latter is well-suited to cope with object oriented applications and to model the test cases by means of sequence diagrams and also to represent the dynamic navigation between pages of the Web application.

Together with the formal models of the web systems, we have presented methods for conformance and non-regression test generation. The conformance is established by a conformance relation between an implementation and a specification. Two techniques have been developed for tests generation purpose, one to handle WS-BPEL specification and one for the UML model.

Finally, we have exercised the proposed methodology on a real case study, an open source e-learning platform dotLRN. We have carried on conformance and non-regression tests with the formal approach and also with a hand crafted methods based on TCL scripts. We have presented how to provide a Web service from a Web application. The obtained Web services can be composed with other available Web services and the WS-BPEL testing methods can be thus applied.

References

1. Alalfi, M.H., Cordy, J.R., Dean, T.R.: A survey of analysis models and methods in website verification and testing. In: Proc. Seventh International Conference on Web Engineering ICWE 2007, pp. 306–311 (2007)
2. Basanieri, F., Bertolino, A., Marchetti, E.: The cow_suite approach to planning and deriving test suites in UML projects. In: J.M. Jézéquel, H. Hussmann, S. Cook (eds.) UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings, *LNCS*, vol. 2460, pp. 383–397. Springer (2002)
3. Bayse, E., Cavalli, A., Núñez, M., Zaidi, F.: A passive testing approach based on invariants: application to the wap. Comput. Netw. ISDN Syst. **48**(2), 247–266 (205)
4. Cambronero, M.E., Diaz, G., Pardo, J.J., Valero, V., Pelayo, F.L.: Rt-uml for modeling real-time web services. In: Proc. IEEE Services Computing Workshops SCW 2006, pp. 131–139 (2006)
5. Cardoso, J.: Approaches to developing semantic web services. International Journal of Computer Science (IJCS) **1**(1), 8–21 (2006)
6. Cavalli, A.R., Lee, D., Rinderknecht, C., Zaïdi, F.: Hit-or-jump: An algorithm for embedded testing with applications to in services. In: FORTE XII / PSTV XIX 1999: Proc. of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), pp. 41–56. Kluwer, B.V., Deventer, The Netherlands, The Netherlands (1999)
7. Christensen, Curbera, F., Meredith, G., Weerawarana, S.: Web services description language wsdl ver. 1.1 (March 2001). http://www.w3.org/TR/wsdl
8. Christopher A. Jones & Fred L. Drake, J.: Python & XML, 1st edition edn. O'Reilly & Associates (2001)
9. Cockburn, A.: Writing Effective Use Cases, 1st edition edn. Addison-Wesley (2000)
10. CollabNet: Argouml (2008). http://argouml.tigris.org/
11. Community, O.: Openacs. http://openacs.org
12. Community, O.: Selenium ide. http://www.openqa.org/selenium-ide/
13. dotLRN: Learn, research, network. http://www.dotlrn.org
14. EVIWARE: soapui: the web services testing tool (2009). http://www.soapui.org/
15. Farahbod, R., Glasser, U., Vajihollahi, M.: Specification and validation of the business process execution language for web services. In: Abstract State Machines, pp. 78–94 (2004)
16. Fouchal, H., Petitjean, E., Salva, S.: Testing timed systems with timed purposes. In: Proc. Seventh International Conference on Real-Time Computing Systems and Applications, pp. 166–171 (2000)
17. Garca-Fanjul, J., Tuya, J., de la Riva, C.: Generating test cases specifications for bpel compositions of web services using spin. In: Proc. International Workshop on Web Services - Modeling and Testing WS-MaTe 2006, pp. 83–94. Palermo, Italy (2006)
18. Group, O.M.: BPMN, business process modeling notation (2009). http://www.bpmn.org/
19. Hinz, S., Schmidt, K., Stahl, C.: Transforming bpel to petri nets. In: Business Process Management, pp. 220–235 (2005)
20. Holzmann, G.J.: The spin model checker: Primer and reference manual (2003)
21. Hong, H.S., Cha, S.D., Lee, I., Sokolsky, O., Ural, H.: Data flow testing as model checking. International Conference on Software Engineering **0**, 232 (2003)
22. IBM: Rational rose. http://www-306.ibm.com/software/awdtools/developer/rose/index.html
23. Kazhamiakin, R., Pandya, P., Pistore, M.: Timed modelling and analysis in web service compositions. In: Proc. First International Conference on Availability, Reliability and Security ARES 2006, pp. 7 pp.– (2006)
24. Krichen, M., Tripakis, S.: An expressive and implementable formal framework for testing real-time systems. In: Proc. 16th IFIP International Conference on Testing of Communicating Systems TestCom 2005, pp. 209–225 (2005)

25. Kung, D.C., Liu, C.H., Hsia, P.: An object-oriented web test model for testing web applications. Asia-Pacific Conference on Quality Software **0**, 111 (2000)
26. Lallali, M., Zaidi, F., Cavalli, A.: Timed modeling of web services composition for automatic testing. In: Proc. Third International IEEE Conference on Signal-Image Technologies and Internet-Based System SITIS 2007, pp. 417–426 (2007)
27. Li, N., qin Ma, Q., Wu, J., zhong Jin, M., Liu, C.: A framework of model-driven web application testing. Computer Software and Applications Conference, Annual International **2**, 157–162 (2006)
28. Liu, C., Kung, D., Hsia, P., Hsu, C.: Structure testing of web applications. In: Proc. 11th Annual International Symposium on Software Reliability Engineering, pp. 84–96 (October 2000)
29. Lorenzo, G.D., Fasolino, A.R., Melcarne, L., Tramontana, P., Vittorini, V.: Turning web applications into web services by wrapping techniques. Working Conference on Reverse Engineering **0**, 199–208 (2007)
30. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0. oasis standard, 12 october 2006. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
31. NAKAJIMA, S.: Lightweight formal analysis of web service flows. Progress in Informatics **2**, 57–76 (2005)
32. OASIS: Organization for the advancement of structured information standards. http://www.oasis-open.org/specs/index.php
33. OASIS: Universal description discovery and integration. http://uddi.xml.org/uddi-org
34. OASIS: Wsbpel ver. 2.0 (April 2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
35. OMG: Unified modeling language (uml). http://www.uml.org/
36. OMG: Xml metadata interchange (xmi). http://www.omg.org/spec/XMI/2.1.1/
37. OpenSTA: Open system testing architecture. http://www.opensta.org/
38. P., E.J., la Fuente Valentn L, D., S, G., A, P., C, D.K.: Approaches to developing semantic web services. International Journal of Computer Science **1**(1), 8–2 (2006)
39. Paradkar, A.M., Sinha, A., Williams, C., Johnson, R.D., Outterson, S., Shriver, C., Liang, C.: Automated functional conformance test generation for semantic web services. IEEE International Conference on Web Services ICWS **0**, 110–117 (2007)
40. probleme: System modeling workbench tool. http://www.abo.fi/~iporres/html/smw.html
41. Realfsen, A.S.: Tclwebtest recorder. http://www.km.co.at/km/twtr
42. Ricca, F., Tonella, P.: Analysis and testing of web applications. International Conference on Software Engineering **0**, 0025 (2001)
43. Sheshagiri, M.: Automatic composition and invocation of semantic web services. Master's thesis, UMBC (2004)
44. SOASTA: Soasta cloudtest. http://www.soasta.com/
45. Softlogica: Wapt: Web application testing. http://www.loadtestingtool.com/
46. Software QA and Testing Resource Center: Web site test tools and site management tools. http://www.softwareqatest.com/qatweb1.html
47. W3C: extensible markup language xml. http://www.w3.org/XML
48. W3C: Ontology web language (owl). http://www.w3.org/TR/owl-features/
49. W3C: Resource definition framework (rdf). http://www.w3.org/RDF/
50. W3C: Web application formats working group. http://www.w3.org/2006/appformats/
51. W3C: Web ontology web language for services (owl-s). http://www.w3.org/Submission/OWL-S/
52. W3C: Web service description language with semantics (wsdl-s). http://www.w3.org/Submission/WSDL-S/
53. W3C: Web services modeling ontology (wsmo). http://www.wsmo.org/
54. W3C: The world wide web consortium. http://www.w3.org/
55. W3C: Simple object access protocol soap (version 1.1) (May 2000). http://www.w3.org/TR/soap/

56. Wombacher, A., Fankhauser, P., Neuhold, E.: Transforming bpel into annotated deterministic finite state automata for service discovery. In: Proc. IEEE International Conference on Web Services ICWS 2004, pp. 316–323 (2004)
57. Xu, L., Xu, B., Chen, Z., Jiang, J., Chen, H.: Regression testing for web applications based on slicing. Computer Software and Applications Conference, Annual International **0**, 652 (2003)
58. Yang, Y., Tan, Q., Yu, J., Liu, F.: Transformation bpel to cp-nets for verifying web services composition. In: Proc. International Conference on Next Generation Web Services Practices NWeSP 2005, pp. 6 pp.– (2005)
59. Yu, Y., Huang, N., Luo, Q.: Owl-s based interaction testing of web service-based system. Next Generation Web Services Practices, International Conference on **0**, 31–34 (2007)
60. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: International Conference on Web Services ICWS 2003, pp. 38–41 (2003)

Appendix

```
<process>
  <sequence>
    <receive name="receiveInput" partnerLink="client" portType="tns:Pick"
                  operation="initiate" variable="input" createInstance="yes"/>
    <assign>
      <copy>
        <from variable="input" part="payload"/>
        <to variable="request" part="payload"/>
      </copy>
    </assign>
    <invoke name="invokeAsyncService" partnerLink="AsyncBPELService"
                   portType="services:AsyncBPELService" operation="initiate"
                                                        inputVariable="request"/>
    <pick name="receiveResult">
      <onMessage partnerLink="AsyncBPELService"
          portType="services:AsyncBPELServiceCallback" operation="onResult"
                                                              variable="response">
        <assign>
          <copy>
            <from variable="response" part="payload"/>
            <to variable="output" part="payload"/>
          </copy>
        </assign>
      </onMessage>
      <onAlarm for="'PT30S'">
        <terminate/>
      </onAlarm>
    </pick>
    <invoke name="replyOutput" partnerLink="client" portType="tns:PickCallback"
                                    operation="onResult" inputVariable="output"/>
  </sequence>
</process>
```

Fig. 18 The WS-BPEL Description of The PICK Web Service

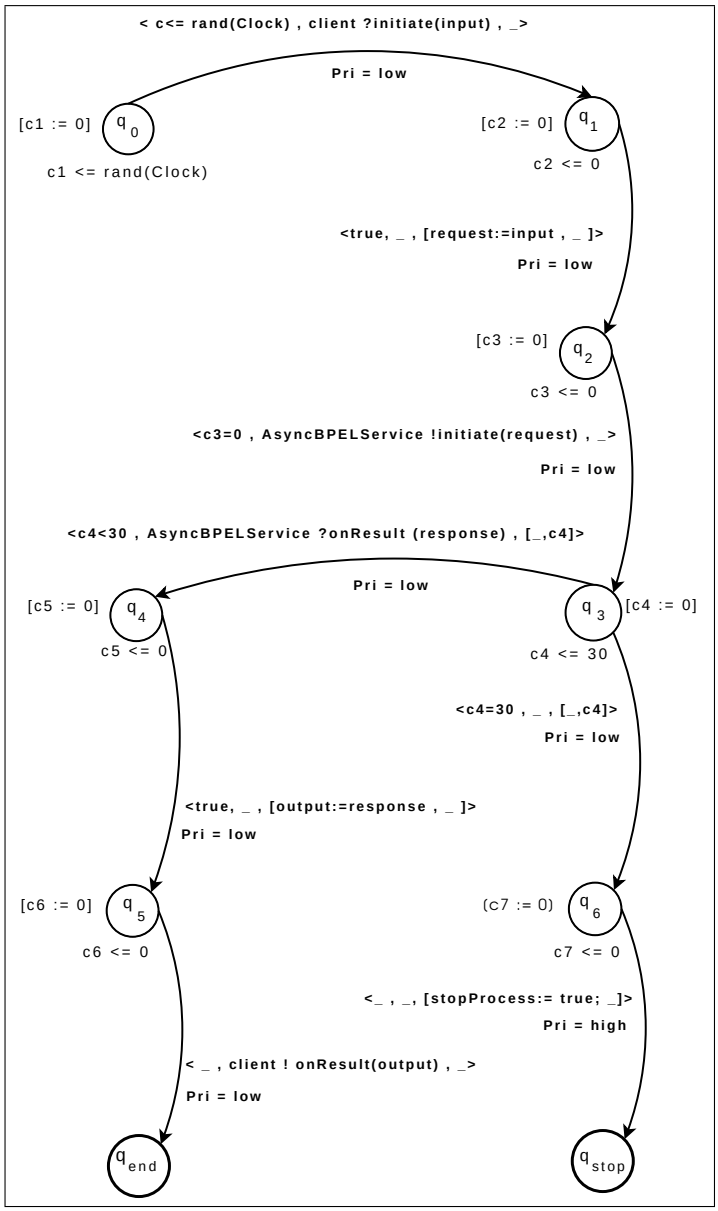

Fig. 19 The WS-TEFSM of the PICK Web Service without termination handling