An Industrial Experience on using Models to Test Web Service-Oriented Applications

Andre Takeshi Endo[1,2], Maicon Bernardino[3], Elder Macedo Rodrigues[3], Adenilso Simao[2], Flavio M. de Oliveira[3], Avelino F. Zorzo[3], Rodrigo Saad[4]
[1] Universidade Tecnologica Federal do Parana (UTFPR)
Cornelio Procopio – PR – Brazil
[2] Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
PO Box 668, 13560–970 Sao Carlos, SP, Brazil
[3] Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil
[4] Dell Inc., Porto Alegre – RS – Brazil
andreendo@utfpr.edu.br, adenilso@icmc.usp.br, {bernardino, elderrodrigues}@acm.org, {flavio.oliveira, avelino.zorzo}@pucrs.br, rodrigo_saad@dell.com

ABSTRACT

Service-oriented architectures and Web services have been widely adopted by enterprises to pervade integration among software systems. As reliable services are essential to assure that these systems work correctly, formal and systematic testing should be performed. This paper reports the application of a model-based approach to test Web services in the context of real-world applications of a multinational computer technology corporation. The employed approach is called ESG4WSC, in which an event-driven model is provided to support modeling and test case generation, as well as an environment to support the concretization and test execution.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability

Keywords

model based testing, service composition, event sequence graph, experience report

1. INTRODUCTION

Service-Oriented Architecture (SOA) is an architectural style in which functionalities are decomposed into distinct units named services [16]. These services, distributed over a global or internal network, can be reused and combined to create new and more complex enterprise applications. In this process, a composite service is developed by integrating the functionalities of partner services. There exist various languages and specifications that have been used to describe or execute service compositions, such as WS-CDL [14] and BPEL [12]. Service-oriented applications have been mainly developed using a set of XML standards (WSDL, SOAP, and UDDI), known as Web Services technology, which is implemented by various programming languages.

Since SOA was proposed as an architectural style and Web services as its main implementation technology, service testing for SOA has been investigated [2, 3]. Furthermore, there is a research effort on proposing formal testing approaches that support the verification of atomic and composite Web services [9]. Among them, *Model-Based Testing* (MBT) has been advocated as a promising approach to provide more systematic and automated testing [4, 6, 11, 21]. In MBT, a model of the *system under test* (SUT) is designed in order to derive test cases automatically. Endo and Simao [8] propose an MBT process for testing service-oriented applications, describing the artifacts, tools and revisiting different steps. A more specific MBT approach to test Web service compositions is presented in [1]. The authors propose a modeling technique, named *Event Sequence Graph for Web Service Composition* (ESG4WSC), that is able to represent communication between the partner services and the composition. Tools were also developed to support modeling and generation, as well as a test execution environment supported by an *Enterprise Service Bus* (ESB).

In this paper, we report an experience on applying a model-based testing approach in the context of service-oriented applications developed within a multinational computer technology corporation. The ESG4WSC approach [1] was adopted to model and test the Web services; moreover, the testing process proposed in [8] was followed. First, a set of BPEL-based composite services were modeled and test suites generated to evaluate the practical capabilities of the modeling and test generation. Second, the complete approach, from modeling to test execution, was applied to an ongoing project. We show the study configuration, the achieved results, and the lessons learned during this project.

The main contribution of this paper is the experience that was conducted in an industrial setting and that shed light on the MBT of real-world service-oriented applications.

The remainder of this paper is organized as follows. Section 2 briefly describes the MBT process adopted and presents the modeling technique applied to design the test models. Section 3 reports the study configuration and adopted tools. Section 4 analyzes the results. Section 5 presents the lessons learned. Threats to validity and a discussion on the results are shown in Section 6. Section 7 summarizes the related work. Finally, Section 8 shows the conclusion and discusses future work.

2. THE TESTING APPROACH

In this study, an MBT process was employed to test service-oriented applications. This process is divided in four main steps:

1. *modeling*: test models are designed by the tester;
2. *test generation*: the designed test model is used to derive test cases;
3. *concretization*: adaptors are implemented to handle the different levels of abstraction between the abstract test suite generated and the SUT; and
4. *test execution*: adaptors and test cases are used to execute the tests in the SUT.

We adopted the ESG4WSC approach to model and generate the test suites [1, 7]. An ESG4WSC model is a directed graph in which nodes are events and edges represent valid sequences of events. Figure 1 illustrates an ESG4WSC model for the `BCS-05` Service[1]; this test model intends to represent the messages exchanged in a composition. In Web service composition, request messages are represented as light gray circles and response messages as dark gray ellipses. The messages of the composition itself are represented with a bold line. As the example is a composition with a request-only operation, the unique event with a bold line is `CS01:operation01`. These events are referred to as public events, while the messages exchanges with the partner services are referred to as private events. There are two special nodes, '[' and ']', that represent the entry and exit nodes, respectively.

A sequence of nodes that are connected by edges is called *event sequence*. Any event sequence that starts with a direct successor of '[' is called *partial event sequence* (PES). If a PES ends with a direct predecessor of ']', it is called *complete event sequence* (CES). There are two assumptions in the ESG4WSC graph: (i) every event must be reachable from the entry node by an event sequence, and (ii) the exit node must be reachable from any event by an event sequence.

It is possible to associate *decision tables* (DTs) with public request events when input parameters may cause different events. In Figure 1, event `CS01:operation01` has two next private request events that are provoked according to an input parameter. Events with associated DTs are double circled. Table 1 shows a simple DT for event `CS01:operation01`. It contains constraint *constraint*01 = '*smd*', which can be evaluated as **true** or **false**, and two rules (R1 and R2). R1 means that if *constraint*01 ='*smd*' is evaluated as ***true***, the next event should be `PS01:op02_smd`. On the other hand, R2 means that if *constraint*01 ='*smd*' is evaluated as ***false***, the next event should be `PS01:op02`. The ESG4WSC model has other features, such as refined events (which contain one or more refining ESG4WSCs) and parallel representation. The approach supports a holistic generation of test cases, covering not only desirable cases (*positive testing*), but also unexpected behaviors (*negative testing*). Formal definitions of the ESG4WSC model and its test generation algorithms can be found in [1].

[1]This service was developed by the multinational computer technology corporation; for the sake of confidentiality, we replace the original labels by generic ones for all services along the text.

Table 1: Decision table for event `CS01:operation01`.

		rules	
		R1	R2
constraints	*constraint*01 ='*smd*'	True	False
next actions	`PS01:op02`		✓
	`PS01:op02_smd`	✓	

Positive Testing

To generate the *positive test cases*, a test suite composed of CESs is produced automatically to cover all edges (event pairs) in the test model. The constraints in the DTs are also solved and new edges are added and covered if necessary [1]. Using the model in Figure 1, 14 test cases were generated; a sample test case is presented as follows.

$\langle$ `CS01:operation01 (R1-constraint01='smd' true), PS01:op02_smd, PS01:op02Response, PS02:op03, PS02:op03Response_na, PS05:op06` $\rangle$

Negative Testing

Test cases can also be generated for undesirable situations. These cases are referred to as *negative test cases*, which can be applied to public or private events. In public negative testing, unexpected cases are tested between public events that were not predicted in the model. The test cases generated for this situation are called *public faulty event sequence* (PubFES). For the example in Figure 1, only one test case is generated (there is only one public event) and represents two request events `CS01:operation01` invoked in parallel by different clients.

In private negative testing, the approach intends to verify the service composition's behavior when unexpected situations occur in the partner services, such timeouts and exceptions. To do so, different fault classes are defined and simulated during the tests. Table 2 shows the fault classes we used, in which type of event is applied, and the number of negative test cases generated for the `BCS-05` service. The test cases generated for this situation are called *private faulty event sequence* (PrivFES). In total, 89 test cases were generated from the model in Figure 1. A sample PrivFES is presented as follows; the first part is a PES to reach private response event `PS02:op03Response_na` and the second part (F_{LR}) indicates that event `PS02:op03Response_na` should take quite some time to be returned to the composition (i.e., a longtime response).

($\langle$ `CS01:operation01 (R1), PS01:op02_smd, PS01:op02Response, PS02:op03, PS02:op03Response_na` $\rangle$, F_{LR})

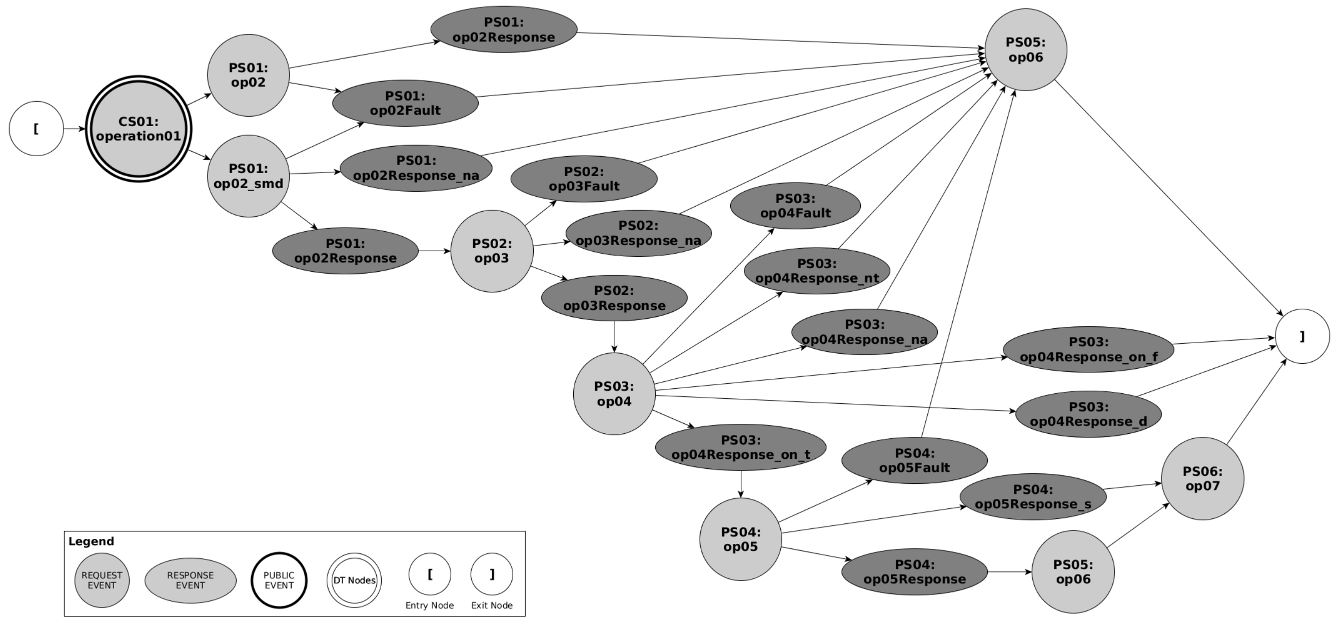


Figure 1: Example of an ESG4WSC model.

Table 2: Fault classes and the number of negative test cases.

Fault Class	Event	#Test cases
No Response (NR)	request	8
Missing Service (MS)		
Unexpected Fault (UF)		
Longtime Response (LR)	response	16
Wrong XML Schema (WSc)		
Wrong XML Syntax (WSy)		
Right Schema, Wrong Data (WD)		

3. STUDY CONFIGURATION

This study was conducted in cooperation with the Technology Development Lab (TDL) of a multinational computer technology corporation that provided access to some of its applications, as well as technological support and documentation. Our research group on Performance Testing from PUCRS (one of the top 7 research universities in Brazil) works in cooperation with this TDL, which has development and testing teams in different countries worldwide. This cooperation is set to experiment and develop new strategies for testing.

The subject applications are all real-world and up-and-running systems. Applications were selected by the TDL based on availability and technical support. All data presented in this paper was produced and collected using Web service-oriented applications developed and used internally by the TDL.

The testing process adopted by the TDL is briefly described as follows. The development team writes automated unit tests, the application is deployed in a production server and developers perform manual tests in integration and system levels (some tests through the Web service, but most of them through the GUI). The development team releases a stable version that is then tested by the performance team, reusing functional test suites or including new ones focused on workload and concurrent users.

We had two researchers that performed the role of testers, namely, tester A and B. Tester A is an expert in the ESG4WSC approach (he participated in the proposal of the approach and in the development of supporting tools). Tester B is a PhD student that had no knowledge in the approach, though he had previous experience with MBT and Web services. Tests were carried out individually or in pair. From the TDL, there were two professionals involved: a performance engineer and a software developer. The communication with the TDL was online (via e-mail and instant messaging) and in face-to-face meetings. Overall, the cooperation between the researchers and the TDL was carried out in one month; the testers were working full time in the project and two professionals provided support on-demand.

We adopted a set of tools to support the conduction of the study and automation. The *Test Suite Designer* (TSD) tool was used to design ESG4WSC models and to generate positive and negative test suites [1]. The *Event Runner for Test Execution* (ERunTE) tool was adopted to support the test concretization and execution [1]. ERunTE is integrated with Mule-ESB [17] to record and control all messages produced during the tests. SoapUI [10] was employed to mock some services. Finally, Eclipse [5] and JUnit [13] were used to support the development of adaptors (programmed in Java) and the test execution.

In this study, the research goal was to evaluate the feasibility of MBT, specifically the ESG4WSC approach, in real-world service-oriented applications. In particular, we aim to validate the modeling and test generation capabilities, as well as to analyze the concretization and test execution steps and associated tools. Thus, we divide the study as follows:

1. *Modeling and test generation:* we focused on the modeling and test generation over BPEL-based composite

services. Further details are shown in Section 4.1.

2. *Concretization and test execution:* we analyzed the full application of ESG4WSC approach in the context of an ongoing project to deploy the ABC application, emphasizing the concretization and test execution. Further details are shown in Section 4.2.

4. ANALYSIS OF RESULTS

4.1 Part 1: Modeling and Test Generation

In this part, we used 23 composite services specified in BPEL to evaluate the ESG4WSC's modeling capabilities. The sources of information about the 23 services were the BPEL specification itself, WSDL files, logs of the BPEL engine, and meetings with the developers. Then, we evaluated the test suites generated from these models.

Table 3 shows the test model's characteristics for each of the modeled services. Observe that all services are asynchronous (request-only) since the number of public response events (Column #4) is zero for all services. The number of public request events (Column #1) is low, most of them with one or two requests. Only services BCS-03 and BCS-20 have more public requests. The number of private requests and response events (Columns #2, #5) reflects the complexity of communication with the partner services. The overall complexity of the test models can be summarized by the number of events and edges (Columns #9, #10). Service BCS-11 has the largest test model with 447 events and 501 edges, followed by BCS-22, BCS-09, and BCS-20. In services BCS-11, BCS-20, and BCS-22, refined events and refining ESG4WSCs (Columns #7, #11) were adopted to deal with the complexity of many events and edges. ESG4WSCs in parallel (Column #12) were not used in these models. Most of the branches in the models are caused by different types of responses, instead of input parameters. This is observed by the number of DTs (and their elements, constraints, actions, and rules) that is low (Columns #13-#16). BCS-20 has the highest number of DTs, which is consistent with its public request events; BCS-10 has the highest number of constraints for one DT (Column #14). Ten out of 23 services do not have associated DTs (Column #13).

From the test models, positive and negative test suites were generated. Table 4 shows the test suite's information divided by the type of testing. For each type of test suite, the number of executed events is also shown as a cost measure (Columns #2, #4, #6, .., #20). All test suites were automatically generated using the TSD tool. The cost of positive test cases is highly dependent on number of events and edges in the model (Columns #9, #10 in Table 3). Service BCS-11 has the highest number of test cases (40) and BCS-01, BCS-15, and BCS-17 has the lowest number of test cases (2) (in Column #1).

Columns #3-#20 refer to negative test suites and their costs. For PubFESs, the cost is related to the number of public request and response events (Columns #1, #4 in Table 3). Services BCS-03 and BCS-20 have the highest number of test cases and cost (Columns #3, #4), 49 test cases executing 98 events and 25 test cases executing 222 events, respectively. Sixteen out of 23 services have only one test case (Column #3).

The PrivFESs are divided in accordance with the fault classes described in Table 2. Notice that PrivFESs are related to private events (as described in Section 2) since for each private event, a test case is generated to simulate a different fault class. The test suites for the fault classes NR, MS, and UF (Columns #5-#10) are dependent on the number of private request events (Column #2 in Table 3). They have different costs (Columns #6, #8, #10) because each fault class has its own characteristics during the execution. For instance, in class NR, the affected request event happens and no response is produced, while in class UF a response event is provoked. Services BCS-11, BCS-09, and BCS-22 have the highest number of test cases.

The test suites for the fault classes LR, WSc, WSy, and WD (Columns #11-#18) are dependent on the number of private response events (Column #5 in Table 3). Services BCS-11, BCS-09, and BCS-22 also have the highest number of test cases. Services BCS-03, BCS-08, and BCS-23 have no test case for these fault classes since their models do not contain private response events (Column #5 in Table 3).

The last Columns #19, #20 show the total number of PrivFESs, including all seven fault classes. As the negative testing for private events produces test suites that cover all request and response events in combination with the fault classes, a high number of negative test cases is generated. For all models, its cost (Column #20) exceeds the cost of positive testing (Column #2).

Test model information gives an idea on the human effort that would be spent since the tester is supposed to design it manually. Although the modeling effort is directly related to the application's size and complexity, we observe that other factors may also influence it. The restricted access to information sources and unclear test purposes may increase the cost during the modeling.

The cost of generating test suites is low since it is automatically performed by the tool. For the largest model, the tool took less than 11 seconds to produce the positive and negative tests. We have provided the test suite information as an additional measure to predict the cost of execution. It is important to emphasize that a development effort is also needed to implement adaptors during the concretization. This topic was not investigated in this part of the study. We provide further discussion on it in Section 4.2.

4.2 Part 2: Concretization and Test Execution

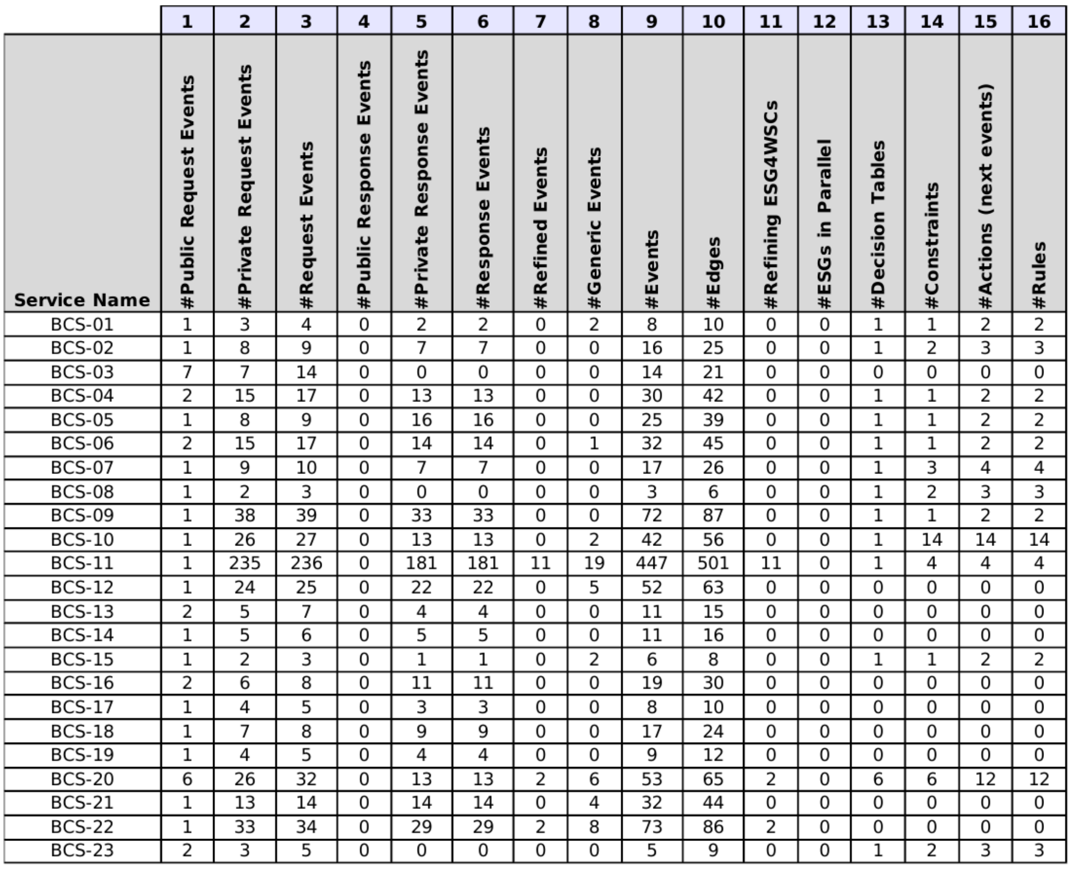
In this part, we applied the ESG4WSC approach in the ABC application (an ongoing project), specifically in its composite service ABCService (ABCS). This service interacts with three other services: PartnerService01 (PS01), PartnerService02 (PS02), and PartnerService03 (PS03). Table 5 shows the tested services, if they are composite or not, their total number of operations, and number of operations involved in the tests.

Table 5: Information about services.

Service Name	composite service?	#operations	#involved operations
ABCService	yes	8	2
PartnerService01	no	26	2
PartnerService02	no	12	1
PartnerService03	no	10	1

Testers A and B worked together to perform this part

Table 3: Test model information.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Service Name	#Public Request Events	#Private Request Events	#Request Events	#Public Response Events	#Private Response Events	#Response Events	#Refined Events	#Generic Events	#Events	#Edges	#Refining ESG4WSCs	#ESGs in Parallel	#Decision Tables	#Constraints	#Actions (next events)	#Rules
BCS-01	1	3	4	0	2	2	0	2	8	10	0	0	1	1	2	2
BCS-02	1	8	9	0	7	7	0	0	16	25	0	0	1	2	3	3
BCS-03	7	7	14	0	0	0	0	0	14	21	0	0	0	0	0	0
BCS-04	2	15	17	0	13	13	0	0	30	42	0	0	1	1	2	2
BCS-05	1	8	9	0	16	16	0	0	25	39	0	0	1	1	2	2
BCS-06	2	15	17	0	14	14	0	1	32	45	0	0	1	1	2	2
BCS-07	1	9	10	0	7	7	0	0	17	26	0	0	1	3	4	4
BCS-08	1	2	3	0	0	0	0	0	3	6	0	0	1	2	3	3
BCS-09	1	38	39	0	33	33	0	0	72	87	0	0	1	1	2	2
BCS-10	1	26	27	0	13	13	0	2	42	56	0	0	1	14	14	14
BCS-11	1	235	236	0	181	181	11	19	447	501	11	0	1	4	4	4
BCS-12	1	24	25	0	22	22	0	5	52	63	0	0	0	0	0	0
BCS-13	2	5	7	0	4	4	0	0	11	15	0	0	0	0	0	0
BCS-14	1	5	6	0	5	5	0	0	11	16	0	0	0	0	0	0
BCS-15	1	2	3	0	1	1	0	2	6	8	0	0	1	1	2	2
BCS-16	2	6	8	0	11	11	0	0	19	30	0	0	0	0	0	0
BCS-17	1	4	5	0	3	3	0	0	8	10	0	0	0	0	0	0
BCS-18	1	7	8	0	9	9	0	0	17	24	0	0	0	0	0	0
BCS-19	1	4	5	0	4	4	0	0	9	12	0	0	0	0	0	0
BCS-20	6	26	32	0	13	13	2	6	53	65	2	0	6	6	12	12
BCS-21	1	13	14	0	14	14	0	4	32	44	0	0	0	0	0	0
BCS-22	1	33	34	0	29	29	2	8	73	86	2	0	0	0	0	0
BCS-23	2	3	5	0	0	0	0	0	5	9	0	0	1	2	3	3

of the study. Based on performance test scripts and on meetings with the development team, the test model shown in Figure 2 was designed. It focuses on the flow of messages triggered by operations operation01 and operation02 of `ABCService`. This model was augmented with SOAP messages for the public request events and private response events. These messages are necessary to provoke some sequences.

The four services involved in the tests were deployed in Mule-ESB. Thus, all messages produced during the tests will be passed through the bus and controlled by module *esbcomp* of the ERunTE tool. We used SoapUI to mock services `PartnerService01`, `PartnerService02`, and `PartnerService03`. A simple modification was performed in `ABCService`, the original service endpoints were changed to the ones provided by the ESB. Thus, all messages produced during the tests are controlled by the ESB.

Using the model in Figure 2, test cases were generated using the TSD tool. Table 6 summarizes the test suites generated for positive and negative testing. In total, 68 test cases were generated.

The next step was the concretization of tests. Two adaptors were implemented, PublicEventAdaptor and MessageCheckingAdaptor [1, 7]. PublicEventAdaptor implements the calls for public events, *i.e.*, invoking the composite service, and checks its responses. Each event has an associated method that is annotated with the event name. MessageCheckingAdaptor implements individual verifications for all events. Its purpose is to verify, after executing a test case, whether all messages were produced and in the defined order. Moreover, a couple of additional classes were implemented to configure and run the tests. Modules *runner* and *service* of ERunTE were used to support the test execution with the developed adaptors. Table 7 shows the number of lines of code and the cyclomatic complexity for the two adaptors and the entire project. The metrics were collected using Eclipse Metrics plugin [20].

All test cases were successfully executed; the approximate execution time is also presented in Table 6. The test suite for fault class MS took more time than the tests for other classes

Table 6: Number of positive and negative test cases.

Positive testing		
Test Suite	**#Test cases**	**Exec. time**
#CESs	7	$\approx$ 13s
Negative testing		
Test Suite	**#Test cases**	**Exec. time**
#PubFESs	4	$\approx$ 5s
#PrivFESs (NR)	7	$\approx$ 75s
#PrivFESs (MS)	7	$\approx$ 515s
#PrivFESs (UF)	7	$\approx$ 13s
#PrivFESs (LR)	9	$\approx$ 96s
#PrivFESs (WSc)	9	$\approx$ 12s
#PrivFESs (WSy)	9	$\approx$ 5s
#PrivFESs (WD)	9	$\approx$ 12s

Table 4: Test suite information.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Service Name	#Positive Test Cases	#Executed Events	#Public Faulty Event Sequences	#Executed Events	#Private Faulty Event Sequences (NR)	#Executed Events	#Private Faulty Event Sequences (MS)	#Executed Events	#Private Faulty Event Sequences (UF)	#Executed Events	#Private Faulty Event Sequences (LR)	#Executed Events	#Private Faulty Event Sequences (WSc)	#Executed Events	#Private Faulty Event Sequences (WSy)	#Executed Events	#Private Faulty Event Sequences (WD)	#Executed Events	Total # #Private Faulty Event Sequences	#Executed Events
BCS-01	2	7	1	2	3	12	3	9	3	15	2	6	2	8	2	8	2	8	18	68
BCS-02	8	47	1	2	8	36	8	28	8	44	7	33	7	33	7	33	7	33	53	242
BCS-03	7	14	49	98	7	14	7	7	7	21	0	0	0	0	0	0	0	0	70	140
BCS-04	10	80	6	28	15	103	15	88	15	118	13	54	13	67	13	67	13	67	103	592
BCS-05	14	97	1	2	8	46	8	38	8	54	16	80	16	96	16	96	16	96	89	508
BCS-06	11	85	6	28	15	103	15	88	15	118	14	56	14	70	14	70	14	70	107	603
BCS-07	9	39	1	2	9	30	9	21	9	39	7	18	7	25	7	25	7	25	56	185
BCS-08	3	5	1	2	2	4	2	2	2	6	0	0	0	0	0	0	0	0	7	14
BCS-09	15	159	1	2	38	327	38	289	38	365	33	210	33	243	33	243	33	243	247	1922
BCS-10	14	53	1	2	26	78	26	52	26	104	13	26	13	39	13	39	13	39	131	379
BCS-11	40	822	1	2	235	3923	235	3688	235	4158	181	2623	181	2804	181	2804	181	2804	1430	22806
BCS-12	10	173	1	2	24	285	24	261	24	309	22	243	22	265	22	265	22	265	161	1895
BCS-13	4	24	5	17	5	23	5	18	5	28	4	13	4	17	4	17	4	17	36	150
BCS-14	5	27	1	2	5	22	5	17	5	27	5	16	5	21	5	21	5	21	36	147
BCS-15	2	5	1	2	2	6	2	4	2	8	1	2	1	3	1	3	1	3	11	31
BCS-16	10	55	4	8	6	28	6	22	6	34	11	40	11	51	11	51	11	51	66	285
BCS-17	2	13	1	2	4	18	4	14	4	22	3	10	3	13	3	13	3	13	25	105
BCS-18	7	47	1	2	7	36	7	29	7	43	9	38	9	47	9	47	9	47	58	289
BCS-19	3	15	1	2	4	14	4	10	4	18	4	10	4	14	4	14	4	14	29	96
BCS-20	9	112	25	222	26	250	26	224	26	276	13	88	13	101	13	101	13	101	155	1363
BCS-21	11	140	1	2	13	103	13	90	13	116	14	103	14	117	14	117	14	117	96	765
BCS-22	8	132	1	2	33	402	33	369	33	435	29	310	29	339	29	339	29	339	216	2535
BCS-23	4	7	4	8	3	6	3	3	3	9	0	0	0	0	0	0	0	0	13	26

Table 7: Code metrics for the test project (adaptors, setup code).

	Lines of Code	average cyclomatic complexity
PublicEventAdaptor	94	1.5
MessageCheckingAdaptor	231	1.8
Entire test project	900	1.6

(around 515 seconds). This happened due to a limitation in the ERunTE tool that requires more time to simulate a missing service. The test suites for fault classes NR and LR also took more time since the ERunTE tool simulates timeouts for these PrivFESs.

5. LESSONS LEARNED

At the beginning of the study, there were two testers: tester A was an expert and tester B that needed training in the approach. We opted by a strategy similar to pair programming [23]. In the first sessions, tester A employed the approach while tester B observed and asked questions. Next, tester B took control and performed the work while tester A observed and inspected the tasks. When tester B was comfortable with the approach and tools, testers A and B worked independently. A lesson we learned is that working in pairs was effective to introduce the approach since we saved time with training and yet part of job were performed.

Other lesson is that the models were designed more efficiently in two steps:

1. *Exploratory modeling:* an initial ESG4WSC model was designed, identifying request and response events. The order among them was also modeled. During this step, the global communication is prioritized and branches and DTs are not taken into account. Generic events and comments were used to recall that these issues need to be handled in future.

2. *Test-driven modeling:* using the model designed in the previous step, a more detailed analysis was conducted to identify and model DTs (constraints, rules, and actions). Generic events and comments were removed and branches along the model were solved. The goal of this step is to set up a model that is adequate to generate test cases.

This configuration of steps was intuitively performed during the modeling of the first four services. After a phase of identification, all services were modeled in two steps and therefore with two model versions.

During the modeling of the 23 services, limitations were identified on the ESG4WSC modeling technique and tool.

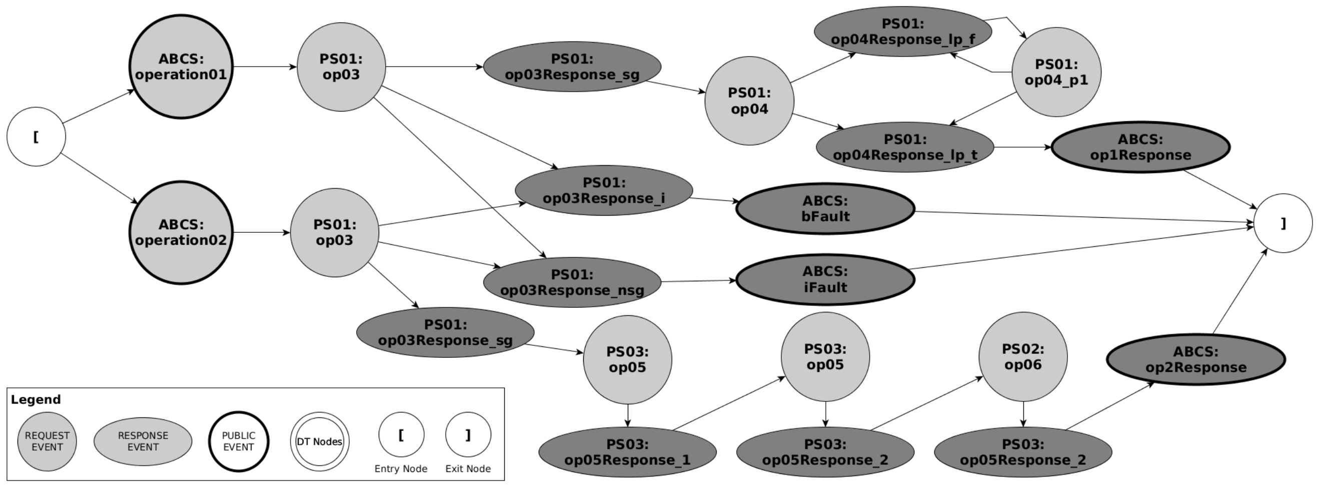


Figure 2: ESG4WSC model for the `ABC` application.

However, we learned that these limitations can be handled pragmatically. We describe the limitations and possible solutions as follows.

Event branch: it happens when some event branch is solved by some event or input parameter (in a DT) that happens previously in the workflow. In Figure 3(a), events `resp01_1` and `resp01_2` are used to select the branch in event `req03`. Although this design is acceptable in the exploratory modeling, Figure 3(b) shows a solution to support test case generation. The event sequence between the solving events (`resp01_1` and `resp01_2`) and the branch (`req03`) needs to be replicated. A drawback is that the replicated piece of model can be large and difficult to manipulate.

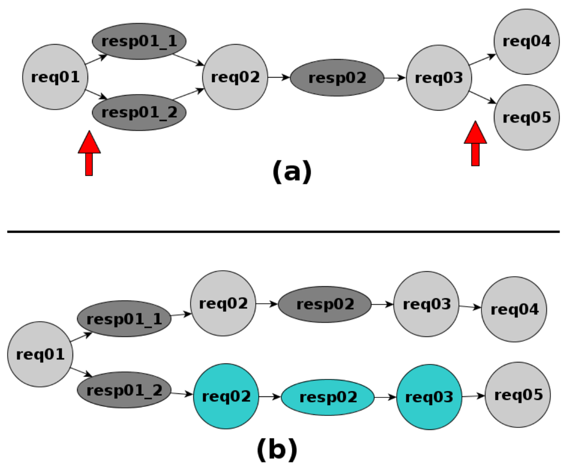


Figure 3: Model snippets for the event branch issue.

ForEach in parallel: the activity `ForEach in parallel` from BPEL 2.0 and the **`flowN`** extension of Oracle BPEL engine introduce the possibility of executing n request events in parallel. As the parallelism is implicitly introduced and the number of threads is only decided during runtime, ESG4WSCs in parallel are not able to directly represent this case in the proposed model. Figure 4(a) shows an example so that the graph within the box can be executed n times in parallel. Figure 4(b) illustrates a solution assuming that there will be two instances (threads). Thus, the tester defines the number of instances before the test generation and replicates the ESG4WSCs in parallel in a refined event. The tester needs to know and define the number of instances (threads) in modeling time, which is a drawback.

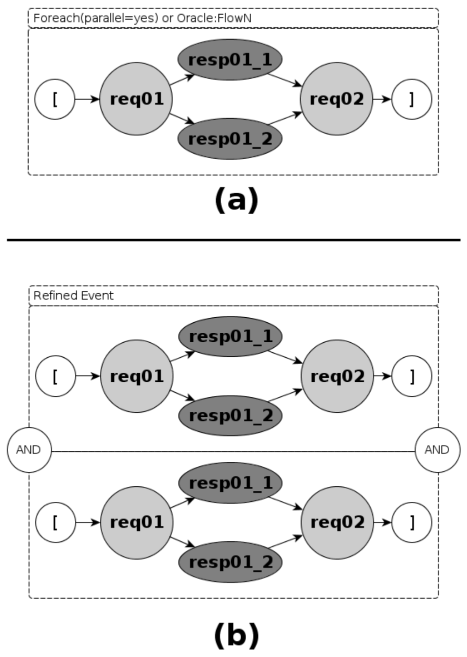


Figure 4: Model snippets for the `ForEach in parallel` issue.

Private events within loops: this case is similar to the previous one, as the number of iterations in a loop is decided during runtime. Figure 5(a) shows a model snippet usually obtained during the exploratory modeling (events `req01` and `resp01_1` are within a loop). Figure 5(b) depicts a solution so that the loop is extended in three iterations.

The tester should identify the number of iterations and repeat the instances in the model before generating the tests. The drawback here is also to have some previous knowledge on the runtime execution of the composite service.

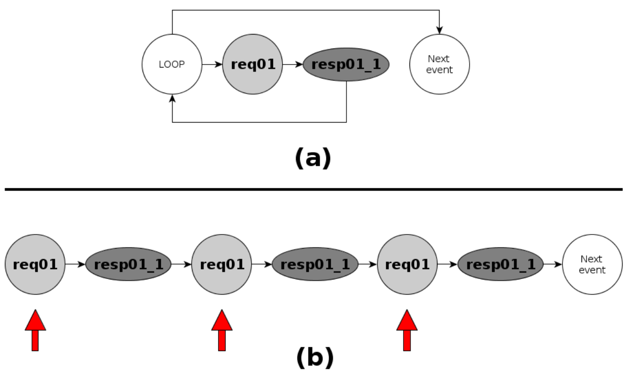


Figure 5: Model snippets for the loop issue.

Global/internal variables: BPEL engines have the concept of global variables that are independent and assigned outside the scope of the composition. However, they can be referred to within the BPEL and define different branches. Internal variables are more common in composite services implemented in traditional programming languages, instead of BPEL. These composite services tend to interact with databases and modify the workflow depending on internal variables. These global and internal variables cannot be represented in the ESG4WSC model. Figure 6(a) illustrates a case in which the branch after event `resp01_1` is solved by global variable `var`. A practical solution is to establish preconditions to the model or to the test cases. Figure 6(b) depicts the splitting in two models with preconditions (`var=true` and `var=false`). These preconditions have to be handled during the test execution that may require an extra effort from testers.

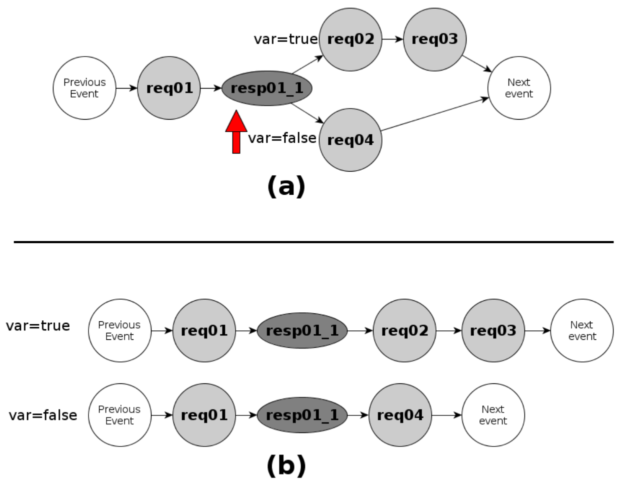


Figure 6: Model snippets for the variables issue.

Finally, the concretization and execution steps involved technical issues. We had problems to generate façade classes that interact with the services, and even accessing them due to security control. The SoapUI tool was particularly helpful by supporting the mocking of services and the production of template SOAP messages.

6. THREATS TO VALIDITY AND DISCUSSION

The first threat to this study is that the IT corporation and its applications may not be representative of the universal set of existing companies and applications that employ SOA and Web services. We recognize that the presented results cannot be generalized, albeit relevant insights were obtained about the adoption of the approach (and also MBT).

A limitation of this study is that we did not obtain information about the fault detection capability. The available subject applications selected may not be approapriate to experiment a new testing approach, once they are stable and likely have few if not none functional faults. Due to time limitations from both researchers and professionals from the IT corporation, we could not conduct further analysis. In future work, we intend to search for more evidences by, *e.g.*, analyzing the code coverage and previously-detected faults (stored in some issue tracker). However, we believe the obtained results have provided insights and preliminary evidences that motivate further investigation and investments from the industry.

Some may also wonder how representative is the adopted approach for MBT as a whole. The ESG4WSC approach was proposed inspired in the state-of-art in MBT [4, 6, 11, 21]. By the description of the approach and the results, it is not difficult to realize that an MBT process is followed. We are not aware of experiments or case studies that adopt a widely accepted MBT approach. There are several competing approaches/tools and one is selected for the experimental study usually based on the expertise, as we did in this paper. Nevertheless, comparisons with similar approaches are essential and planned as future work.

The corporation has cooperated with the PUCRS university for more than 10 years. Although the Technology Development Lab (TDL) of the corporation has a cost-effective testing process, they have recently demonstrated interest in the adoption of MBT. The effort has been spent mainly on performance testing. However, this work describes an initial effort towards the use of MBT for functional testing. As the corporation has no tool/approach that automates the testing of Web services, we could not compare the ESG4WSC approach with the way services have been tested in the company.

During the tests, no fault was detected in the SUT. This can be explained by the application's stability. The application has been released for more than two years and many cycles of testing/maintenance were performed. Although most of the generated test cases (scenarios) were likely covered by previous tests (performed by the development team), there is a lack of automated solutions for testing Web services. Our impressions are that the MBT approach can be useful in scenarios, similar to the presented one, that have complex workflows and mocking different services and sequences of messages are too complex and error-prone. However, more robust and automated tools would be essential to a large scale adoption.

We observed that there is still room for improvements in the approach automation. In the adaptors development, MessageCheckingAdaptor may use the XML schema in the WSDL files to support automatic verification of messages. Most of the written code can be generated automatically. Moreover, XPath queries that currently are evaluated in the adaptor code can be included directly in the model (and in the XML test cases as well). Thus, ERunTE-runner would be in charge of reading XPath queries and evaluated them, working as a test oracle. ESB configurations may also be automatically performed and integrated with the development environment. We also noticed opportunities for supporting performance testing. For instance, the module *esbcomp* (integrated with the ESB) of ERunTE could be extended to capture metrics used to evaluate and monitor performance, such as response time and throughput. Extensions could also be carried out in the model in a way that performance testing information is introduced and used to guide the generation of model-based performance tests.

7. RELATED WORK

The testing of service-oriented applications has been investigated to deal with specific features found in this class of software (comprehensive surveys on SOA testing can be seen in [2, 3]). However, as surveyed by Bozkurt et al. [2], just 11% of 177 analyzed papers about testing and verification of Web services have conducted some experimental validation with real world case studies. This section focuses on the Li et al. [15]'s and Wieczorek et al. [22]' studies which represent a thin effort on evaluating model-based approaches to test Web service compositions in real-world and industrial scenarios.

Li et al. [15] propose a gray-box approach to test business processes specified in BPEL. The approach involves coverage for composition, regression testing for loose-coupling SOA, and test generation. To reach this goal, the authors use three key enablers: test-path exploration, trace analysis, and regression testing selection. A testing tool named BPELTester was implemented to support the approach. They carried out a preliminary case study with 12 BPEL processes from various industry projects. While Li et al. focus mainly on white box testing, our study (which is black box testing) analyzed 23 BPEL processes and yet a service composition implemented in Java.

Wieczorek et al. [22] present a case study on the application of an MBT approach to test service choreographies in a real-world project. They use a proprietary model, called Message Choreography Model (MCM), to design the test models and generate test cases. The study involved two groups of users: integration experts and integration testers. The authors claim that the obtained results confirmed the approach applicability and the potential to save resources. This study differs from ours mainly regarding the users. While our goal was to cover several samples of real-world applications, the tester role was performed by researchers (not real users). The experience reported in this paper can be extended in future with group of users like in [22].

This paper intends to contribute on Web service testing applied to industrial cases, as well as to provide more evidences on the practical application of MBT in service-oriented applications. The experience reported herein also complements previous experimental studies conducted in controlled contexts, proving a more comprehensive evaluation of the ESG4WSC approach [1, 7].

8. CONCLUSION

In this paper, we have presented an experience report on applying a model-based approach to test Web services. A set of real-world applications of a multinational computer technology corporation was used in the study. In the first part, the results on modeling and test generation of 23 BPEL-based composite services have been described. In the second part, we have provided more details on the concretization and test execution for the `ABC` application.

The experience reported in this paper has given preliminary evidences that model-based testing, more specifically, the ESG4WSC (Event Sequence Graph for Web Service Composition) approach, is applicable to test service-oriented applications in real and less controlled scenarios within an IT corporation. From the results, we have analyzed a set of issues that impacts the approach and tools, and discussed how they can be overcome. More investigation on how to deploy the approach in an ongoing project and its cost-effectiveness is still necessary. Another topic that needs further investigation is to research the automatic generation of event-driven models out of BPEL specifications, reducing the initial effort to design test models.

Moreover, we intend to implement the approach in a component-based architecture. Hence, it is necessary to modularize each feature of the approach into components, allowing to reuse and extend the ESG4WSC approach in other domains, *e.g.*, performance testing. Once the components have been implemented, it is possible to deploy them into a software product line for MBT tools called PLeTs [18, 19]. A tool derived from PLeTs is assembled by installing a set of selected components on a common software base. As mentioned earlier, this thread aims to more robust and automated tools for a large scale adoption in industry.

9. ACKNOWLEDGMENTS

Andre T. Endo was financially supported by FAPESP/Brazil (grant 2012/21083-9). In this study, he was also partially supported by the project PROCAD/CAPES 191/2007 – *"Integrando e aprimorando atividades de pesquisa, ensino/treinamento e transferência tecnológica em teste e validação de software"*. Study partially developed by the Research Group of the PDTI 001/2013, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91. We thank CNPq/Brazil, CAPES/Brazil, INCT-SEC, and Dell Brazil for the support in the development of this work. Avelino Zorzo is a researcher financed by CNPq/Brazil.

10. REFERENCES

[1] F. Belli, A. T. Endo, M. Linschulte, and A. Simao. A holistic approach to model-based testing of web service compositions. *Software: Practice and Experience*, pages n/a–n/a, 2012.

[2] M. Bozkurt, M. Harman, and Y. Hassoun. Testing and verification in service-oriented architecture: a survey. *Software Testing, Verification and Reliability*, 23(4):261–313, 2013.

[3] G. Canfora and M. Di Penta. Service-oriented architectures testing: A survey. In *Software Engineering: International Summer Schools (ISSSE)*, pages 78–105, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *International conference on Software engineering (ICSE)*, pages 285–294, Los Angeles, USA, 1999. ACM.

[5] Eclipse.org. Eclipse, 2012. [23 May 2013].

[6] I. K. El-Far and J. A. Whittaker. Model-based software testing. In *Encyclopedia on Software Engineering*, pages 825–837. Wiley, 2001.

[7] A. T. Endo. *Model based testing of service oriented applications.* PhD thesis, Instituto de Ciencias Matematicas e de Computacao, Universidade de Sao Paulo (USP), Sao Carlos, SP, Brazil, Apr. 2013.

[8] A. T. Endo and A. Simao. Model-based testing of service-oriented applications via state models. In *IEEE International Conference on Services Computing (SCC 2011)*, pages 432–439, Washington, DC, USA, 2011.

[9] A. T. Endo and A. S. Simao. A systematic review on formal testing approaches for web services. In *4th Brazilian Workshop on Systematic and Automated Software Testing (SAST)*, pages 89–98, Natal, Brazil, 2010.

[10] Eviware. soapUI, 2012. [23 May 2013].

[11] W. Grieskamp, N. Kicillof, K. Stobie, and V. A. Braberman. Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability*, 21(1):55–71, 2011.

[12] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guï£¡zar, N. Kartha, C. K. Liu, R. Khalaf, D. Konig, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. OASIS web services business process execution language (WSBPEL) v2.0, 2007.

[13] junit.org. JUnit, 2012. [23 May 2013].

[14] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0, 2005.

[15] Z. J. Li, H. F. Tan, H. H. Liu, J. Zhu, and N. M. Mitsumori. Business-process-driven gray-box soa testing. *IBM Systems Journal*, 47(3):457–472, 2008.

[16] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton. OASIS reference model for service oriented architecture 1.0, 2006.

[17] MuleSoft. Mule ESB: Open source ESB and integration platform. [23 May 2013].

[18] E. M. Rodrigues, L. D. Viccari, A. F. Zorzo, and I. M. Gimenes. PLeTs Tool - Test Automation using Software Product Lines and Model Based Testing. In *Proceedings of the 22th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 483–488, Redwood City, California, USA, 2010.

[19] M. B. d. Silveira, E. d. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, and F. M. de Oliveira. Generation of Scripts for Performance Testing Based on UML Models. In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 258–263, Miami, Florida, USA, 2011.

[20] SourceForge.net. Eclipse metrics 1.3.6, 2005. [23 May 2013].

[21] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[22] S. Wieczorek, A. Stefanescu, and A. Roth. Model-driven service integration testing - a case study. In *International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 292–297, Washington, DC, USA, 2010. IEEE Computer Society.

[23] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25, 2000.