Specification -based Test Cases Generation for Multi-Level Service Composition

Shymaa Sobhy, Mahmoud Hussein and Ashraf B. El sisi
Faculty of Computers and Information,
Menofia University, Egypt ,
shaymaa.abdelaal@ci.menofia.edu.eg , mahmoud.hussein@ci.menofia.edu.eg and
ashraf.elsisi@ci.menofia.edu.eg

***Abstract*- Testing is the traditional validation method in the software industry. To ensure the delivery of high quality and robust service-oriented applications, testing of web services composition has received much attention. These services have become more and more complex, where they have to cope with strict requirements of business processes and their dynamic evolution, and interactions among different companies. In this context, the analysis and testing of such services demand a large amount of effort. To reduce the effort required for web-services testing, in this paper, we propose a specification-based approach to automatically generate test cases for web services composition that is modeled at different levels of abstraction. This approach specifies a service structure as multi-level models. To generate the test cases, it checks if the first level of the model has a parallel execution or a decision table to be solved by an algorithm that solves Chinese postman problem. Then, it identifies paths for last level of the model and relates the results of all levels with each other. To evaluate our approach, we applied it to four cases study using our developed tool. Compared to exiting approaches, our approach reduces testing cost and execution time, and increases testing reliability.**

Keywords--Service-oriented Applications, Web Services Composition, Model-based Approach, and Event-driven Model.

I. INTRODUCTION

Service-oriented architectures (SOAs) and web services have been used to enable loosely-coupled, distributed applications by using independent and self-contained services [1]. These services can be combined in a workflow that characterizes a new composite service. The resulting composite service is also called a web service composition (WSC [2, 4]. Such services have complex communications where the service behavior depends not only on the composition but also on the integrated services. Therefore, the testing process becomes complicated.

A common problem in testing any kind of application is to automatically generate meaningful test cases [3, 5]. The strategy of using models for test case generation is known as model based testing [6, 7 and 8]. Web applications are evolving rapidly, as many new technologies, languages, and programming models are used to increase the interactivity and the usability of web applications [8, 14]. This inherent complexity brings challenges to modeling, analysis, testing, and verification of this kind of applications.

To reduce the complexity of designing large composite services, the designers apply service decomposition where services are model at different levels of abstraction [10]. Many techniques have been introduced to support automatic testing of composite services. For example, a multi-observer architecture is proposed to detect and locate faults in composite web services [6], and a new model to describe a service choreography that manipulates data flow by means of XPath queries is introduced [18]. In addition, a model-based integration testing for service choreography using a proprietary model, called message choreography model (MCM) is proposed [24]. But, these techniques do not support testing service compositions that are modeled at different levels of abstraction.

In this paper, we propose an approach for generating test cases for composite web services that are modeled at different levels of abstractions. We use a model based technique called "Event Sequence Graph for Web Services Composition" to generate cost-effective test cases for service compositions that modeled at one level [9]. We improve that technique to generate test cases for web services modeled at a multiple level of abstraction. Our approach check if the first level has a parallel execution or decision tables to be solved using an algorithm that solves Chinese postman problem for a directed graph to identify paths by generating a Euler network. Then it identifies paths for last level of the service model by same algorithm. Finally, the approach relates results of all levels with each other. We evaluate our approach by applying it to four case studies that have different complexity. We also developed a tool to generate test cases for the case studies.

The remainder of this paper is organized as follows. Related work is analyzed in Section 2. Proposed approach in Section 3. Section 4 experimental results. Section 5 presents conclusion.

II. Related Work

This section introduces work that is related to our approach. We also explain in detail an existing technique that we use as a base for our approach. Web service testing has been studied intensively in the last years [1, 3, and 4] with a particular effort on formal testing (for a systematic review of the literature, see [17]). The service testing provides the reliability analysis and formal reviews to the service. In the following, we describe some of the existing approaches.

A model that describes a service choreography that manipulates data flow by XPath queries is introduced [18]. In the choreography, XPath queries can handle different XML schema files. This work is focused on test case generation for web service composition but modeled at one level only. Benares [6] proposes a multi-observer architecture to detect and locate faults in composite web services. The proposed architecture is composed of a global observer and local observers that cooperate to collect and manage faults found in the composite service. Their approach aims at testing the service composition that modeled as one level. Wieczorek [24] proposes a model-based integration testing for service choreography using a proprietary model, called message choreography model (MCM). The work is close to the proposed approach, with difference that, our approach uses a more abstracted model compared to MCMs, which form a domain-specific language created to design service choreography. Fevzi Belli and Christof Budnik proposed an approach for generation and selection of test cases based on statecharts [22]. This approach with scalable way uses regular expressions and regular expressions are used in the test process. But here we introduce event sequences graph that generate tests for multi-level graphs. JanTretmans proposed an overview of formal, model-based testing in general and of model-based testing for labeled transition system models in particular [23]. Also, it introduces the same concept of model-based testing but this still not deal with multilevel graphs. Belli and Endo [9] have proposed an event-based model, named (ESG4WSC) Event Sequence Graph for Web Services Composition that represents the request and response messages exchanged between services involved in a WSC. This approach is for generate tests for graph that modeled at one level only. We will improve it to deal with multilevel graphs. Table 1 shows algorithm for deriving CESs from an ESG4WSC.

III. The Proposed Approach

This section introduces firstly background secondly present a running example thirdly the proposed approach.

A. Background

This section introduces formal notions and algorithms that are relevant to the proposed approach. It also presents the underlying fault model, test case generation and minimizing test set [9, 15 and 21]. Event Sequence Graph for Web Services Composition: Event Sequence Graph (ESG) for Web Services Composition represents the request and response messages exchanged between services involved in a service composition. When a given event is refined by input parameters that determine the next events, decision tables (DTs) are associated to augment this representation. Decision tables are widely employed in information processing and are also traditionally used for testing. Table 2 shows decision table for xloan case study. ***Definition 1:*** A (simple/binary) decision table DT ={C, E, R} represents events that depend on certain constraints, where: C is nonempty finite set of constraints (conditions), which can be evaluated as either true or false, E is the nonempty finite set of events, and R is the nonempty finite set of rules each of which forms a Boolean expression connecting true/false configurations of constraints and determines executable or waited events. ***Definition 2:*** An ESG for web service ESG4WSC = {V, E, M, R, DT, f, Ξ, Γ} is a directed graph, where:

- V is a nonempty finite set of vertices representing events;
- E⊆V x V is a finite set of arcs (edges);
- M is a finite set of refining Event Sequence Graph for Web Services Composition models;
- R⊆V x M is a relation that specifies which Event Sequence Graph for Web Services Compositions are connected to a refined vertex;
- DT is a set of DTs that refine events according to function f;
- F: V→DT {ε} is a function that maps a decision table dt ∈DT to a vertex v ∈ V. If v ∈ V is not associated with a DT, then f(v)=ε;
- Ξ , Γ ⊆V are finite sets of distinguished vertices with ξ ∈ Ξ and γ ∈ Γ called entry nodes and exit nodes, respectively, wherein for each v ∈ V there exists at least one sequence of vertices (ξ ,v0, . . .,vk) from ξ ∈ Ξ to vk=v and one sequence of vertices (v0,…,vk, γ) from v0=v to γ∈ Γ with (vi,vi+1) ∈E for i=0,….,k-1and v≠ ξ, γ. ***Definition 3:*** Let V be as in Definition 2. Then, the set of vertices V is partitioned into V_e, $V_{refined}$,

V_{req}, and V_{resp} that is, $V = V_e \cup V_{refined} \cup V_{req} \cup V_{resp}$ and V_e, $V_{refined}$, V_{req}, and V_{resp} are pairwise disjointing, where:

- Ve is a set of generic events,
- Vrefined = $\{v \in V \setminus \exists m \in \bigwedge (v, m) \in R\}$ is a set of vertices refined by one or more Event Sequence Graph for Web Services Compositions. A refinement with more than one Event Sequence Graph for Web Services Compositions represents behavior running in parallel,
- Vreq is a set of vertices modeling a request to its own interface/operations (public) or an invoked service (private), and Vresp is a set of responses to a public or private request. Therefore, it is also remarked as public or private. ***Definition 4:*** Let DT be defined as in Definition 2. Then, the set of decision tables is partitioned into DT_{seq} and DT_{input}, where: DTseq is the set of DTs that model the execution restrictions for following events and DTinput is the set of DTs that model constraints for input parameter of invoked operations.

B. Fault Model: An ES (see Definition 5) describes a specific execution of a WSC that has to be enforced during testing. Thus, it is expected that exactly those events in the specified order are executed [9].

C. Test Case Generation: To cause and control a specific CES of the WSC, it is often inevitable to take control of partner services because they communicate with the system under test (SUT). And the flow of the WSC might depend on a returned response. The modeled constraints of DTs enable to validate the data passed to the service operations [13, 15 and 19].

D. *Minimizing the Test Sets*: The total number of CESs with minimal total length that cover the ESs of a required length is called Minimal Spanning Set of Complete Event Sequences (MSCES). Eentire walk occurs when the CES contains all EPs at least once [28]. The Chinese Postman Problem is expected to have a higher degree of complexity than MSCES problem introduced here as the edges of the ESG are not weighted [11, 12, 23 and 20].

E. Running example: The example involves three services: LoanService (LS), BankService (BS), and BlackListInformationService (BLIS). LoanService represents the business process xLoan. It has three operations: request, cancel, and select. BankService represents the financial agency that approves (or not) loans, and provides loan offers to its clients. The operations used in the example are approved, offer, confirm, and cancel. The BlackListInformationService provides an operation checkBL to check if a client has debits with other financial organization. The example is extended to add parallel flow in the process by including CommercialAssociationService (CAS). Similar to BlackListInformationService, CAS provides operations to check whether a client has debits with other commercial organization. In the extension, both services are supposed to be called in parallel. If the client has debit according to one of them, the client needs the bank approval [9]. The multi-level Event Sequence Graph for the running example (xloan) is in Figure 1.

F. *The proposed approach*: When the input is multi-level graphs, we apply our proposed approach. For increase efficiency, we apply the CPP algorithm only to two levels the first level and the last level. And then relate the results to each other. First, we apply (CPP) to first level. Second, we check if last level is a parallel execution. If true, we find the CESs for it. Then, we identify the valid successor for each CES with respect to the DT. Thirdly, we find the CESs of the inner sublevels. Then, we make replacing operation. Table 3 shows the algorithm for the proposed approach.

Table 1. An algorithm for deriving CESs from an Event Sequence Graph

Input: An Event Sequence Graph for Web Services Composition (one level).
Output: CESs.
1. **Foreach** (vertex = refined vertices) **do**
  Go to step 3 to Generate CESs for the refined vertices first.
2. **Add** multiple edges (representing EPs) to Event Sequence Graph for Web Services Compositions:
  If (refined vertex has a DT restricting the ongoing execution)
    Identify the valid successor for each CES with respect to the DT.
    Add an edge from the refined vertex to the allowed successor.
  Else
    Add an edge from the refined vertex to the successor (there should be only one) for each CES.
3. **Generate** CESs according to the CPP algorithm (i.e., cover all EPs by CESs of minimal total length).
4. **Replace** refined vertices in the resulting CES set of Step 3 with the CESs derived in Step 1 with respect to their allowed successors.
Return CESs

Table 2. The decision tabel for check Refined

Dtcheck	***R1***	***R2***	***R3***	***R4***
Event :BLIST: inBlist happen	T	T	F	F
Event :BLIST: NotinBlist happen	F	F	T	T
Event :CAS:DebetorTrue happen	T	F	T	F
Event :CAS:DebetorFalse happen	F	T	F	T
BSoffer				√
BSapproveBank	√	√	√	

Table 3 . The proposed approach

Input: Event Sequence Graph for Web Services Compositions that decomposed into different levels of abstraction (multiple levels).
Output: CESs.
Step 1: **Apply** (CPP algorithm) to (Frist level) to generate CESs for it and save it to an array data structure.
If (last level = parallel execution) **Go to** step 2
If (refined vertex has a DT restricting the ongoing execution)
Identify the valid successor for each CES with respect to the DT.
Else
Put the sequence from the refined vertex to the successor (there should be only one) for each CES.
Step 2: **Apply** (CPP algorithm) to (last level) to generate CESs for it and do and operation on results.
Step 3: **Get** the CESs of the inner sublevels manually.
Step 4: **Replace** the abstracted frist level node in CESs from Step 1 with other sublevels nodes result from step 3.
Step 5: **Replace** CESs from step 4 with CESs result from Step 2.
Return CESs

I.

IV. EXPERIMENTAL RESULTS

In this section, we show applying the proposed approach to the xloan example then show the evaluation by applying it to other three cases study

A. Applying proposed approach to the xloan example:

We will now apply the proposed approach for the xloan case study. Table 4 apply CPP algorithm to the first level that list two only sequences [16]. Table 5 shows the application of CPP algorithm to the last level to get the CESs and shows four sequences. Table 6 shows applying the AND operation to them. Finally, we go to replacing operation to relate the results. Table 7 shows the CESs of level 1 after replacing by level 2. Table 8 shows replacing results with results from table 6 related to DT constrains. The new propose approach is applied only at two levels rather than calling it three time as done at the old approach and its result is related to each other. This increase efficacy by reduce execution time and number of iteration of CPP algorithm.

B. Evaluation

We evaluate our approach by applying it to three other cases studies. The first one called Travel Agent Service. It provides a set of facilities to query and book a trip. It interacts with two services: ISELTA-hotel and Airlines services. It combines these two services to provide operations for searching and booking a travel involving flight and hotel reservations.

As the flight ticket and hotel reservation are essential in any travel, a successful booking using this service guarantees hotel and flight reservations [9]. The second called ABC services case study [25]. This service interacts with three other services: PartnerService01 (PS01), PartnerService02 (PS02), and PartnerService03 (PS03). It focuses on the flow of messages triggered by the operations of ABCService. The third called BCS-05 service which is a version from Business Connectivity Services. We have applied the old and proposed approach to these three cases studies (for the details, see [26]). In the following, we give the information about the case studies including test model information for higher length, and the execution time and number of iteration for the old algorithm and the proposed approach.

Table 4. CES1,2 after apply CPP for level 1of multi level graph

CES1	Start LSrequestLoan BSapproveBank BSapproved BSoffer BSOffers LSreplyOffers LSSelectOffers LSWrongOffer LSSelectOffers BSconfirmBank LSreplySelect End
CES2	Start LSrequestLoan **check** BSapproveBank BSapproved BSoffer BSOffers LSreplyOffers LSSelectOffers LSWrongOffer Timeout >2h BScancelBank End

Table 5. CESG after apply cpp for last level

CES1	[BLIS:checkBL , BLIS:inBList]

Tabel 6. CESG after AND operation for last level

CES1	[BLIS:checkBL BLIS:inBList] || [CAS:inDebtorsList CAS:debtorsTrue]

Table 7. CES1,2 after replacing level 2 for the proposed approach

CES1	No replace
CES2	Start LSrequestLoan **checkBLIS** BSapproveBank BSapproved BSoffer BSOffers LSreplyOffers LSSelectOffers LSWrongOffer Timeout>2h BScancelBank End
	Start LSrequestLoan **checkCAS** BSapproveBank BSapproved BSoffer BSOffers LSreplyOffers LSSelectOffers LSWrongOffer Timeout>2h BScancelBank End

Table 8. Replace with the original node from step one by DT Constrains

CES1	No Replace
CES2	Start LSrequestLoan **BLIS:checkBL BLIS:inBList CAS:inDebtorsList CAS:debtorsTrue** BSapproveBank BSapproved BSoffer BSOffers LSreplyOffers LSSelectOffers LSWrongOffer Timeout>2h BScancelBank End

C. Generating Event Sequences

A phenomenon in testing interactive systems that most testers seem to be familiar with is that faults can be frequently detected and reproduced only in some context. Further, the coverage criteria can be made more powerful by increasing the value of length coverage to be obtained thereby further reducing any negative effect of reducing the length of tests on the fault detection effectiveness. This makes a test sequence of a length greater than 2 is necessary since repetitive occurrences of some subsequences are needed to a failure to occur. Summary of the results are in Table 9. The Xloan service has execution time 280 ms for length 3 and 276 ms for length 4 while the number of test cases for length 3 is 8 and for length 4 are 11. The Travel agent service has execution time 276 ms for length 3 and 350 ms for length 4 while the number of test cases for length 3 is 15 and for length 4 are 17. The ABC services has execution time 359 ms for length 3 and 400 ms for length 4 while the number of test cases for length 3 is 9, and for length 4 are 10. The BCS-05 Service has execution time 372 ms for length 3 and 500 ms for length 4 while the number of test cases for length 3 is 15 and for length 4 are 18.

D. The Execution Time and Number of Iteration

The execution time and the number of iterations of our approach and the old algorithms are summarized in Table 10. First, the Xloan service has execution time of 414 ms for old algorithm and 260 ms for proposed approach while the number of iterations for old algorithm is 16 and for proposed approach are 10. Second, the travel agent service has execution time 860 ms for old algorithm and 607 ms for proposed approach and number of iterations for old algorithm is 34, and for proposed approach are 18. Third, the ABC service has execution time of 285 ms for old algorithm and 209 ms for proposed approach while the number of iterations for old algorithm is 10 and for the proposed approach are 8. Third, the BCS-05 service has execution time 332 ms for old algorithm and 274 ms for proposed approach, number of iteration for old algorithm is 17, and for proposed approach are 15. We found that when applying the old algorithm to the decomposed graphs this will require repeated algorithm N time equals to N levels of graphs which increases the execution time and the number of iterations of algorithm. Figure 2 shows execution time in millisecond and Figure 3 shows number of iterations for the old and the proposed approaches.

Table 9. Test model information for higher length

Length greater than 2	Execution time		Test cases	
	Length (K=3)	*Length (K=4)*	*Length (K=3)*	*Length (K=4)*
Xloan service	280ms	377ms	8	11
Travel agent	276ms	350ms	15	17
ABC services	359ms	400ms	9	10
BCS-05 Service	372ms	500ms	15	18

Table 10. The execution time and number of iteration for the old algorithm and our proposed approach

Cases	Execution time		Iterations	
	Old	*proposed*	*Old*	*proposed*
Xloan service	414	260	16	10
Travel agent service	860	607	34	18
ABC services	285	209	10	8
BCS-05 Service	332	274	17	15

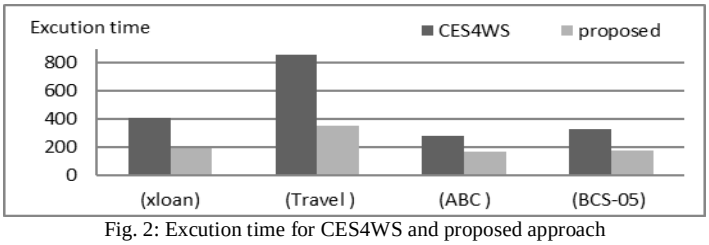

Fig. 2: Excution time for CES4WS and proposed approach

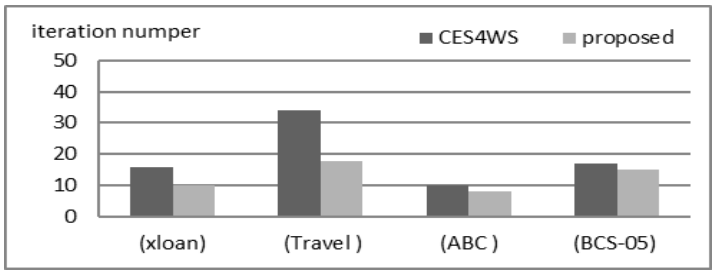

Fig. 3. Iteration time for CES4WS and proposed approach

V. CONCLUSION

Testing is the most critical and expensive phase of the software development life cycle. In this paper, we have improved a technique called Event Sequence Graph for Web services compositions to generate cost-effective test cases for WSCs that decomposed at different levels of abstraction. We found that when the input is multiple graphs, the old approach generates unrelated test cases and takes more iteration and execution time. When our approach is applied, it works properly with less iteration and less execution time and gives related test cases. We introduce algorithms to generate test cases from multi-level graphs. We also generate test cases for length greater than 2 that is necessary since repetitive occurrences of some subsequences are needed to a failure to occur/reoccur. We have evaluated our approach by four case studies with different complexity, parallel execution and decision tables. In the future, we will make our approach holistic to perform positive testing as we do here and also to test undesirable situations (i.e. negative testing) based on the service model. We will also perform other testing stages such as test cases execution. Finally, it is essential to conduct experimental comparisons with other approaches, such as structural testing for web service compositions.

REFERENCES

[1] G. Canfora, and M. Penta, "Service-oriented architectures testing: a survey", In Software Engineering: International Summer Schools (ISSSE), Springer, 2009.

[2] M.Papazoglou,W. Heuvel," Service oriented architectures: approaches, technologies and research issues", The International Journal on Very Large Databases The VLDB journal,vol. 16(3),pp. 389-415,2007.

[3] D. Kung, C.Liu, and P. Hsia, "A model-based approach for testing Web applications", In: Proc. of Twelfth International Conference on Software Engineering and Knowledge Engineering, Chicago, July, 2000.

[4] M.Schmidt, B.Hutchison,and P.Lambros, "The enterprise service bus: making service-oriented architecture real", IBM Systems Journal, vol. 44(4), pp.781-797, 2005.

[5] F. Lars, T.Jan, and R. de Vrie., "Towards model –based testing of web services", International Workshop on Web Services Modeling and Testing, 2006.

[6] A. Benharref, R.Dssouli, R. Glitho,and M.Serhani,"Towards the testing of composed Web services in 3rd generation networks", In IFIP International Conference on Testing of Communicating Systems (TESTCOM), Vol. 3964, pp. 118-133,2006.

[7] F.Belli,and B.Christof, and W.Lee,"Event-based modeling, analysis and testing of user interactions: approach and casestudy", Software Testing, Verification and Reliability, vol.16(1), pp.3-32 ,2006. Van der Aalst WMP. Formalization and verification of event-driven process chains. Information and Software technology,1999;

[8] F.Robert, and B.Rumpe,"Model-driven Development of Complex Software", A research roadmap, Future of Software Engineering,IEEE Computer Society,pp. 37-54,2007.

[9] F.Belli, A.Endo, M. Linschulte, and A.Simao, "A holistic approach to model-based testing of Web service compositions", Software – Practice and Experience, vol. 44(2), pp. 201-234 ,2014.

[10] F.Belli,and B.Christof, and W.Lee,"Event-based modeling, analysis and testing of user interactions: approach and casestudy", Software Testing, Verification and Reliability, vol.16(1), pp.3-32 ,2006.

[11] F.Belli,and M. Linschulte,"Event-driven modeling and testing of real-time Web services", Service Oriented Computing and Applications, 4(1), pp.3-15, 2010.

[12] F.Belli, A.Endo, M.Linschulte,and A.Simao,"Model-based testing of Web service compositions", Service Oriented System Engineering (SOSE), pp. 181-192, IEEE, 2011.

[13] Z.Hong, P.Hall, and J.May,"Software unit test coverage and adequacy", ACM Computing Surveys (CSUR),vol. 29(4), pp.366-427 1997.

[14] A.Paul, and J.Offutt, "Introduction to software testing", Cambridge University Press, 2008.

[15] F.Belli, and C.Budnik,"Minimal spanning set for coverage testing of interactive systems",In First International Colloquium on Theoretical Aspects and Computing (ICTAC),Springer, pp. 220-234, 2004.

[16] Y. Lin, and Z.Yongchang,"A new algorithm for the directed Chinese postman problem" ,Computers & operations research ,vol.15(6), pp. 577-584,1988.

[17] A.Endo, A.Takeshi, and A.Simao,"A systematic review on formal testing approaches for Web services", Brazilian Workshop on Systematic and Automated Software Testing, International Conference on Testing Software and Systems, pp.89, 2010.

[18] L.Mei, C.W, and T.Tse, "Data flow testing of service choreography", Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 151-160, 2009.

[19] V.Stoyanova, P.Dessislava, and I.Sylvia, "Automation of test case generation and execution for testing web service orchestrations",Service Oriented System Engineering (SOSE), IEEE, pp. 274-279, 2013.

[20] F.Belli, and C.Budnik,"Towards optimization of the coverage testing of interactive systems", Computer Software and Applications Conference, Vol. 2, pp. 18-19, IEEE, 2004.

[21] W.Douglas " Introduction to graph theory. ",Vol. 2,Upper Saddle River: Prentice hall, 2001.

[22] F.Belli and A.Hollmann, "A holistic approach to testing of interactive systems using statecharts". InProceedings of 2nd South-East European Workshop on Formal Methods (SEEFM 05), South-Eastern European Research Center SEERC 2005 (pp. 1-15).

[23] J.Tretmans ,"Model-based testing and some steps towards test-based modelling". InFormal Methods for Eternal Networked Software Systems 2011 (pp. 297-326). Springer Berlin Heidelberg.

[24] A. Cavalli, TD.Cao, W.Mallouli,"Webmov: A dedicated framework for the modelling and testing of web services composition",InWeb Services (ICWS), 2010 IEEE International Conference,2010 Jul 5 (pp. 377-384),IEEE.

[25] A.Endo,"Using models to test web service-oriented applications.", an experience report, 2012.

[26] https://www.docdroid.net/cbR9qpv/version7detalid.pdf.html, Last accessed: July 2016.