Automatic Test Case Generation from UML State Chart Diagram: A Survey

Yasir Dawood Salman and Nor Laily Hashim

Abstract The need for developing high-quality systems and applications with the minimum faults and error in them has been increased recently. In addition, a question of time and expenses that should be as low as possible always is concerned. Thus, it needs to be updated with the testing techniques that are more structured and automated which are used during the analysis and design phase. The significant role of automated testing techniques is that it helps speeding up the delivery of services of the products to the market with little chances of loss, and increases the software value. If the purpose were to decrease the expenses and getting closer to technology, then testing automation would be a crucial choice. The aim of this survey is to improve the understanding of UML diagram based testing techniques. Test case generation from state chart has been have focused on. Also, classify the various research approaches to their methods. The issues of test coverage associated with these methods have been discussed also.

1 Introduction

Testing is considered an essential part of the today's software development and has proven to be a useful tool to enhance the programming coding quality. Testing can help to detect the software bugs which compilers are not usually able to detect [1]. However, whether a program is correct or not, its correctness can be guaranteed through using the testing in order to enhance the quality of its code. The task of testing in software is so complicated, that it includes the software evaluation to show whether it meets the needs. Theoretically or practically, this is usually a difficult task.

Y.D. Salman (✉) · N.L. Hashim
Universiti Utara Malaysia, Kedah, Malaysia
e-mail: yasir.dawod@gmail.com

N.L. Hashim
e-mail: laily@uum.edu.my

H.A. Sulaiman et al. (eds.), *Advanced Computer and Communication Engineering Technology*, Lecture Notes in Electrical Engineering 362,
DOI 10.1007/978-3-319-24584-3_12

Come as no surprise that for the past decade a great amount of research work has been conducted over automatic test case generation [2–8]. Therefore, a great amount of different techniques to generate the test case has been explored intensively and propelled. In contrast, software systems have gotten to be progressively complex, for instance, with components developed by diverse vendors, utilizing distinctive techniques within diverse programming languages and actually running on diverse platforms.

State chart diagrams in UML can be used to construct the dynamic aspects of a system. This diagram consists of transitions, states, actions, and events [9] and by emphasizing the flow of control from state to state as it shows a state machine. State chart is comprehensive Finite State Machine (FSM) with concurrency, hierarchy, and communication, and these extensions allow small diagrams to express complex behavior in a modular method [10].

The purpose of generating test case using UML state chart diagram is to verify the relations between the behavior, state transition, state, action, and event. This technique is used to determine if one can fulfil the system specifications through the state based motion of the system. In the state based system, there are three reasons for the fault. The first is when the state diagram cannot accurately transfer the system function specification. Secondly is when the state chart diagram configuration is wrongly or unreliable. The final is when converting form a state chart diagram to programmable code [11]. Several of surveys and reviews study have been conducted on test case generation [12–18], where they cover in general the generation of test cases a variety of inputs. This paper on the other hand focuses on varies methods of test case generation from UML state chart diagram, where the commonalties and trends in the methods used are explored. The following section presents test case generation approaches using state chart diagram.

The paper is structured as follows. Section 2 a review related work of automatic test case generation using UML state chart diagram. Section 3 presents the trends and gaps identified from the survey. Finally, Sect. 4 outlines conclusion and future research work.

2 Test Generation Approaches Using UML State Chart Diagram

Researchers such as [19, 20] have paid considerable attentions to automatic test case generation from UML diagrams. At the same time, there were more researchers who work on generating test cases from UML state chart diagrams [21–23].

Kansomkeat and Rivepiboon [20] develop a transformation method from state chart diagrams into intermediate diagrams that are used to generate test sequences. The test cases are generated automatically from state chart diagrams created by the Rational Rose tool. The testing coverage criterion is used to guide the generation of

test cases and to produce the intermediate model Testing Flow Graph (TFG) from the all-state coverage and all transition coverage. Based on their fault detection abilities, their test cases measured the effectiveness of test case generation. From the generated test cases, results of simple test experiments revealed high effectiveness in the test case generation. However, usually more than one object often participates in the execution of a use case. Therefore, it will be difficult to test using this approach with the chance of such behavior occurred. In addition, it does not generate multi test data due to the lack of coverage choices.

Offutt et al. [24] developed a method to automatically generate the test cases from state chart diagrams, by changing events for Boolean class attributes. The developments of many useful coverage criteria that were centered on the state chart diagrams were effective. Class-level testing is the aim of their approach. This method attains transition-pair coverage, transition coverage, and full predicate coverage.

Gnesi et al. [19] offered a formal test case generation by providing a mathematical basis for conformance testing and automatic test case generation for state chart diagrams that was established on an operational semantic. With transitions labelled by input/output pairs, they proposed a formal conformance testing relation for input enabled transition systems. In order to succeed in the specified requirements, testing software is identified as conformance testing. Considering the formal specification, a conformance relation defines the accuracy criterion of the implementation. However, in order to practice this technology proper test selection strategies are needed to use the test generation algorithm practically.

Briand et al. [25] focused on creating a methodology using state chart diagram to define the system state required for each event or transition, where parts of the paths, input values for the parameter for all actions and events associated with transitions have to be defined. Their work generates a test case specification involving a possible sequence of transitions. A requested sequence tree is constructed and then it will be used to develop the test restraints for the transition sequences to test it, to get the interactions among state dependent objects in their work.

Li and Lam [26] presented an approach to generate test sequences from state chart diagrams using ant colony optimization. A UML state chart diagram is transformed into intermediate model called a directed graph. By exploring the directed graph by a group of ants cooperatively, the test sequences are generated. From this generation they achieve the all-state coverage in the coverage criteria.

Santiago et al. [27] presented a method to automate test case generation from UML state chart diagrams using a software specification model. This method converts the UML state chart diagrams model into an XML-based language table, and by using the Perform Charts tool, and generates intermediate model from state based on control flow. Their indication is to determine that by using a higher-level technique, such as UML state chart diagrams; there will be possibility to represented complex software with clarity and rich details. UML state chart diagrams can enable to model a complex system more realistically and provide hierarchy and parallelism for it. Although these conditions are not enough in order to guarantee

that a test case generation approach is successful, but still they show an improvement when they have been compared with the use of the Condado as an unconnected tool with a FSM specification. In addition, the Condado implements the switch cover method for the control part. A switch is a transition-to-transition pair, and their method generates test cases to cover all pairs of transitions in the model in the coverage criteria.

Murthy et al. [28] suggested a new foundation for generating test cases using the UML state chart diagram as a base model of behavior. They also defined a test ready state chart diagram, which indicates that the model is ready with data for a test generator to generate test scripts automatically from it. To generate the paths, they start from the start node with a state transition and reconnoitering each next node subsequent of its state transitions, if any along a state transition is satisfied, it provides guard condition. They solve the problem of generating test case from UML state chart diagram by defining all the sentential forms derivable from an equivalent extended context free grammar model. Additionally, coverage criteria that were achieved are path coverage and basic path coverage.

Ali et al. [21] have projected a method for state-based integration testing. Their work forms an intermediate test model named State Collaboration Test Model (SCOTEM) from the corresponding state chart diagrams and UML collaboration diagrams. SCOTEM copies all possible paths for object state changes that message sequences may cause. Then SCOTEM produces test paths centered on several coverage criteria. For them revealing the state-dependent interaction errors is the goal behind the generated test cases. Their work reflects the analysis of all possible states of cooperating levels in an interface.

Santiago et al. [29] presented an environment name automated Generated Test case based on State Charts (GTSC) which allows a test designer to generate test cases based on state charts test criteria and FSM methods. This interesting characteristic allows test sequence generation from both state chart and FSM techniques based on the same FSM. However, other comparisons needs to be made namely all-paths-k-C0-onfiguration of the state chart Coverage Criteria Family (SCCF) as well as the round-trip route testing offered by Binder [30] and all-paths-k-configurations. Similarly, there can be more comparisons between the latest FSM-based methods, such as state counting, and some SCCF criteria. Such an analysis will be enabled with the help of mutation testing by GTSC in applying these test criteria methods.

Kosindrdecha and Daengdej [22] proposed a new method to generate and prepare both test data and test case based on state chart diagram, called "TGfMMD" method. This method has been developed to verify the state chart diagram before generation of both test cases, and test data from extended state chart diagram. However, this method has not yet been tested with a complex state chart diagram.

Swain et al. [31] proposed a novel technique to generate test cases automatically from UML state chart diagram and activity diagram. They construct it based on the model an intermediate representation, that they named it state-activity diagram (SAD). They generate the test case from the use of SAD generation, Depth First Search (DFS) and mutation analysis. In addition, in order to detect harmonization of

state chart diagram as well as activity diagram faults within a use case of the system exercise, an activity synchronization in the context of multiple state combinations has been used. They also achieves transition coverage and state/activity path coverage. For the testing, they have implemented a prototype tool based on their approach. However, in their work the tester should select the test data for each test case manually.

Shirole et al. [32] have also worked on the automatic generation of test case using UML state chart diagram. They used the Genetic Algorithm (GA) as medium for their tool by combining information from state chart diagram in it. They propose a search-based approach to handle infeasible paths and test data generation. They use the following steps to generate the test cases. First is to transform the UML Specifications into Extended Finite State Machine (EFSM), secondly to transform the EFSM into Extended Control Flow Graph, third is to generate test sequences using GA and DFS, and finally, select the test cases using data-flow techniques. In the coverage criteria, they focused on state cover, transition cover, all-definition cover, and all du-path. However, the state chart diagrams that they considered are very simple, what will lead to less coverage when dealing with scenarios that are more complex. Also because of using DFS and fitness function, all path coverage is not fully obtained.

Li et al. [33] presented a test case generation approach, which takes UML state chart diagrams as inputs. They first construct the state chart diagram conforming to system requirement. Then, analyze the .mdl file of state chart diagram, extract the main information of the state chart diagram and convert it to a directed graph. Finally, they designed an algorithm to construct the Euler circuit based on a directed graph and generate test cases automatically by Euler circuit algorithm. Their specified test coverage criteria are the state coverage and the transition coverage of state chart diagram, also to minimize the number of test cases.

In an earlier study Swain et al. [23] proposed an approach to generate test cases from UML state chart diagram. They have named their approach, Automatically Generating Test cases from State Chart Diagram (AGeTeSC). First, they have constructed the state chart diagram for a given object. Then the state chart diagram is traversed, conditional predicates are selected and these conditional predicates are transformed into source code. Then, the test cases are generated and stored by using function minimization technique. From the state chart diagram, they perform a DFS to select the associated predicates. After selecting the predicates, they guess an initial dataset. They have generated test predicate conditions from a state chart diagram, which are used to generate test cases. Their technique accomplishes little coverage in test case like transition pair coverage, state coverage, action coverage, and transition coverage. It also achieves fully predicate coverage by generating a test data for each conditional clause. Besides that, it can handle transitions with guards and achieves transition path coverage. Here the quantity of test cases is minimized and they reach transition path coverage in testing the limitations decided by simple predicates, but the test case needs to be optimized.

Additionally Swain et al. [34] proposed an approach for test cases generation named, Test Generation and Minimization for O-O software with State Charts

(TeGeMiOOSc). It starts by analyzing the system, which is going to be tested and accepted by user, then build the state chart diagram. After that, they convert the given UML state chart diagram into an intermediate model, that they named it a state transition graph. DFS is used to form test sequences and generating all the possible paths. Then obtain all the valid sequences of the application until final edge is reached. Finally, they minimize a set of test cases by calculating node coverage for each test sequence. In the same year a work of Swain et al. [35] has performed a similar experiment to generate test case from UML state chart diagram and they have named it, Generation and Minimization of test cases from State Charts (GeMiTefSc). Their approach at first build a state chart diagram model for SUT, next, they conjugated state transition graph from state chart diagram. Then, by using the graph, all the required information is extracted. Then, by applying Wang's algorithm [36] they generated the test cases. Finally, they minimized the set of test cases by calculating node coverage for each test case and this help them to determine which test case are covered by other test cases. However in their works, after creating the intermediate graph they rely on the DFS to generate the paths, what will lead to less in coverage when the state chart diagram have loops and feedbacks in it. Also by using minimization, they minimize a set of test cases what will cause to overlap or ignoring some of the important data therefore will have less coverage.

Chimisliu and Wotawa [37] in their earlier work have proposed a method for generating test cases automatically aiming at achieving transition coverage and state coverage of the model. Their proposed approach presents an automatically transformation of the system composed of communicating state charts diagram into a Language of Temporal Ordering Specification (LOTOS). They also showed how to generate test cases in semi-automatically way by making use of an input from the user as explanations on the UML diagram. In their work, generated test cases coverage criteria did not contain any reject transitions. Thus, the generation process was not as efficient as in the case when the user provides explanations that can be used as refuse transitions in the test purpose.

In their more recent work, Chimisliu and Wotawa [38] and Chimisliu and Wotawa [39] proposed an improved tool for test case generating from UML state chart diagram by using control, data and communication dependencies. They generated the test cases by using the Test Generation with Verification (TGV) technology [40], a test case generator from the analysis and the construction of distributed processes toolbox. For the coverage criteria, their generation technique aimed at achieving transition coverage only. Therefore, the lack of coverage will indicate to the need to enhance this method or obtaining a novel one.

3 Findings from the Survey

The following section presents the trends, gaps and commonalities found from the 21 studies that conducted test cases generation from UML start Chart diagram.

Table 1 reviews the past decade researchers and their studies. The input model column shows that the current researchers use state chart diagram or combinations of other diagrams and also the method they used to generate the test cases. Furthermore, the intermediate models that are generated as intermediary between

Table 1 Test case generated methods using UML state chart diagram

Author(s)	Input model	Method	Intermediate model	Coverage criteria
Kansomkeat and Rivepiboon [20]	State chart	Parsing TFG, mutation analysis	Testing flow graph (TFG)	State and transition
Offutt et al. [24]	State chart	Spec test	Specification graph	Transition coverage Full predicate coverage Transition-pair coverage Complete sequence
Gnesi et al. [19]	State chart	Input/output label transition systems (IOLTSs), random test selection	–	–
Briand et al. [25]	State chart	Normalization and analysis of operation contracts and transition guards	Invocation sequence tree (IST)	All transitions, all transition pairs, full predicate, and all round-trip paths
Li and Lam [26]	State chart	Ant colony optimization	Directed graph	All states
Santiago et al. [27]	State chart	Perform charts and Condado	FSM	All pairs of transitions
Murthy et al. [28]	State chart	Extended UML state chart model	Context free grammar model	Path coverage, Basic path coverage
Ali et al. [21]	Collaboration diagrams and state chart	State collaboration test model (SCOTEM)	Testing flow graph (TFG)	Single-path coverage All-transition coverage, N-path coverage All-path coverage

(continued)

Table 1 (continued)

Author(s)	Input model	Method	Intermediate model	Coverage criteria
Santiago et al. [29]	Finite state Machines and state Charts	Switch cover, distinguishing Sequence and unique Input/output methods	FSM	All transitions
Kosindrdecha and Daengdej [22]	State chart	TGfMMD method	Sketch diagram-based technique	All nodes
Swain et al. [31]	State chart and activity chart	SAD generation, DFS, mutation analysis	State activity diagram (SAD)	Transition coverage and activity path coverage
Shirole et al. [32]	State chart	Genetic algorithm	Extended control flow graph	State cover, Transition cover, All-definition cover, and All du-path
Li et al. [33]	State chart	Euler circuit algorithm	Directed graph	State coverage criteria, Transition coverage criteria
Swain et al. [23]	State chart	Depth first search (DFS), Model J unit	State chart graph	State coverage, Transition coverage, Transition pair coverage
Swain et al. [34]	State chart	Test generation and Minimization for O-O software with state charts (TeGeMiOOSc)	State graph	State coverage, Action coverage, Transition coverage, Transition path coverage, Condition coverage
Swain et al. [35]	State chart	Generation and minimization of test cases from State Charts (GeMiTefSc)	State graph	State coverage, Action coverage, Transition coverage, Ppath coverage, Condition coverage
Chimisliu and Wotawa [37]	State chart		Test purpose	Transition coverage, State coverage

(continued)

Table 1 (continued)

Author(s)	Input model	Method	Intermediate model	Coverage criteria
Chimisliu and Wotawa [38]	State chart	Test generation with Verification technology (TGV)	Test purpose	Transition coverage
Chimisliu and Wotawa [39]	State chart		Test purpose	Transition coverage

the input model and the generated paths, and coverage criteria are also clarified in this table.

These studies illustrate the importance of integrating of UML state chart diagram with other intermediate model to generate the test cases. The conclusion from these studies describe that most of them need to translate UML state chart diagram into another description, such as a graph or a table (intermediate model), which will be used to derive the test cases. Furthermore, many studies worked on DFS algorithm [23] to generate the test paths. However, this algorithm will lead to loss of paths, especially for the loops [31]. Therefore, there is a need to redefine the path coverage criterion through loop path coverage and generate an enhanced DFS algorithm or other algorithm to generate the paths [8].

In generating test case using UML state chart diagrams very few studies reveal their proposed algorithms or their testing implementation conducted during the testing. Among the studies are [20, 22, 27, 29, 41]. This scenario will lead to difficulties in updating their work or do enhancement on them. Furthermore, it is hard to implement it in a larger scale or produce it to generate test case in a fully automated way.

In rationalizing the generation of test case, the quality or the adequacy of test cases is often described with coverage criteria. From these studies, the most common coverage criteria are path coverage, transition coverage, and state coverage, what will be necessary to cover in future studies.

There are many approaches, like GA, model checking, or graph search algorithms, that are used to cover such coverage criteria for graph based models. However, there are few problems with the existing UML state chart diagrams test generation approaches. One of them is selecting the right input graph that has enough complexity to generate the accurate coverage percentage. Studies from [23, 34, 35] used very simple state chart diagrams. In addition, there are approaches by [37–39] that selected very few coverage criteria, where they just cover the transaction coverage or unnecessary one.

Many of the test case tools were not integrated. The one that have been used for test case generation, they demand several effort from the software testers since all the testing steps require manual interference in order to make appropriate adjustments on the output of a tool to be used as input to another tool [29].

4 Conclusion and Future Work

UML recently become a focused model in the field of software testing. New techniques and methods for the generation of test case from these UML diagrams needs to be identified. To identify them in this paper, a literature survey on generating test cases from UML state chart diagram has been conducted.

From this survey, it shows that existing methods on state chart test case generation methods mostly concentrate on the DFS algorithm, in which some do not work for maximum test coverage, while some methods prepare and generate a significant number of tests with less test coverage. In the future, we have planned to develop a test case generation method that minimizes the size of tests, time and cost, while preserving maximum test coverage using Modified Condition/Decision Coverage criterion. In addition, this method will prove that this technique is more capable of detecting more number of faults than compared to current existing techniques.

References

1. Patwa, S., Malviya, A.K.: Impact of coding phase on object oriented software testing. Covenant J. Inform. Commun. Technol. (CJICT) **2**, 57–67 (2014)
2. Cartaxo, E.G., Neto, F.G.O., Machado, P.D: Test Case Generation by Means of UML Sequence Diagrams and Labeled Transition Systems. In SMC, pp. 1292–1297 (2007)
3. Mingsong, C., Qiu, X., Xu, W., Wang, L., Zhao, J., Li, X.: UML activity diagram-based automatic test case generation for Java programs. Comput. J. **52**, 545–556 (2009)
4. Javed, A.Z., Strooper, P.A., Watson, G.: Automated generation of test cases using model-driven architecture. In: Automation of Software Test, AST'07. Second International Workshop on, pp. 3–3 (2007)
5. Kim, H., Kang, S., Baik, J., Ko, I.: Test Cases Generation from UML Activity Diagrams. In: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, SNPD, pp. 556–561 (2007)
6. Kundu, D., Samanta, D.: A novel approach to generate test cases from UML activity diagrams. J. Object Technol. **8**, 65–83 (2009)
7. Lilly, R., Uma, U.G.: Reliable Mining of Automatically Generated Test Cases from Software Requirements Specification. IJCSI, pp. 87–91 (2010)
8. Mingsong, C., Xiaokang, Q., Xuandong, L.: Automatic test case generation for UML activity diagrams. In: Proceedings of the 2006 International Workshop on Automation of Software Test, pp. 2–8 (2006)
9. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual. Pearson Higher Education, New York (2004)
10. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, US (2010)
11. Kim, W.Y., Son, H.S., Kim, R.Y.C.: A study on test case generation based on state diagram in modeling and simulation environment. In: Advanced Communication and Networking. Springer, Berlin, pp. 298–305 (2011)
12. Karambir, Kuldeep, K.: Survey of software test case generation techniques. Int. J. Adv. Res. Comput. Sci. Softw. Eng., pp. 937–942 (2013)

13. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., et al.: An orchestrated survey of methodologies for automated software test case generation. J. Syst. Softw. **86**, 1978–2001 (2013)
14. Shirole, M., Kumar, R.: UML behavioral model based test case generation: A survey. ACM SIGSOFT Softw. Eng. Notes **38**, 1–13 (2013)
15. Rafi, D.M, Moses, K.R.K., Petersen, K., Mäntylä, M.V.: Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: Proceedings of the 7th International Workshop on Automation of Software Test, pp. 36–42 (2012)
16. Prasanna, M., Sivanandam, S., Venkatesan, R., Sundarrajan, R.: A survey on automatic test case generation. Acad. Open Internet J. **7**:1–6(2005)
17. Pahwa, N., Solanki, K.: UML based test case generation methods: A review. Int. J. Comput. Appl. **95**, 1–6 (2014)
18. Shamsoddin-Motlagh, E.: A review of automatic test cases generation. Int. J. Comput. Appl. **57** (2012)
19. Gnesi, S., Latella, D., Massink, M.: Formal test case generation for UML statecharts. In: Proceedings of Ninth IEEE International Conference on Engineering Complex Computer Systems, pp. 75–84 (2004)
20. Kansomkeat, S., Rivepiboon, W.: Automated generating test case using UML statechart diagrams. In: Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, pp. 296–300 (2003)
21. Ali, S., Briand, L.C., Rehman, M.J.-U., Asghar, H., Iqbal, M.Z.Z., Nadeem, A.: A state-based approach to integration testing based on UML models. Inf. Softw. Technol. **49**, 1087–1106 (2007)
22. Kosindrdecha, N., Daengdej, J.: A test generation method based on state diagram. JATIT pp. 28–44 (2010)
23. Swain, R.K., Panthi, V., Behera, P., Mohapatra, D.: Automatic test case generation from UML state chart diagram. Int. J. Comput. Appl. pp. 26–36 (2012)
24. Offutt, J., Liu, S., Abdurazik, A., Ammann, P.: Generating test data from state-based specifications. Softw. Test. Verification Reliab. **13**, 25–53 (2003)
25. Briand, L.C., Labiche, Y., Cui, J.: Automated support for deriving test requirements from UML statecharts. Softw. Syst. Model. **4**, 399–423 (2005)
26. Li, H., Lam, C.P.: An ant colony optimization approach to test sequence generation for state-based software testing. In: Fifth International Conference on Quality Software, (QSIC 2005), pp. 255–262 (2005)
27. Santiago, V., do Amaral, A.S.M, Vijaykumar, N., Mattiello-Francisco, M.F., Martins, E., Lopes, O.C.: A practical approach for automated test case generation using statecharts. In: 30th Annual International Computer Software and Applications Conference, 2006 (COMPSAC'06), pp. 183–188 (2006)
28. Murthy, P., Anitha, P., Mahesh, M., Subramanyan, R.: Test ready UML statechart models. In: Proceedings of the 2006 International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools, pp. 75–81 (2006)
29. Santiago, V., Vijaykumar, N.L., Guimarães, D., Amaral, A.S., Ferreira, É.: An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In IEEE International Conference on Software Testing Verification and Validation Workshop, 2008 (ICSTW'08), pp. 63–72 (2008)
30. Binder, R.V.: Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley Professional, Boston (2000)
31. Swain, S.K., Mohapatra, D.P., Mall, R.: Test case generation based on state and activity models. J. Object Technol. **9**, 1–27 (2010)
32. Shirole, M., Suthar, A., Kumar, R.: Generation of improved test cases from UML state diagram using genetic algorithm. In: Proceedings of the 4th India Software Engineering Conference, pp. 125–134 (2011)

33. Li, L., He, T., Wu, J.: Automatic test generation from UML statechart diagram based on euler circuit. Int. J. Digit. Content Technol. Appl. **6** (2012)
34. Swain, R.K., Behera, P.K., Mohapatra, D.P.: Minimal Test Case Generation for Object-Oriented Software with State Charts. arXiv preprint arXiv:1208.2265 (2012)
35. Swain, R.K., Behera, P.K., Mohapatra, D.P.: Generation and Optimization of Test cases for Object-Oriented Software Using State Chart Diagram. arXiv preprint arXiv:1206.0373 (2012)
36. Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., Guoliang, Z.: Generating test cases from UML activity diagram based on gray-box method. 11th Asia-Pacific Presented at the Software Engineering Conference (2004)
37. Chimisliu, V., Wotawa, F.: Model based test case generation for distributed embedded systems. In: IEEE International Conference on Industrial Technology (ICIT), pp. 656–661 (2012)
38. Chimisliu, V., Wotawa, F.: Improving test case generation from UML statecharts by using control, data and communication dependencies. In: 13th International Conference on Quality Software (QSIC), pp. 125–134 (2013)
39. Chimisliu, V., Wotawa, F.: Using dependency relations to improve test case generation from UML statecharts. In: IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 71–76 (2013)
40. Claude, J., Thierry, J.: TGV: Theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. Softw. Tools Technol. Transf. **7**, 297–315 (2002)
41. Hartmann, J., Imoberdorf, C., Meisinger, M.: UML-based integration testing. In: ACM SIGSOFT Software Engineering Notes, pp. 60–70 (2000)