International Journal of Computer Engineering and Information Technology

VOL. 9, NO. 11, November 2017, 258–263
Available online at: www.ijceit.org
E-ISSN 2412-8856 (Online)

A Test Purpose-Driven Testing Method

A. Guerrouat

aguerrouat@yahoo.com

ABSTRACT

The paper presents a framework for the automation of testing of component-based systems and embedded systems by considering test purposes. We deal with the component-based systems from communication point of view. Hence, only the external behavior of the components at the interaction points can be observed. For test generation, we adapt a generic FSM-based method by including test purposes to optimize the amount of the generated test sequences for practicability reasons. These methods have been generically since decades successfully used for testing both hardware and software including protocols and communicating systems.

Keywords: *Formal Methods, Embedded Systems, Black-Box Testing, Automation Of Test Generation and Testing.*

1. INTRODUCTION

While in the last decade component-based software development gained a considerable importance from both researchers and practitioners, automated testing of such software systems, however, did not reach a similar interest. In addition to the difficulty and the expensiveness of the manual testing procedure, many assume that once a component is sufficiently tested, there will be no need to retest it when reused. This, however, is not always true, since components may satisfy a certain application domain and fails a new environment [1] [2].

Component based systems and embedded systems are becoming more and more the key technology of any kind of complex technical systems, ranging from telecommunications devices to aircrafts and automobiles including driverless and smart cars. An embedded computer system is a computer system that represents a part of larger system and performs some requirements of that system [8] [9].

The growing amount and complexity of requirements on embedded systems regarding properties like safety and real-time behavior make the software development process a costly and error-prone activity. The cost factor plays, however, a central role in today's industrial competition, for instance, between car manufacturers. The development of competitive and efficient products is imposing more and more constraints to the design of embedded systems. One of the means to reach this goal are formal methods such as state-transition based methods for supporting the different stages of system development process especially the validation by means of testing. There are several requirements for those methods that should be among others qualities abstract, understandable, analyzable, scalable and unambiguous specification formalisms.

Since decades the automation of testing communication systems based on formal methods was the main focus of academia. This was due to the increasing complexity and growing use of communication protocols and systems as the later are pervading our today's life in different ways. Conformance testing that corresponds to the black-box testing was one of the most important testing types. Black-box test is also the most dominant test type in the area of component-based systems due to the nature of their constituents. Indeed, components are independent and replaceable parts and should be conform to and provide a set of interfaces. They also consist usually of special components, called COTS (commercial-off-the-shelf) that can be purchased on a component market. These are often delivered without their source code which makes other types of tests like white or grey tests less appropriate.

Finite state machines are very popular in the control flow specification of state/transition-based systems and many related analysis methods have been developed [3] [4] [5]. They aim in particular to support a formal generic test derivation for validation and testing purposes. If data flow should be considered, finite state machines are enlarged to extended finite state machines (EFSMs) to deal with both the data flow and control flow of a system. Test derivation methods developed around state transition machines generate potentially a huge amount of test data, commonly called test suites, due among other to the recursive and repetitive character of the behavior of communication systems and protocols. For cost and practical reasons generic test suites resulting from test derivation methods such as the transition tour method cannot be executed.

In this paper, we present the basic principle of a test method that optimize the amount of the generic test data

based primarily on the test purposes, as it is well-known that an exhaustive testing is impossible.
The rest of the paper is organized as follows. Section 2 reminds the basic structure of the communication in component-based systems and embedded systems to which the proposed test method is applied. In Section 3, we reviews the FSM based test generation method and the appropriate fault model on which the present approach is founded. Section 4 illustrate the basic principle of the present approach. Finally, Section 5 concludes the paper.

2. BACKGROUNDS

2.1 Embedded System Environment

An embedded system is any computer system or computing device that performs a dedicated function or is designed for use with a specific embedded software application, for example, smart phone, mobile phone, E-book, robot, car, aircraft etc. That is, an embedded system is a special-purpose system built into a larger device. Figure 1 shows some examples of such devices and objects. It is embedded as a subsystem in a larger system which may or may not be a computer system. An embedded system is typically required to meet specific requirements.

Fig. 1. Examples of Devices and Objects with Embedded Systems (*Source: http://www.polytechnichub.com)*

At the communication level, embedded systems are commonly connected to a physical environment through sensors and actuators. They are typically reactive systems. A reactive system is in continuous interaction with its environment and executes at a pace determined by that environment. The behavior depends on input and current state for which the automata model is often the most appropriate.

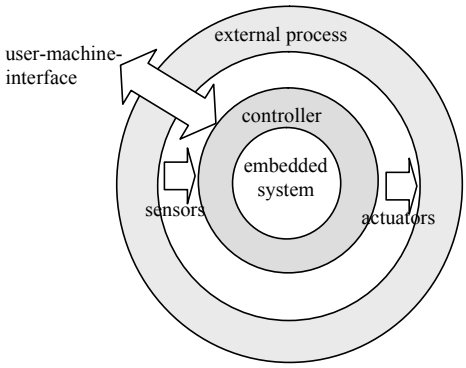

Fig. 2. A Generic Communication Model for Embedded Systems

Figure 2 illustrates the main constituents of an embedded system and its environment comprising an external process, sensors, actuators, and a controller. The external process is a process that can be of physical, mechanical, or electrical nature. Sensors provide information about the current state of the external process by means of so-called monitoring events. These are communicated to the controller as input events. The later are considered as stimuli for the controller. The controller must react to each received event, i.e. input event, which usually originate from sensors. Depending on the received event, the external process will be conducted to a corresponding state. The controller's behavior is depending on that of the external process. The controller commands the behavior of the external process taking into consideration requirements on the process and its characteristics, such as physical laws, real time and other constraints. Actuators receive the results determined by the controller which are communicated to the external process by means of so-called controlling events.

2.2 Specification Model

In the component-based approach for embedded systems one distinguishes a component repository, a composition environment and a run-time environment. The component repository consists of collections of single specifications of each component of the embedded system, sensors, controller and actuators. These are modeled as finite state machines that collectively build functional modules. The composition environment is the embedded system specification which consists of the specification of its environment and its controller. This takes places by linking the single modules to each other by means of channels via interfaces, called interaction points (Figure 3). These modules interact with each other via broadcasting events via these interaction points. However, a logic sequence should be respected in this communication. For instance, the direct communication of a module of an actuator with a sensor is not allowed. Run-time environment consists of the instantiated

embedded system specification issued from the former step, i.e. the composition.

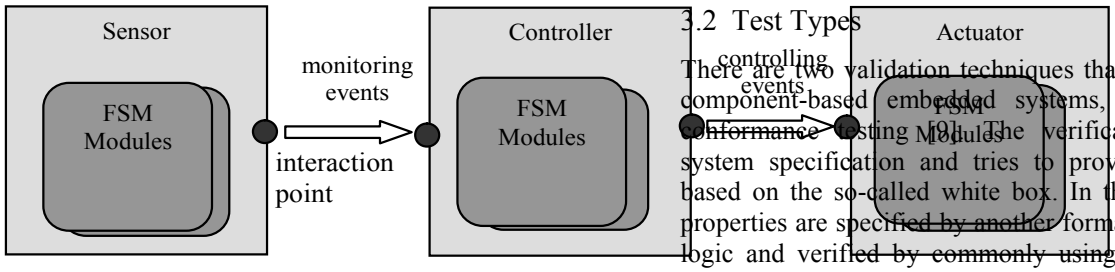

Fig. 3. Overview of the composition environment for embedded systems based on FSM model

The most important component of an embedded system consists of the controller that communicates with its environment, i.e. sensors and actuators, via signals (i.e. input/output events). The output events of sensors represent input events for the controller. The events from the controller to the actuators are output events and represent input events for the actuators. They result from new computations performed by the controller that is triggered by the received input events.
Based on the occurred sensor events (e.g. indicating the power on/of state for an electrical unit, the speed of a mobile object such as a car, etc.), the corresponding finite sate machine of the component is triggered and the involved transition(s) is performed. This would lead to trigger the FSMs of the controller whose states now change. Note that transitions in the controller can be spontaneously triggered by other events, e.g. time out. The subsequent state of the external process is computed and communicated as output events by the actuators.

3. TEST GENERATION METHODS REVIEW

3.1 Overview

There exist since decades a variety of test generation methods that are based on FSMs [6] [7]. Each of them is characterised error class it can cover, and thus by its suitability for a given test type. These methods share is basic idea. The target test sequence should be short and composed of consecutive transitions that contains every transition of the FSM at least once and allows to check whether every transition is implemented as defined. To test a transition, one has to apply the input and to check whether the correct output occurs and the expected target state is reached. Depending on the test type, checking the target state might be omitted (transition tour method) or be carried out by means of distinguishing sequences (checking experiments method), characterizing sequences (W-method), or unique input/output sequences (UIO methods) [6] [7] [10].

3.2 Test Types

There are two validation techniques that can be used for component-based embedded systems, verification and conformance testing Methods [8]. The verification deals with system specification and tries to prove its correctness based on the so-called white box. In this case, the user properties are specified by another formalism as temporal logic and verified by commonly using model-checkers. The second approach, the conformance testing that is subject of the current work, deals with a system implementation and tries to find incorrect behavior on it without considering its internal structure. This type of test is called black-box test. Test sequences (called test suites, i.e. sets of test cases) are derived from the specification and are executed on the implementation (IUT: implementation under test). Based on the test purposes of the executed test cases and the analysis of the verdict of the test execution, it will be stated whether the implementation is conform to the specification or not. A test case is composed of a test preamble (a sequence of actions leading to a given state or action), test body (actions to test), and of a test postamble (sequence of action allowing to go back to the initial state).

3.3 Fault Model

As mentioned above, for FSM-based test generation methods, there exist some fault models that define the semantic of errors categorized in error classes [10].
Following are the most well-know error classes based on the FSM model:

- Output errors class: A transition has an output fault if, for the corresponding state and input received, the embedded system implementation provides an output different from the one specified.
- Transfer errors class: A transition has a transfer fault if, for the corresponding state and input received, the embedded system implementation enters a different state than expected. (Even it produces the expected output).
- Transfer errors with additional states class: The embedded system implementation reaches more states than expected with possible transfer faults.
- Missing states errors class: The embedded system implementation has less states as specified. This is usually due to non-deterministic behaviours and/or incompleteness.

3.4 Transition-Tour Test Method

3.4.1 Overview

As explained above, the black-box test is the most convenient test type for component-based systems. Thus, the appropriate test generation method among those invoked previously is the Transition-Tour method. Indeed, this method does not require information about the internal structure of embedded system components, i.e. only the external behaviour is observed. This makes it a better candidate to be used in conformance testing (s. Figure 4).

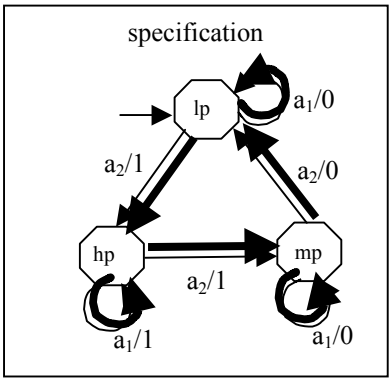

Fig. 4. Principle of TT method

3.4.2 Test Generation and Fault Model based on TT

The TT method is able to detect any set of output faults in the absence of transfer faults. A transition tour of a FSM is a path (test sequence) starting at the initial state, traverse every transition at least once, and returns to the initial state (Figure 5). From a transition tour, one can identify a test suite consisting of an input sequence and its expected output sequence based on the specification: the input sequence $a_1a_2a_1a_2a_1a_2$ and its expected output sequence 011100. The transition tour can find all output faults, e.g. the faulty observed output sequence 111100 is detected when executing the input sequence $a_1a_2a_1a_2a_1a_2$ on the implementation. However, the TT-method is not able to detect transfer errors (Figure 5).

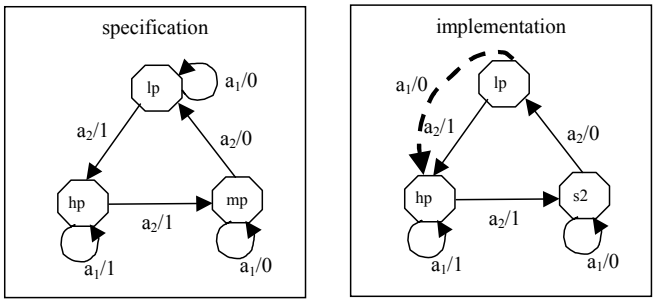

Fig. 5. TT-method can detect only output errors

3.4.3 Discussion of TT Method

As it could be stated from the above example, in addition to its simplicity, the TT method is generically well suitable for a black box test that applies to conformance testing. However, the TT method is not able to specify that part of the implementation under test to be tested. It does only require that each transition should be visited at least once in order to be able to detect any output error in the IUT. On the other side, it is well-known in testing that an exhaustive testing is practically impossible due to the fact that this would require an enormous amount of testing infrastructures, a huge amount of time and very high costs. This statement is still true even for testing only the control flow, i.e. without considering the data flow. Based on the principle of the test generation of the TT method, the generated test suites (also called test sequences) could be theoretically very extensive and even unending. This is due to the nature of the behaviour of communication systems and protocols such those used in the embedded system area which is commonly a repetitive and recursive behaviour. If an implementation under test passes a test for the behaviour occurrence n, this does necessarily mean that it will pass the test for the behaviour occurrence m.

4. TEST PURPOSE-BASED APPROACH

4.1 Meaning of Conformance Testing

An implementation under test I is conform to a specification S with respect to a conformance relationship conf (I conf S) if for any environment E the following is true (Figure 6). Each behavior of I in E that is observed at the external interaction points is also possible for S in E at the corresponding interaction points. The conformance relation conf formally defines the meaning of correctness of an implementation I w.r.t. its specification S. In our case, the correctness is based on the fault model defined for the TT method, namely the output error class.

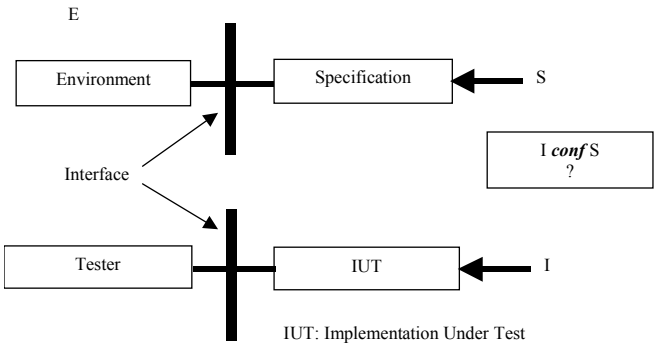

Fig. 6. Principle of conformance testing

4.2 Basic Principle of the Approach

The present approach combines the simplicity of the automated test generation based on the TT method and practicality of the generated test sequences or testsuites (a set of test cases) based on the test purpose concept. Test cases are usually built from the generic testsuites by primarily considering the test architecture and test environment, for example local or distributed, under which the IUT will be tested. For example, Figure 7 presents an example of a local test architecture in which the tester and the implementation under test (IUT) are located in the same physical location and there is a direct contact of the tester with the IUT via the PCOs (Points of Control and Observation).

4.2 Test Purposes

The test purposes allow to focus the test execution on some behaviour parts of the implementation based on the test experiences and knowledge of the test expert. Based on this knowledge, a test expert may drive the test to those behaviour parts that are more likely to present vulnerability. The test purposes could be designated in different ways:

- By specifying the behaviour occurrence targeted by the test. The behaviour occurrence consists of one or more transition tours assuming that each occurrence brings back the IUT to its initial state.
- By specifying explicitly a path (test preamble) that should conduct the tester to the behaviour part that is targeted by the test (test body). This must not necessarily a complete transition tour.
- By specifying only the test length of the targeted behaviour part (test body). In this case, a path (test preamble) will be derived that randomly locates any behaviour part with the required test extensive to keep the test cost within a reasonable amount.

4.3 Test Architecture

To apply the above presented test generation method resulted, we adapt an OSI (Open Systems Interconnection) test architecture for conformance testing, called abstract test methods, namely a conceptual test architecture. As shown in Figure 7, a conceptual test architecture describes how the behavior of the IUT in principle can be controlled and observed.

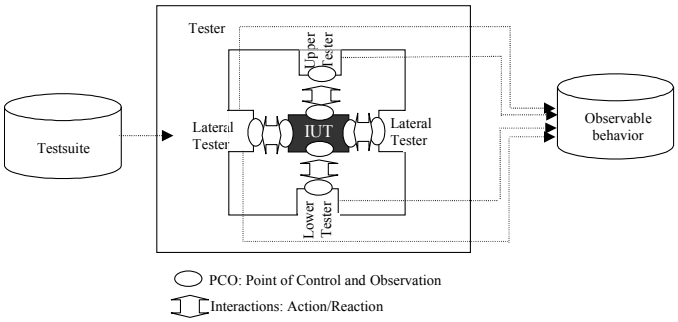

Fig. 7. Conformance testing execution

The advantage of the above test architecture can be applicable not only for testing protocol stacks for open communication systems (OSI), but also for any other communicating systems including component-based and embedded systems. Note that in the case of OSI test architectures, the communication takes place vertically, but not necessarily for all communicating systems.

As depicted by Figure 7, in conformance testing the external behavior of the IUT is observed at accessible interaction interfaces, called PCOs (Points of Control and Observation) in the testing context. The tester communicates directly with the IUT via PCOs. These are supposed to correspond with the interaction interfaces (called also service interfaces). The activities at the interfaces can be provided via three classes of tester components: upper testers (UT), lower testers (LT) and lateral testers (AT). Upper tester and lower testers are a special case of the CB under test, namely when the IUT represents a protocol stack (OSI layer model) seen as CBS. In the later case, lateral testers do not exist since in a component-based communication system w.r.t. OSI layer model the direct communication between two neighbor layers takes places only vertically and thus, cross and direct horizontal communication are not allowed (except for physical layer).

5. CONCLUSION

In this paper, we have presented the basic principle a test generation method that is founded on the well-known TT method [6]. We have proposed to consider the test purposes in order to keep the test extent, efforts and costs reasonable and practical. Indeed, it is well-known that an exhaustive testing is impossible. The test purposes allow to reduce the test sequences by directing and focussing on

the test process to IUT parts that could be more likely to behave erroneously based on test expert knowledge and experiences. The test type considered here is the conformance testing which is black-box test. It is convenient for testing component-based and embedded systems as mostly the tester has no access to the internal structure of the components but only to their interaction interface.

We are refining the proposed improvement by formalizing the designation of test purposes and defining a complete related testing process and environment dedicated particularly to embedded systems. Furthermore, we plan to investigate real-life embedded systems especially from the automotive area including driverless cars.

REFERENCES

[1] K. H. Pries and J. M. Quigley. Testing Complex and Embedded Systems. CRC Press, 2016. ISBN 9781439821411.

[2] H.-K Kim and R. Y. Lee. Components Based Testing Using Optimization AOP. In R. Lee (ed.), Computer and Information Science, Springer, 2012. ISBN: 978-3642304538.

[3] P. Ammann and J. Offnutt. Introduction to Software Testing, 2nd Edition. Cambridge University Press, 2017. ISBN: 978-1107172012.

[4] R. Buessow, R. Geisler, and M. Klar. Specifying safety-critical embedded systems with statecharts and Z: A case study. In Proceedings of Fundamental Approaches to Software Engineering (FASE'98), Lisbon, 1998.

[5] P. E. Black. Finite State Machine. Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology, 2008.

[6] A.T. Dahbura, K.K. Sabnani, and M.U. Uyar. Formal methods for generating protocol conformance test sequences. In Proc. IEEE 78 (8), 1990.

[7] A. V. Aho et al. An optimisation technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. In S. Aggarwal and K. Sabnani (eds.), Protocol Specification, Testing, and Verification, New Jersey, 1988.

[8] I. Crnkovic. Component-based approach for embedded systems. Ninth International Workshop on Component-Oriented Programming, Oslo, 2000.

[9] S. Beydeda, and V. Gruhn. Testing Component-Based Systems Using FSMs. In Beydeda and Gruhn (Eds.), Springer-Verlag, 263-280, 2004.

[10] S. Fujiwara et al. Test selection based on finite state models. IEEE transaction on Software Engineering 17(6), 1991.