GENERATING SYNCHRONIZABLE TEST SEQUENCES BASED ON FINITE STATE MACHINE WITH DISTRIBUTED PORTS[1]

Gang Luo[a], Rachida Dssouli[a], Gregor v. Bochmann[a], Pallapa Venkataram[b] and Abderrazak Ghedamsi[a]

[a] Departement d'IRO, Universite de Montreal,
C.P. 6128,Succ.A, Montreal, P.Q., H3C 3J7, Canada
e-mail:luo@iro.umontreal.ca, Fax: (514) 343-5834

[b] Dept. of Electrical Communication Engineering,
Indian Institute of Science, Bangalore-560 012, Indian
E-mail: pallapa@ece.iisc.ernet.in

Abstract

In the area of testing communication systems, the interfaces between systems to be tested and their testers have great impact on test generation and fault detectability. Several types of such interfaces have been standardized by the International Standardization Organization (ISO). A general distributed test architecture, containing distributed interfaces, has been presented in the literature for testing distributed systems based on the Open Distributing Processing (ODP) Basic Reference Model (BRM), which is a generalized version of ISO distributed test architecture. We study in this paper the issue of test selection with respect to such an test architecture. In particular, we consider communication systems that can be modeled by finite state machines with several distributed interfaces, called ports. A test generation method is developed for generating test sequences for such finite state machines, which is based on the idea of synchronizable test sequences.

Starting from the initial effort by Sarikaya, a certain amount of work has been done for generating test sequences for finite state machines with respect to the ISO distributed test architecture, all based on the idea of modifying existing test generation methods to generate synchronizable test sequences. However, none studies the fault coverage provided by their methods. We investigate the issue of fault coverage and point out a fact that the methods given in the literature for the distributed test architecture cannot ensure the same fault coverage as the corresponding original testing methods. We also study the limitation of fault detectability in the distributed test architecture.

Keyword codes: C.2.4; C.2.2; D.2.5
Keywords: Distributed Systems; Network Protocols; Testing and debugging

---

[1] This work was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at the University of Montreal (Canada).

1. INTRODUCTION

A general distributed test architecture where the IUT (implementation under test) contains several distributed ports has been studied in [7] for testing distributed systems. It is based on the Open Distributing Processing (ODP) Basic Reference Model (BRM) (see Figure 2.4). In this architecture, the IUT contains several distributed interfaces, called ports (i.e., points of control and observation), the testers cannot communicate and synchronize with one another unless they communicate through IUT, and no global clock is available in the system. This could be a test architecture of a communication network with n accessing nodes, where the testers reside in these nodes. When n=2, this general distributed test architecture is reduced to the ISO distributed test architecture [ISO87] for communication protocol testing. We study in this paper test selection methods with respect to a general distributed test architecture.

Usually, in the so-called local test architecture developed by ISO [13], the specifications of communication protocols are first abstracted into state machines [15], then test cases are generated from the resulting machines. A number of methods have been developed to generate test cases for finite state machines (FSMs) [9, 18, 6, 19, 17], however, they are not directly applicable to the distributed test architecture, because of the synchronization problem between distributed testers.

In distributed test architectures, testing is relatively difficult because certain problems of synchronization between the testers may arise during the application of test sequences. To solve this problem, with respect to the ISO distributed test architecture [13] where there are only two ports, an approach for test generation has been developed in [20] by modifying the existing test generation methods for FSMs such as the transition tour [17], the DS-method [14], and the W-method [6] such that the resulting sequences are so-called synchronizable test sequences.

We develop in Section 3 an approach to generating test sequences for FSMs with n distributed ports ($n \geq 2$), with respect a general distributed test architecture (see Figure 2.4), after defining and explaining several concepts related to distributed test architecture and FSMs. Our approach is a generalized version of the approach given in [20].

Starting from the initial effort by Sarikaya, a certain amount of work has been done for generating test sequences for finite state machines for ISO distributed test architecture, all based on the idea of modifying existing test generation methods to generate synchronizable test sequences. [5, 3] have studied the computation complexity issue of the approach of [20], and they have pointed out that there is no efficient (i.e., polynomial time) algorithm for obtaining a minimum length synchronizable test sequences from a given FSM. A heuristic approach to obtaining synchronizable test sequences is given in [5]. Another approach to obtaining synchronizable test sequences, presented in [12], is to insert additional synchronization statements (i.e., software probes) into the IUT. This approach is not applicable when the conformance testing is conducted by a second party other than the implementor as the party does not have the access to the internal information of the IUT. Although so much work has been done on the different aspects of synchronizable test sequence generation, yet none studies the fault coverage provided by their methods.

We explore in Section 4 the issue of fault coverage and discuss the observability and fault detectability in the distributed test architecture. We come out with a fact that these modified methods in the literature cannot ensure the same fault coverage as the corresponding original testing methods. For instance the modified transition tour cannot detect all output faults for FSMs with two distributed ports although transition tour can detect all output faults for the FSMs with only one port. A class of output faults may remain undetected if the modified transition tour is applied to an FSM with two ports. Unfortunately, none of the former articles

on generating synchronizable test sequences [20, 12, 5, 3] had realized such weakened fault coverage. We also present an improved approach that has a better fault detection power. We finally show a limitation of fault coverage in the distributed test architecture that not all faults of the IUT are detectable, showing the impossibility of complete fault coverage. Therefore, the testability of IUTs under the distributed test architecture need to be studied. We conclude in Section 5 by discussing some future research issues.

2. DISTRIBUTED TEST ARCHITECTURE AND FSMS WITH SEVERAL PORTS

We discuss in this section different test architectures used for testing communication systems, and finite protocol machine (i.e., finite state machine) model related to these architectures. We also present a fault model for finite state machines.

2.1. Distributed test architectures for communication systems

A. ISO standard test architectures for protocol conformance testing

ISO has defined four types of test architectures for protocol conformance testing, called local, distributed, coordinated and remote test architectures, respectively [13]. Local test architecture corresponds to traditional software testing, as shown in Figure 2.1, where PCO and IUT refer to point of control and observation, and the implementation under test, respectively. In this architecture, the two PCOs of the IUT can be viewed as a single port since the IUT and the tester are located in the same place. In the distributed test architecture, the test system is divided into so-called upper and lower testers which access the PCOs 1 and 2 of the IUT, respectively, as shown in Figure 2.2. The lower interface PCO 2 is accessed over distance and indirectly through the underlying communication service. The coordinated test architecture is similar to the distributed test architecture, and the only difference between the two is that the former has some kind of coordination between upper and lower testers, established using a so-called test coordination protocol through a (possibly separate) communication channel between upper and lower testers. The remote test architecture (see Figure 2.3) corresponds to the distributed test architecture where only a lower tester is used; instead of upper tester, the IUT may include a stack of several protocol layers above the layer being tested.

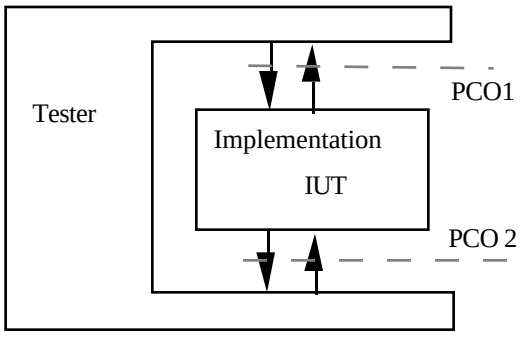


Figure 2.1. Local test architecture

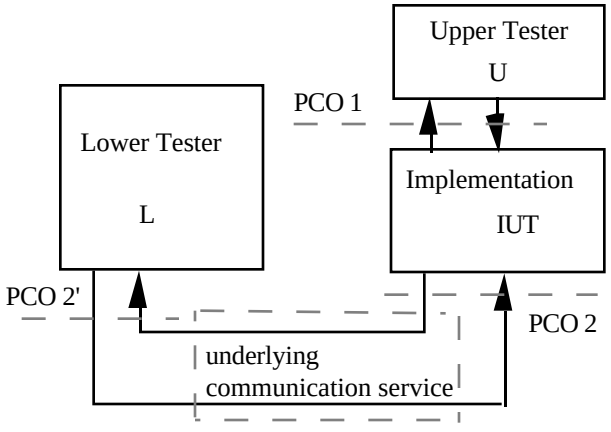


Figure 2.2. Distributed test architecture

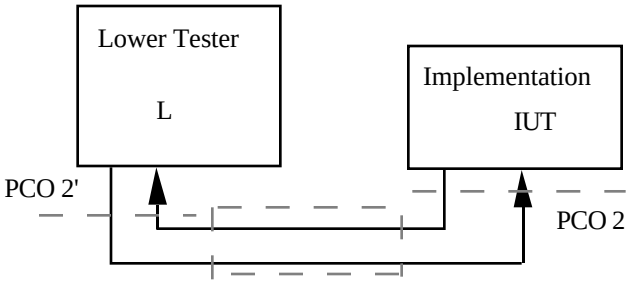


Figure 2.3. Remote test architecture

B. A general distributed test architecture

As mentioned before, the ISO test architectures only deal with an individual single protocol entity. However, to test the overall properties of distributed database systems and communication networks, we may face a general distributed test architecture, as shown in Figure 2.4. In this general distributed test architecture, (i) the IUT contains several ports, (ii) the testers cannot communicate and synchronize with one another unless they communicate through the IUT, and (iii) no global clock is available.

In testing a communication network, this general architecture can model a communication network with n accessing nodes; the IUT models the communication network, and the testers are located in these nodes. When n=2, the above model is reduced to the ISO standard distributed test architecture, which is shown in Figure 2.2.

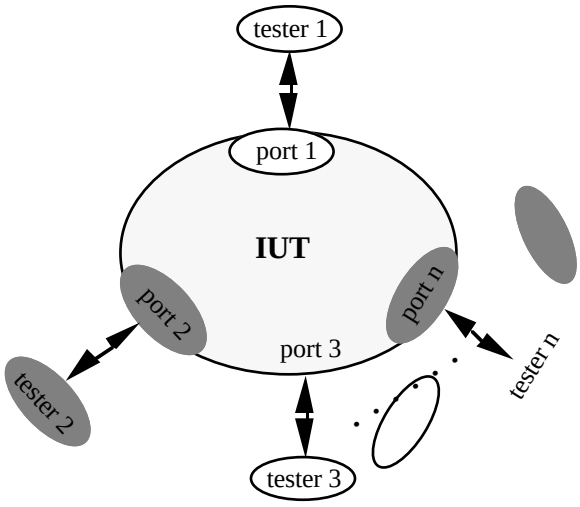


Figure 2.4. A general distributed test architecture

2.2. Finite state machines with n ports

A network with distributed terminals sometime can be modeled as a finite state machine with several ports. We define in the following the concept of *multi-port finite state machines*, which is a generalization of finite state machines with two ports given in [20].

DEFINITION *Multi-port finite state machine:*

A *multi-port finite state machine with n ports* (np-FSM) [illegible] as a 6-tuple (s_0) where $n \geq 1$ and

(1) St is a finite set of labels, called states.

(2) Σ is a n-tuple ($Li_1, Li_2, ..., Li_n$) where Li_k is a set of inputs for *port* k, and $Li_k \cap Li_j = \emptyset$ ($\emptyset$ denotes the empty set) when $k \neq j$ and k, j = 1, 2, ..., n.
Furthermore, we assume $Li = Li_1 \cup Li_2 \cup ... \cup Li_n$.

(3) Γ is a n-tuple ($Lo_1, Lo_2, ..., Lo_n$) where Lo_k is a set of outputs for port k, and $Lo_k \cap Lo_j = \emptyset$ when $k \neq j$, with k, j=1, 2, [illegible]
Furthermore, we assume $Lo = \{ <a_1, ..., a_m> \mid \exists Lo_{k1}, Lo_{k2}, ..., Lo_{km}$ [illegible] $a_1 \in Lo_{k1}, a_2 \in Lo_{k2}, ..., a_m \in Lo_{km}$ [illegible] $1 \leq k1 < k2 < ... < km \leq n \}$
where $<a_1, a_2, ..., a_m>$ is said to be an *output tuple* [illegible]

(4) T is a transition function: [illegible] $St \times Li$ [illegible]

(5) O is an output function. [illegible] for the [illegible] holds for empty output. (Note that the FSM is completely specified if $St \times Li$ [illegible]

(6) $s_0 \in St$ is the initial state. □

In other words, a np-FSM has n ports, say $p_1, p_2, ..., p_n$; each port p[illegible] is associated with a set of inputs Li_k and a set of outputs Lo_k. For any two different ports, [illegible]ere are no common inputs (outputs) in the two associated input sets (output sets). A np-FSM can also be represented by a directed graph in which the nodes are the states and the directed edges are transitions linking the states. Figure 2.5 shows an example of 3p-FSM under the distributed test architecture.

From this definition, a np-FSM is deterministic since a state and an input uniquely determines a next state and an output tuple; but it is not necessarily *completely* specified in that not every state/input pair has a defined next state and an output tuple.

Let $a \in Li$, $b \in Lo$, and P, Q $\in$ [illegible] we write P a/b Q to represent that $T(P, a) = Q$ and $O(P,a) = b$. P a/b Q is called *a transition from* P *to* Q *with the label* a/b. For the example of 3p-FSM shown in Figure 2.5, we have A 1/<a,c> B.

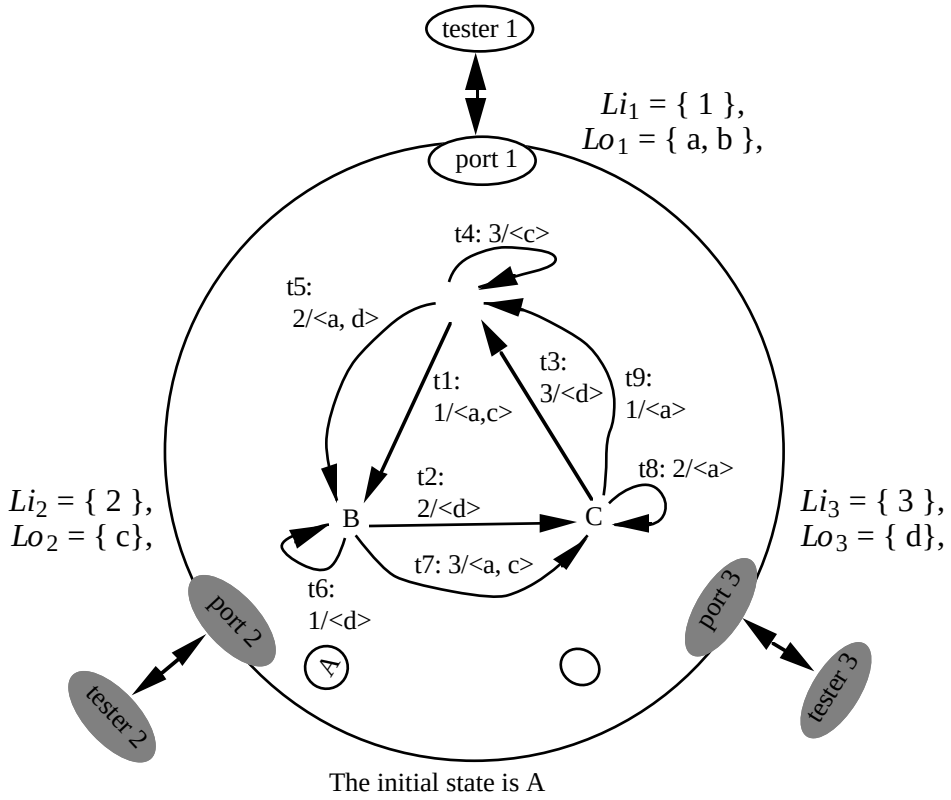


Figure 2.5. An example of 3p-FSM under distributed test architecture

The np-FSMs can model a communication network with n terminals where each terminal is viewed as a port. This model could be useful for network testing and verification. When n=2, 2p-FSMs are finite state machines defined in [20, 12, 3], which are viewed as IUTs under the distributed test architecture.

2.3. Fault Model

We present a fault model for np-FSMs. Like other state machine based test generation, fault models serve as a guide to test generation and as a basis for test coverage analysis, as described in [2]. Let SP and IUT be two np-FSMs, representing a specification and its implementation, respectively. Assume that they have the same Σ and Γ. Then the fault types are defined as follows:

(1) *Output fault:* We say that IUT has output faults if SP can be obtained from IUT by modifying the outputs of one or more transitions in IUT.
(2) *Transfer fault:* We say that IUT has transfer faults if SP can be obtained from IUT by modifying the end states of one or more transitions in IUT.
(3) *Hybrid fault:* We say that IUT has hybrid faults if SP can be obtained from IUT by changing the outputs and/or the end states of one or more transitions, in IUT.

3. A GENERALIZED SYNCHRONIZABLE TEST SEQUENCE METHOD

Although a number of test generation methods have been developed based on FSMs, they cannot be applied to the FSMs with two or more distributed ports directly, as pointed out by [20], since a so-called *synchronization problem* exists in the distributed test architecture. We generalize in this section the concept of so-called *synchronizable test sequence* given in [20] for 2p-FSMs to the case of np-FSMs, dealing with the synchronization problem. Based on the generalized concept, we present an approach to modifying the existing test generation methods for FSMs, such as the transition tour [17], the W-method [6], the DS-method [14] and so on, to obtain synchronizable test sequences for np-FSMs. For certain protocol specifications, it is impossible to avoid synchronization problems; we use techniques for testing nondeterministic FSMs to handle such specifications.

3.1. Synchronization problem and synchronizable test sequences

Under the architecture given in Figure 2.4, the testers are distributed over several sites. They are only synchronized through the interactions with the implementation. In this situation, considering two consecutive transitions t1 and t2 of a given np-FSM I (n≥2), one of the testers is said to face a *synchronization problem* if this tester did not take part in the first transition and if the second transition requires that it sends a message to the machine I.

For example, consider the 3p-FSM shown in Figure 2.5, tester3 faces a synchronization problem when considering the consecutive transitions t1 and t7. tester3 is supposed to send the input 3 to the machine after the t1 has been executed; however, since t1 neither receives any input from, nor sends any output to the tester3 (i.e., tester3 did not take part in the transition t1), tester3 does not know whether t1 has been executed, i.e., a synchronization problem.

We now define the concept of synchronizable test sequences for np-FSMs, n≥2, for handling the synchronization problem of general np-FSMs. Given a np-FSM I with the ports 1, 2, ..., n, we require the following concepts for the

DEFINITION *Interaction ports* (IP)*of a given transition* [pi, PO]**:**
Let pi∈ {1, 2, ..., n} and PO {1,2, ..., n} (n is the number of ports), [pi, PO] is said to be an *interaction port* (IP) *of a given transition* t if t receives an input from the port pi and sends to each port in PO an output (if PO=∅, t does not send any output). □

DEFINITION *Synchronizable test sequences***:** Given an ordered pair of transitions t1 and t2 of I, let [pi, PO] and [pi', PO'] be their IPs, respectively. t1 and t2 are said to be *synchronizable* if (1) pi=pi', or (2) pi'∈ PO.
A given test sequence is said to be *synchronizable* if any two consecutive transitions of the sequence are synchronizable. □

Formerly, people define the synchronizable test sequences for 2p-FSMs by means of the so-called basic transition list [20]. However, for np-FSMs, a generalized version of this list could be too large to handle. The concept of *interaction ports* provides a nice means to define the synchronizable test sequences, having avoided a cumbersome presentation**.**

As an example, for the 3p-FSM shown in Figure 2.5, the IPs of the transitions t1 and t2 are [1, {1,2}] and [2,{3}], respectively; the transitions t1 and t2 are synchronizable. It is easy to see that testers will not face any synchronization problem if synchronizable test sequences are used. Therefore, it is desirable to generate synchronizable test sequences for testing np-FSMs.

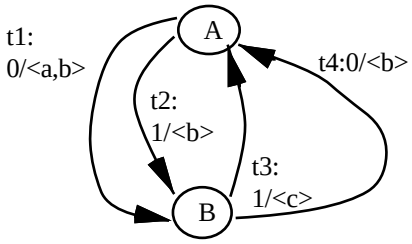
3.2. Generating synchronizable test sequences

Guided by the idea of synchronizable test sequences, we present a modified transition tour such that the resulting test sequences are all synchronizable, using an approach similar to the one given in [20]. The other existing test generation methods for FSMs can be modified to obtain synchronizable test sequences for np-FSMs, in a similar manner.

Any graph traversal algorithm such as the one given in [21] can be modified to obtain a transition tour. Assume that an algorithm *TT* produces a transition tour; we present in the following a procedure for generating synchronizable transition tours, by modifying the algorithm *TT*.

Generating synchronizable transition tour:

Each transition *t* to be added to the sequence by the algorithm *TT* is first checked whether it forms a synchronizable pair together with the last transition *tt* of the sequence; that is, assuming the IPs of the transitions *tt* and *t* to be [pi, PO] and [pi', PO'], respectively, check whether pi=pi' or (2) pi'∈ PO are true. If the transitions *tt* and *t* are not synchronizable, a different transition from the present state is considered. If no suitable transition exists from the present state, the algorithm *TT* backtracks to the previous state, continuing the tour from there in a different way. This process continues until all the transitions of the machine are covered.



Li_1 = { 0 }, Li_2 = {1 }
Lo_1 = { b }, Lo_2 = { a, c}

(a)

Input sequence: 0, 1, 1, 0

Resulting global sequence:

Port 1:	0	b				b	0	b	
Port 2:		a	1	c	1				

PIs: <1,{1,2}>, <2,{2}>,<2,{1}>,<1,{1}>

tester1 -- port 1
tester2 -- port 2
The initial state is A

(b)

Figure 3.1. An example of a 2p-FSM

For example, Figure 3.1b shows a synchronizable transition tour for the 2p-FSM shown in Figure 3.1a. Furthermore, for the example of 3p-FSM shown in Figure 2.5, a synchronizable transition tour is "t1, t2, t3, t4, t5, t6, t7, t8, t9".

Using an approach similar to the above, one can obtain the test generation methods for np-FSMs by modifying the DS-method [14], the W-method [6], the UIO-method [19] the Wp-method [9] and the Petrenko'method [18] such that the resulting test sequences are all synchronizable.

We note that for certain np-FSMs, given a transition, there may not exist any synchronizable test sequence that can force the machine to traverse this transition. Such an example was given in [20] for a 2p-FSM. In this case, one cannot obtain any test suite using the modified methods.

4. FAULT COVERAGE

In the distributed test architecture, since there is no other synchronization channel instead of IUT, the observability and fault detectability is weaker than that in the local test architecture. Therefore, it is difficult to achieve good fault coverage. We find that the modified methods presented in [20] do not ensure the same fault coverage as the corresponding original testing methods; and we also present an improved approach that provides better fault coverage. We finally show that no method can ensure full fault coverage for finite state machines in the distributed architecture.

4.1. Observability and fault detectability

For a system (software or communication system) that receives inputs and produces outputs, *observability* refers to the ease of determining if specified inputs affect the outputs; *fault detectability* refers to the ease of detecting specified fault. Observability and fault detectability of IUT vary in different test architectures.

Consider the ISO test architectures presented in the earlier section. An IUT has the highest observability and fault detectability in the local test architecture, and has lower ones in the coordinated, distributed architectures, and the lowest one in the remote test architecture. If one associates each test architecture (local, coordinated, distributed and remote test architecture) with a set of detectable faults (S_l, S_c, S_d,S_r), as shown in [8], the fault detectability in these architectures is reflected by the following:

$$S_r \subseteq S_d \subseteq S_c \subseteq S_l.$$

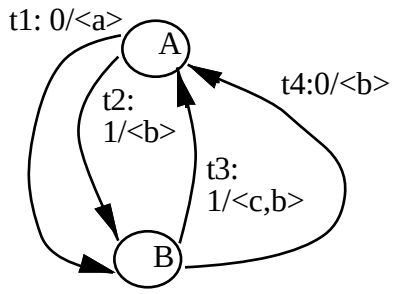
In the distributed testing architecture, the observability and fault detectability of IUT are limited [1]; therefore, it is difficult to achieve good fault coverage in the distributed test architecture, which is further explored in the following subsections.

4.2. A class of undetectable output faults by synchronizable test sequences

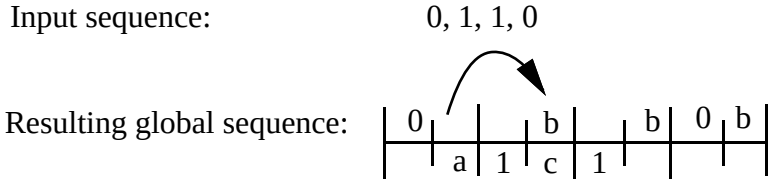
For FSMs, as we know, the transition tour [17] can detect all output faults if no transfer faults occur. However, for 2p-FSMs, a transition tour which is a synchronizable test sequence does not necessarily detect all output faults. For example, the 2p-FSM shown in Figure 4.1a is a faulty implementation of the 2p-FSM shown in Figure 3.1a; and it has only output faults. The input sequence "0,1,1,0" is a synchronizable transition tour, as described in Figure 3.1b. When the input sequence "0,1,1,0" is applied to the faulty implementation, we obtain the global

sequence shown in Figure 4.1b. Although the global sequence for the original 2p-FSM is different from the global sequence for the faulty one, the difference cannot be seen from the two local ports U and L since no global clock is assumed in the distributed test architecture. Therefore, a modified transition tour does not necessarily detect all output faults for 2p-FSMs.

Using similar arguments, it is easy to prove that the modified W-method and DS-method [20] do not necessarily detect all output and transfer faults for 2p-FSMs.



(a)



(b)

Figure 4.1. An example of output-shifting fault in a 2p-FSM

[12] presented a method to solve the synchronization problem by instrumenting probes into the IUT. However, since the method only guarantees that input sequence is synchronized, the above problem still exists. Furthermore, the instrumentation of the IUT is not permissible in the conformance testing when the testing is conducted by a second party other than the producer.

We formalize the class of output faults which may remain undetected by the modified methods as follows: For two consecutive transitions, say t1 and t2, in a given specification S, the faulty implementation I can be obtained from S by removing an output from one of the two transitions and adding the output to the other transition. We call this class of output faults *output-shifting faults.* We present in the following an outline of an approach which can detect this class of output faults.

The outline of the approach:

1. Generate a set of synchronizable test sequences, say Π, by using the approach of [20], with respect to one of the test generation methods for FSMs such as the transition tour [17], the W-method [6], and so on.

2. Find a set of all transition pairs, say Ω, along the paths caused by applying the sequences of Π, each pair of which may have an output-shifting fault which is undetectable using Π.

3. If Ω is empty, stop. Otherwise, add a set of additional synchronizable test sequences to Π, or concatenate some synchronizable sub-sequences to the sequences in Π, such that Π can ensure the absence of output-shifting faults in the transition pairs of Ω. Go to Step 2.

It is not very difficult to give a detailed procedure for Steps 2 and 3, but under what condition an efficient procedure exists and what is efficient procedure, is the work of future research. The worst case computation complexity of the above procedure is not polynomial since the procedure contains the generation of synchronizable test sequences; and no polynomial algorithm exists for the generation [5, 3]. We now explain the approach using the example of Figure 3.1a. Suppose that the transition tour is of our interest. Figure 3.1b shows the resulting test sequence after Step 1. Figure 4.2 shows the result of Step 2; the two arrows in the figure indicate the two possible output-shifting faults which cannot be detected by the test sequence. Figure 4.3 shows the result of Step 3; the test sequence can ensure the absence of any output-shifting faults shown in Figure 4.2. The transition string "t1,t4" ensures the absence of output-shifting fault in <t1,t3>, and the rightmost transition t3 of the test sequence ensures the absence of output-shifting fault in <t3,t2>.

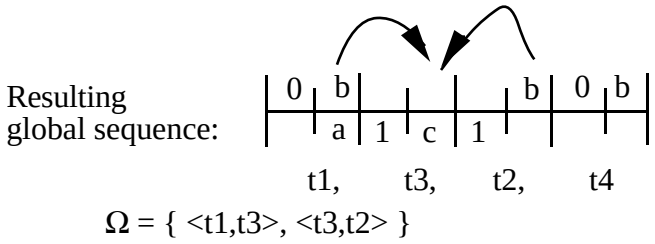


Figure 4.2. The example for Step 2

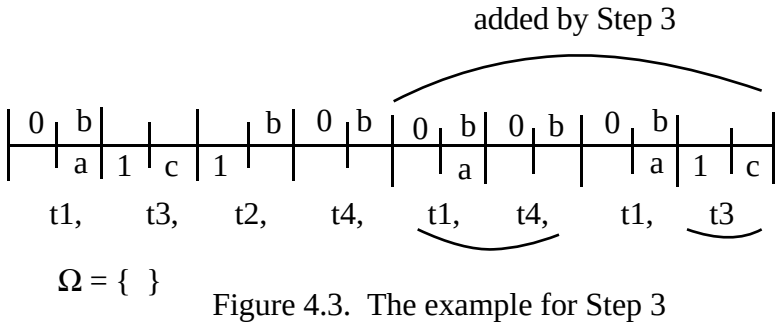


Figure 4.3. The example for Step 3

Although the improved approach detects more faults than the original, it does not necessarily guarantee the total absence of faults. Therefore, the approach is still heuristic in nature since no

precise fault coverage is easily given under the given fault model. We also note that it is impossible to guarantee the total absence of faults in the distributed test architecture, as described in the following section.

4.3. Impossibility of complete test coverage in a distributed test architecture

We show in the following that, in the distributed test architecture, no test suite (i.e., a set of test sequences) can ensure the total absence of the faults specified in the fault model. In other words, for a np-FSM, not all faults specified in the fault model are detectable.

We first note: no method that is based on the concept of synchronizable test sequences can ensure full fault coverage for all np-FSMs. The reason is that, for certain np-FSMs, given a transition, there may not exist any synchronizable test sequence that can force the machine to traverse this transition. Such an example was given in [20] for a 2p-FSM.

One may argue that, for the np-FSMs where some transitions cannot be forced to be traversed by any synchronizable test sequences, to traverse these transitions, certain test sequences can be applied to the machines repeatedly up to a sufficient number of time. Then the probability that not all these transitions are exercised at least once, can usually be reduced to close to zero (note: this is a common assumption for testing nondeterministic finite state machines). We argue that, even using such an approach and the assumption, still no test suite can ensure full fault coverage. We prove this using the 2p-FSM shown in Figure 4.4.

For the sake of convenience, we introduce in the following several terms for 2p-FSMs. A *test case* is denoted as two I/O sequences $[\sigma_U , \sigma_L]$ such that σ_U consists of the inputs and outputs of the port U, and σ_L of the port L, respectively. We say that a 2p-FSM I *passes* a test case $[\sigma_U , \sigma_L]$ if there is a path p in I such that when p is traversed, σ_U and σ_L will be observed at the ports U and L, respectively. A fault in the implementation I is *detectable* against its specification S if and only if there exists a test case $[\sigma_U , \sigma_L]$ such that S passes $[\sigma_U , \sigma_L]$ and I does not.

Consider the example of Figure 4.4, let the 2p-FSM be S (where plain line transitions make the machine complete). Assume that I is a corresponding faulty implementation which results from changing the output "f" of the transition t3 to "g" (see the dashed transition). The transition t3 in I has an output fault. We argue that the fault is not detectable. We note that all the paths which start from the initial state A and traverse the faulty transition t3, begin with the transition string "t1, t2, t3"; and t3 does not appear in the rest of the paths any more. The input sequence pair for the ports U and L, corresponding to the path "t1, t2, t3", is [a.b, c]. Since no synchronization exists between the ports U and L, the [a.b, c] may cause the machine to be executed along one of the three paths: "t1, t2, t3", "t1, t5, t7", "t4, t6, t7". The transitions of these paths are indicated by bold lines shown in Figure 4.4. The total of two test cases, [a/d.b/e, c/f] and [a/d.b/e, c/g] are derived from the three paths. If we assume on the contrary that the fault in t3 is detectable, it is not difficult to see that the fault must be detected using the two test cases [a/d.b/e, c/f] and [a/d.b/e, c/g]. However, from Figure 4.4, both S and I pass the test cases [a/d.b/e, c/f] and [a/d.b/e, c/g]. Therefore, the fault is not detectable.

Nevertheless, all faults are detectable if synchronization exists between the distributed ports or there is a global clock -- in this case, the test generation methods for FSMs can be applied directly. Therefore, the testability of IUTs under the distributed test architecture should be studied.

The initial state is A

Li_U = { a, b }, Li_L = {c }
Lo_U = { d, e }, Lo_L = { f, g}

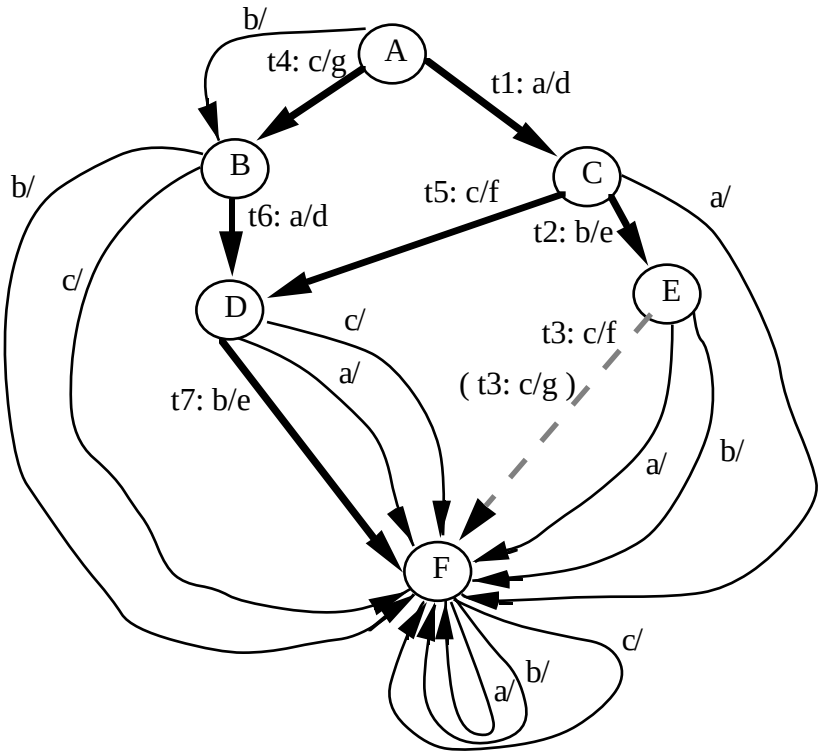


Figure 4.4. An example of 2p-FSM

5. CONCLUSION

Test architectures represent interfaces between systems to be tested and their testers; and they are important topics in the International Standardization Organization (ISO). A general distributed test architecture, which is a generalized version of the ISO distributed architecture, has been presented in the literature for testing distributed systems based on the Open Distributing Processing (ODP) Basic Reference Model (BRM). Based on this test architecture, for the systems modeled by finite state machines, we developed a test generation method. As an application example, we have applied this method to generate test sequences for a so-called quorum protocol [16] (the protocol is described in [11, 4]).

We investigated the issue of fault coverage. We pointed out that the methods given in the literature for the distributed test architecture, modified from certain existing methods, cannot ensure the same fault coverage as the corresponding original testing methods. We also studied the limitation of fault detectability in the distributed test architecture. The distributed test

architecture also gives rise to new problems of FSM based diagnosis, and the diagnosis method of FSMs of one port [10] may be modified for np-FSMs by applying the concept of synchronizable test sequences.

REFERENCES

1. G. v. Bochmann, R. Dssouli and J. Zhao, "Trace Analysis for Conformance and Arbitration Testing", IEEE Transactions on Software Engineering, Vol. SE-15, No.11, 1989, pp.1347-1356.
2. G.v. Bochmann, A. Das, R. Dssouli, M.Dubuc, A.Ghedamsi, and G.Luo, "Fault Models in Testing", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.17-30.
3. Sylvia Boyd and Hasan Ural, "The Syncronization Problem in Protocol Testing and Its Complexity", Information Processing Letters Vol.40, No. 8, (Nov. 1991) pp.131-136.
4. S. Ceri and G. Pelagatti, Distributed Batabases: Principles and systems, McGraw-Hill, New York 1984.
5. Wen-Huei Chen, Ching-sung Lu, Jinn-Tuu Wang, and Richard-Jinjr Lee, "Constrained Chinese Postman Problem with Its Application on Synchronizable Protocol Test Generation", Journal of Information and Engineering Vol.6, (1990), pp.149-157.
6. T.S.Chow, "Testing Software Design Modeled by Finite-State Machines, IEEE Trans. on Software Eng., Vol. SE-4, No.3, 1978.
7. J. de Meer, V.Heymer, J. Burmeister, R. Hirr and A.Rennoch, "Distributed Testing", Participants Proceedings of International Workshop on Protocol Testing Systems, Oct. 15-17th, 1991, the Netherlands, pp.IV43--51.
8. R. Dssouli and G. v. Bochmann, *Conformance Testing with Multiple Observers*, Proc. IFIP Workshop on Prot. Specification, Testing and Validation, 1986, North-Holland Publ., pp. 217-229.
9. S.Fujiwara, Gregor von Bochmann,F.Khendek,M.Amalou & A.Ghedamsi, "Test Selection Based on Finite State Models", IEEE Transactions on Software Engineering, Vol SE-17, No.6, June, 1991, pp.591-603.
10. A.Ghedamsi and G.v. Bochmann, "Diagnostic Tests for Finite State Machines", the proceedings of 12th International Conference on Distributed Computing Systems, in Japan, 1992, IEEE coputer society press, pp.244-251.
11. Maurice Herlihy, "A Quotum-Concensus Replication Method for Abstract Data Types", ACM Transactions on Computer Systems, Vol.4, No.1, Feb 1986, pp.32-53.
12. Darrell Hubbard, "Deterministic Execution Testing of FSM-Based Protocols", AT&T Technical Journal, Vol.69, No.1, 1990, pp.119-128.
13. ISO/TC97/SC21, OSI conformance Testing Methodology and Framework - Parts 1- 5, ISO, 1991.
14. Z. Kohavi, Switching and Finite Automata Theory, New York: McGraw-hil, 1978.
15. D.Y. Lee and J.Y. Lee, "A Well-Defined Estelle Specification for the Automatic Test Generation", IEEE Transactions on Computers, Vol.40, No.4, April, 1991, pp.526-542.

16. Gang Luo, Rachida Dssouli, Gregor v. Bochmann, Pallapa Venkataram and Abderrazak Ghedamsi, "Generating Synchronizable Test Sequences Based on Finite State Machine with Distributed Ports", Internal Report (available upon request).
17. S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", in Proc. IEEE Fault Tolerant Comput. Conf., 1981.
18. Alexandre Petrenko, "Checking Experiments with Protocol Machines", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.83-94.
19. K.Sabnani & A.T.Dahbura, "A Protocol Test Generation Procedure", Computer Networks and ISDN, Vol.15, No.4, 1988, North-Holland, pp.285-297.
20. Behcet Sarikaya and Gregor v. Bochmann, "Synchronization and Specification issues in Protocol Testing", IEEE Transactions on Communications, Vol.COM-32, No.4, April 1984, pp.389-395.
21. R.Tarjan, "Depth-first search and linear graph algorithms", SIAM J. Comput., vol.1, no.2, 1972.