Generating Interoperability Test Sequence for Distributed Test Architecture: 
A Generic Formal Framework *

Zhiliang Wang\textsuperscript{1}, Jianping Wu\textsuperscript{2}, and Xia Yin\textsuperscript{1}

Department of Computer Science, Tsinghua University, Beijing 100084, P.R. China
\textsuperscript{1}\{wzl,yxia\}@csnet1.cs.tsinghua.edu.cn
\textsuperscript{2}jianping@cernet.edu.cn

Abstract. Protocol interoperability testing is an important complement of conformance testing to ensure the quality of implementations of network communication protocols. In this paper, we present a generic formal framework of interoperability test sequence generation considering distributed test architecture. We firstly give an extended CFSM model, which represents each component machine as a multi-port FSM, to specify system under test conveniently. In the framework, the first step is to generate the centralized test sequence using reachability analysis approach; Then suitable distributed test architecture should be selected; After that, we discuss the controllability and observability problems in interoperability testing. In order to solve these problems, we present a formal algorithm to generate synchronizable interoperability test sequence. An example is given to illustrate this formal framework.

1 Introduction

In order to ensure the quality of network communication software, protocol test techniques are widely used. Conformance testing is a basic method of protocol testing, which can be used to test whether an implementation conforms to its protocol specification. As the complement of conformance testing, interoperability testing is often used to test whether two or more protocol implementations can communicate with each other correctly and inter-operate as a whole system to perform functions specified in protocol specifications. Interoperability testing is also being performed by IETF and ETSI in the process of protocol design [4].

A lot of researches on interoperability testing have been done, such as [3][4][9][12]. Interoperability test sequence generation is an important issue in this field. [2][5][10] have focused on it. In most of these works, the basic idea is to model the interoperability system under test as a system of Communicating

* This work is partially supported by the National Natural Science Foundation of China under Grant No. 90104002, the National High Technology Development 863 Program of China under Grant No. 2001AA112061, and 973 Program of China under Grant No. 2003CB314801.
Finite State Machines (for short, CFSM) and generate test sequence for the composition of these machines [9].

But in most of these previous works, only the situation of “1 against 1” has been considered, that is, in “interoperability system” under test there are only two IUTs; and no work considered the more common situation, such as “1 against N”. An example of the latter situation is that in mobile IPv6 protocol [8], there exist three different kinds of nodes: Correspondent Node (CN), Home Agent (HA) and Mobile Node (MN). On the other hand, interoperability system under test has the nature of distributed and concurrent, so distributed test architecture should be used to perform interoperability testing. But as we know, there is no work to generate interoperability test sequence for distributed test architecture. In this paper, we present a generic formal framework for distributed interoperability test sequence generation. In the process of generating test, controllability and observability problems may occur [1]. Distributed synchronizable test sequence should be generated to solve these problems. This formal framework can also be applied to the more common situation of interoperability testing.

The remainder of this paper is structured as follows. Section 2 gives the formal model used in this paper. We extend CFSM model to represent each component machine as a multi-port FSM in order to specify interoperability system under test definitely. Section 3 proposes distributed test architecture for interoperability testing. In section 4, we present our generic formal framework of interoperability test suite generation under distributed test architecture. Section 5 applies our method to an example. Conclusion and future work are given in section 6.

2 Formal Model

An interoperability system under test is composed of two or more protocol entities. To test interoperability of such a system, we assume specifications of all the protocol entities known, which is called specification-based interoperability testing [12]. From their specifications, interoperability test sequence can be derived. In order to specify interoperability system definitely, we should specify, (1) the behavior of each IUT; (2) abstract topology of network containing all the IUTs. In this paper, we extend CFSM model to represent each component machine as a multi-port FSM [6]. We also define the mapping relations of ports between component machines to specify the network topology.

**Definition 1.** Communicating multi-port FSM

A communicating multi-port finite state machine is a set of machines \( M = \{M_1, M_2, \ldots, M_m\} \), where \( m \) is the total number of component machines in the model; component machine \( M_i (i = 1, 2, \ldots, m) \) is an np-FSM (multi-port FSM with \( n \) ports) and it is an 8-tuple: \( M_i = (S, I, O, \delta, \lambda, s_0, Q, R) \), where,

1. \( S \) is the finite set of states of \( M_i \);
2. \( I \) is an \( n \)-tuple \( I = \{I_1, I_2, \ldots, I_n\} \), where \( I_k (k = 1, 2, \ldots, n) \) is the set of input symbols of port \( k \), which can be denoted as \( k:a(a \in I_k); T = I_1 \cup I_2 \cup \cdots \cup I_n \) is the set of input symbols of \( M_i \);
(3) $O$ is an n-tuple $O = \{O_1, O_2, \ldots, O_n\}$, where $O_k (k = 1, 2, \ldots, n)$ is the set of output symbols of port $k$, which can be denoted as $k : b (b \in O_k)$; $\overline{O} = (O_1 \cup \{\varepsilon\}) \times (O_2 \cup \{\varepsilon\}) \times \cdots \times (O_n \cup \{\varepsilon\})$ is the set of output symbols of $M_i$;

(4) $\delta : S \times I \rightarrow S$ is the state transition function;

(5) $\lambda : S \times I \rightarrow O$ is the output function;

(6) $s_0$ is the initial state of $M_i$;

(7) $Q$ is an n-tuple: $Q = (Q_1, Q_2, \ldots, Q_n)$, where $Q_k (k = 1, 2, \ldots, n)$ is the input queue of port $k$. These queues are all FIFO (First In First Out) ones;

(8) $R$ is the Port Mapping Relations. $R$ is an n-tuple: $R = (R_1, R_2, \ldots, R_n)$, where $R_k (k = 1, 2, \ldots, n)$ can be the format of 1) $M_j : h (j \neq i)$, which means the port $k$ of $M_i$ is connected to the port $h$ of $M_j$; 2) env, which means the port $k$ of $M_i$ is connected to external environment.

To simplify the discussions below, we assume that only one output is produced in one transition at most. In this definition, we also denote inputs (outputs) from (to) external environment as external inputs (outputs), and I/O symbols that interchanged between different IUTs as internal ones. 

Example:

Fig. 1. an example of communicating multi-port FSM of interoperability system

Fig. 1 is an example of communicating multi-port FSM model of interoperability system, where all the three component machines are 2p-FSMs: in machine $M_1$, two ports are $P_{11}$ and $P_{12}$; and in machine $M_2$, $P_{21}$ and $P_{22}$; $M_3$, $P_{31}$ and $P_{32}$. In all these ports, $P_{11}$ and $P_{21}$ are external ports and the others are internal ones. The port mapping Relations are (1) for $M_1$, $R_1 = env$ and $R_2 = M_3 : 1$; (2) for $M_2$, $R_1 = env$ and $R_2 = M_3 : 2$; (3) for $M_3$, $R_1 = M_2 : 2$ and $R_2 = M_2 : 2$. We use an example to illustrate the semantics of port mapping relations: say port $P_{12}$ of component machine $M_1$ is mapped into port $P_{31}$ of component machine $M_3$ (e.g. for $M_1$, $R_2 = M_3 : 1$), so when $M_1$ produces an output $u$ to port $P_{12}$, $u$
will become the input of port $P_{11}$ of $M_1$, and so on. The sets of I/O symbols are shown as Table 1.

<table>
<thead>
<tr>
<th>Port</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$i_1, i_2$</td>
<td>$z$</td>
<td>$x_1, x_2$</td>
</tr>
<tr>
<td>Output</td>
<td>$o_1, o_2$</td>
<td></td>
<td>$y_1, y_2$</td>
</tr>
</tbody>
</table>

Table 1: I/O symbols of Communicating np-FSM $M$

3 Distributed Interoperability Test Architecture

In interoperability testing, two or more IUTs can be contained in system under test. IUTs communicate with external environment via external ports, and communicate with each other via internal ports. We can consider interoperability testing as a black-box test method or grey-box test method. In the two methods, access point PCOs (Points of Control and Observation) can be used to control and observe external behaviors of the IUTs' external ports. In a grey-box test model, besides PCOs, another kind of access point POs (Points of Observation) should be used to observe the interactions between the IUTs. While in a black-box test model, only PCOs are used, and no POs. Obviously, gray-box test method has more powerful fault detection ability than black-box one. Fig. 2 shows a gray-box test method for SUT in fig. 1.

To perform interoperability testing, distributed test architecture should be used. We introduce the concept Test Component [11] into test architecture. Each test component can work with one or more access points to SUT (usually one$^1$), which executes the part of test cases relative to its own access points. According to different test scenarios, different test architecture using different test configurations should be selected. Fig. 2 is an example of test configuration for SUT in fig. 1. Test components PTC1, PTC2, PTC3, PTC4 are used to work with PCO1, PCO2, PO1, PO2 respectively; MTC is used to create the other four PTCs, and calculate the global verdict of test cases. All the test components can communicate with each others via a communication channel.

4 Proposed Generic Formal Framework

As described above, protocol interoperability testing should be put into practice under distributed testing architecture. In order to adapt to test architecture, the generated interoperability test sequence must be distributed to test components contained in the distributed architecture. We denote the test sequence, which is generated from formal model and contains behavior descriptions of all

$^1$ In order not to weaken the control ability of test system, we use one test component to work with just one PCO (or PO).
ports of SUT, as **Centralized Test Sequence** (for short, CTS), and test sequence distributed to each test component as **Distributed Test Sequence** (for short, DTS). As in conformance testing, in this process, controllability and observability problems[1] may occur. To solve these problems, coordination messages between test components should be added to the distributed test sequence. The resulting test sequence is called “**Distributed Synchronizable Test Sequence**”, which can be applied in the real distributed testing architecture. The whole framework of interoperability test sequence generation is shown in fig. 3.

![Fig. 3. Framework of interoperability test sequence generation](image)

### 4.1 Centralized Interoperability Test Sequence Generation

In our formal framework, the first step is to generate centralized interoperability test sequence. According to the specifications of SUT, we can generate global state reachability graph, from which centralized interoperability test sequences can be generated. In order to avoid the famous state explosion problem, in [5], two assumptions have been made: (1) system works in a slow environment; (2) no livelock exists in specification and implementation. Based on these assumptions, in [5], two principles of interoperability test generation have been proposed: (1) principle of stable states; (2) single stimulus principle. With these, only **Stable States**, which are unchangeable and stable if no external inputs from environment are applied, should be considered in global state reachability graph, and other states, namely, **Transient States**, are the global states that cannot be observed and controlled. So under these two assumptions, only in the stable states, an external input can be applied to SUT.

We extend the method used in [5] to our generic formal model - Communicating np-FSM, in which two or more machines can be contained. Say a communicating np-FSM is $M = \{M_1, M_2, \cdots, M_m\}$, and the state of $M_i (i = 1, 2, \cdots, m)$ is $S_i$. So the global state of $M$ is a vector $(S_1, S_2, \cdots, S_m)$; The transitions between stable global states in global reachability graph are denoted as global...
transitions (while transitions in each component machine are denoted as **local transitions**).

Fig. 4 shows an example of global states and transitions of model in fig. 1. Between the two stable global states there are three local transitions. They constitute a global transition, which we denote as \((i_1/u, u/z, z/o_2)\).

In interoperability testing, basic test requirements are interactions between different IUTs. So the purpose of interoperability testing is to find out all possible interactions between specifications of different IUTs and to check whether IUTs can act as them correctly. An interoperability test suite can be divided into several test cases. A test case can be denoted as the format \((\text{Preamble}, \text{Test body}, \text{Postamble})\). In such a test case, Test body, which focuses on the interactions between different IUTs, is the main part to test; Preamble is used to lead SUT to the initial global state of test body and Postamble can be used to lead SUT back to the initial global state of the system after test body\(^2\). In the framework, test bodies are just global transitions. For convenience, we make an assumption that all the machines contained in system have reset capability, that is, there exists an input \(r\) that takes the machines to their initial states\(^7\). For these integrated-structured test cases, they are all independent of each other: that is, they can be executed separately in any order.

**Test Selection Criterion:** Only global transitions containing more than one IUTs’ behaviors can be considered as test bodies.

A DFS (Depth First Search) or BFS (Breadth First Search) based algorithm can be used to generate all the test bodies and preambles. After that, an additional step is used to assemble them into centralized interoperability test cases. Fig. 5 shows an example of an interoperability test case: test body is the global transition \((i_1/u, u/v, v/w, w/z, z/o_2)\). From initial global state, the test case leads SUT to the global state \((a,2,D)\) using the preamble \((i_1/u, u/z, z/o_2)(x_1/y_1)\), and then executes the test body. So the test case is composed of three global transitions or nine local transitions.

In fact, an interoperability test case can be denoted as a sequence of global transitions:

\(^2\) In fact, Postamble should also check if the SUT is in a correct global state in the strict-defined test cases. But for convenience, we only consider one function of leading SUT back to the initial global state.
Definition 2. test case (sequence) for interoperability testing

An interoperability test case (sequence) for Communicating np-FSM M is \( \text{IoTC}(M) = Gt_1 \cdot Gt_2 \cdot \ldots \cdot Gt_L \), where, a) \( L \) is the total number of global transitions; b) \( Gt_i (i = 1, 2, \ldots, L) \) is the \( i \)th global transition in the test case: \( Gt_i = Lt_{i,1} \cdot Lt_{i,2} \cdot \ldots \cdot Lt_{i,\ell_i} \), where, \( \ell_i \) is the total number of local transitions in \( Gt_i \).

4.2 Distributed Synchronizable Interoperability Test Sequence

After generating centralized test sequence, the next step is to generate distributed synchronizable test sequences considering the real distributed test architecture. This process is known as Test Sequence Distribution.

Controllability and Observability Problems in interoperability testing

We assume a given interoperability test sequence as \( \text{IoTS}(M) = Gt_1 \cdot Gt_2 \cdot \ldots \cdot Gt_L \), so:

(1) controllability problem:

Intuitively, controllability problem is that test components cannot determine when to apply the external inputs to its PCO(s). To satisfy the assumption of slow environment, an external input must be applied to SUT only when SUT is in stable state. So test system must assure this point when applying an input.

Condition of controllability problem:

For the consecutive two global transitions \( Gt_i, Gt_{i+1} (i = 1, 2, \ldots, L - 1) \) in \( \text{IoTS} \): If \( \text{Port}(\text{Input}(Gt_{i+1})) \not\subseteq \text{Port}(\text{Output}(Lt_{i,j})) \)\(^3\), that is, for the consecutive two global transitions, PCO of the second global transition’s external input is not contained in PCOs (POs) of output(s) in the last local transition of the first global transition, controllability problem will occur when external input of \( Gt_{i+1} \) is applied to SUT.

In order to solve the controllability problem, a coordination message should be sent to the test component \( TC_j \), which will apply the external input of the second global transition, by the test component \( TC_i \) that has received the last output of the first global transition. We denote such a coordination message as \( \text{CM}(TC_i \rightarrow TC_j) \).

(2) observability problem:

Intuitively, observability problem is that test components cannot determine when to begin and stop observing (receiving) I/O symbols from its PCOs (or POs). Observability problem will leave the output-shift fault undetectable\(^{[13]} \).

Condition of observability problem:

For a global transition \( Gt_i \) in \( \text{IoTS} \):

If \( \text{Port}(\text{Input}(Gt_i)) \neq \text{Port}(\text{Output}(Lt_{i,j}))(j = 1, 2, \ldots, \ell_i) \), that is, PCO (PO) of a global transition’s output is not the same with PCO of this global transition’s external input, observability problem will occur when observing these outputs.

\(^3\) NOTE: function(1) \( \text{Input}(Gt) \) returns the set of external input symbols of global transition \( Gt \); (2) \( \text{Output}(Lt) \) returns the set of output symbols of local transition \( Lt \); (3) \( \text{Port}(s) \) returns the set of all access points associated with the symbols in set \( s \).
In order to solve the observability problem, coordination messages should be sent to all the test components which will receive the outputs as responses to the external input, by the test component applying the external input.

**Distributed Synchronizable Test Sequence Generation Algorithm**

**Algorithm 1**

**INPUT:** CTS IonTS(M) = Gt1 · Gt2 · · · · · Gl, test components TCi (k = 1, 2, · · · , n).

**OUTPUT:** DTS(TCi) (k = 1, 2, · · · , n).

**BEGIN**

1. for i from 1 to L do
2. Get TCi such that Port(Input(Gi)) is contained in TCi;
3. if (i > 1) then /* need not consider the first global transition */
4. if Port(Input(Gi)) \notin Port(Output(L(i−1)∪j=1−1)) then
   /* controllability problem occurs */
5. Get TCi such that Port(Output(L(i−1)∪j=1−1)) is contained in TCi;
6. DTS(TCi) ← DTS(TCi) ∪ CM(TCi → TCi);
7. DTS(TCi) ← DTS(TCi) ∪ CM(TCi → TCi);
8. outputPCO_set ← φ;
9. for j from 1 to l, do
10. outputPCO_set ← outputPCO_set ∪ Port(Output(Lj,));
11. for each PCOi in outputPCO_set do
12. Get TCi such that PCOi is contained in TCi;
13. if (Port(Input(Gi)) \notin PCOi) then /* observability problem occurs */
14. DTS(TCi) ← DTS(TCi) ∪ CM(TCi → TCi);
15. DTS(TCi) ← DTS(TCi) ∪ CM(TCi → TCi);
16. DTS(TCi) ← DTS(TCi) · Input(Gi), /* add external input */
17. for j from 1 to l do/* add all outputs */
18. if (Port(Output(Lj,)) \notin φ) then /* PCO(PO) is defined */
19. Get TCi such that Port(Output(Lj,)) is contained in TCi;
20. DTS(TCi) ← DTS(TCi) · outputsym;

**END**

NOTE: “!” (“?”) means “send” (“receive”) a coordination message.

According to the above discussions, it is obvious that:

**Proposition 1.** Resulting distributed test sequence from Algorithm 1 is distributed synchronizable test sequence, that is, the sequence is free from controllability and observability problems.

Now we analyze the complexity of algorithm 1. In the loop of Line (11)-(15), PCO number in the set outputPCO_set must be less than or equal to l, So the computational expense of algorithm 1 is less than or equal to 3 ∑L

Thus the upper bound of complexity is O(Llocal), where Llocal = ∑L

is just the total number of local transitions in the given CTS. On the other hand, maximal number of the coordination messages added to the given test sequence is 2 * L + 2 * (n − 1) * L, so its upper bound is O(nL). From the resulting test sequences, concurrent TTCN (TTCN-2)[11] test cases can be generated conveniently.
5 Case Study

We use the formal model in Fig. 1 to illustrate our framework. We apply two methods to generate CTSs. The result is as shown in Table 2:

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Test Cases</th>
<th>Number of Global Trans</th>
<th>Number of Local Trans</th>
<th>Number of Test Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS-based</td>
<td>16</td>
<td>40</td>
<td>122</td>
<td>162</td>
</tr>
<tr>
<td>DFS-based</td>
<td>16</td>
<td>44</td>
<td>138</td>
<td>182</td>
</tr>
</tbody>
</table>

Two methods obtain the same number of interoperability test cases, but BFS-based method obtains interoperability test suite of smaller length on global transitions, local transitions and test events. The main cause is that BFS-based method can obtain preambles of smaller length on global transitions than DFS-based method.

Under distributed test architecture shown in fig. 2, we apply algorithm 1 to the given test case in fig. 5. Fig. 6 is the MSC-like result.

Fig. 6. MSC-like result of Distributed Synchronizable Test Sequence generation

In the resulting test case of fig. 6(b), five coordination messages are added. It is easy to determine that the resulting test case is free from controllability and observability problems, so it is a distributed synchronizable test case.

6 Conclusion

In this paper, we present a generic formal framework of protocol interoperability test generation for distributed test architecture. At first, centralized interoperability test sequence should be generated from formal model using the method
of reachability analysis. Then suitable distributed test architecture should be selected. The next step is to generate distributed synchronizable test sequence that can be performed in real distributed test architecture. We discuss the controllability and observability problems existing in interoperability testing. To solve these problems, a formal algorithm is presented. The essential cause of leading to these problems is that the introduction of multiple test components weakens the power of control and observation of test system, so coordination messages between test components should be added into test sequence to enhance fault detection ability. We also give an example to illustrate such a formal framework, which can be proven very efficient and feasible.

In our future work, this method will be applied in real protocol interoperability testing. And we will also investigate more deeply test generation method for interoperability testing based on formal model.

Acknowledgments

We would like to thank Kai Wang and Mingli You, Tsinghua University, for implementing part of algorithms in this paper.

References