INTEROPERABILITY TEST GENERATION FOR COMMUNICATION PROTOCOLS BASED ON MULTIPLE STIMULI PRINCIPLE[3]

Soonuk Seol[†], Myungchul Kim[†], and Samuel T. Chanson[††]
[†]*Information and Communications University, Korea*
{ suseol, mckim }@icu.ac.kr
[††]*Hong Kong University of Science & Technology, China*
chanson@cs.ust.hk

Abstract This paper presents a method for generating interoperability test suites for the class of communication protocols such as the ATM/B-ISDN signaling protocol and TCP that are modeled as communicating nondeterministic finite-state machines. To our knowledge, no test generation method exists for protocol interoperability testing that supports multiple simultaneous inputs to the implementations. In other words, it has been assumed that communicating systems adhere to the single stimulus principle in generating test cases. In practice, however, there exists the possibility that additional messages will be sent to a machine while the previous message is still being processed, and/or simultaneous messages will be sent to several machines at the same time (e.g., TCP's simultaneous open/close). To cope with these situations, we have developed an interoperability test suite derivation algorithm based on formal models. Experimental results show that our method is applicable to practical systems and can generate validation-equivalent interoperability test suites in terms of transition coverage.

Keywords: Interoperability testing, protocol testing, test case generation, validation, single stimulus, multiple stimuli, transition coverage, stable state, TCP, ATM

1. INTRODUCTION

To increase the confidence that protocol products conform to international standards, various protocol testing methodologies have been developed and applied. In particular, conformance testing that checks whether an implementation is correct with respect to the relevant standards

[3] This work has been supported by KOSEF (No. 995-0900-007-2).

has been standardized by ISO [1] and ITU-T [12] and applied to protocols such as N-ISDN and ATM/B-ISDN. Nevertheless, it is well known that conformance testing has limitations in ensuring interoperability. Thus, even two conforming implementations may fail to interoperate [2, 3]. The reasons include ambiguity of protocol standards, incompatible option settings and incomplete conformance testing [4].

There has been some research work on interoperability testing in the literature. Rafiq and Castanet [2] dealt with interoperability test suite generation based on reachability analysis. However, a rigorous definition of interoperability was not used and the work considered only the case where lower testers exist between two Implementations Under Test (IUTs). Vermeer and Blik [5] described experiences with interoperability testing of the FTAM protocol. The work used a single tester between two IUTs and thus had limited capability. Arakawa et al. [3] derived a conformance test suite and an interoperability test suite separately and later manually combined them to reduce the number of conformance test cases. Kang and Kim [6] developed a method for systematically dealing with symmetric communication protocols and used an interoperability test architecture that did not observe the interface behavior between two IUTs. Shin and Kang [4] proposed and applied a test derivation method suitable for testing interoperability for the class of communication protocols such as the ATM/B-ISDN signaling protocol. Seol et al. [7] later enhanced the algorithm in [6, 4] and applied the algorithm to the ATM/B-ISDN signaling protocol and TCP with analysis on transition coverage. Kang et al. [8] developed a coherent framework of interoperability testing and a systematic interoperability test suite derivation method based on the framework.

Quite a number of methods and tools for test case generation get inputs in the form of deterministic, nondeterministic, or extended Finite State Machine (FSM) specifications. However, these methods are generally applicable only when the protocol consists of a single FSM. For communication protocols, most interoperability test suite derivation methods are based on some form of reachability analysis and the protocols are modeled by communicating finite state machines. To our knowledge, the single stimulus principle has always been explicitly or implicitly assumed. These models therefore do not support multiple stimuli. The single stimulus principle means that only a single stimulus can be given at each stable state. A stable state for concurrent systems or communicating protocols is defined in [9] as the system state (i) that is reachable from the initial state adhering to the single stimulus principle, and (ii) from which no change can occur without another stimulus. Systems adhering to the single stimulus principle are said to be running in a slow environment [10]. In a slow environment, inputs can be sent from the environment to the system only when all the queues and all the channels are empty. According to the single stimulus principle, simultaneous stimuli and stimuli during transitions are not considered.

When protocol implementations interact with one another, it is possible that additional messages are sent to the implementation(s) while the previous message is still being processed and/or simultaneous messages are sent to each implementation at the same time. For instance, TCP allows simultaneous open and simultaneous close of a TCP connection. Therefore, checking multiple stimuli in interoperability testing is important because multiple stimuli are an essential part of many protocol behaviors and their implementation seems to be especially error-prone.

In this paper, we propose a new method to support concurrent behavior of protocols in generating test cases for interoperability testing. The rest of this paper is organized as follows: Section 2 presents some related work. In Section 3, we define formal models for communication protocols and interoperability test cases. Section 4 explains our approach to interoperability testing and the procedure for generating interoperability test suites. In Section 5, we present the results of applying our proposal to the TCP and ATM/B-ISDN signaling protocols with transition coverage analysis. Finally, Section 6 summarizes the contributions of the paper and suggests further research directions.

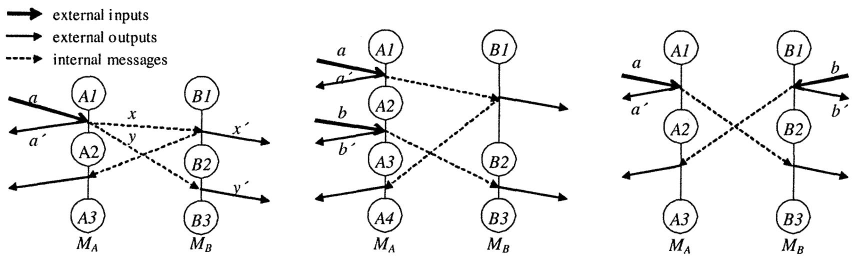
2. RELATED WORK AND MOTIVATION

In this section, we present some existing methods that are related to our work and discuss their limitations due to the single stimuli assumption.

Luo et al. [10] proposed a method of generating test sequences for concurrent programs and communication protocols modeled as communicating nondetermini-stic finite state machines (CNFSMs). The method first reduces a system of CNFSMs into a single NFSM by reachability analysis. The NFSM is then transformed into a trace-equivalent observable NFSM from which test sequences are generated. Basically, a global state denotes contents of channels, input queues, and states in each machine. As mentioned in their paper, it is obvious that the number of states in a global state machine becomes infinite in case of unlimited channels and/or queues. For that reason, they made the assumptions that queues and channels are bounded, and that the system runs in a slow environment. The first assumption is reasonable since unbounded queues or unbounded channels cannot be implemented in practice. The second assumption, however, is not reasonable for those protocols that allow multiple and simultaneous messages.

Paula Lima and Cavalli [11] proposed a pragmatic approach to generating test sequences for embedded components of complex systems. Embedded testing and interoperability testing are similar in that internal actions are usually invisible and thus hard to control. Their method provides means to support multiple outputs (i.e., when one signal from the environment stimulates several simultaneous outputs) and non-determinism of components. A drawback of their method is that it limits the number of external inputs between stable states to one. In other words, an input can be given to the system only if the previous input has been completely processed and the system is not undergoing any state change at the time.

Recently, Kang et al. [8] proposed a framework for interoperability testing of communication protocols. They developed an interoperability test derivation method based on stable states, and applied their method to the ATM signaling protocol. This method also has the same drawback as the other methods mentioned above. As their method adheres to the single stimulus principle so that multiple stimuli and stimuli during transitions between stable states cannot be handled.



(a) multiple internal outputs for one input (b) multiple stimuli during a transition between stable states (c) simultaneous stimuli

- from one machine to another - from the environment to the system of machines

Figure 1. Multiple stimuli.

In practice, multiple messages between communicating entities often occur. Figure 1 shows three cases of multiple messages in MSC-like charts. There are two machines which exchange messages with each other and with the environment. In this paper, we shall call the former messages internal messages and the latter external messages. Case (a) involves multiple stimuli from one entity to the peer entity for a given external input while cases (b) and (c) involve multiple stimuli from the environment to the system of machines. In (a), the machine MA produces two internal messages x and y arising from the external input a. Note that multiple output messages can be produced not only from an external input but is possible also from an internal input. In (b), external input b is given while the previous input a is still being processed. This case is an example of multiple stimuli during a transition from one stable state to another. Note that the global states (A1, B1) and (A4, B3) in Figure 1(b) are stable states. A similar case is shown in Figure 1(c) where one input goes to MA and another input goes to MB at the same time. This is known as simultaneous stimuli which is a special case of multiple stimuli. Simultaneous stimuli can be seen in TCP's simultaneous open and close behaviors. These cases can often happen in the real world and may also occur in faulty implementations. Therefore, interoperability testing should test for these behaviors as well.

3. FORMAL MODELS

This section defines formal models for communicating entities and for interoperability test cases. The specifications and implementations of communicating entities can be modelled using a special type of FSMs known as input output state machine (IOSM). IOSM is defined as follows:

Definition 1. An IOSM is a 5-tuple ($\mathbf{S}$, S_0, $\mathbf{L_{in}}$, $\mathbf{L_{out}}$, $\mathbf{Tr}$) where:

(1) $\mathbf{S} = \{ S_0, S_1, \ldots, S_{n-1} \}$ is a set of states;

(2) $S_0 \in \mathbf{S}$ is the initial state;

(3) $\mathbf{L_{in}} = \{ v_1, v_2, \ldots, v_m \}$ is a set of input symbols;

(4) $\mathbf{L_{out}} = \{ u_1, u_2, \ldots, u_k \}$ is a set of output symbols;

(5) $\mathbf{Tr} \subseteq \{ S_s \text{—} v/U \rightarrow S_d \mid S_s, S_d \in \mathbf{S} \wedge v \in \mathbf{L_{in}} \wedge U \in \mathbf{P}(\mathbf{L^*_{out}}) \}$ is a set of transitions where $\mathbf{L^*_{out}}$ denotes the set of sequences of symbols in $\mathbf{L_{out}}$ and $\mathbf{P(X)}$ denotes the power set of the set $\mathbf{X}$.

Bold letters in Definition 1 represent sets. $\mathbf{L_{in}}$ is divided into two sets $\mathbf{L_{in,E}}$ and $\mathbf{L_{in,I}}$ that are, respectively, the set of external input symbols and the set of internal input symbols. Similarly, $\mathbf{L_{out}}$ is divided into two sets $\mathbf{L_{out,E}}$ and $\mathbf{L_{out,I}}$ that are, respectively, the set of external output symbols and the set of internal output symbols. For example, symbol a in Figure 1(a) is an external input symbol and a' is an external output symbol. Symbols x and y are internal output symbols for machine M_A and, at the same time, are internal input symbols for machine M_B. "$S_s \text{—} v/U \rightarrow S_d$" is a transition where S_s, S_d, v, and U are, respectively, the starting state, the destination state, an input symbol and a sequence of output symbols. For a transition tr in $\mathbf{Tr}$, the '.' notation is used to refer to one of the elements of the following sets such as S_s, S_d, v, and U. For example, $tr.S_s$ represents the starting state of the transition tr. In case of a deterministic IOSM, there is at most one transition in $\mathbf{Tr}$ for each input symbol while there is at least one transition in a completely defined IOSM.

Definition 2. For every transition tr ($\forall tr \in \mathbf{Tr}$) whose starting state is S_i ($tr.S_s = S_i$), the state $S_i \in \mathbf{S}$ is said to be:
(1) *controllable* iff $tr.v \in \mathbf{L_{in,E}}$;
(2) *uncontrollable* iff $tr.v \in \mathbf{L_{in,I}}$;
(3) *semi-controllable* iff $\exists tr_j.v \in \mathbf{L_{in,E}}$ and $\exists tr_k.v \in \mathbf{L_{in,I}}$ where $j \neq k$.

We classify states in an IOSM into three types according to Definition 2. If a state can be changed only by external inputs, the state is said to be *controllable.* Such a state is controlled only by the testers' inputs. If all possible inputs at a state are internal ones, the state is said to be *uncontrollable.* If some possible inputs at a state are internal and others are external, the state is *semi-controllable.* This classification is illustrated in Figure 2. In IOSM M_A, a part of which is depicted in the box in Figure 2, state *A1* is controllable, *A3* is uncontrollable, and *A2* is semi-controllable. There are two transitions from state *A2* as depicted in the box in Figure 2. One transition goes to state *A3* with external input y and the other goes to state *A4* with internal input c. Let us trace the behavior of multiple stimuli in this system. When the system's global state is (*A1*, *B1*) which is a stable state, an external input x is given. Then, M_A goes to state *A2*, sending an external output x' and an internal output a, and M_B replies with an internal message c. Before the message c reaches M_A, another external message y (multiple stimuli from the environment) is given. This is because the state *A2* of M_A is a semi-controllable state. If M_A receives the internal message c in state *A2*, it will go to state *A4*, but it will go to state *A3* if it receives the external input y. Although the system is in a transient global state (*A2*, *B2*), the next stable state of the system is unpredictable since the transient global state includes a semi-controllable state. In general, if a machine in a system is in a semi-controllable state and an internal signal comes from a peer communication machine, the existing test generation methods cannot examine the external inputs since the methods adhere to the single stimulus principle.

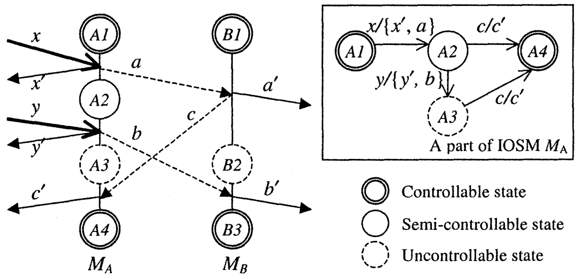


Figure 2. Three types of states in an IOSM.

Now we define a behavior model for two IOSMs communicating with each other and the environment. Let Π be the composition of the two IOSMs, one, M_A, for the specification A and the other, M_B, for the specification B. Then Π is defined as follows:

Definition 3. The composite IOSM Π is a 5-tuple ($\mathbf{S}_\Pi$, $S_{\Pi 0}$, $\mathbf{L}_{\Pi,\mathrm{in}}$, $\mathbf{L}_{\Pi,\mathrm{out}}$, $\mathbf{Tr}_\Pi$, Q_Π) where:

(1) $\mathbf{S}_\Pi = \{ S_{\Pi 0}, S_{\Pi 1}, \ldots, S_{\Pi n-1} \}$ is a set of global states, and $S_{\Pi i} = (S_A, S_B)$, where S_A and S_B are states of M_A and M_B, respectively;

(2) $S_{\Pi 0} \in \mathbf{S}_\Pi$ is the initial state. Namely, $S_{\Pi 0} = (S_{A0}, S_{B0})$;

(3) $\mathbf{L}_{\Pi,\mathrm{in}} = \{ v_{\Pi 1}, v_{\Pi 2}, \ldots, v_{\Pi m} \}$ is a set of input symbols. These messages are external inputs of each machine. Namely, $\mathbf{L}_{\Pi,\mathrm{in}} = (\mathbf{L}^{A}_{\mathrm{in,E}} \cup \mathbf{L}^{B}_{\mathrm{in,E}})$;

(4) $\mathbf{L}_{\Pi,\mathrm{out}} = \{ u_{\Pi 1}, u_{\Pi 2}, \ldots, u_{\Pi k} \}$ is a set of output symbols. These messages are external outputs of each machine. Namely, $\mathbf{L}_{\Pi,\mathrm{out}} = (\mathbf{L}^{A}_{\mathrm{out,E}} \cup \mathbf{L}^{B}_{\mathrm{out,E}})$;

(5) $\mathbf{Tr}_\Pi \subseteq \{ tr_{\Pi 0}, tr_{\Pi 1}, \ldots, tr_{\Pi t-1} \}$ is a set of sequences of transitions, and a sequence of transitions $\{ tr_\Pi \} \Leftrightarrow tr_0, tr_1, \ldots, tr_{l-1}$ where $tr_i \in (\mathbf{Tr}_A \cup \mathbf{Tr}_B)$;

(6) $Q_\Pi = (Q_A, Q_B)$ is a pair of directed channels where Q_A is the channel from M_B to M_A, and Q_B from M_A to M_B; both channels are represented by queues.

A global state and directed channels (modelled by queues) in a composite IOSM Π determine the system state of the IOSM Π. Here, the system state is used to describe the state of the system as a whole while a global state is used to depict each local state of the IOSMs in the system. Note that work in [8, 10] defined a global state as one describing the entire system, including information for input queues and/or channels.

An IOSM receives messages from either the environment or its channel queue. Then, the two IOSMs and the two directed channels constitute a composite IOSM Π as shown in Figure 3. Note that in this model IOSM Π does not have any queue for inputs from the environment. This is the same as a single IOSM because its formal model does not define any queue. Since each state in an IOSM consumes one message at a time, messages following the current message do not affect the next state. We assume that internal outputs are instantly placed in the channel queue directed to the receiving IOSM. This implies *first-come first-served* (FCFS) message queuing and handling. A message leaving one machine before another message will arrive at the other machine and served before the other message.

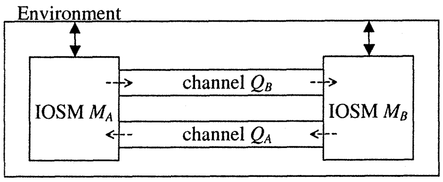


Figure 3. A composite IOSM Π.

A global state is classified into one of two types depending on the channel contents.

Definition 4. The global states of two IOSMs are classified into two types.
$\forall S_\Pi = (S_A, S_B) \in \mathbf{S}_\Pi$,
(1) A global state S_Π is said to be *stable* iff Q_A and Q_B are both empty.
(2) S_Π is in a *transient* state otherwise.

When a system is in a stable state, the system will change its state due to external inputs only. Both local states in a stable state cannot be uncontrollable states, otherwise the system is in a deadlock state because both machines will wait for messages from each other indefinitely. When a system is in a transient state, the system will change its state soon. However, the existing test generation methods do not allow for the system to receive additional external inputs. Our method, on the other hand, examines all possible external inputs. Namely, when one of the two local states in a transient global state is semi-controllable, we examine the cases where the IOSM receives an external input in the semi-controllable state as well as the cases where it consumes an internal message from its receiving channel.

Finally, we define an interoperability test suite in Definition 5. Each machine in a system consumes one input (either internal or external) at each state, that is, one transition at a time for each IOSM. Therefore, an interoperability test case can be represented by a sequence of transitions in either IOSMs in the system. An interoperability test suite is defined in terms of interoperability test cases. Note that a test case whose length is 1 is simply a conformance test case.

Definition 5. An interoperability test suite $\mathbf{IOPTS} \subseteq \mathbf{Tr}_{\Pi}$ is defined as follows: $\mathbf{IOPTS} = \{ iopt_0, iopt_1, \ldots, iopt_{n-1} \}$ is a set of interoperability test cases where an interoperability test case is defined as a set of sequences of transitions,

$\{ iopt \} \Leftrightarrow tr_0, tr_1, \ldots, tr_{l-1}$ where $tr_i \in (\mathbf{Tr_A} \cup \mathbf{Tr_B})$, $0 \leq i \leq l-1$, and $l \geq 2$.

The interoperability test case illustrated in Figure 2 can be represented as follows. The length of this interoperability test case is 4. The starting stable state is (*A1*, *B1*) and the destination stable state is (*A4*, *B3*).

A1-x/{x',a}->A2, A2-y/{y',b}->A3, B1-a/{a',c}->B2, B2-b/b'->B3, A3-c/c'->A4

4. OUR PROPOSED APPROACH

This section describes our approach to generate interoperability test cases based on the formal models defined in Section 3. First, we explain the overall procedure of our approach and the associated assumptions. Next, we select the test architecture that achieves the desired test coverage. Finally, we discuss how to generate interoperability test cases based on the multiple stimuli principle. For the purpose of analysis, we divide the multiple stimuli into two sets classified in Figure 1 and considered in Section 4.3 and 4.4, respectively.

4.1 An overview

In order to derive test cases, we first need to derive IOSMs from the given protocol specifications. We assume that there is no autonomous transition, e.g., a timer. This is because even though we are able to generate test cases involving time constraints, it is difficult to apply them to real test environments. The interoperability test derivation algorithm generates test cases from the IOSMs by constructing an interaction graph. The algorithm starts from the initial stable state consisting of each IOSM's initial states and at each step examines all possible external inputs even in transient states, until a stable state is reached. In the meantime, new paths are generated based on the multiple stimuli principle, and newly found stable states are added to the associated state space. This procedure is repeated for every new stable state until all stable states are covered. One can apply our algorithm for deriving interoperability test cases based on the single stimulus principle as well, especially when resources (such as available points of control and observation) are limited. The input-enumeration procedure used in our approach is similar to what is known as state-perturbation [15]. However, the goal of our procedure is to build a reachability tree and then generate interoperability test cases based on its stable states, while the goal of state-perturbation is to build a reachability tree for validating protocols between interacting processes without the concept of stable state. Moreover, generating interoperability test cases mainly focuses on inputs and outputs to and from the systems of machines, which are regarded as black- or grey-boxes. However, the model (labelled transition systems) used in paper [15] is focused on internal interactions between systems of machines only.

4.2 Test architecture

To cater to the multiple stimuli principle, we need a test architecture with a tester between the two IUTs as well as one at each end as shown in Figure 4. The arrows in Figure 4 represent points of control and observation (PCOs) with both observability and controllability functions. It is assumed that there is a communication channel through which the testers coordinate with each other. Tester C should be able to hold messages passing through it for a while. In this way, simulating multiple stimuli is feasible. In the case of TCP/IP, for example, tester C will be configured with two network interface cards for each IUT. The tester duplicates and drops all the messages between the two IUTs using libpcap [14], a library for system-independent packet capture, and then sends the messages according to the interoperability test cases, coordinating with the other two testers in doing so.

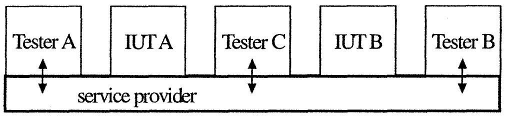


Figure 4. Interoperability test architecture.

4.3 Multiple stimuli from one machine to another

Figure 5 shows the channel status (modelled by queues) for every transition execution in deriving an interoperability test case in the given example. Starting with an external input *ex*, the algorithm pushes every internal output generated by the input into the channel Q_{AB}. Next it moves to the peer machine M_B and processes all the messages in the channel and pushes all internal outputs produced during the process into the channel Q_{BA}. This procedure is repeated until both channels are empty. Note that the order of messages, a, b, c (or, 0, 1, 2, 3) remains unchanged since first-come first-served scheduling is used. Moreover, protocol concurrency is still maintained, e.g., $tr_b \mid tr_1$, $tr_c \mid tr_2$, and $tr_c \mid tr_3$.

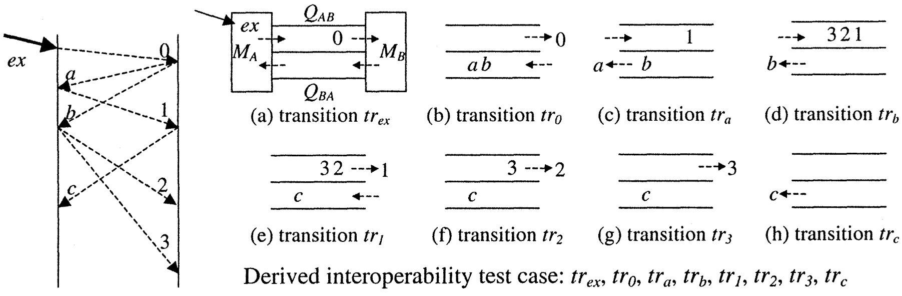


Figure 5. Derivation of interoperability test case for multiple internal outputs.

4.4 Multiple stimuli from the environment

To support multiple messages from the environment, an additional step is needed. Before the algorithm picks up an element from a channel, it examines whether the current local state of the machine is semi-controllable or not. If so, a new path is generated for all possible external inputs in this semi-controllable state. At this point, information of the current global state (which is a transient state) and all queues will be copied and the algorithm will generate test cases for each path recursively.

The procedure of generating interoperability test cases explained so far can be formalized into the algorithm given below. Lines 1 to 10 describe the main (control) part of the algorithm. This part initializes variables and calls the *interaction_sequences*() function for every stable state in the **NEW** set that has been initialized with the initial stable state (S_{A0}, S_{B0}). The function described in the rest of the algorithm generates all possible transitions based on the multiple stimuli principle. From lines 14 to 24, it checks the termination condition by examining the queue contents and updating the **NEW**, **IOPTS**, and $\mathbf{Tr}_\Pi$ sets, or calling itself recursively. From lines 25 to 37, it examines all possible transitions whose inputs are either external inputs or the current message from the queue, and then calls itself recursively with appropriate variables for this test case. Finally, the rest of the function checks whether or not the current interoperability test case has encountered a specification error, namely, there are no transition for the current message (so-called *implicit signal consumption)*. This algorithm is implemented in the C language with about 1000 lines of codes. The execution time of the program is negligible (80 milliseconds for generating 1012 test cases from 150 transitions).

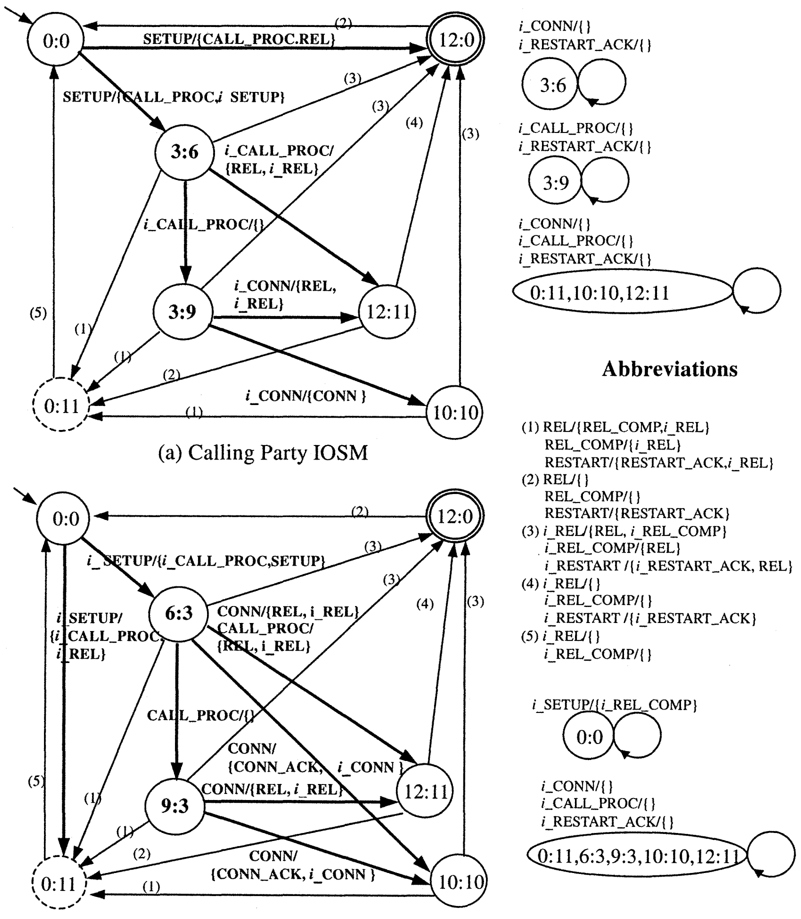
```
$L_{\Pi,in} := L^{A}_{in,E} \cup L^{B}_{in,E}$; $S_{\Pi 0} := (S_{A0}, S_{B0})$; $S_{\Pi} := \{ S_{\Pi,0} \}$; $Tr_{\Pi} := \{\}$; IOPTS:={}; NEW:=$S_{\Pi}$
while NEW≠∅ do begin
    set gs to any global state in NEW
    delete the global state from NEW
    for each possible external input symbol v in global state gs
        prepare two empty queues $Q_A$ and $Q_B$, and an empty transition list iopt
        push v to corresponding queue, $Q_A$ or $Q_B$
        interaction_sequences( $Q_A$ or $Q_B$, $Q_B$ or $Q_A$, gs, iopt, NEW )
    end for
end while

procedure interaction_sequences( var Q1, var Q2, gs, var iopt, var NEW )
begin
    if Q1 is empty then begin
        if Q2 is empty then begin
            if iopt has more than one transition then
                insert this new interoperability test case iopt to IOPTS
            if iopt do not end with an error code then
                insert this new global state gs to NEW
            insert iopt to a set of global transitions $Tr_{\Pi}$
        end
        else interaction_sequences( Q2, Q1, gs, iopt, NEW )
        return
    end
    else begin
        in := pop(Q1);  atLeastOne := false
        for each possible transition tr from the current global state gs
            change gs according to the input in the transition, tr.v
            copy Q1, Q2, and iopt to newQ1, newQ2, and newiopt, respectively
            if the input tr.v (≠ in) is external then
                insert current input in to the back of newQ1
            if the transition tr has internal outputs then
                push all the outputs to newQ2
            append the transition tr to newiopt
            interaction_sequences( newQ1, newQ1, gs, newiopt, NEW )
            atLeastOne := true
        end for
        if atLeastOne := false then begin
            report there is a specification error
            append an error code to iopt to stop expanding the current path
            interaction_sequences(Q1, Q2, gs, iopt, NEW )
        end
    end
end procedure
```

Algorithm 1. Interoperability test suite derivation

5. APPLICATIONS

In this section, we apply our method to the ATM/B-ISDN signaling protocols and TCP. Figures 6 and 7 show the IOSMs of these protocols (for more details, refer to papers [7, 8]) respectively.



Abbreviations

(1) REL/{REL_COMP,i_REL}
REL_COMP/{i_REL}
RESTART/{RESTART_ACK,i_REL}
(2) REL/{}
REL_COMP/{}
RESTART/{RESTART_ACK}
(3) i_REL/{REL, i_REL_COMP}
i_REL_COMP/{REL}
i_RESTART /{i_RESTART_ACK, REL}
(4) i_REL/{}
i_REL_COMP/{}
i_RESTART /{i_RESTART_ACK}
(5) i_REL/{}
i_REL_COMP/{}

Note 1) The Input alphabet and the output alphabet of the Calling Party IOSM and the Called Party IOSM are the same, i.e. {SETUP, CALL_PROC, CONN, CONN_ACK, REL, REL_COMP, RESTART, RESTART_ACK, i_SETUP, i_CALL_PROC, i_CONN, i_REL_COMP, i_RESTART, i_ RESTART_ACK}.

Note 2) PNNI specifies three alternative ways to handle unexpected messages: i.e., ignore the received message, send STATUS and clear the call. Except in 0:0 and 12:0 above, it is assumed (for simplicity) that unexpected messages are ignored.

Note 3) Dot ('.') notation denotes that messages are set one after another successively. For instance, {i_CALL_PROC.i_REL} means that i_CALL_PROC is sent followed by i_REL.

Figure 6. The IOSM of ATM signaling protocol.

Given the IOSMs, the input data for our method can be derived systematically. An IOSM is described in a file, in which each line represents a transition in the IOSM. For instance, "[1]0:0 3:6 SETUPb CALL_PROC *i*_SETUP" is the transition whose starting state is 0:0, destination state is 3:6, input message is SETUPb, and output messages are CALL_PROC and *i*_SETUP. For convenience of reference, index numbers are given to the original data file. The postfix is used to specify specific IOSM, e.g., SETUPb is an external message being sent to IOSM M_B. The prefix '*i*_' is used to denote internal messages.

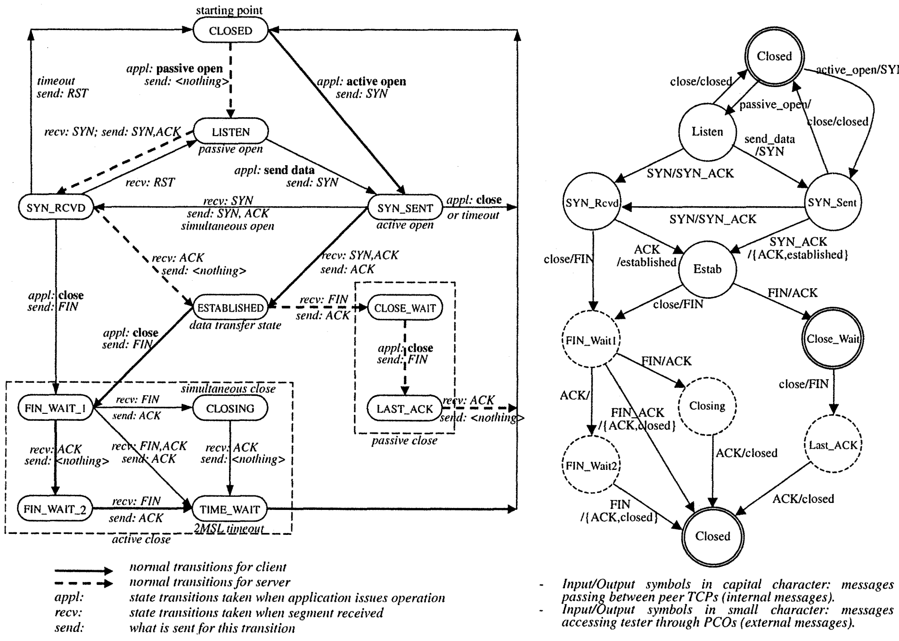
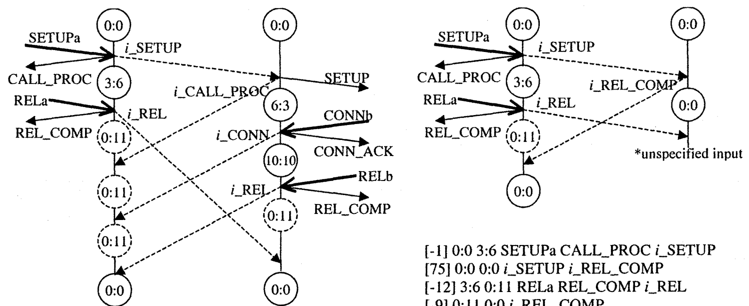


Figure 7. The TCP IOSM and its simplified one.

The results of applying our method to the two protocols are summarized in Table 1. In the case of ATM signaling protocol, a total of 1012 test cases were derived. 16 cases among them are only for one IUT and thus they are for conformance testing since there is no interaction between IUTs. Another 6 cases address specification errors for requesting the IUTs to receive unspecified input messages. Note that our method can also be used in detecting specification errors in the design phase.

The number of interoperability test cases based on the single and multiple stimulus principle are 54 and 990 respectively. Without taking into account of multiple stimuli, we have derived exactly the same 54 test cases reported in a previous work [8].

Figure 8 shows two examples in the derived interoperability test cases. An interoperability test case consists of several transitions in given IOSMs. Each transition has a unique *id*. Minus sign is used to denote the transitions of one IUT and plus sign to denote the transitions of the other one. Illustrated in Figure 8(a), the interoperability test case, <-1, 4, -12, -61, 5, 16, 8, -60, -8>, involves multiple stimuli. This test case is based on the specification and represents one of the correct behaviors of the protocol. Depicted in Figure 8(b), the interoperability test case, <-1, 75, -12, -9, -999.>, reveals specification error.



[-1] 0:0 3:6 SETUPa CALL_PROC *i*_SETUP
[4] 0:0 6:3 *i*_SETUP *i*_CALL_PROC SETUP
[-12] 3:6 0:11 RELa REL_COMP *i*_REL
[-61] 0:11 0:11 *i*_CALL_PROC
[5] 6:3 10:10 CONNb CONN_ACK *i*_CONN
[16] 10:10 0:11 RELb REL_COMP *i*_REL
[8] 0:11 0:0 *i*_REL
[-60] 0:11 0:11 *i*_CONN
[-8] 0:11 0:0 *i*_REL

(a)

[-1] 0:0 3:6 SETUPa CALL_PROC *i*_SETUP
[75] 0:0 0:0 *i*_SETUP *i*_REL_COMP
[-12] 3:6 0:11 RELa REL_COMP *i*_REL
[-9] 0:11 0:0 *i*_REL_COMP
[-999] specification error: unspecified input

(b)

Figure 8. Examples of interoperability test cases.

Table 1 summarizes the results of our applications to the TCP and ATM signaling protocols. The test cases are classified into test cases for checking specification errors, conformance test cases, and interoperability test cases. It is shown that our method generates more test cases than existing methods [7, 8] in order to support multiple stimuli. For example, in the case of the ATM signaling protocol, we have derived 990 interoperability test cases while 54 were derived by the method [7] that assumes the single stimulus principle. Therefore, it is clear the existing methods cannot detect faults due to multiple stimuli.

Table 1. Test case generation for the IOSMs of TCP's connection establishment phase and ATM signaling protocol

TCP's connection establishment phase (Each IOSM consists of 19 transitions)		
used principle	single stimulus principle	multiple stimuli principle
test cases	20	74
test cases due to specification error	4	54
conformance test cases	8	8
interoperability test cases	8	12

ATM signaling protocol (Each IOSM consists of 75 transitions)		
used principle	single stimulus principle	multiple stimuli principle
test cases	70	1012
test cases due to specification error	0	6
conformance test cases	16	16
interoperability test cases	54	990

Table 2 gives the transition coverage for the IOSMs of TCP's connection establishment phase and the ATM signaling protocol. It is shown that our method, which is based on the multiple stimuli principle, has higher transition coverage than the existing methods, which are based on the single stimulus principle. Our method traverses 5 more transitions of TCP and 9 more transitions of ATM than the existing methods in [7, 8]. Our coverage is exactly the same as that covered by validation[4] techniques. A SDL commercial tool [13] was used to obtain the result on validation reported in Table 2. In other words, our method can generate interoperability test suite whose transition coverage is equivalent to that of validation.

Table 2. Transition coverage

methods	the existing methods [7, 8] (single stimulus principle)		our method (multiple stimuli principle)		validation	
protocols	TCP	ATM	TCP	ATM	TCP	ATM
traversed tr.	13 (68%)	39 (52%)	18 (95%)	48 (64%)	18	48
untraversed tr.	6 (32%)	36 (48%)	1 (5%)	27 (36%)	1	27

Table 3. Untraversed transitions

	the existing methods [7, 8]		our method	
protocols (untraversed tr.)	TCP (6)	ATM (36)	TCP (1)	ATM (27)
wrongly specified	1 (5%)	13 (17%)	1	13
over-specified	0 (0%)	14 (19%)	0	14
limitation of the method	5 (26%)	9 (12%)	0	0

[4] Validation is based on the state space exploration technique. It automatically checks a distributed system's design for errors and makes it possible to prove a system will work before implementation. Validation executes all possible combinations of events that can happen, and reports any indication that something has gone wrong.

Table 3 shows the reasons why some transitions are not traversed. First, some transitions are wrongly specified and will never be executed due to absence of transitions that send the corresponding signals. For instance, some transitions in Figure 6 need an input signal i_RESTART, but there is no transition that sends the required signal. Note that a wrongly specified transition may give rise to subsequent untraversed transitions. Also, some transitions are over-specified. Over-specified transitions are superfluous and are unnecessary for our purpose. They include self-transitions whose start state and final state are the same. Actually they are sometimes used for error handling. Since checking correct behavior is the objective of our method, such transitions are considered as over-specified in this paper. Finally, some transitions are not traversed due to the limitation of the method used. As shown in the table, our method can traverse all the transitions that cannot be handled by the existing methods.

6. CONCLUSION AND FUTURE WORK

We have proposed a new method for generating interoperability test suites that support multiple inputs to the system under test. This means the method adopts the multiple stimuli principle rather than the traditional single stimulus principle used in existing methods. To do this, we have developed appropriate formal models on which the interoperability test suite derivation algorithm is based. We have applied our method to the ATM signaling protocol and also to a part of TCP. Experimental results have shown that our method has higher transition coverage than the existing methods: 26% higher for TCP and 12% higher for ATM. Our transition coverage is equivalent to that managed by the validation technique. As future work, the derived test cases will be applied to production protocols for interoperability testing in a real-life environment.

REFERENCES

[1] ISO/IEC 9646-1, "Information Technology – OSI Conformance Testing Methodology and Framework. Part 1: General Concepts", 1994.

[2] Rafiq, O. and Castanet, R., "From Conformance Testing to Interoperability testing", Proceedings of the 3rd International Workshop on Protocol Test System, 1990.

[3] Arakawa, N., Phalippou, M., Risser, N. and Soneoka, T., "Combination of conformance and interoperability testing", Formal Description Techniques, V (C-10) M. Diaz and R. Groz (Eds.), Elsevier Science Publishers, 1993.

[4] Shin, J. and Kang, S., "Interoperability Test Suite Derivation for the ATM/B-ISDN Signaling Protocol", Testing of Communicating Systems, Vol 11, Kluwer Academic Publishers, pp. 313-330, 1998.
[5] Vermeer, G. S. and Blik, H., "Interoperability Testing: Basis for the Acceptance of Communicating Systems", Protocol Test System, VI, Elsevier Science Publishers, 1994.
[6] Kang, S. and Kim, M., "Interoperability Test Suite Derivation for Symmetric Communication Protocols", IFIP Joint International Conference on Formal Description Techniques (FORTE X) and Protocol Specification Testing and Verification (PSTV XVII), pp. 57-72, November 1997.
[7] Seol, S., Kim, M., Kang, S. and Park, Y., "Interoperability Test Suite Derivation for the TCP protocol", IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification Testing and Verification (PSTV XIX), 1999.
[8] Kang, S., Shin, J. and Kim, M., "Interoperability Test Suite Derivation for Communication Protocols", Computer Networks, 2000.
[9] Arakawa, N. and Soneoka, T., "A Test Case Generation Method for Concurrent Programs", Protocol Test Systems, IV, Elsevier Science Publishers, 1992.
[10] Luo, G., Bochmann G., and Petrenko, A., "Test Selection Based on communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method", IEEE Transactions on S.E., Vol 20, No. 2, pp. 149-162, February 1994.
[11] Lima Jr., L. Paula and Cavalli, A., A Pragmatic Approach to Generating Test Sequences for Embedded Systems, IWTCS, 1997.
[12] ITU-T X.290 Series, Conformance Testing Methodology and Framework, 1994.
[13] "Telelogic SDT 3.2 Manuals", Telelogic, September 1997.
[14] The Tcpdump Group, "libpcap", available at http://www.tcpdump.org.
[15] Zafiropulo P., West C.H., Rudin H., Cowan D.D., and Brand D., "Towards Analyzing and Synthesizing Protocols (Computer Networks)", IEEE Transactions on Communications, vol.28, no.4, pp.651-661, April 1980.