4

Interoperability Test Suite Derivation for Symmetric Communication Protocols

Sungwon Kang and Myungchul Kim
Korea Telecom Research & Development Group
Sochogu Umyundong 17, Seoul 137-792, Korea
e-mail: kangsw@sava.kotel.co.kr, mckim@sava.kotel.co.kr

Abstract

Communication protocols are commonly designed in such a way that implementations of the same protocol can be used as peers for communication. Such a protocol is said to be symmetric. When two or more entities are employed to perform a certain task as in the case of communication protocols, the capability to do so is called interoperability and considered as the essential aspect of correctness of communicating systems. This paper deals with the problem of deriving interoperability test suite for control part of symmetric protocols. A new approach to efficient interoperability testing is described with justifications and the method of interoperability test suite derivation is shown with the example of the ATM Signaling protocol.

Keywords

Interoperability testing, test suite generation, symmetric protocol

1. INTRODUCTION

When more than one objects are employed to perform a certain function, there arises the problem of whether they together behave correctly. This is the problem of

Formal Description Techniques and Protocol Specification, Testing and Verification
T. Mizuno, N. Shiratori, T. Higashino & A. Togashi (Eds.)

interoperation in a general sense. Here (1) involvement of more than one objects and (2) the objects together behaving as expected are the two key characteristics of (correct) interoperation. Thus it can be said that two or more objects *interoperate* if they together behave as expected. For interoperability of communication protocols, such expectations are documented in specifications. Usually the expectations about interoperation are not described in a single dedicated document. Rather they should be inferred or derived from the relevant specifications.

Within the context of communication networks, an object mentioned above can be a network node, a layer of a node or a component of a layer or a plane or even a network, i.e. anything we decide to view as a whole. These notions of interoperation and object allow us to view various kinds of interacting behavior within a network and between networks as special kinds of interoperation. Thus *internetworking* can be seen as interoperation of objects which are networks. *Interworking* of two nodes or two networks utilizing an interworking function unit becomes interoperation of the three objects, i.e. two nodes or networks together with the interworking function unit in between.

An abstract view of communication can be conceived when protocols of network nodes are layered and underlying layers are regarded as the service provider for the layer above. Then by similarly abstracting from the underlying layers, we can focus on the behavior of a certain layer and think about interoperation of the objects which realize the particular layer under consideration. In this way, the above definition of interoperation remains valid even when we take an abstract view of network nodes and networks.

Once the notion of interoperation is clearly understood, there arises the problem of verifying interoperation for target implementations which interact with each other. This is the task of interoperability testing. By virtue of the two characteristics of interoperation noted at the beginning of this section, interoperability testing differs from conformance testing. Because of the first characteristic, more varied test architectures are possible than with conformance testing. Because of the second characteristic, test suite derivation become a more challenging task for which the expected behavior need be inferred first.

1.1 Related Work

In the past, research on protocol testing mainly concentrated on conformance test sequence generation (cf. [Chow 78] [Sidh 90] and the bibliographies therein). Accordingly, it didn't take long before the international standard for the methodology and framework for conformance testing has come into existence [ISO/IEC 9646]. Although conformance testing is regarded as a necessary step on the way to achieving interoperation, it is agreed that it is insufficient to ensure interoperation of communication network entities. Some sort of direct testing of interoperation is considered indispensable. Work on interoperability testing can be classified into two categories depending on whether it is more geared to practical things such as clarification and implications of interoperability testing or to systematic generation of interoperability test suite. As work along the former line are [Bonn 90] [GRSS 90] [Cast 91] [VerB 94]. [Bonn 90] [VerB 94] present

interoperability testing experiences. [GRSS 90] gives a comprehensive discussion on various aspects related to interoperability testing.

For the latter line of work, there are [RafC 90] [AraS 92] [APRS 93] [CasK 94] [LuBP 94] [KanK 95]. All these base interoperability test suite derivation on some sort of reachability analysis. For interoperability test architecture, [RafC 90] uses upper testers as well as lower testers. [AraS 92] [LuBP 94] introduces notions of stable state to reduce the size of relevant state space. [CasK 94] develops interoperability test suite method for synchronous models. [KanK 95] shows how to derive test suites for dynamic testing of interoperability.

The previous work, however, did not provide a coherent framework for interoperability testing in that the notions of interoperability, interoperability testing, interoperability test case and interoperability test architecture were not presented in an integrated manner nor were interrelated for the purpose of interoperability test suite development. In particular, there is no work specially treating symmetric protocols. So the consequences and possible optimizations for interoperability test derivation that may arise from a communication protocol being symmetric remained unexplored.

This paper, which belong to the second line of work on interoperability testing, addresses these issues. Starting from the general definition of interoperability which we already gave in the previous section, we carefully select an interoperability test architecture. The chosen architecture, combined with natural assumptions and inherent limitations for testing, is shown to induce a notion of interoperability test case. And this notion of interoperability test case allows us to focus on genuine interoperability aspect and at the same time to derive interoperability test suite in a cost-effective manner. It is shown that the test suite derivation and the actual testing itself can be made very efficient in particular when the protocol under consideration is symmetric. The remainder of this paper describes in detail this approach to efficient interoperability testing and the method of interoperability test suite derivation with the example of the ATM Signaling protocol.

1.2 Our Approach

We consider interoperability testing of two interacting implementations as the most basic type of interoperability testing. When more than two objects are involved, interoperability testing requires a more complicated test architecture. Also the test derivation, the test execution and the test result analysis become more complicated.

In this paper, we restrict our attention to control part of symmetric communication protocols. A communication protocol is said to be *symmetric* if it is designed in such a way that the implementations of the same protocol can be used as communication peers. In order to be symmetric, a protocol should be such that its peers have exactly the same functional features. Peers for a symmetric protocol need not be placed adjacent. Commonly protocols are symmetric or near symmetric (with asymmetry resulting only from differences in some features). Examples of asymmetric protocol are master-slave protocol and client-server protocol.

In our approach to interoperability test suite derivation, it is assumed that (1) the communication protocol is structured as a finite state machine (FSM) and that (2) a

complete set of conformance abstract test cases has been already developed (elsewhere) based on the FSM structure. Because of this second assumption, our approach will yield an interoperability test suite which has no overlapping with conformance test suites as we will see later. Furthermore, (3) we adopt the test architecture in Figure 1. We call an individual object involved in interoperation an Implementation Under Test (IUT). In spite of the term IUT, it is important to note that the target of testing here is not the individual objects (which are the targets of conformance testing) but the system as a whole which consists of those objects. Still, IUT is a convenient term and will be used throughout the paper.

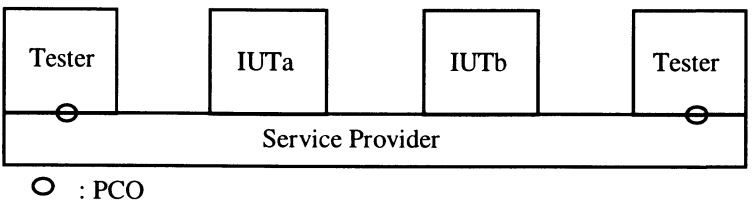

Figure 1. Test architecture for interoperability testing.

Note that there are only two Points of Control and Observation (PCO's) in Figure 1. Often in practice a monitoring point or Point of Observation (PO) is set between IUT's. We do not set such PO. For with such PO, (1) it would be more costly to generate test suite, (2) it would be more costly to perform testing and (3) interoperability testing would have much overlapping with conformance testing.

As with conformance testing, interoperability testing is restricted by observability and controllability that is allowed in a chosen test architecture. In addition, it is usually assumed that the slowness of the environment [LuBP 94] places practical limit on observability and controllability. That is, transient states cannot be observed or controlled in a predictable way and hence are considered useless for the purpose of testing. This justifies basing any notion of test case on stable states as we do in this paper.

The interoperability test approach of this paper is to *test interoperability of implementations against interoperability of specifications*. In this approach, interoperability testing is a (yet another) conformance testing, i.e. conformance testing of the system which is composed of the actual subsystem implementations. Any notion of interoperation, if any, should be expressed in principle in a given specification or should be agreed upon in advance by specification writers and implementers. Ascertaining correctness of specifications in this respect is the task of verification and validation, which goes beyond the proper scope of testing. For the target of testing is the relation between specification and implementation rather than between specifications. However, if specific interoperability requirements are subject to validation and verification at the specifications level, then the same validation and verification method can be applied to a system of implementations for testing.

As the result of interoperability testing of implementations, problems residing in specifications may be found and reported. Then the additional ingredient

interoperability testing provides in addition to verification of interoperability of specifications is to fill the gap left by usual conformance testing. The real work of interoperability assurance should come at the stage of specification development. But still the fine points that may have been missed at the time of specification writing (including validation and prototyping) can be augmented through testing.

2. COMMUNICATING SYSTEMS OF IOSM'S

In this paper, we take the view that specifications and implementations of communicating entities can be modeled as some sorts of FSM's. This makes possible rigorous discussion on conformance and interoperability issues which would provide the basis for automating test suite derivation process. In this section, we make precise the FSM model to be called IOSM and its communication behavior.

Definition 1 An IOSM is a 5-tuple <St, s_0, L_{in}, L_{out}, Tr> where:
(1) St = $\{s_0, ..., s_{n-1}\}$ is a set of states,
(2) $L_{in} = \{v_1, ..., v_m\}$ is a set of input symbols,
(3) $L_{out} = \{u_1, ..., u_p\}$ is a set of output symbols,
(4) $Tr \subseteq \{s\text{–}v/U \rightarrow s' \mid s,s' \in St \wedge v \in L_{in} \wedge U \in P(L_{out})\}$ and
(5) $s_0 \in St$ is the initial state.

In the definition, $P(X)$ denotes the power set of the set X. L_{in} and L_{out} are respectively called *input alphabet* and *output alphabet*. L_{in} and L_{out} can be further subdivided into two classes: one prefixed with '*i_*' and the other without the prefix. '*i_*' is used to indicate internal messages as opposed to external messages. So for example when two IOSM's M_1 and M_2 are communicating, the messages between M_1 and M_2 are internal messages and are prefixed with '*i_*' but messages between M_1 (or M_2) and their environment are external messages and are not prefixed. Tr is a set of transitions. 'v/U' is called a *label*. The message before '/' is the received message and the set of messages after '/' are messages sent. The set notation is used because upon receiving a message IOSM can send zero or more messages. In the set Tr of transitions of a *deterministic* IOSM, for any state there is only one transition with the same input symbol. Note that Definition 1 does not restrict IOSM to be deterministic.

Definition 2 Let M be an IOSM. Let $v, v_i \in L_{in}$, $u, u_i \in L_{out}$, $\mathbf{v} \in L_{in}^*$ and $s, s' \in St$. Then:
(1) $\sigma(s,v) = \{(s',v/u) \mid (s\text{–}v/U \rightarrow s') \in Tr \wedge u \in U\}$
(2) $\sigma(s,\mathbf{v}) = \{(s', v_1/u_1 ... v_k/u_k) \mid \mathbf{v} = v_1 ... v_k \wedge$
$(s\text{–}v_1/U_1 \rightarrow s_1) \in Tr \wedge ... \wedge (s_{k-1}\text{–}v_k/U_k \rightarrow s') \in Tr \wedge u_1 \in U_1 \wedge ... \wedge u_k \in U_k\}$
(3) $M(\mathbf{v}) = \sigma(s_0, \mathbf{v})$

A member of $M(\mathbf{v})$ is a sequence of labels and is called a *run* (or *trace*) of M for the input sequence **v**. Next we define communicating system of IOSM's.

Definition 3 A (communicating) system Σ of n IOSM's is $<\{(M_i,Q_i) \mid 1 \le i \le n\}, L_{\Sigma,in}, L_{\Sigma,out}, s_{\Sigma,0}>$ where:
(1) M_i, $1 \le i \le n$, is a IOSM $<St_i, s_{i,0}, L_{i,in}, L_{i,out}, Tr_i>$ as in Definition 1.

(2) Q_i, $1 \leq i \leq n$, is an input queue for M_i.
(3) $L_{\Sigma,in}$ is a set of external input symbols.
(4) $L_{\Sigma,out}$ is a set of external output symbols.
(5) The initial state of the system is $s_{\Sigma,0} = <(s_{1,0},Q_0), ...,(s_{n,0},Q_n)>$ where Q_i is empty and $s_{i,0}$ is the initial state of M_i, $1 \leq i \leq n$.

In this model, there is one explicitly defined input queue for each IOSM and implicit bi-directional communication channels between each pair of IOSM's. In the next definition, we give a precise description of the global behavior of a communicating system of IOSM's. It is assumed that each message contains enough information to identify the receiving IOSM.

Definition 4 (Communicating system of IOSM's) Let Σ be as in Definition 3. Let $v \in L_{\Sigma,in}$, $U \in P(L_{\Sigma,out})$, $\mathbf{v} \in L_{\Sigma,in}^*$, $\mathbf{u},\mathbf{u_1},\mathbf{u_2} \in L_{\Sigma,out}^*$, s, s′ be states of Σ. Then

(1) $\sigma(s,v) = \{(s',v/\mathbf{u_1}.\mathbf{u_2}) \mid s=<(s_{1,j1},\varepsilon),...,(s_{i,ji},\varepsilon),...,(s_{n,jn},\varepsilon)> \wedge$
$\exists 1 \leq i \leq k: (s_{i,j} - v/U \rightarrow s_{i,j'}) \in Tr_i \wedge \mathbf{u_1} \in \varphi(U) \wedge$
$(s',\mathbf{u_2}) \in \sigma'(\mu(s,\{(s_{i,j'},\varepsilon)/(s_{i,j},\varepsilon)\} \cup \{(s_{i,j},Q_i)/(s_{i,j},i\_w.Q_i) \mid i\_w \in U\}))\}$

where $\sigma'(s) = \{(s',\mathbf{u_1}.\mathbf{u_2}) \mid \mathbf{u_1} \in \varphi(U) \wedge (s',\mathbf{u_2}) \in \sigma'(\mu(s,\{(s_{k,j},Q_i)/(s_{k,j},i\_w.Q_k) \mid i\_w \in U\}))\}$
if $\exists 1 \leq i \leq n$: $s = <...,(s_{i,j},Q_i.i\_w),...> \wedge (s_{i,j} - i\_w/U \rightarrow s_{i,j'}) \in Tr_i$
$\sigma'(s) = \{(s,\varepsilon)\}$ otherwise
where $\mu(s,\varnothing) = s$
$\mu(<...,(s,Q),...>, P \cup \{(s',Q')/(s,Q)\}) = \mu(<...,(s',Q'),...>,P)$
$\varphi(\varnothing) = \{\varepsilon\}$
$\varphi(\{i\_w\} \cup U) = \varphi(U)$
$\varphi(\{u\} \cup U) = \{u.\mathbf{u} \mid \mathbf{u} \in \varphi(U)\}$

(2) $\sigma(s,vv) = \{(s'',v_1/u_1...v_k/u_k v/u) \mid$
$\mathbf{v} = v_1...v_k \wedge (s',v_1/u_1..v_k/u_k) \in \sigma(s,\mathbf{v}) \wedge (s'',v/u) \in \sigma(s',v)\}$
$\sigma(s,\boldsymbol{\varepsilon}) = \{(s,\boldsymbol{\varepsilon})\}$

(3) $\Sigma(\mathbf{v}) = \sigma(s_{\Sigma,0},\mathbf{v})$

A system state in which input queues are all empty is called a *stable state* [LuBP 94]. The communicating system as defined above takes a sequence of inputs from the environment one by one. The next input from the environment is processed only when the system is in a stable state. This definition of stable state makes it unnecessary in later sections to describe input queues to show (stable) states of a communicating system. In (1), σ with a single external input defines a set of new states reached by processing of a single input from the environment followed by sequences of messages (including sequences of internal state changes) sent by the receiving IOSM. The reception of a message and sending messages as its response constitutes an atomic action. Thus inputs from the environment or IOSM's is instantly placed into the input queues of the receiving IOSM's. μ is a function that updates states. φ takes a set of messages and returns the set of all possible orderings of external messages excluding internal messages. (2) extends σ to a sequence of external inputs. (1)-(3) for communicating systems are similar to (1)-(3) for IOSM's of Definition 2. In a communicating system of IOSM's, *deadlock* is said to occur when any combination of (external and internal) inputs cannot change the

system state. *Livelock* is said to occur when system state changes forever without reaching a stable state. Note that Definition 4 can also be regarded as defining a composition of n IOSM's, denoted $M_1 \times M_2 \times ... \times M_n$, for which

(1) the states are stable states of Σ

(2) between its states s and s′ there is a transition with the label 'v/**u**' if and only if $(s', \mathbf{u}) \in \sigma(s, v)$

(3) its initial state is the same as the initial state of Σ.

Therefore we will use Σ and $M_1 \times M_2 \times ... \times M_n$ in an interchangeable way.

In conformance testing, conformance test suite is developed in such a way that for the given specification with FSM structure, absence of operation errors (or correctness of input/output behavior) and absence of transfer errors (or correctness of the state reached after the transition) are examined with respect to each transition of the FSM [Chow 78]. For this purpose, a conformance test case consists of three parts: preamble, test body and postamble. *Test body* is the part of a conformance test case that checks operation errors and transfer errors. Test body ends by reaching a stable state. New test cases are needed to examine the behavior of the machine M at the stable state reached after the transition. *Preamble* is the part that takes M to the stable state at which the test body can be started. *Postamble*, which is often empty, is the part that takes M to a stable state. Methods to generate such a conformance test suite are well-known [Chow 78][Sidh 90]. Let *TS*(M) be the set of all such conformance test cases, *conformance test suite* in short, for M. We say that a conformance test suite is *complete* if each and every transition in the specification modeled as IOSM has a corresponding conformance test case as defined above.

3. INTEROPERABILITY TEST SUITE DERIVATION

To illustrate our method we use as an example the ATM Signaling Protocol for the ATM switch. The behavior of the protocol at the user-network interface is specified in [ITU-T Q.2931] and also in ATM Forum UNI specification [AF UNI]. The two specifications have much in common. The behavior of the protocol at the network-network interface differs from that of user-network side. Among ATM Forum specifications, PNNI specification [AF PNNI] is the one that gives the most complete description of protocol behavior at the network-network interface. We use ATM Forum UNI 3.1 and PNNI specifications as the definitive specifications of the interface behavior of the ATM switch which together constitute the ATM signaling protocol for the ATM switch (to be subsequently abbreviated as Signaling Protocol).

3.1 Pruning Transitions

The first step is to derive an IOSM, say *S*, from the relevant informal specifications. Assuming that we already derived *S*, let *pr*(*S*) be the IOSM which is the same as *S* except that *pr*(*S*) does not contain the transitions in *S* which (1) do not change state and (2) do not contain '*i*_' prefixed messages. For example, upon receiving

STATUS_ENQUIRY message at any state, the signaling protocol entity should send STATUS message reporting its current state [AF UNI][AF PNNI]. Such transitions would not appear in $pr(S)$. In this way, we come up with an IOSM depicted in Figure 2 from Signaling Protocol.

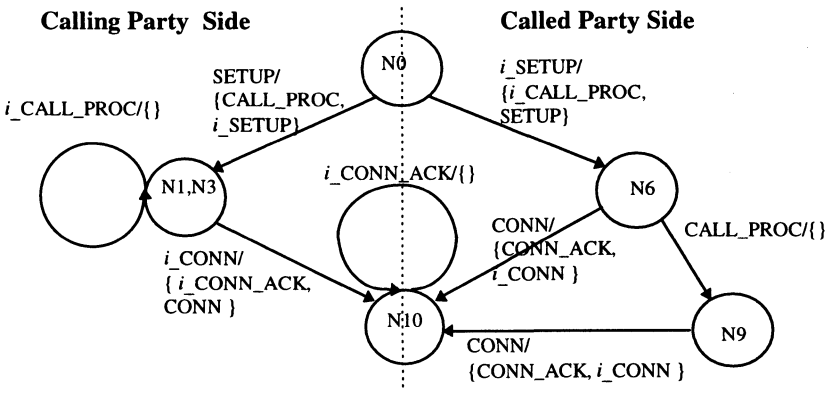

Figure 2. Call establishment part of the pruned IOSM for Signaling Protocol.

Conceptually an IOSM is one connected graph. With this example, the picture in one graph would be severely cluttered due to its complexity. Therefore we present the graph partitioned with respect to functionalities. Figure 2 describes state transitions for call establishment. At any moment, signaling protocol entity functions either as a calling party or as a called party (but not both). In the figure, it is made clear with the dotted line in the middle. There are two kinds of messages (or PDU's): one appearing at the UNI interface and the other appearing at the NNI interface. In Figure 2, NNI messages are prefixed with '*i_*'. The reason for the prefix is the interactions occurring at NNI are internal events between two ATM switches. As explained with Figure 1 in Section 1.2, we take the view that, by not monitoring NNI, interactions at NNI do not directly affect test results.

When combined with the conformance test suites for S_a and S_b, the interoperability test suite for $pr(S_a)\times pr(S_b)$ covers all the test cases for $S_a\times S_b$. The following theorem states this completeness formally.

Theorem 1 Let S_a, S_b be two IOSM's. Then

$$TS(S_a\times S_b) \subseteq \{TS(pr(S_a)\times pr(S_b)) \cup TS(S_a) \cup TS(S_b)\}$$

Note that this step of optimization is applicable regardless of whether the protocol is symmetric or not. For symmetric protocols, the behavior of the global system can be described as the composition of $pr(S_a)$ with itself, i.e. $pr(S_a)\times pr(S_a)$.

3.2 Interoperability Test Case

Figure 3 depicts a typical message interaction sequence for successful call establishment. The terminal equipment TEa initiates a call to TEb by sending

SETUP through the switch IUTa. TEb accepts the call by sending CONN through the switch IUTb. The shaded area represents a black-box whose internals we decide not to observe.

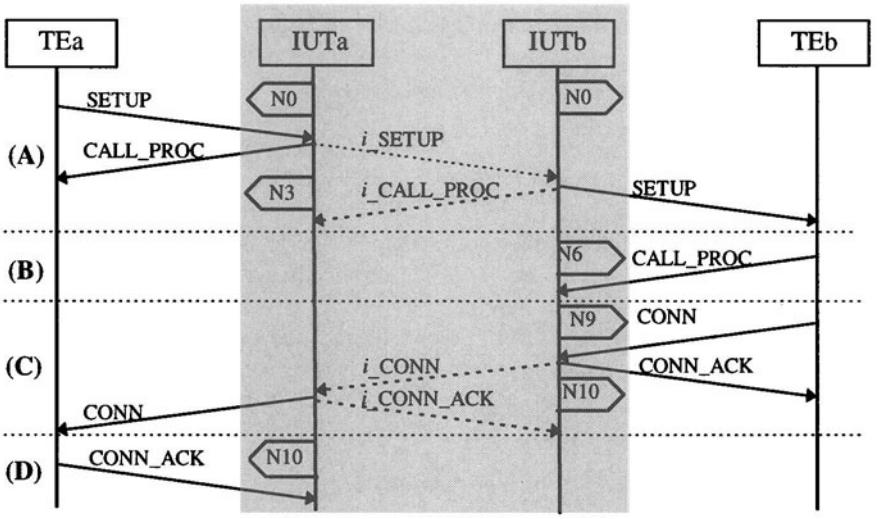

Figure 3. Message interaction sequence for call establishment.

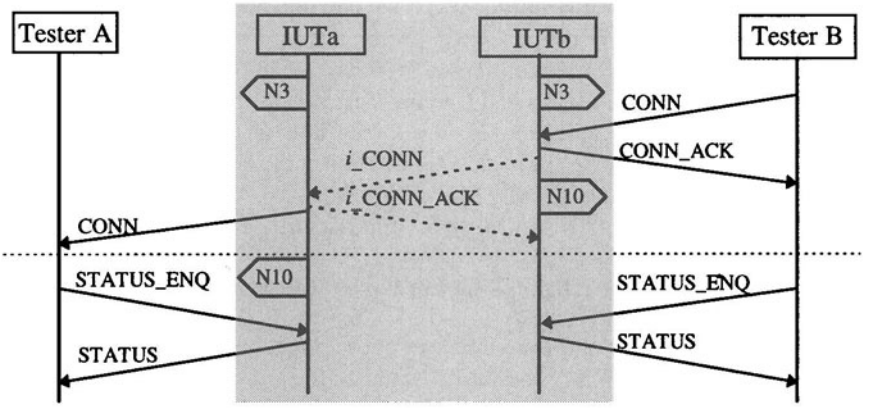

Figure 4. An interoperability test case for subsequence (C).

As with the conventional conformance testing, unstable states are not available for control purposes and can be utilized to reduce the size of state space of protocol entities logically built from specifications and at the same time the number of test cases. There are 4 *stable states*, (N0,N0), (N3,N6), (N3,N9) and (N10,N10) in the interaction sequence of Figure 3. Since we assume that test bodies are from a stable

state to a stable state, there are 4 subsequences which are candidates for testing as marked with (A), (B), (C) and (D). However, (B) and (D) are actually conformance test cases for IUTb and IUTa, respectively, since only one IUT is involved. (A) and (C) are genuine interoperability test cases. Figure 4 is one of the interoperability test cases derivable from the interaction sequence of Figure 3.

3.3 Optimizing Test Suite Derivation Process

In the previous sections, in the course of optimizing interoperability testing, we (1) pruned transitions of IOSM and (2) gave definition of interoperability test case in such a way that it is based on stable states and abstracts from internal interactions. In this section, we consider further optimizations. To do that, we need the notions of initiator and responder. An active tester which always initiates a test body is called *initiator*. A tester which may be active or passive but never initiates a test body is called *responder*. An active responder may send messages during test case execution. For example, it may initiate preamble part by sending a message.

In the test architecture of Figure 1, suppose that IUTa and IUTb are implementations of the same protocol specification. One of Tester A or Tester B must send a message to initiate a test body. If Tester B were to send a message to initiate a test body, then by making Tester A initiate the test body with the positions of Tester A and Tester B interchanged the mirror image of the same sequence of interactions would occur. This observation can be stated as follows:

Theorem 2 Given the test architecture of Figure 1, for interoperability testing of symmetric protocols, it is sufficient to have one initiator and one responder.

For example, the test execution shown in Figure 4 can be achieved by switching the positions of IUTa and IUTb and keeping Tester A as the initiator and Tester B as the responder. An implication of the observation is that the same test suite should be run twice, the second run after switching the positions of Initiator and Responder. However, the necessary test suite size is now reduced approximately to half. In general Theorem 2 is not true of asymmetric protocols.

3.4 An Efficient Algorithm to Generate Stable States

Not only can we reduce the test suite size approximately to half as shown in the previous section, but also it is possible to optimize the test generation process itself. Relying on Theorem 2, we show below an algorithm to generate all stable states which form the starting and the ending states of interoperability test cases. Actual test cases generation is realized by decorating the algorithm with the step to store the applicable sequence of events between stable states. In our approach to interoperability testing, the number of stable states is small enough that it can be used as a suitable basis for manual calculation as well as for automatic generation of interoperability test suites.

```
input:   IOSM's S1 and S2
output:  Stable[j] contains all the stable states of S1×S2
j :=0;
Stable[j]  := {(s1,0, s2,0)};        /* All stable states generated up to Stage j */
New        := {(s1,0, s2,0)};        /* Stable states to be expanded at Stage j+1 */
while New ≠ ∅ do begin
    <New, (s1, s2)> := delete-one-element(New);
    OldFrontier  := ∅;               /* Already expanded nodes at Stage j */
    NewFrontier := {(s1, s2)};       /* Nodes to be expanded at Stage j+1 */
    while  (NewFrontier ≠ ∅)  do begin
        <NewFrontier, (s1, s2)> := delete-one-element(NewFrontier);
        OldFrontier := OldFrontier ∪ {(s1, s2)};
        Event_Seq_Set := interaction-sequences((s1, s2));
        while  Event_Seq_Set ≠ ∅ do begin
            <Event_Seq_Set, Event_Seq> := delete-one-element(Event_Seq_Set);
            if Event_Seq ends with a stable state (s1,s2)
            then begin
                    if s'∉ (Stable[j] ∪ New ∪ OldFrontier)
                    then NewFrontier := NewFrontier ∪ {(s1,s2)};
                 end
            else begin
                    There is an error in the specifications.
                    Stop and fix the error.
                 end
        end while;
    end while;
    j := j+1;
    Stable[j] := Stable[j−1] ∪ OldFrontier;
    New := {(s2,s1) | (s1,s2)∈ Stable[j] ∧ (s2,s1)∉ Stable[j] };
end while;
```

Figure 5. An efficient algorithm to generate stable states.

In the algorithm, (s_1, s_2) denotes the global system state where s_1 and s_2 are stable states of S_1 and S_2, respectively. *delete-one-element(Set)* is a function that choose an arbitrary element from the set *Set* and removes it. The output of this function is a pair of which the first argument is the chosen element and the second argument is the resulting *Set* without the chosen element. *interaction-sequences*((s_1, s_2)) generates all possible sequences of interactions which are initiated by the IOSM which is in state s_1. Let $TS'(S_1, S_2)$ denote the test suite generated by the algorithm.

An application of the algorithm is shown in Figure 6. The global states in boxes are stable states and those not in boxes are transient states. Underline for instance in $\underline{s}$ indicates that the tester for the IUT with the underlined state is the initiator. After the first stage is over, the only node of which the full expansion has been postponed

is **(N3,N6)** marked in bold face. Therefore the 2nd stage would begin with the expansion of (N6,N3). In general, there can be more than one such nodes. Similarly, in the 2nd stage the only node which would be not fully expanded is **(N9,N3)**. Hence in the 3rd stage (N3,N9) should be expanded.

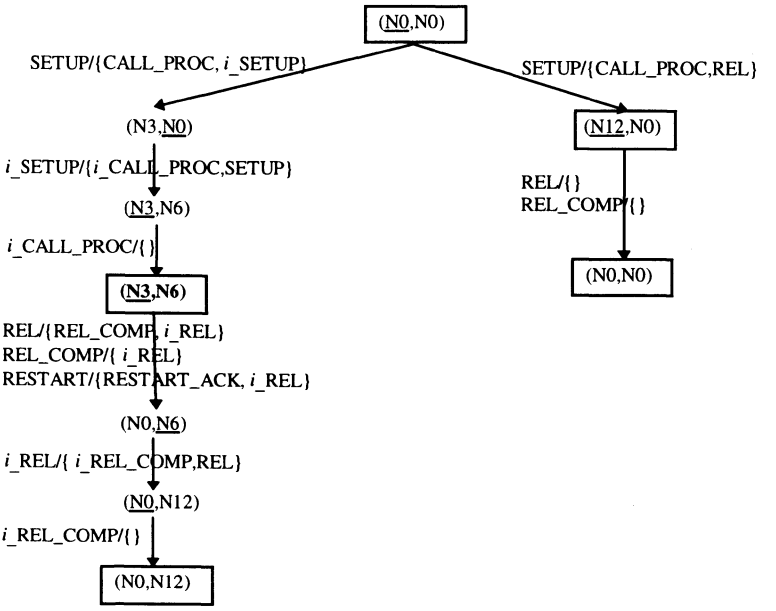

Figure 6. The 1st stage of test suite generation algorithm.

Let $tc_{S1,\,S2}$ be one of the test cases generated from the algorithm. Let $tc_{S2,\,S1}$ denote the test case with exactly the same structure as $tc_{S1,\,S2}$ except that in the former the roles of S_1 and S_2 are exchanged. For example, if $tc_{Sa,\,Sb}$ to be obtained from the 2nd stage is stated as:

The initial state is (N6,N3).

After IUTa receives CALL_PROC from LTa, IUTa does not respond.

The final state is (N9,N3).

then $tc_{Sb,\,Sa}$ is as follows:

The initial state is (N3,N6).

After IUTb receives CALL_PROC from LTb, IUTb does not respond.

The final state is (N3,N9).

It is the characteristic of the algorithm that it only generates one of these two test cases. But neither of the test cases can be dispensed with. The following states an obvious fact about the algorithm.

Lemma 3 Let S_1, S_2 be IOSM's and $tc_{S1,\,S2}$ and $tc_{S2,\,S1}$ be as defined above. Then $tc_{S1,\,S2} \in TS'(S_1, S_2)$ if and only if $tc_{S2,\,S1} \in TS'(S_2, S_1)$

Thus the algorithm can be said to be complete in the sense that if it is applied twice, the second time with the positions of S_1 and S_2 interchanged, all the test cases generated by the conventional method would be covered. The next theorem states this completeness.

Theorem 4 Let S_1, S_2 be IOSM's. Then

$$TS(S_1 \times S_2) \subseteq TS'(S_1, S_2) \cup TS'(S_2, S_1)$$

Symmetry of $TS'(S_1, S_2)$ and $TS'(S_2, S_1)$ implies that for actual application the effect of applying the two test suites would be achieved simply applying one of the suites twice, the second time with the positions of two IUT's interchanged.

When it is known that the two IUT's are identical, possible nonconforming behavior which would be detected with the positions of IUT's interchanged would be detected without interchanging their positions. Let $exec(tc, I_a \times I_b)$ denote the result of applying the test case tc to the system of IUT's $I_a \times I_b$. For test suite TS, define

$$exec(TS, I_a \times I_b) = \{exec(tc, I_a \times I_b) \mid tc \in TS\}.$$

Corollary 5 Let S_1, S_2 be IOSM's and I_a, I_b be IUT's . Then

$$exec(TS(S_1 \times S_2), I_a \times I_b) \subseteq exec(TS'(S_1, S_2), I_a \times I_b)$$

This means that if $I_a = I_b$ it is sufficient to apply $TS'(S_1,S_2)$ (or $TS'(S_2,S_1)$) once for interoperability testing. Note that Lemma 3, Theorem 4 and Corollary 5 are stated generally and they remain valid in the special case when the relevant IOSM's are pruned machines.

4. FURTHER DEVELOPMENT

When the approach shown in this paper is fully applied to Signaling Protocol, we obtain 19 test case skeletons (3 for call establishment, 11 for call clearing, 5 for restarting). In [KanC 96], they were called *test case skeletons* because actual abstract test cases can be derived from them by embellishing PDU's. In general, more than one abstract test cases may be constructed depending on the chosen test approach. With the test architecture used in this paper (Figure 1), interoperability test cases can be adequately described in TTCN.

Signaling Protocol is structured in such a way that depending on whether each IUT supports CALL_PROC, there are four different possible sets of test cases as follows:

		IUTa	IUTb
Upon receiving SETUP Does IUT send CALL_PROC to the user?	(1)	Yes	Yes
	(2)	No	Yes
	(3)	Yes	No
	(4)	No	No

We used the case (1) for the example in this paper. The case (4) can be handled in the exactly the same manner shown in this paper.

5. CONCLUSION

In this paper, first we clarified the notions of interoperation and interoperability testing and showed a specific interoperability test architecture and testing approach in accordance with those notions. We applied the approach to symmetric protocols and showed how various optimizations can be achieved. The details of optimizations were exhibited with the example of the ATM Signaling Protocol. In order to do that, firstly, we defined as the conceptual basis interoperability test case such that it utilizes stable states, abstracts from interactions internal to the two communicating protocol entities and is disjoint from conformance test. Secondly, we showed how to prune IOSM's so that, provided that the relevant conformance test suites already exist, the interoperability test suite derived from the pruned IOSM has no overlapping with them and covers all the proper interoperability test cases. Thirdly, we developed an efficient algorithm to generate all stable states to optimize test suite generation process itself. As the result, the test suite size can be reduced approximately to half in the case of symmetric protocols. Moreover, it was shown that when it comes to interoperability testing of two identical IUT's the number of test cases to be actually executed can be reduced to half.

There are some preliminary steps necessary before the interoperability testing can be applied. It is usually taken for granted that interoperability testing should be done *after* IUT's pass conformance testing [ISO 9646-1]. The first reason for this is that if IUT's are non-conforming most likely they would not interoperate. Moreover, when a system of implementations fails, we would not know which one is responsible for non-interoperation or whether there is an interoperation problem residing in specifications themselves. In addition, before the interoperability testing can be applied, implementations of the underlying service need to be thoroughly tested and should be known to be conforming. These assumptions are in line with the established conformance test methodology [ETSI 96].

Since the purpose of the B-ISDN signaling protocol is to provide ATM layer connection, one may think that one should check that traffic does indeed go through ATM layer connection after a successful call setup. This involves not only the behavior of the Signaling layer but also the behavior of the ATM layer. Thus testing of such interoperation behavior requires a more complicated test architecture with additional PCO's for the User Plane above the ATM layer. Being based on the test architecture shown in Figure 1, our interoperability test cases do not cover such cases. A similar view has been taken in ATM Forum signaling user-side ATS [AF Sig U-ATS] and network-side ATS [AF Sig N-ATS] and also CCITT ISDN Layer 3 D-channel testing [ITU-T Q.931bis], where data transport through B-channel and ATM connection, respectively, are not examined. Verdicts for nondeterministic test cases should be given by adopting *all-weather conditions assumption* as in other work [LuBP 94][Brin 88], i.e. run them sufficiently many times to expose any nonconformance. In the example in Section 3, such test cases are those beginning at (N0,N0) with SETUP and at (N6,N3) with CALL_PROC.

The basic idea of interoperability testing framework and approach developed in this paper was partially proposed and was used for interoperability test suite development inside the ATM Forum [KKCS 96][KanC 96]. In Section 4, we

mentioned the cases where the protocols are asymmetric. Our approach would not directly work for those cases. However, if protocols are *almost* symmetric then it is very likely that modification of our approach would work for them. We also plan to extend our application target to include point-to-multipoint signaling feature. Furthermore, we plan to extend our method for testing protocol data incorporating data flow analysis.

Acknowledgment We thank Dr. David Su, David Cypher and Leslie Collica at NIST, USA, for having numerous discussions with us on the subject of interoperability testing. Special thanks are due to David Cypher who clarified many subtle issues related to the ATM signaling protocol.

6. REFERENCES

[APRS 93] 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 B. V. (North-Holland), 1993.

[AraS 92] Arakawa, N. and Soneoka, T., "A Test Case Generation Method for Concurrent Programs", *Protocol Test Systems, IV*, J. Kroon, R. J. Heijink and E. Brinksma (Eds.), Elsevier Science Publishers B. V. (North-Holland), 1992.

[AF UNI] ATM Forum, *ATM User-Network Interface Specification, Version 3.1*, 1994.

[AF PNNI] ATM Forum, *ATM Forum PNNI Draft Specification*, 1996.

[AF Sig U-ATS] ATM Forum, "ATM Forum 96-0979: ATS for the UNI3.1 Signaling - User Side", 1996.

[AF Sig N-ATS] ATM Forum, "ATM Forum 95-1145R1: ATS for the UNI3.1 Signaling - Network Side (Part I)", 1996.

[Bonn 90] Bonnes, G., "IBM OSI Interoperability Verification Services", IFIP TC6/WG6.1 The 3rd International Workshop on Protocol Test Systems, 1990.

[Brin 88] Brinksma, E., "A Theory for the Derivation of Tests", *Protocol Specification, Testing and Verification, VIII*, S. Aggarwal, K. Sabnani (eds.), North-Holland, Amsterdam, pp. 63-74. 1988.

[CasK 94] Castanet, R. and Kone, O., "Deriving Coordinated Testers for Interoperability", *Protocol Test Systems, VI(C-19)*, O. Rafiq(Ed.), Elsevier Science B. V. (North-Holland), IFIP, 1994.

[Chow 78] Chow, T. S., "Testing Software Design Modeled by Finite-State Machines", *IEEE Trans. on SE, Vol. SE-4, No. 3*, May 1978.

[ETSI 96] ETSI, *Making Better Standards: Practical Ways to Greater Efficiency and Success*, ETSI MTS, 1996.

[GRSS 90] Gadre, J., Rohre, C., Summers, C., and Symington, S., "A COS Study of OSI Interoperability", *Computer Standards and Interfaces, Vol. 9, No. 3*, pp 217-237, 1990.

[ISO 95] ISO/IEC 9646-3, "Information Technology - OSI Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)", 1995

[ITU-T Q.2931] ITU-T, "ITU-T draft Recommendation Q.2931: B-ISDN User-Network Interface Layer 3 Specification for Basic Call/Bearer Control", 1994.
[ITU-T Q.931bis] ITU-T, "ITU-T Rec. Q.931bis: Abstract Test Suite for Basic Call Control Conformance Testing", 1994.
[KKCS 96] Kang, S., Kim, M., Cypher, D., and Su, D., "A Proposal for ATM Signaling Protocols Interoperability Test Suite Development", ATM Forum/96-0167, February 1996.
[KanC 96] Kang, S., and Cypher, D., "Test Case Skeletons for Interoperability Testing of ATM Signaling Network-Side Protocols", ATM Forum/96-0336r2, Baltimore, August 1996.
[KanK 95] Kang, S., and Kim, M., "Test Sequence Generation for Adaptive Interoperability Testing", IFIP TC6/WG6.1 The 8th International Workshop on Protocol Test Systems, Evry, France, September 1995.
[LuBP 94] 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*, February 1994.
[RafC 90] Rafiq, O. and Castanet, R., "From Conformance Testing to Interoperability Testing", The 3rd International Workshop on Protocol Test Systems, 1990.
[Sidh 90] Sidhu, D. P., "Protocol Testing: The First Ten Years, The Next Ten Years", *Protocol Specification, Testing and Verification, X*, L. Logrippo, R. R. Probert & H. Ural (Eds.), Elsevier Science Publishers B. V. (North-Holland), 1990.
[VerB 94] Vermeer, G.S., and Blik, H., "Interoperability Testing: Basis for the Acceptance of Communication Systems", *Protocol Test Systems, VI (C-19)*, Elsevier Science Publishers B. V. (North-Holland), 1994.

Sungwon Kang received B.A. degree from Seoul National University in Korea in 1982 and M.S. and Ph.D. in computer science from the University of Iowa in U.S.A in 1989 and 1992. Since 1993, he has been a senior researcher at Korea Telecom. From 1995 to 1996, he was a guest researcher at National Institute of Standards and Technology of U.S.A. In 1997, he was the co-chair of the 10th International Workshop on Testing of Communicating Systems and currently is the head of Protocol Testing Technology Team at Korea Telecom R&D Group. His research interests include communication protocol testing, program optimization and programming languages.

Myungchul Kim received B.A. in electronics engineering from Ajou Univ. in 1982, M.S. in computer science from the Korea Advanced Institute of Science and Technology in 1984, and Ph.D. in computer science from the Univ. of British Columbia in 1992. Since 1984, he has been working for Korea Telecom. In 1997, he was co-chair of the 10th International Workshop on Testing of Communicating Systems. Currently he is managing director of Testing Technology Research Section at Korea Telecom R&D Group and chairman of Profile Test Specifications - Special Interest Group of Asia-Oceania Workshop. His interests include protocol engineering on multimedia and telecommunications.