27

Test Management and TTCN based Test Sequencing

Jining Tian and Jianping Wu
Department of Computer Science
Tsinghua University, Beijing, 100084, P. R. China
wjp@tsinghua.edu.cn

Abstract

This paper presents a test management system named TMS which is capable of providing necessary management and organization of conformance testing according to the definition of conformance assessment process in the OSI Conformance Testing Methodology and Framework [1]. TMS is also able to dynamically sequence the TTCN test cases during test execution. The test sequencing is based on the ordering relation among test cases, which can solely derived from the TTCN test suite. An operational semantics of TTCN is defined here using the Labeled Transition System (LTS) to model the derivation of this relation. Design of the TMS also provided. Moreover, the TMS is an integrated part of our Protocol Conformance Testing System (PCTS). This paper also demonstrates how certain features of TMS, when combined with the characteristics of PCTS, can promote the overall system performance.

Keywords

Conformance Testing, Test System, Test Management, Test Sequencing

1 INTRODUCTION

The ISO has investigated considerable efforts in the elaboration of a common framework for protocol conformance testing, namely the OSI Conformance Testing Methodology and Framework [1]. According to this document, the Conformance Assessment Process (CAP) is the key activity in protocol conformance testing and the Means of Test (MOT) is the principal facility to carry out the process. Test Management System plays an important role in the protocol conformance testing process. Its target is to manage and organize the conformance assessment process carried out by the test laboratory for a specific test client. It covers the complete test preparation phase and the test operation phase as defined in [1].

The Protocol Conformance Testing System (PCTS) developed by Tsinghua University is an integrated platform for protocol conformance testing. PCTS has three major subsystems: the Test Management System, the Test Execution System, and the Test Result Report and

A. Cavalli et al. (eds.), *Protocol Test Systems VIII*

Analysis System. Three main functions of TMS are test organization, test suite management, and test document management.

The test management system we presented here has some unique characteristics. First, we claim a strict accordance to ISO standards. All the concepts, methods and procedures adopted in PCTS should strictly follow their definitions in the conformance test standard. Second, we have a TTCN based test suite management. TTCN is adopted as the basic notation in TMS. It is the only tangible form of test suite that exists in our TMS and PCTS. Third, we have a TTCN based test sequencing. All the ordering relations are derived solely from the TTCN test suite. By doing so, the TMS has become protocol independent and has gained great flexibility. Moreover, we adopted formal methods in the specification of our ordering relations. we here put forward a formal model of the operational semantics of TTCN in terms of the Labeled Transition System (LTS). Finally, we provide a user friendly interface and a convenient document management system.

The importance of test management can be seen from the following several aspects. First, it is an important step towards test automation. The test organizer will guide you step by step through the conformance assessment process without human interference. Second, the test sequencing module within TMS will systematically execute the test cases in the most efficient order. Besides, the TMS is also an important way to ensure the accordance with the standard and to enhance the comparability and the acceptability of the test results. Finally, the TMS provides a bunch of tools to promote test efficiency. The TMS, together with our test case generation tool TUGEN [4], the TTCN editing tool and the test execution system, have covered the whole life cycle of a test case. These tools thus formed an integrated platform for developing protocol conformance test suites.

The outline of this paper is as follows. Section 2 presents the basic functions of test management. Section 3 gives out a formal approach towards TTCN based test sequencing. Section 4 describes a design of the TMS according to the models presented above and discusses some technical issues. Section 5 shows an example of test sequencing. Section 6 concludes the paper.

2 TEST MANAGEMENT

2.1 Overview

There exists many useful tools for protocol conformance testing, each of which can more or less perform the management functions defined above. The XRTLE is a general purpose OSI tester from Alcatel TITN Inc. It can perform the SUT management function and can provide some test case selection criterion such as selection by PICS/PIXIT, by historical information or by human interference [18]. The IDACOM PT500 series testers are proprietary. They enjoy a high reputation in the testing of OSI lower layer protocols. They have a terminal based management system and also provide PICS/PIXIT selection [19]. The Test Case Management System (TCMS) from UBC can manage multiple relations among the test cases. Its static selection module supports PICS/PIXIT selection and its dynamic selection module supports test sequencing [6]. But none of them can perform the whole range of functions

provided by TMS and they seldom can directly manage the TTCN test suites. The TMS distinguishes itself by the following characteristics.

Test organization has a full coverage of all the steps that must be carried out in the test preparation and the test operation phase. It is a powerful tool which can provide useful guidance through out these two phases.

Test suite management is based on TTCN, which is now the only standard test notation in CTMF and is widely accepted as a test specification language. We gained remarkable advantages by adopting it as the basic format of test suite in TMS.

- The TMS is protocol independent and protocol test method independent;
- The test case can be created dynamically with our embedded TTCN editor;
- Confusion, disintegration and ambiguity caused by intermediate notations can be avoided;
- The exchange of test suite with other test laboratories will be easy;
- Interpretation based test execution strategy [3];
- Powerful test suite management and selection functions.

Convenient document management and user friendly interface have been invested to ease the cross-reference and information retrieval.

The structure of TMS is shown in Figure 1. Following are the detailed discussions of the major roles of TMS.

2.2 Test Organization

The test organization is responsible for the organization of the conformance assessment process. It is used here to ensure that all the procedures involved in PCTS are carried out in exactly the same order and manner as they are defined in CTMF. It contains the following sub-modules.

Static Conformance Review
This step is used to check whether the mandatory options in the conformance requirements has been claimed in the PICS. The capability test is the actual test of these options.

PICS/PIXIT Based Test Case Selection
The TMS must maintain a relation between the test cases and the PICS/PIXIT items, then select the test cases accordingly. Other selection methods are also available: selection by test group, by PDU/ASP used, by variables involved, by test steps it invoked, or by any attribute associated with test cases. Manual adjustment of the selection is also possible.

Test Sequencing
Test selection can be extended into the test campaign by the test sequencing. This action is beyond the scope of CTMF. It is done solely for the purpose of promoting test efficiency.

Test Parameter Preparation
Since we adopt the interpretation based test execution, no tangible Executable Test Suite (ETS) exists in the system. Only necessary parameter tables are obtained from the client information module and are provided to the TE.

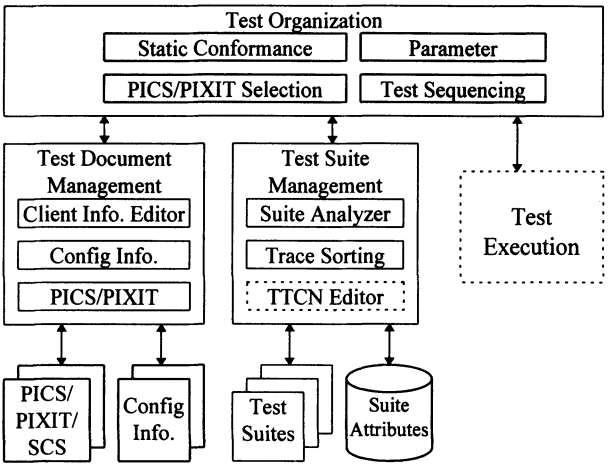

Figure 1 Structure of Test Management System

2.3 Test Suite Management

Test suite is the basis of the whole protocol conformance testing. It is involved in nearly every part of the conformance assessment process. This function of the TMS is to prepare the test suite for the conformance testing, especially for the PICS/PIXIT selection and test sequencing. It can be further divided into suite analysis and trace sorting.

The suite analyzer extracts various attributes from the TTCN test suite and stores them in a relational database. These attributes include test suite structure, selection expression, and other attributes of each test case such as its ID, variables used, PDU/ASP involved, etc. The analyzer also calculates the traces of each test case according to our definition of the TTCN machine which is introduced in section 3. These traces are then used by the trace sorting module to obtain the ordering relations which are defined in section 3. These relations are used by the sequencing module to determine the execution sequence of the test cases.

The TTCN editor can also be seen as an integrated part of TMS although it can also be invoked as an independent utility. This X-windows based editor can perform the basic maintenance role for TTCN. It can be invoked from any point within PCTS or TMS to support on-the-spot modifications of the test suite. It can translate test cases between GR form and MP form. The embedded TTCN editor together with our interpretation based TTCN execution gives PCTS the greatest flexibility. We can write a new test case on the spot and then execute it immediately.

2.4 Test Document Management

Several X-window based tools are provided here to facilitate user information registration and document generation.

- Check list generator: It indicates the information provided and needed by the test laboratory. The exchange of checklists are helpful in the choice of Abstract Test Method (ATM) and Abstract Test Suite (ATS).
- Client Information Editor: This editor defines the SUT and the IUT based on the user information, then generates the SCS describing the relation among SUT, IUT and relevant protocol standard. Administrative information is stored in database to ease future information retrieval.
- Universal PICS/PIXIT editor: It can read the PICS/PIXIT proforma written in a special format, display it in a window after the user has filled it in, then generate the PICS and PIXIT document.
- Configuration Information Editor: According to the PICS/PIXIT document, especially based on the ATM and ATS selected, the configuration of the tester can be determined. This includes the configuration of underlying service provider, auxiliary channels, and all the other entities in the test environment. Entities in PCTS are connected by a global message delivery system. Virtual channels must be established between related entities.

3 TTCN BASED TEST SEQUENCING

3.1 Basic Concepts

The test cases in TTCN are organized into a hierarchy of test groups. This grouping can provide some information about the relations among test cases, because test cases within the same group usually have related test purposes. However this do not imply any execution order among test cases.

The CTMF distinguishes four types of testing according to the extent to which they provide indication of conformance [1]: basic interconnection test (BIT), capability test, behavior test, and conformance resolution test (CRT). A list may be included in the test suite indicating which test cases are appropriate for BIT, and capability tests may be grouped in a separate group. These are just some rough indication of the test execution order, e.g., usually the BIT are execute first, then capability test, last the behavior test. They can not provide any finer information regarding to testing order within a certain test type, especially within the behavior tests which is the major part of an ATS. The above distinction is still an insufficient indication of execution order.

According to CTMF, each test case can be executed independently. But there do exists some relation among the results of test cases [5]. When a test case fails, some other test cases may be very likely to fail. We use "very likely" here because we even can not guarantee the failure of the same test case if it is executed again. In order to determine the test case execution order, we can define a "ordering relation" among test cases.

3.2 Obtain ordering relation from protocol specification

The traditional way of deriving the ordering relation is to start from the protocol specification. The Test Case Management System developed at University of British

Columbia [5] adopted this method. They called it "dynamic selection". According to their method, a relational model named Abstract Test Case Relation Model (ATCRM) can be constructed from the state machine of the protocol specification. Then, the test cases in a test suite can be mapped onto this relational model according to their IO paths to establish the ordering relations.

This method has several disadvantages. First, if we want to derive a relation on a test suite automatically, we must first obtain the formal descriptions of the related protocol specification written in a certain FDT. Second, the relation model may not match well with the test suite, in terms of the granularity of states. Third, the test case and the specification must be written in the same FDT or at least, has the same semantic model. Otherwise the mapping from test cases to the relational model will be very difficult.

3.3 Obtain ordering relations from test suite

In order to get rid of the above restrictions, we adopted a new approach to derive the ordering relation solely from the test suite. Here we can not obtain the state information from the protocol specification, but this information is equally unavailable during the test campaign. The MOT can understand the IUT only through interactions at PCO. We thus can only model the IUT together with the underlying service provider as a black box. In another word, we are going to give the MOT some intelligence to let the MOT learn gradually from the IUT, predict its future based on its past, and adjust the tester's behavior accordingly.

This approach has the following advantages. Since it is derived solely from the test suite, there will be no matching or mapping problems. Moreover, we make the sequencing method protocol independent because we do not assume the existence of protocol specification. Last, we can enjoy great flexibility introduced by this method due to its minimum requirements on outer conditions. Once we obtained a test suite in TTCN, we can right away carry out the sequencing against it.

3.4 A formal approach towards test sequencing

Although TTCN has already become an international standard, a standard concerning its formal semantics has yet to be developed. An operational semantics of TTCN based on the Labeled Transition System (LTS) is discussed in [8,9] which emphasizes its compositional structure.

In the following discussion, we are going to establish a basic semantic model for TTCN using the LTS. Our model focuses on the traces of a test case and it intends to provide an unambiguous definition of our ordering relations and also to achieve a sound mathematical foundation for test sequencing. This enables us to reason about the properties of these relations so that further optimization of test execution is possible. This formal model is also of great importance to our PCTS system because it has a TTCN based test case interpreter which bares a great similarity to the model. The properties of this model may also provide some useful guidance to the further development of the TTCN interpreter.

We first give out the definition of the TTCN machine and a simplified basic TTCN machine in terms of the LTS. Then, according to this model, the traces of the test cases and

the prefixing relations among them are presented. We further put forward some auxiliary notations based on the previous definitions. Next, the ordering relations are discussed. Unless explicitly explained, we always assume that the test results are repeatable. Of course this may not always be the case. At last, the non-determinism is discussed.

3.5 Labeled Transition System and TTCN machine

Def. 1 A LTS is a 4-tuple <S, L, T, s_0>, where S is a set of states, L is a set of observable actions. Let τ denotes the unobservable action, and δ denotes the successful termination. $T \subseteq S \times \{L \cup \{\tau\} \cup \{\delta\}\} \times S$ is the set of transitions. It can also be written as $s \xrightarrow{u} s'$ where $s, s' \in S$, $\mu \in L \cup \{\tau\}$, $(s,\mu,s') \in T$.

Def. 2 The syntax of a Test Behavior Expression (TBE) is defined as follows:
$B =_{def}$ stop | exit | ?a;B | !a;B | B[]B | B>>B | [q];B | [I:=v];B

The operator precedence is ';' > '[]' > '>>'. q is the qualifier which is a Boolean expression. Its syntax and semantics are ignored here just for simplicity. This is an algebraic representation of the Abstract Evaluation Tree defined in [1]. The TBE can express the most important syntactical components in the dynamic part of TTCN, but it is much simpler. So that it can be handled by our basic TTCN machine defined below. This test behavior expression is a formal representation of the Abstract Evaluation Tree defined in [1].

We can first translate TTCN into TBE. Let *Texp* denotes the set of all TBEs. Let *Texpr*(t) denotes the function that does this translation. *Texpr*(t) = B_t . $B_t \in$ *Texp*.

Ex 1: Sequential composition is mapped onto action prefix.

```
!a
  ?b        ⇒    !a (?b; stop [] ?c; stop )
  ?c
```

Ex 2. Attached trees are viewed as a separate process:

```
!a          B:   ?d
  +B              !e  ⇒          !a ; B >> ?c
    ?c            ?f(Inconc)  or  !a ; ( ?d ; !e ; exit [] ?f ; stop ) >> ?c
```

Ex 3. Loops are translated into recursions:

```
?a          L1
  !b              ⇒    L1 = ?a ; !b ; L1
    → L1
```

Def. 3 A TTCN Machine, is a virtual machine that can perform the evaluation of the TTCN test case. The state of this TTCN machine can be represented by a 4-tuple: (*stack*, *env*, *ctrl*, *sto*). It has four parts:

1) Internal stack: *stack* is the place to store some intermediate results;
2) Environment: *env* preserves the information about variable type and scope;
3) Control Part: *ctrl* contains the TTCN test case to be evaluated;
4) Storage Part: *sto*=(m, i, o) $\in$ *Mem*×*Input*×*Output*, where m $\subseteq$ *Mem*=*Ident*×*Value*, is a set of pairs represents the identifier and its

corresponding value. *i* and *o* are the input queue and output queue respectively. Elements in these queues are called messages and each representing an event. Each element has a PCOid and an event type.

Since our aim is to specify the ordering relations, we can omit some of the details in this module, such as the stack for the calculation of expressions and the analysis of complex sentence constructions, and the environment.

Def. 4 A Basic TTCN Machine is a simplified TTCN machine. Its state has only two parts: the control part and the storage part:

1) Control part: $ctrl \in Texp$, is the TBE to be evaluated. In order to avoid getting entangled into the complicated TTCN syntax, we use TBE instead of TTCN. Thus we can achieve a simple and elegant model which is nevertheless sufficient to describe our ordering relations.
2) The storage part is the same as that of a TTCN machine.

The behavior of a Basic TTCN machine can be modeled by a Labeled Transition System:

Def. 5 A Test Behavior Expression B_t is defined as in Def 2, its semantics is defined by a Labeled Transition System $Lts(B_t)$: $Lts(B_t) =_{def} <S, L', Tran, s_0>$ where
$S=\{(ctrl, sto)\} \subseteq (Texp \times Store)$ denotes all the possible states of the TTCN machine;
$L'=\{L \cup \{\tau\} \cup \{\delta\}\}$ is the set of all possible actions. ?x denotes a receive event and !x denotes a send event;
$s_0 = (B_t, sto_0)$ where $sto_0 = (m_0, i_0, o_0)$ is the initial state;
$Tran \subseteq S \times L' \times S$ are the transitions satisfying the following rules:

$(exit, sto) \xrightarrow{\delta} (stop, sto)$;
$(?a;B, (m, a \cdot i, o)) \xrightarrow{?a} (B, (m, i, o))$;
$(!a;B, (m, i, o)) \xrightarrow{!a} (B, (m, i, a \cdot o))$;
$(B_1[]B_2, sto) \xrightarrow{u} (B_1', sto')$, if $(B_1, sto) \xrightarrow{u} (B_1', sto')$;
$(B_1[]B_2, sto) \xrightarrow{u} (B_2', sto')$, if $(B_2, sto) \xrightarrow{u} (B_2', sto')$;
$(B_1>>B_2, sto) \xrightarrow{u} (B_1'>>B_2, sto')$, if $(B_1, sto) \xrightarrow{u} (B_1', sto')$;
$(B_1>>B_2, sto) \xrightarrow{\tau} (B_2, sto')$, if $(B_1, sto) \xrightarrow{\delta} (B_1', sto')$;
$([q];B, sto) \xrightarrow{\tau} (B, sto)$, if q=Ture;
$([q];B, sto) \xrightarrow{\tau}$ stop, if q=False;
$([I:=v];B, (m, i, o)) \xrightarrow{\tau} (B, (m[v/I], i, o))$ where $m[v/I] = (m-\{(I, x)\}) \cup \{(I, v)\}$

3.6 Traces and Prefix

Def. 6 Traces of a LTS
Let $<S, L, T, s_0>$ be a LTS, $s, s', s_1, s_2, \ldots s_n, s_{n+1}, s_1', s_2', \ldots s_n' \in S$, $\mu_1, \ldots \mu_n \in L'$.
Let $\sigma = \mu_1 \cdot \mu_2 \ldots \mu_n$ be a sequence of labels from L', then σ is said to be a trace over L'.

The set of all possible traces over L′ is noted as L′*. We further have the following notions:

1) if $s = s_1 \xrightarrow{\mu_1} s_2 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_n} s_{n+1} = s'$, then $s \xrightarrow{\sigma} s'$.
2) if $s = s_1 \xrightarrow{\tau^*} s_1' \xrightarrow{\mu_1} s_2 \xrightarrow{\tau^*} \ldots \xrightarrow{\tau^*} s_n' \xrightarrow{\mu_n} s_{n+1} = s'$, then $s \overset{\sigma}{\Rightarrow} s'$.
3) if $\exists s', s \xrightarrow{\sigma} s'$, then $s \xrightarrow{\sigma}$.
4) if $\exists s', s \overset{\sigma}{\Rightarrow} s'$, then $s \overset{\sigma}{\Rightarrow}$.

Def. 7 B_t is the test behavior expression of test case t and $Lts(B_t) = <S, L, T, s_0>$ is the LTS for B_t where $s_0 = <B_t, sto_0>$. Let T_c denotes the set of all test cases in a test suite. t is a test case.

$Trace(t) =_{def} \{ \sigma | <B_t, sto_0> \overset{\sigma}{\Rightarrow} \}$

$AllTrace(t) =_{def} \{ \sigma | <B_t, sto_0> \xrightarrow{\sigma} \}$

Trace(t) denotes all the possible observable traces of test case t, and *AllTrace*(t) denotes all possible observable and unobservable traces of test case t.

Def. 8 Prefix is a relation among traces: $\prec \subseteq L^* \times L^*$. Let '.' denotes the concatenation of traces. $\sigma,\sigma' \in L^*$, if $\exists \sigma'' \in L^* \wedge \sigma = \sigma' \cdot \sigma''$, then σ is a *prefix* of σ, noted as $\sigma' \prec \sigma$, and $\sigma'' = \sigma \backslash \sigma'$.

3.7 Auxiliary Definitions

Def. 9 Auxiliary Definitions

1) *Log*: $T_c \mapsto L^*$. $Log(t) =_{def} \sigma$, where $\sigma \in Trace(t)$.
   Log emulates the function of the MOT: it executes the test case and gives out a log. σ is the trace observed during the test campaign.
2) *Verdict*: $L^* \times T_c \mapsto \{Pass, Inconc, Fail\}$
   $Verdict(\sigma,t) =_{def} v$, where $\sigma = Log(t)$, $v \in \{Pass, Inconc, Fail\}$.
   v is the verdict assigned in test case t after trace σ is observed.
3) $Purpose(t) =_{def} \{ \sigma_p \in Trace(t) \mid Verdict(\sigma_p,t) = Pass \}$
   These are the purpose traces, ie. traces that have "Pass" verdicts.
4) $Expect(\sigma,t) =_{def} \{ x \mid \exists \sigma' \in Purpose(t)$ satisfy $\sigma \cdot x \leq \sigma' \}$
   Expect(σ,t) denotes the proper actions expected by the test case after trace σ has been observed.
5) Let σ_F denotes a trace, $t \in Tc$, $(Log(t) = \sigma_F) \wedge (Verdict(\sigma_F, t) = Fail \mid Inconc)$, then $Correct(\sigma_F, t) =_{def} \sigma_c$ where $\sigma_F = \sigma_c \cdot f \wedge f \notin expect(\sigma_c)$
   Correct(σF, t) denotes the correct trace that had been observed, just before the even leading to a fail or "Inconc" verdict happened.
6) $Proper(\sigma, t) =_{def} \{ \sigma_p \mid \sigma_p = \sigma \cdot x, x \in Expect(\sigma, t)\}$
   Proper(σ, t) denotes the proper traces expected by the test case t after trace σ has been observed.
7) $Der(Tr) =_{def} \{ \sigma \mid \sigma' \prec \sigma \wedge \sigma' \in Tr\}$, where $Tr \subseteq L^*$.
   Der(Tr) denotes all the possible extensions of the traces in set Tr.

3.8 Ordering Relations

Def. 10 Ordering relation 0:

If $\forall\ \sigma_2 \in Purpose(t_2)$, $\exists\ \sigma_1 \in Purpose(t_1)$ satisfy $\sigma_1 \prec \sigma_2$, then $t_1 \leq_0 t_2$.

The meaning is if the test purpose traces of t_2, ie. those have "Pass" verdicts, are prefixed by the test purpose traces of t_1, then t_1, t_2 has ordering relation $\leq_0$.

Def. 11 Ordering relation 1

If $\forall \sigma \in Purpose(t)$, $\exists t_i \in T_c$ satisfy $\exists \sigma_i \in Purpose(t_i) \wedge \sigma_i \prec \sigma$, then $\{ t_i \} \leq_1 t$.

This relation takes into account the situation that several test cases together determines the result of another test case. We can further refine this relation by using a more detailed test report , we can refer to the log to obtain the trace of the test case that has just been observed. Since we broke the test cases into traces, then instead of obtaining a relation among test cases, we get a relation between a set of traces and a test case.

Def. 12 Ordering relation 2

$Tr \subseteq L^*$, $t \in Tc$. If $Purpose(t) \subseteq Der(Tr)$, then $Tr \leq_2 t$.

We can obtain more relations according to this definition because usually σ_F is shorter than *Purpose*(t), so it has a higher possibility of becoming a prefix of another test case.

3.9 Properties of Ordering Relations

Here we demonstrate how to use the ordering relations to carry out test sequencing. Using the following theorems, we can determine the result of a test case t without executing it.

Theorem 1. t', t are test cases. If

1) $Verdict(Log(t'), t) = Fail \mid Inconc$,
2) ttt $t' \leq_0 t$,
3) test results are repeatable, then $Verdict(Log(t), t) = Fail \mid Inconc$.

Theorem 2. T_c is a set of test cases, t is a test case. If

1) $\forall\ t_i \in Tc$, $Verdict(Log(t_i), t_i) = Fail \mid Inconc$,
2) $T_c \leq_1 t$,
3) test results are repeatable, then $Verdict(Log(t), t) = Fail \mid Inconc$.

Theorem 3. T_c is a set of test cases, t is a test case. If

1) $\forall\ t_i \in T_c$, $Log(t_i) = \sigma_{Fi}$, $Verdict(\sigma_{Fi}, t_i) = Fail \mid Inconc$,
2) $\exists\ Tr_{unreach} = \bigcup_i Proper(Correct(\sigma_{Fi}, t_i)) \wedge Tr_{unreach} \leq_2 t$,
3) test results are repeatable, then $Verdict(Log(t), t) = Fail \mid Inconc$.

The proofs are straight forward except the last theorem which needs some explanation. When test case t_i fails (or get the "Inconc" verdict) with trace σ_{Fi} observed, we can mark *Proper*(*Correct*(σ_{Fi}, t_i)) as an unreachable set of traces provided that the test results are repeatable. The size of the set of unreachable traces grows as the test continues. This set can be written as $Tr_{unreach} = \bigcup_{t_i} Proper(Correct(\sigma_{Fi}, t_i))$ where t_i is the test case failed(or get the "Inconc" verdict). Whenever the purpose traces of a test case t are covered by *Der*($Tr_{unreach}$), or, in another word $Tr_{unreach} \leq_2 t$, then t is doomed to fail or (to get the "Inconc" verdict), and can thus be skipped. In short, the sequencing can be done according to the above relations. Efficient algorithms are needed to calculate the $Tr_{unreach}$ and to determine the '$\leq$' relations.

3.10 Non-determinism

Actually the test results may not be repeatable. Even the re-execution of the same test case against the same IUT may give out different results. The IUT's choices at the same forking point may not be the same. But there may exist some statistical probabilities among the IUT's choices. Those test cases whose purposes are within *Der*($Tr_{unreach}$) merely have higher probability to fail. We say a test case is doomed to fail under the assumption that the test results are repeatable. In reality, some adjustment is needed: we mark a trace unreachable only when it still fails after a certain number of retries. This number is proportional to the number of valid IUT responses at this branching point.

4 DESIGN OF THE TEST MANAGEMENT SYSTEM

4.1 PCTS and its TMS

The general design guidelines of PCTS are as follows.

- Strict accordance with CTMF
- Use of Formal Methods
- TTCN based Test Execution
- X-window based test presentation
- Object-Oriented software environment

Because of the unique functions carried out by it, the TMS also has its own highlights when applying these general guidelines. Since it has a broad coverage of the Conformance Assessment Process, the procedures defined in CTMF can be best presented here. TTCN based test notation is also a unique characteristic of TMS. All the selection and sequencing are done based on TTCN. Since there are numerous human-machine interactions in the test preparation phase, a convenient and friendly interface is of great importance here. Great efforts have been made in order to ease cross-reference, information retrieval and document generation.

4.2 Modules in TMS

TMS has six major modules as they are shown in Figure 2.

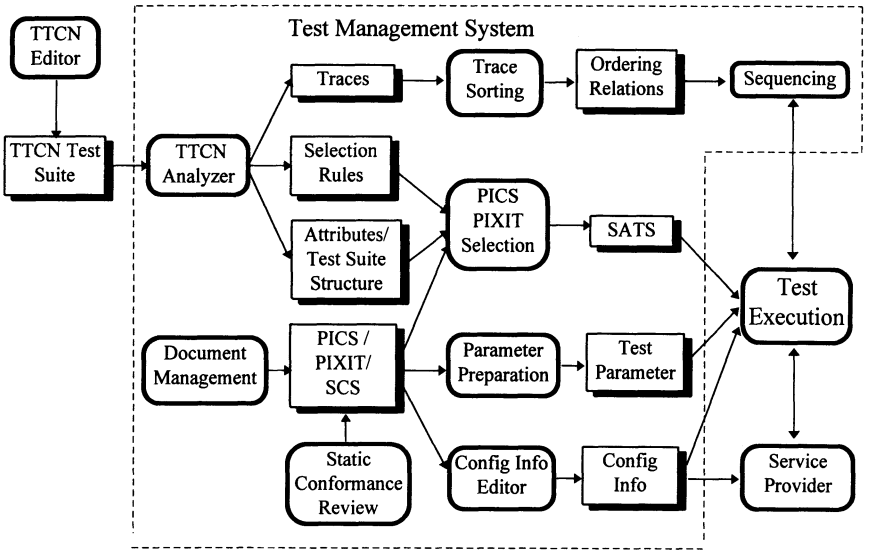

Figure 2 Modules of TMS in Conformance Assessment Process

Document management

It obtains necessary information from the client, this includes PICS/PIXIT information required by PICS/PIXIT selection and parameterization. It will also get other administrative information from the test client. It will provide some necessary check lists to facilitate the negotiation between test laboratory and test client on the selection of test method and test suites. SCS, PICS and PIXIT documents are generated by this module.

TTCN analyzer

This module carries out the syntax analysis of test suites written in TTCN.MP. Because the BNF definition for TTCN has more than 300 production rules, so we use Lex and Yacc to facilitate the analysis. This module will generate three parts of information.

- Test suit structure and test case attributes. The structure is an index of the test suite and the attributes include the test case ID, parameters and PDU/ASP used, etc. They are used for PICS/PIXIT selection.
- PICS/PIXIT selection rules. This information is extracted from the selection expressions part in the test suite, or from a separate selection table that comes along with the test suite. It sets up a selection criterion for each test case by using a Boolean expression of PICS/PIXIT variables.
- Purpose traces of each test case. We first derive the test behavior expression of each test case, then calculate its purpose traces, i. e. the traces with "Pass" verdicts. To avoid infinite traces, loops are unfolded just once.

Thus for each test case, we can create a behavior tree by merging the common prefixes of its traces. Each path between the root and the leaf is a possible trace.

PICS/PIXIT selection

For each test case, its selection expression is calculated, the result will explicitly indicate whether it will be selected or not. Then a new index is generated, only the selected test cases are included. This is the Selected ATS (SATS). User interference and selection according to other criterion can be introduced at any time during the selection.

Trace sorting
This module is the central part of test sequencing. First we construct a global behavior tree by merging the purpose traces of each test case together. This is done also by merging the common prefix of the traces. We then mark the ending node of each purpose trace in the global behavior tree with the identifier of its corresponding test case. Second, in the global behavior tree, at each node that corresponding to a purpose trace, we are going to include a reference to that test case. The reference count of a test case is the number of its purpose traces. With the global behavior tree, it's much easier for us to determine prefix relations among traces: traces corresponding to a parent node in the tree are always the prefix of those traces corresponding to its children.

Sequencing
The execution sequence of the test case will be determined by the following. We traverse the global behavior tree in a width first strategy and execute the test cases corresponding to the node we reached. If the test case fails or gets an "Inconc" verdict with a logged trace σ_F, then the fail count of all the traces within *Purpose*(*Correct*(σ_F,t)) are increased by 1. If the fail count of any trace exceeds its upper limit, it will be marked unreachable and will be added to $Tr_{unreach}$. For each purpose trace within *Der*(*Purpose*(*Correct*(σ_F,t))), the reference count of its corresponding test case will be decreased by 1. If the reference count of any test case reaches 0, then it will not be executed.

Since we are executing the test case in parallel with the interpreting of another [3], the sequencing module is providing the Test Execution module with both the current and the most possible next test cases. When traversing the behavior tree, we adopted the width first strategy, so that the ordering relations between the current and the next test cases will be reduced to the minimum. Thus, when the current test case fails, the next test case which has already been pre-analyzed by the TE is unlikely to become doomed. In this way, we improved overall performance.

Static Conformance Review, Parameter Preparation and Test Configuration
The static conformance review module check the PICS to see if all the mandatory requirements have be fulfilled. The parameter preparation module collects various test parameters from test client and feed them to the test execution. The test configuration module set up the correct environment for testing.

5 EXAMPLE OF TEST SEQUENCING

Suppose we have three test cases Tc1 ={t1, t2, t3}. Their TTCN.GR representations are shown in Table 1. Their test behavior expressions are:

Bt1 = (!a;((?b;exit)[](?c;exit)[](?other;stop)))

Bt2 = (!a;(?b;!g;(?e;exit[]?other;stop)[]?other;stop))
Bt3 = (!a;(?b;!g;(?f;exit[]?other;stop)[]?other;stop))

t_1: TTCN	Verdict
!a	
?b	Pass
?c	Pass
?otherwise	Fail

t_2: TTCN	Verdict
!a	
?b	
!g	
?e	Pass
?otherwise	Fail
?otherwise	Inconc

t_3: TTCN	Verdict
!a	
?b	
!g	
?e	Pass
?otherwise	Fail
?otherwise	Inconc

Table 1 TTCN.GR of test cases

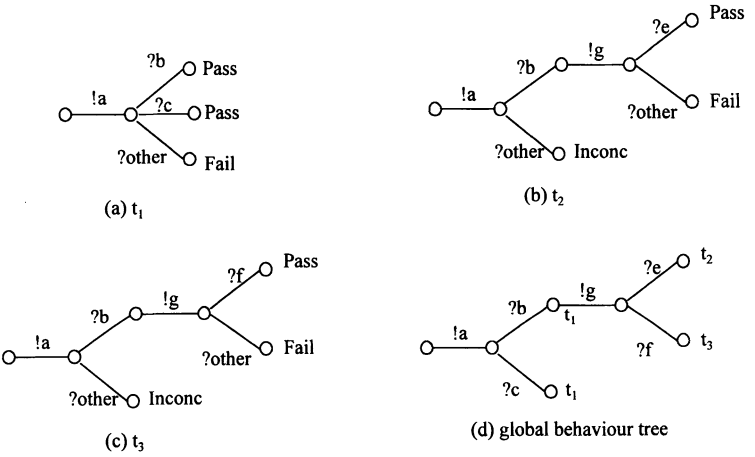

Figure 3 Behaviour tree of test cases

Their behavior trees and the global behavior tree of the three test cases are shown in Figure 3. The purpose traces calculated by the basic TTCN machine are:

Purpose(t_1) = { !a?b, !a?c }, *Purpose*(t_2) = { !a?b!g?e }, *Purpose*(t_3) = { !a?b!g?f }.

According to Def 10, because (!a?b) $\prec$ (!a?b!g?e), (!a?b) $\prec$ (!a?b!g?f), so we have $t_1 \leq_0 t_2$ and $t_1 \leq_0 t_3$. The execution sequence will be (t1,t2,t3) or (t1,t3,t2). According to Theorem 1, if t_1 fails then t_2 and t_3 are skipped.

But consider another test suite $T_{c2}=\{t_2,t_3\}$. This time we still can find some relations between t_2 and t_3. If t_2 fails with $Log(t_2)$=!a?x, x≠b. Then we have *Proper*(*Correct*(!a?x))={!a?b,!a?c}. Then the fail count of !a?b and !a?c increases by one. Here we set the upper limit of all the fail counts to one. Then {!a?b,!a?c} will be marked unreachable. $Tr_{unreach}$={!a?b, !a?c}. Now *Der*({!a?b})={!a?b, !a?b!g?e, !a?b!g?f , ...}.

According to Def 12, we have $\{!a?b,!a?c\} \leq_2 t_2$, and $\{!a?b,!a?c\} \leq_2 t_3$. So we have $Tr_{unreach} \leq_2 t_3$. According to Theorem 3, t_3 is skipped.

In reality, both the reference count of t_1(!a?b!g?e) and t_2(!a?b!g?e) are originally set to one. After t_2 fails, we decrease the reference count of t_3 and it becomes zero. So it is dropped.

6 CONCLUSIONS

The TMS presented above has been used in the testing of many real OSI protocols such as X.25 LAPB, X.25 PLP, TP and X.400, etc. Sequencing has been applied to the X.25 LAPB and X.25 PLP tests. Standard test suites in ISO 8882 are adopted[2]. In X.25 LAPB and PLP test suites, although ordering relations among test cases are few, the common traces between test cases are numerous. Usually the test cases within a same test group has the same preambles. So our sequencing strategy works especially well under the following situations: when a preamble gets "Inconc", and after several retries the whole group will be discarded.

For example, ISO 8882-2 contains 288 test cases for X.25 LAPB and 287 cases are counted for ordering (except DL4_113). In the global behavior tree, totally there are 25888 purpose traces with average length to be 11.57 (test events). The number of common prefixes can be seen from the following Common Prefix Ratio (CPR):

$$CPR =_{def} 1 - \frac{\text{(number of test events in the global behaviour tree)}}{\text{(total number of test events in all purpose traces)}} = 1 - \frac{67062}{299465} = 0.78$$

This means that in the test suite about 80% of test trace length is included in another trace.

The choice of LTS in the establishment of a formal model of TTCN seems to be intuitive. Currently it appears to be the only attempt to formalize TTCN [8,10,23,24]. We believe this is largely due to the tree nature of TTCN test cases and the succinctness of expressing behavior trees in LTS.

In general, the sequencing method we introduced here can greatly increase the testing efficiency especially during the testing of immature systems.

The interpretation based test execution strategy has been proofed to be a great success in the testing of OSI lower level protocols such as X.25 LAPB, X.25 PLP and TP. Now we are focusing on the testing of higher level protocols. Problem aroused when ASN.1 was introduced into TTCN due to the difficulty in handling complex data types which is a common and in-born weakness of all interpreters (e.g. BASIC, Tcl, etc.). This is a new challenge for the TMS to manage TTCN test suite with ASN.1 extensions. However we deployed an interpreting-compiling combined method to overcome this type handling problem. All the ASN.1 definitions embedded in TTCN and the definitions by reference are collected together by the TMS and are then fed to an ASN.1 compiler to create a ASN.1 library which includes encoding, decoding, assignment and comparison functions that can be accessed during test execution. In this way we enjoys the efficiency brought by the compilation while at the same time maintains the flexibility introduced by the interpretation. This method is feasible but is subject to further verification.

Current research works in this area include the development of more efficient algorithms for sequencing and the refinement of the TTCN semantic model .

6 REFERENCES

[1] ISO/IEC IS 9646 1-7 1991(E), OSI-Conformance Testing Methodology and Framework.
[2] ISO/IEC 8882 1-3, ISO/IEC 8882:1992(E), X.25 DTE Conformance Testing.
[3] Yamin Wang and Jianping Wu, An approach to TTCN-based test execution, IWPTS VII, p299-306, IFIP 1994.
[4] Ruibing Hao, Jianping Wu and Samual T. Chanson, Design and implementation of an automatic test suite generator, Chinese Journal of Advanced Software Research, vol. 1, No.2, p152-165 1994.
[5] Samual T. Chanson and Qin Li, On Static and Dynamic Test Case Selections in Protocol Conformance Testing, IWPTS IV, 1991.
[6] Samuel T. Chanson, Sijian Zhang and Qin Li, A Test Case Management System, IWPTS, 1992.
[7] La Briere, Testing in Practice OSI Test Center, IFIP Trans. C vol:C-11, p19-29, 1993.
[8] Walter and B. Plattner, An operational semantics for concurrent TTCN, IWPTS V, p131-143. IFIP, NorthHolland, 1992.
[9] Finn Kristoffersen and Thomas Walter, TTCN test case correctness validation, IWPTS VII, p 37-53, IFIP 1994.
[10] G. J. Tretmans, A Formal Approach to Conformance Testing, PhD Thesis, 1992.
[11] R. Milner, A Calculus of Communicating Systems. Lecture Notes in Computer Science 92. Spring-Verlag, 1980.
[12] Hoare, Communicating Sequential Processes. Prentice-Hall, 1985.
[13] T. Bolognesi and E. Brinksma, Introduction to the ISO specification language LOTOS, Brinksma, et al, A Formal Approach Computer Networks and ISDN Systems, IFIP Transactions. North Holland, 1992.
[14] CCITT. Specification and Description Language (SDL), CCITT Recommendation Z.100, CCITT/ITU, 1992.
[15] ISO/IEC, OSI-Specification of Abstract Syntax Notation 1 (ASN.1), IS 8824, ISO/IEC, 1987.
[16] K. Inan and P. Varaiya, Finitely recursive process models for Discrete Event Systems, IEEE Trans. on Automations. Vol. 33, No. 7, July 1988.
[17] de Meer and V.Heymer,etc, An approach to a Conformance Testing Methodology and the COAST Test System, IFIP, 1991.
[18] Alcatel TITN Inc., XRTLE User Guide, 1992.
[19] UBC/IDACOM, OSI PT Environment, 1990.
[20] H. Ural and Z. Wang, Synchronizable test sequence generation using UIO sequences, Computer Communications, Vol:16, p653-63,1993.
[21] M. E. Koblentz, Issues in testing fast packet services over the broadband (ISDN), IFIP Transaction C vol:c-11, p3-18, 1993.
[22] A. D. Varvitsiotis and G. I. Stassinopoulos, Extending ASN.1 into a full-fledged constraint language in the context of OSI protocol Conformance Testing, Computer Networks and ISDN Systems, p1243-63, 1993.
[23] E.Brinksma, et al, A Formal Approach to Conformance Testing, IWPTS IV, Netherlands, Oct. 1991.
[24] Hogrefe, Status Report on the FMCT Project, IWPTS VII, p.165-180, IFIP, 1994.