1

VERIFICATION OF TEST SUITES

Claude Jard, Thierry Jéron and Pierre Morel *
IRISA, Campus de Beaulieu, F-35042 Rennes, France
{ Claude.Jard, Thierry.Jeron, Pierre.Morel } @irisa.fr

Abstract We present a formal approach to check the correctness and to propose corrections of hand-written test suites with respect to a formal specification of the protocol implementations to test. It is shown that this requires in general a complex algorithmic comparable to model-checking. The principles of a prototype tool, called VTS, and based on the synthesis algorithms of TGV, are presented. We then prove the usefulness of the technique by checking a significant part of the ATM Forum test suite for the SSCOP protocol.

Keywords: Conformance testing, TTCN, Verification, SDL, SSCOP

1. INTRODUCTION

The simple idea developed in this article is that, as soon as one has a formal specification, one can check the correctness of test suites written by hand. It is a useful function since many errors of various types remain in manual test cases. It is in particular the case when designing tests in context or distributed tests. One can also go a little further trying to automatically correct test cases.

Test case verification appears easier than the synthesis problem. This latter, already well studied, must still face problems of state explosion or handling complex symbolic systems. However, test case verification is not a commonplace algorithmic problem. Indeed the test cases do not only arise as sequences of interactions which a simulator can reproduce. Test cases are often real reactive programs which can be abstracted by general graphs.

* respectively researcher at CNRS, at Inria and PhD Student of the University of Rennes. The authors would like to acknowledge reviewers for helpful comments. They also acknowledge Amr Hashem, student of the Information Technology Institute in Cairo, who did his stage on the application of our VTS tool to the SSCOP test suite. This work was part of the FORMA French national project on formal methods.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35516-0_20

H. Ural et al. (eds.), *Testing of Communicating Systems*

Existing approaches of the problem are based on a co-simulation of the test case with the specification. This is the case with TTCN-Link [12] for SDL specifications and TTCN test cases [7]. The same principle was also used in [4] using the Tetra tool for Lotos specifications and test cases translated from TTCN. Co-simulation is useful for the early detection of errors in test cases but is not sufficient in general for exhaustively checking test cases. The reason is that this technique only allows to look at particular sequences. The first problem is due to possible loops in test cases which may be unfolded only a bounded number of times. It is also not sound for non-deterministic specifications (due to hiding of internal actions for example), as correctness involves the comparison of possible outputs of the specification with the inputs of the test case after the same trace. This problem was mentionned but not solved in [4]. Thus correctness of test cases with respect to the behaviors contained in a formal specification requires the installation of a complex algorithmic comparable with model-checking. We precisely have this experience in the development of the test generation tool TGV [8][1].

Based on some basic blocks of TGV, we have developed a toolset called VTS (for Verification of Test Suites), specialized for the verification of tests. In this case, the test case plays the role of a complete test purpose[2] which strongly guides the traversal of the specification state graph. As the algorithm works on-the-fly by building in a lazy way only the part of the specification state graph (and its observable behavior) corresponding to the test case, the performances are very satisfactory. The principal limitations that one met relate to the necessary abstraction of the test cases expressed in TTCN in the general format of graphs we have in VTS. The suggested technique is illustrated by the validation of the test suite of the SSCOP protocol proposed by the ATM Forum. We found several errors and proposed some corrections.

The continuation of the article is structured as follows: Section 2 presents the testing theory which constitutes the basis of the test-checking toolset. Section 3 is devoted to the principles of test-checking. Section 4 presents the application of VTS to the verification of part of the SSCOP test suite. The article ends in a conclusion and some prospects.

2. FORMAL CONFORMANCE TESTING

In this section we introduce the models used to describe specifications, implementations and test cases. We then define a conformance relation that precisely states which implementations conform to a given specification. We report to [13] for a precise definition of the testing theory used.

[1] TGV generates test cases from specifications in SDL and LOTOS and formalized test purposes.

[2] This means that all observable actions are present in the test purpose, while TGV allows more abstraction.

2.1 Models

The model used for specifications, implementations and test cases is based on the classical model of labelled transition systems with distinguished inputs and outputs.

Definition 1 *An IOLTS is an LTS* $M = (Q^M, A^M, \rightarrow_M, q_0^M)$ *with* Q^M *a finite set of states,* A^M *a finite alphabet partitioned into three distinct sets* $A^M = A_I^M \cup A_O^M \cup I^M$ *where* A_I^M *and* A_O^M *are respectively inputs and outputs alphabets and* I^M *is an alphabet of unobservable, internal actions,* $\rightarrow_M \subset Q^M \times A^M \times Q^M$ *is the transition relation and* q_0^M *is the initial state.*

We use the classical following notations of LTS for IOLTS.
Let $q, q', q_{(i)} \in Q^M$, $Q \subseteq Q^M$, $a_{(i)} \in A_I^M \cup A_O^M$, $\tau_{(i)} \in I^M$, and $\sigma \in (A_I^M \cup A_O^M)^*$.

- $q \stackrel{\epsilon}{\Rightarrow}_M q' \equiv (q = q' \vee q \stackrel{\tau_1 \ldots \tau_n}{\longrightarrow}_M q')$
- $q \stackrel{a}{\Rightarrow}_M q' \equiv \exists q_1, q_2 : q \stackrel{\epsilon}{\Rightarrow}_M q_1 \stackrel{a}{\rightarrow}_M q_2 \stackrel{\epsilon}{\Rightarrow}_M q'$
- $q \stackrel{a_1 \ldots a_n}{\Rightarrow}_M q' \equiv \exists q_0, \ldots q_n : q = q_0 \stackrel{a_1}{\Rightarrow}_M q_1 \ldots \stackrel{a_n}{\Rightarrow}_M q_n = q'$.
- $traces_M(q) = \{\sigma | q \stackrel{\sigma}{\Rightarrow}_M\}$ and $traces_M(M) = traces_M(q_0^M)$.
- q **after**$_M$ $\sigma = \{q' | q \stackrel{\sigma}{\Rightarrow}_M q'\}$ and Q **after**$_M$ $\sigma = \cup_{q \in Q} q$ **after**$_M$ σ.
  For an IOLTS M, we sometimes use M **after** σ for q_0^M **after**$_M$ σ.
- $out_M(q) = \{a \in A_O^M | q \stackrel{a}{\Rightarrow}_M\}$ and $out_M(Q) = \{out_M(q) | q \in Q\}$.

Specifications. A specification is modelled by an IOLTS $S = (Q^S, A^S, \rightarrow_S, q_0^S)$. This IOLTS describes the complete behavior of the specification, including internal actions. We consider quiescence (livelock and output quiescence) as observable (by timeouts). So we need to model possible quiescence in the specification. Formally, a state q of S is quiescent if $\neg(\exists a \in A_O^S \cup I^S, q \stackrel{a}{\rightarrow}_S)$ (output quiescence)[3] or $\exists \alpha \in I^{S*}, q \stackrel{\alpha}{\rightarrow}_S^* q$ (livelock)[4]. The *suspension automaton* S^δ of S is then obtained by considering the special label δ as an output and adding self loops labelled by δ in all quiescent states. This corresponds to [13] except that we also consider livelocks. In practice, S^δ is not build but its construction is mixed with τ-closure (see below).

Now, as testing only deals with observable events (including quiescence), we define a deterministic IOLTS S_{VIS} with same observable behavior as S^δ.
$S_{VIS} = (Q^{VIS}, A^{VIS}, \rightarrow_{VIS}, q_0^{VIS})$ where $Q^{VIS} \subseteq 2^{Q^S}$, $A^{VIS} = A_I^{VIS} \cup A_O^{VIS}$ with $A_O^{VIS} = A_O^S \cup \{\delta\}$ and $A_I^{VIS} = A_I^S$, $q_0^{VIS} = q_0^S$ **after**$_S$ ϵ, $\forall a \in A^{VIS}, \forall P, P' \in Q^{VIS}$, $P \stackrel{a}{\rightarrow}_{VIS} P' \iff P' = P$ **after**$_S$ a.

[3] A deadlock ($\neg(\exists a \in A^S, q \stackrel{a}{\rightarrow}_S)$) is a particular case of output quiescence.
[4] As we consider finite state IOLTS, a livelock is a loop of internal actions. A livelock in the specification is not necessarily an error as it may occur due to abstraction.

S^δ does not need to be built because transitions labelled with δ can be added directly to S_{VIS} during τ-closure. This is easy for deadlocks and output quiescence but involves the computation of strongly connected components (SCCs) of τ actions for livelocks (δ are only synthesized on SCC roots). This is done on-the-fly by a part of TGV called FERMDET [9, 8] which adapts Tarjan's algorithm [11].

Implementations. We assume (usual test hypothesis) that an implementation can be modelled by an IOLTS $Imp = (Q^{\text{Imp}}, A^{\text{Imp}}, \rightarrow_{\text{Imp}}, q_0^{\text{Imp}})$ with $A^{\text{Imp}} = A_{\text{I}}^{\text{Imp}} \cup A_{\text{O}}^{\text{Imp}} \cup I^{\text{Imp}}$ and $A_{\text{I}}^{\text{S}} \subseteq A_{\text{I}}^{\text{Imp}}$ and $A_{\text{O}}^{\text{S}} \subseteq A_{\text{O}}^{\text{Imp}}$. As usually, we assume that implementations can never refuse an input. We note $\overline{IOLTS}$ the set of input-complete IOLTS. For the definition of conformance, we also need to consider the suspension automaton Imp^δ of Imp.

Test cases. A test case is modelled by a deterministic IOLTS $TC = (Q^{\text{TC}}, A^{\text{TC}}, \rightarrow_{\text{TC}}, q_0^{\text{TC}})$ where $A^{\text{TC}} = A_{\text{I}}^{\text{TC}} \cup A_{\text{O}}^{\text{TC}}$ with $A_{\text{O}}^{\text{TC}} \subseteq A_{\text{I}}^{\text{S}}$ and $A_{\text{I}}^{\text{TC}} \subseteq A_{\text{O}}^{\text{Imp}}$. Two disjoint subsets of states $Pass \subseteq Q^{\text{TC}}$ and $Inconc \subseteq Q^{\text{TC}}$ and a state $Fail \in Q^{\text{TC}} (Fail \notin Pass \cup Inconc)$ are associated to TC. They correspond to arrival states of transitions carrying verdicts as in TTCN. We assume that a test case is complete for inputs in A_{I}^{TC} in each non controllable state (state where no output is possible). This is in general the case also for TTCN test cases with the special label $?otherwise$. We restrict ourselves to deterministic test cases without internal actions. This restriction could be avoided to deal with more general test cases including internal actions such as distributed tests. In this case, τ-reduction and determinization should be applied to test cases with FERMDET.

2.2 Conformance Testing

In order to speak about correctness of test cases, we need to define the conformance relation that the test cases are supposed to check. As in [8], we consider the **ioco** relation [13]. Note that it is in fact an extension of **ioco** as we consider livelocks. It says that conformant implementations are IOLTS that allow only outputs of the specification (including δ) after any trace of S^δ (also called suspension trace of S in [13]). It is defined as follows:

Definition 2 *Let Imp (implementation) and S (specification) be two IOLTS,*
Imp **ioco** $S \equiv \forall \sigma \in traces(S^\delta), out(Imp^\delta \textbf{ after } \sigma) \subseteq out(S^\delta \textbf{ after } \sigma)$.

3. VERIFICATION PRINCIPLE

Different properties can be checked on test cases. First some static properties can be checked such as syntactical correctness, existence of verdicts, input completeness, controllability, timer management. These properties can be

checked using test cases only. We are more interested in dynamic properties which involve the observable behavior of the specification. We do not pretend to check all properties but only some of them in particular those involving the specification. We could also check some properties involving the test purpose (are Pass verdicts correctly assigned) but this neccessitates also to formalize test purpose which are often very informal.

First, we tackle the problems of laxness and unsoundness. A test is lax if it accepts non conformant implementations which it could be able to reject. Almost conversely, a test is unsound if it rejects conformant implementations. Then in a second part, we deal with the problems of controllability.

This separation corresponds to a difference in algorithmic design. The first problems are solved by a forward traversal of state graphs, while some controllability conflicts are only corrigible by a backward traversal. Moreover, the problems of laxness and unsoundness are strongly dependent on the specification. This is not the case for controllability.

3.1 Test Case against Specification

In this part, we define the types of errors of a test case which are detectable by comparing the behavior of the specification with that of the test.

The concept of comparison leads us to define the synchronous product denoted by PS_{VTS} between a test case TC and the observable behavior of the specification S_{VIS}. Let $TC = (Q^{\text{TC}},\ A^{\text{TC}},\ \rightarrow_{\text{TC}},\ q_0^{\text{TC}})$ provided with two sets of states $Pass^{\text{TC}}$ and $Inconc^{\text{TC}}$ a Fail state and let $S_{\text{VIS}} = (Q^{\text{Svis}},\ A^{\text{Svis}},\ \rightarrow_{\text{Svis}},\ q_0^{\text{Svis}})$ be the τ-reduced and determinized specification.

Definition 3 *The synchronous product is an IOLTS* $PS_{VTS} = (Q^{VTS},\ A^{VTS},\ \rightarrow_{VTS},\ q_0^{VTS})$ *where*

- $A^{VTS} = A_o^{VTS} \cup A_i^{VTS}$, *with* $A_o^{VTS} = A_o^{TC} \cup A_i^{VIS}$, *outputs of the product are the outputs of the test case and the inputs of the specification;*
  $A_i^{VTS} = A_i^{TC} \cup A_o^{VIS}$, *inputs of the product are the inputs of the test case and the outputs of the specification,*
- $Q^{VTS} \subseteq (Q^{VIS} \cup \{\bot\}) \times (Q^{TC} \cup \{\bot\})$ *and* $\rightarrow_{VTS}$ *are the smallest sets defined by application of the following rules.*
  - $q_0^{VTS} = (q_0^{VIS},\ q_0^{TC})$
  - $$\frac{(q^{VIS},\ q^{TC}) \in Q^{VTS} \wedge q^{VIS} \xrightarrow{a}_{VIS} q'^{VIS} \wedge q^{TC} \xrightarrow{a}_{TC} q'^{TC}}{(q'^{VIS},\ q'^{TC}) \in Q^{VTS} \wedge (q^{VIS},\ q^{TC}) \xrightarrow{a}_{VTS} (q'^{VIS},\ q'^{TC})}$$
  - $$\frac{(q^{VIS},\ q^{TC}) \in Q^{VTS} \wedge q^{VIS} \not\xrightarrow{a}_{VIS} \wedge q^{TC} \xrightarrow{a}_{TC} q'^{TC},}{(\bot,\ q'^{TC}) \in Q^{VTS} \wedge (q^{VIS},\ q^{TC}) \xrightarrow{a}_{VTS} (\bot,\ q'^{TC})}$$
  - $$\frac{(q^{VIS},\ q^{TC}) \in Q^{VTS} \wedge q^{VIS} \xrightarrow{a}_{VIS} q'^{VIS} \wedge q^{TC} \not\xrightarrow{a}_{TC}}{(q'^{VIS},\ \bot) \in Q^{VTS} \wedge (q^{VIS},\ q^{TC}) \xrightarrow{a}_{VTS} (q'^{VIS},\ \bot)}$$

The two last rules say that the traces of the specification (resp. of the test) which do not exist in the test (resp. in the specification) end in particular states noted $\perp$ in the synchronous product.

Verification and correction of laxness. A test case is lax if it could reject a non-conformant implementation but does not. More precisely, TC is lax if there exists an implementation Imp which does not conform to S because after a trace σ it allows an output a that S does not, TC can perform the trace $\sigma.a$ but does not produce a *Fail* verdict. Formally:

Definition 4 *Let S be a specification and TC a test case. TC is lax w.r.t S for* **ioco** *iff* $\exists Imp \in \overline{IOLTS}, \exists \sigma \in traces(S^\delta) \cap traces(Imp^\delta) \cap traces(TC)$, $\exists a \in A_o^s$ *such that* $\sigma.a \in traces(TC)$, $a \in out(Imp^\delta$ **after** $\sigma) \wedge$ $a \notin out(S^\delta$ **after** $\sigma) \wedge TC$ **after** $\sigma.a \neq Fail$.

If we notice that Imp^δ is characterized by $\sigma.a$, it is easy to see that the existential quantification on Imp can be eliminated. Thus the laxness property can be reformulated while using only the traces of TC and S^δ, thus the product PS_{VTS}.

Proposition 1 *A test case is lax iff*
$\exists (q^{\text{VIS}}, q^{\text{TC}}) \in Q^{\text{VTS}}, a \in A_I^{\text{VTS}}, q'^{\text{TC}} \neq Fail \in Q^{\text{TC}} : (q^{\text{VIS}}, q^{\text{TC}}) \xrightarrow{a}_{\text{VTS}} (\perp, q'^{\text{TC}})$

We propose a correction which eliminates any laxness from a given test case. Each time an input of the test case TC not leading to *Fail* does not correspond to an output of the specification, this transition is replaced by a transition leading to *Fail* in the corrected test case TC'. We thus obtain the inclusion of the outputs of the specification in the inputs of the test in each state of the test where an input is possible and accessible by a trace of the specification. This is formalized by the following transformation rule:

$$\frac{(q^{\text{VIS}}, q^{\text{TC}}) \xrightarrow{a}_{\text{VTS}} (\perp, q'^{\text{TC}}) \wedge a \in A_I^{\text{VTS}} \wedge q'^{\text{TC}} \neq Fail}{q^{\text{TC}} \not\xrightarrow{a}_{\text{TC}} q'^{\text{TC}} \wedge q^{\text{TC}} \xrightarrow{a}_{\text{TC}} Fail}$$

Verification and correction of unsoundness. A test is sound if it rejects only non conformant implementations. Conversely, it is unsound if there exists a conformant implementation which can be rejected by the test case. Formally:

Definition 5 *Let S be a specification and TC a test case.*
TC is unsound w.r.t. S for **ioco** *iff*
$\exists Imp \in \overline{IOLTS}, Imp$ **ioco** $S \wedge \exists \sigma \in traces(S^\delta) \cap traces(Imp^\delta) \cap traces(TC)$, $\exists a \in A_O^s, TC$ **after** $\sigma.a = Fail$

Again, the existential quantification on Imp can be suppressed and unsoundness can be expressed on PS_{VTS}.

Proposition 2 *A test case TC is unsound iff*
$\exists (q^{VIS}, q^{TC}) \in Q^{VTS}, a \in A_I^{VTS}, q'^{VIS} \neq \perp \in Q^{VIS}: (q^{VIS}, q^{TC}) \xrightarrow{a}_{VTS} (q'^{VIS}, Fail)$

In this case, the correction consists in replacing the incorrect transition leading to *Fail* in TC with a new transition in the corrected test case TC' with same label and leading to a new state in the INCONCLUSIVE set. This correction is reflected by the following rule:

$$\frac{(q^{VIS}, q^{TC}) \xrightarrow{a}_{VTS} (q'^{VIS}, \perp) \wedge q'^{VIS} \neq \perp \wedge a \in A_I^{VTS} \wedge q'^{TC} \notin Q^{TC}}{q'^{TC} \in Q^{TC'} \wedge q'^{TC} \in Inconc^{TC'} \wedge q^{TC} \xrightarrow{a}_{TC'} q'^{TC}}$$

It is easy to see that corrections of laxness and unsoundness do not interfere (correction of laxness cannot produce unsoundness and vice versa). Correction of laxness replaces lax inputs by sound Fail verdicts while correction of unsoundness remove unsound Fail verdicts by unlax inputs leading to Inconc.

Verification and correction of controllability conflicts. Test cases should be controllable in the sense that they should never have the choice between an output and another output or input.

Definition 6 *A test case has a controllability conflict if:*
$\exists q^{TC} \in Q^{TC}, \exists a \in A_o^{TC}, \exists x \in A_I^{TC} \cup A_o^{TC} \backslash \{a\} : q^{TC} \xrightarrow{a}_{TC}$ *and* $q^{TC} \xrightarrow{x}_{TC}$

Detection of controllability conflict can be done by any forward search in the test case. But correction is more difficult in the general case where test cases have loops. While pruning a test case, accessibility to *Pass* states must be preserved. The solution is then to perform a search (breadth-first or depth first) on the reverse transition relation, to prune other transitions in case of conflict and to forget parts of the test case that become unreachable. This algorithm is detailed in [8] as it is also part of TGV.

Implementation in VTS. The algorithm takes as input a test case in Aldébaran format (general purpose graph format) and a specification described in LOTOS, SDL, BCG (compressed format for graphs) or Aldébaran. In the case of testing in context, the specification should include this context as conformance is defined for the specification in its context. It checks the correctness of the test case (laxness and unsoundness) with respect to the specification behavior. For SDL and LOTOS this behavior is given by simulators (respectively ObjectGéode [14] and the OPEN/CÆSAR interface [5]) which are driven by VTS. VTS implements these verifications by a breadth-first traversal of the synchronous product between the τ-reduced and determinized specification and the test case. This τ-reduction and determinization are performed on-the-fly by the FERMDET tool only on the common traces of the test case and the specification (in fact traces of PS_{VTS}). The traces of the test case not leading to *Fail* must be included in those of the specification. Thus for any trace of the test case, we check two aspects. On the one hand, if inputs of the test case are possible, the

algorithm checks the equality between these possible inputs not leading to *Fail* and the possible outputs of the specification after the same trace. In addition, if outputs are possible in the test case, then they should be possible inputs of the specification (the equality is not required in this case). The controllability conflicts are detected when a state of the test case has several possible outputs or an output and inputs. Correction is performed by a backward traversal.

4. APPLICATION TO THE ATM SSCOP TEST SUITE

We have decided to experiment VTS with a real case study. We chose the B-ISDN ATM Adaptation Layer-Service Specific Connection Oriented Protocol (SSCOP) from the ITU Q.2110 document [10]. It presents several advantages:

- this protocol has been studied for test generation with various tools such as Samstag [6], TVeda [3] and TestGen [2],
- we have a formal SDL specification, which has already been validated and used for automatic test generation [1];
- there is a complete conformance test suite, standardized by the ATM Forum, publicly available at http://www.atmforum.com

4.1 The SSCOP Protocol

The Service Specific Connection Oriented Protocol resides in the Service Specific Convergence Sublayer (SSCS) of the ATM Adaptation Layer (AAL) (see figure 1). SSCOP is used to transfer variable length Service Data Units (SDUs) between SSCOP users. SSCOP provides its service to a Service Specific Coordination Function (SSCF). The SSCF maps the service of SSCOP to the needs of the AAL user. SSCOP uses the service of the CPCS (Common Part Convergence Sublayer) and SAR protocols which provide an un-assured information transfer and a mechanism for detecting corruption of SSCOP Protocol Data Units (PDUs). One currently defined use of SSCOP is within the signaling AAL (SAAL).

SSCOP performs the following functions:

- Sequence integrity: this function preserves the order of SSCOP SDUs that were submitted for transfer by SSCOP.
- Error correction by selective retransmission: through a sequencing mechanism, the receiving SSCOP entity can detect missing SDUs. This function corrects sequence errors through retransmission.
- Flow control.
- Error reporting to layer management.

- Keep alive: this function verifies that the two peer SSCOP entities participating in a connection are remaining in a link connection established state even in the case of a prolonged absence of data transfer.
- Local data retrieval: this function allows the local SSCOP user to retrieve in-sequence SDUs which have not yet been released by the SSCOP entity.
- Connection control: this function performs the establishment, release, and re-synchronization of an SSCOP connection. It also allows the transmission of variable length user-to-user information without a guarantee of delivery.
- Transfer of user data: SSCOP supports both assured and un-assured data transfer.
- Protocol error detection and recovery.
- Status reporting.

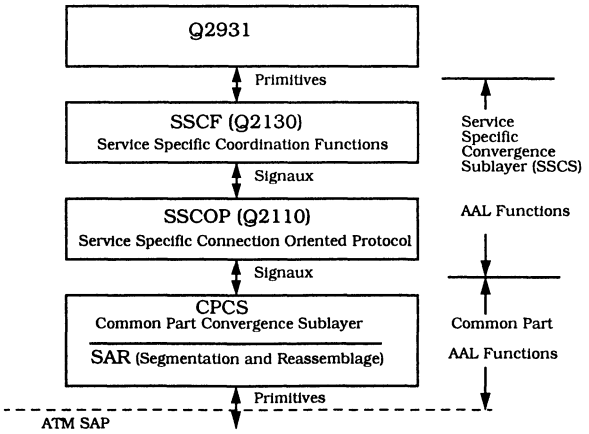

Figure 1. Situation of SSCOP in the ATM stack

The SDL executable specification of SSCOP was written by Serge Gauthier from CNET (the research center for France-Telecom). The specification was written in 1995 using SDL based on the SDL description given in the final draft document XI/Q2210 of ITU-T study group. It consists in approximately 5000 lines of textual SDL code. The specification was dedicated to test generation, thus it makes some simplifications which do not comply with all the aspects of the standard. Later on, the formal specification has been slightly corrected during the verification works of the FORMA project [1].

4.2 The ATM Test Suite

We considered the conformance abstract test suite for SSCOP, which was published by the ATM Forum Technical Committee on September 1996 under

the title "Conformance Abstract Test Suite for the SSCOP for UNI 3.1.". This test suite aligns with the principles defined in the OSI conformance testing methodology and framework ISO 9646 Parts 1-2 [7]. The test scripts are written in TTCN.

The testing architecture considered is the remote testing architecture (see figure 2) with only one lower tester (and PCO). The asynchronous communication on this PCO is reflected in the formal specification by the intercalation of a retransmission process between the SSCOP entity and the environment. This palliates the absence of queue between the SDL model and its environment in the ObjectGeode simulator.

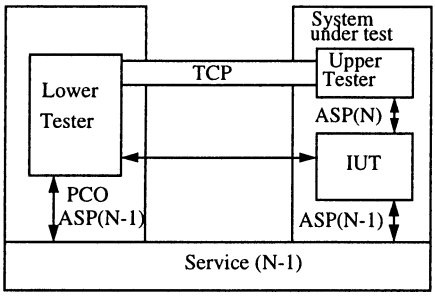

Figure 2. Remote testing architecture

While the test suite consists of 317 test cases (approximately 500 pages of TTCN), the test cases viable for verification based on our formal specification of the protocol were found to be 110 test cases. The abstraction resulted in the following:

- the test cases testing invalid PDUs (INV) were not considered as the specification does not describe the behavior of the SSCOP on reception of invalid PDUs. INV test cases arise to 186 cases.
- UD and MD PDUs were not considered in the SDL specification. Thus these PDUs have been abstracted in TTCN test cases. Moreover the 12 test cases related to valid UD and MD PDUs were not considered.
- the test cases testing the behavior of clocks and timeouts were abstracted as timers are treated as internal unobservable actions. Timer test cases arise to 9 cases.

Most of the test cases have the same structure: a preamble tries to drive the implementation under test (IUT) to a particular control state of the SSCOP entity as defined in the specification; then a test body is applied to check that the IUT behaves correctly; then follows a state identification behavior which checks the arrival control state; finally a postamble drives the IUT to the initial state.

4.3 Our Approach to Use VTS

We proceeded through the successive following steps.
1. The development of a compiler for the generation of test cases from the test suite. The input is a test suite written in TTCN machine processable (TTCN.MP) format. The output is a set of automata (one for each test case) in the Aldébaran format that the VTS program can process. The compiler also maintains a set of state variables similar to the set maintained by SSCOP. The values of the state variables are used to generate the values of the constraints that VTS can deal with. The compiler has to simulate the operations that are to be executed on the contents of the state variables of the SSCOP protocol mentioned by the test case. This simulation is essential for the sake of generation of PDUs and ASPs to be fired by the tester. It also simulates the changes in the state variables associated with firing transitions that are normally not explicitly mentioned in the test case. This is essential for the sake of generation of PDUs and ASPs expected from the specification.
2. The development of a program for the automatic generation of the supplementary files needed as inputs to the VTS program, namely the "feed" and the "hide" files. The feed file is used during the construction of the transition system of the specification as an input to the ObjectGeode API. The feed file contains the various signals that to be feed into the SSCOP specification from the external environment. The environment refers to both upper and lower layers. The hide file is used to hide the unobservable transitions that are generated but are not observable at the point of control and observation (PCO) due to the test architecture. Unobservable transitions in the test suite may be 1/ inputs or outputs of the queue of the retransmission process representing the asynchronous channel, 2/ signals between the user and the SSCOP specification that cannot be observed form the lower tester, and thus are not mentioned in the test script, 3/ actions on timers of the specification.
3. The use by VTS of the test cases generated by the compiler from the TTCN test suite and of the supplementary files in order to check and correct test cases.
4. The analysis of the errors found by the VTS tool and the variance between the hand written test cases of the ATM Forum test suite and the corrected test cases generated by VTS.
5. The proposal for corrections of the ATM test suite by providing alternatives to incorrect test cases with respect to the SSCOP specification.

4.4 Results

Most test cases failed because of forgotten signals due the classical problem of message crossing inherent to asynchronous communication. In fact these signals are observable due to the expiration of timers before the reception by SSCOP of a signal sent by the tester. One can imagine that those are not real

errors but are due to implicit assumptions on the transmission delay between the tester and the IUT. Nevertheless, as the remote testing architecture is considered this assumption should not be done or should at least be documented.

Out of the 110 test cases, 16 test cases failed for other reasons. They all fail in the state identification step: 2 tests in control state 4, 1 in control state 5, 4 in control state 7 and 9 in control state 10.

Out of the 16 defective tests, the following have been corrected:

1. 2 tests of state 4, 1 test of state 5, 2 tests of state 10 by adding a step to correct the value of the state counters. For example, in state 4, VR_SQ is not changed on receiving RS or ER PDUs. The S4_VERIFY test step fails if the last transition performs an RS or ER PDUs as VR_SQ is not incremented. Thus the BGN PDU of the test step is not detected as a retransmission. As VTS does not manipulate symbolic variables but only values, it does not provide a useful correction. Once the problem identified, it can nevertheless be easily corrected by hand by decrementing the value of VT_SQ in the test case before conducting the state verification step.

2. 4 tests of state 7 by adding a step to initialize the values of the state counters. This is detected by VTS as incorrect values of the $USTAT$ PDU in the S10_VERIFY step (state 10 is the arrival state of these test cases). The SSCOP specification resets its state variables after sending a $BGAK$ PDU at state 3. The initialization is not reflected in the test cases. This can be manually corrected by inserting the initialization procedure before state verification.

3. 4 tests of state 10 by introducing an alternative sequence for verification of state 10. S10_VERIFY assumes the reception of an $USTAT$ PDU before entering in the verification step. VTS has shown that there exist situations in which the SSCOP entity transfer directly to state 10 without the generation of an $USTAT$ PDU. The only solution is to change completely the S10_VERIFY procedure.

4. 2 tests of state 10 by changing the values of the USTAT PDU in the procedure for verification of state 10.

4.5 Example of Verification/Correction

We present here an example of the experimentation of the VTS tool on the test case numbered S10_V_P17 in the ATM Forum Test Suite. This test case and its following checking sequence (S10_VERIFY) are presented below in TTCN.GR format. They are exact copies of the ATM Forum test case and test step except that UD and MD PDUs have been abstracted. According to the informal test purpose, this test case verifies that the IUT, in control state 10 (DataTransferReady), saves an SD PDU that sequence number is between the sequence number of the next in sequence and the next highest expected SD PDUs.

S10_V_P17				
Nr	Label	Behaviour Description	Constraint Ref	Verdict
1		+S10_PREAMBLE		
2		LT_PCO!SD	SD_S_N_S(VT_S+2)	
3		START T_Wait		
4	LB1	LT_PCO?USTAT(VT_MS:= BIT_TO_INT(USTAT.N_MR))	USTAT_R_LIST(VT_S, VT_S+2, VT_S)	
5		LT_PCO!SD	SD_S_N_S (VT_S+1)	
6		LT_PCO!SD	SD_S_N_S (VT_S)	
7		LT_PCO!POLL(VT_PS:= INC_MOD_24 (VT_PS,1))	POLL_S_N_S(VT_S+3)	
8		START T_Wait		
9	LB2	LT_PCO?STAT[CHECK_N_PS (VT_PA, BIT_2_INT(STAT.N_PS), VT_PS)](VT_MS:=BIT_TO_INT (STAT.N_MR))	STAT_R_N_R(VT_S+3)	(P)
10		+S10_VERIFY		
11		+POSTAMBLE		
12		LT_PCO?POLL	POLL_R_GEN	
13		GOTO LB2		
14		+TS_Wait		
15		LT_PCO?POLL	POLL_R_GEN	
16		GOTO LB1		
17		+TS_Wait		

S10_VERIFY				
Nr	Label	Behaviour Description	Constraint Ref	Verdict
1		LT_PCO!SD	SD_S_N_S(VT_MS+3)	
2		START T_Wait		
3	LB1	LT_PCO?USTAT(VT_MS:= BIT_TO_INT(USTAT.N_MR))	USTAT_R_LIST(VT_S, VT_MS, VT_S)	(P)
4		LT_PCO?POLL	POLL_R_GEN	
4		GOTO LB1		
5		+TS_Wait		

The graph drawn on the left hand side of Figure 3 represents the test case (body) with its different test steps (preamble, checking sequence and postamble) in a graph format as processed by our TTCN compiler. For the sake of clarity we did not represent the transitions *?otherwise* producing *Inconclusive* verdicts in states 2 and 4 and *Fail* verdicts in states 6, 10, 12 and 15. In the SDL specification and the test suite we have set the parameter VT_MS to 20 and Max_CC to 1 in order to shorten the preambles. The variable VT_S is initially set to 0.

As for most test cases, the VTS tool detects unsoundness (forgotten inputs of END PDU in states 6, 10 and 12) because of a bad treatment of asynchronism. According to the specification, once in control state DataTransferReady an END PDU can be sent if TimerNoResponse expires. The implicit hypothesis made in this test is that the PDUs SD (lines 2 and 5 or transitions $5 \rightarrow 6$ and $11 \rightarrow 12$) and POLL (line 7, transition $8 \rightarrow 9$) sent by the tester are received before the timer expires.

Formally speaking, the test case is also lax as *?otherwise* in states 2 and 4 should produce a *Fail* verdict and not an *Inconclusive* verdict. In fact it is common practice to deliver only *Inconclusive* verdicts in preambles but the VTS tool does not make this distinction.

The test case is really incorrect in the S10_VERIFY step as the value of the first and last parameters of the USTAT PDU (line 3, transition $12 \rightarrow 13$) are not correct. The VTS tool found this error (which is both laxness and

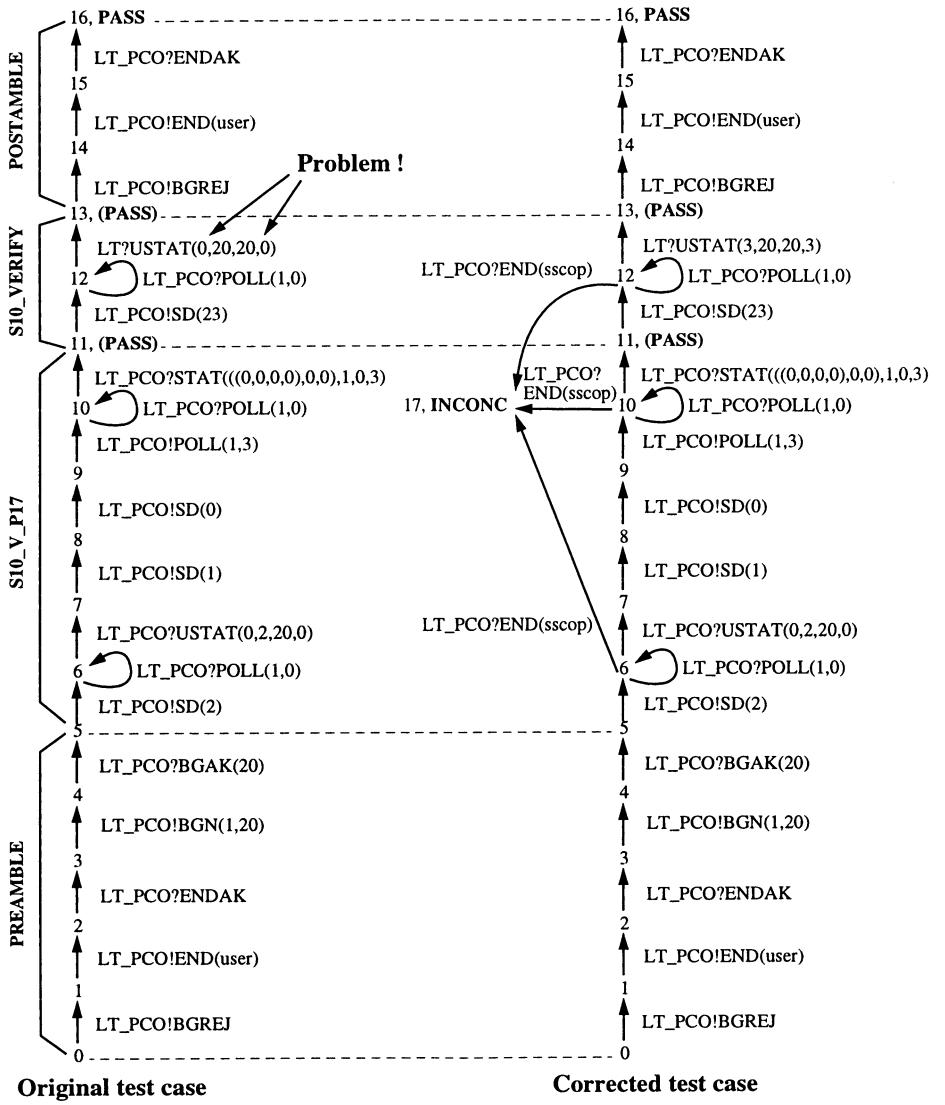

Figure 3. Initial and corrected test cases

unsoundness) and proposed to correct the test case. The right hand side of Figure 3 corresponds to the corrected test case produced by the VTS tool. New transitions lead to the *Inconclusive* state 17 for missing END PDUs and the transition labelled by USTAT is corrected by changing the first and last parameters values from 0 to 3.

This error was further analyzed by a close look to the TTCN test case. The real reason is that the variable VT_S of the test case is not updated when sending an SD PDU (lines 2, 5, 6 in S10_V_P17) but this variable is used in USTAT PDU (line 3 of S10_VERIFY). Thus it still has value 0 which is sent in USTAT while in the specification the corresponding variable is incremented and the resulting parameter has value 3.

As the VTS tool manipulates instantiated test cases, it was only possible to find the correct values. But finding that incrementations were missing was done by hand. Finding this kind of correction automatically is not possible in general. But some improvements can be made by a symbolic treatment of variables, the help of provers, abstractions and static analysis.

5. CONCLUSION AND PERSPECTIVES

VTS was originally conceived for testing the TGV tool. Test cases produced by TGV were checked by VTS in order to track bugs in the main algorithm of TGV. But the main interest of VTS is to check manual test cases. This was demonstrated here by its application on an industrial size specification and a significant part of a test suite in which some errors have been found.

The interest of such a tool is evident for complex systems. This is in particular the case for distributed systems because of the difficulty to foresee all behaviors of these systems due to asynchronism, hiding of internal actions and non-determinism. This leads directly to checking distributed test cases. VTS can be easily extended to check the correctness of this kind of test cases by using FERMDET as a front end. Nevertheless correction is more problematic as observed errors can be caused by internal actions.

But VTS suffers from the limitation inherent to enumerative tools. Parameters of specifications and test cases have to be fixed thus correctness cannot be guaranteed for all values of these parameters. Moreover, some of the errors detected by VTS on the SSCOP test suite are errors on values of message parameters. Analyzing the errors and correcting them would be easier with a symbolic treatment of data.

References

[1] M. Bozga, J.-C. Fernandez, L. Ghirvu, C. Jard, T. Jéron, A. Kerbrat, P. Morel, and L. Mounier. Verification and test generation for the SSCOP protocol. *Journal of Science of Computer Programming*, 36(1), 2000.

[2] A. Cavalli, B. Lee, and T. Macavei. Test generation for SSCOP-ATM networks protocol. In *SDL Forum, (INT, Evry)*. Elsevier, September 1997.

[3] I. Disenmayer, S. Gauthier, and L. Boullier. L'outil tveda dans une chaˆne de production de tests d'un protocole de télécommunication. In *CFIP'97 : Ingénierie des Protocoles*, pages 271–286. Hermes, September 1997.

[4] M. Dubuc, G. Bochmann, O. Bellal, and F. Saba. Translation from ttcn to lotos and the validation of test cases. Technical Report PUB 732, Université de Montréal, may 1990.

[5] Hubert Garavel. Open/cæsar: An open software architecture for verification, simulation, and testing. In *Proceedings of TACAS'98 (Lisbon, Portugal)*, volume 1384 of *LNCS*, pages 68–84, Berlin, March 1998. Springer Verlag.

[6] J. Grabowski, R. Scheurer, and D. Hogrefe. Applying SAMSTAG to the B-ISDN Protocol SSCOP. Technical Report A-97-01, part I, University of Lübeck, January 1997.

[7] OSI-Open Systems Interconnection, Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework. ISO/IEC International Standard 9646-1/2/3, 1992.

[8] Thierry Jéron and Pierre Morel. Test generation derived from model-checking. In Nicolas Halbwachs and Doron Peled, editors, *CAV'99, Trento, Italy*, pages 108–122. Springer, LNCS 1633, July 1999.

[9] P. Morel. *Une algorithmique efficace pour la génération automatique de tests de conformité*. PhD thesis, Rennes I Univ., France, February 2000.

[10] ITU Q.2110: B-ISDN ATM Adaptation Layer - Service Specifi c Connection Oriented Protocol (SSCOP), 1994.

[11] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal Computing*, 1(2):146–160, June 1972.

[12] Telelogic. The SDT TTCN Link Reference Manual, 1997.

[13] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.

[14] Verilog. ObjectGeode SDL Simulator Reference Manual, 1996.