Translation from TTCN to LOTOS and the validation of test cases

M. Dubuc, G. v. Bochmann, O. Bellal and F. Saba

Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, Succ. "A", Montréal, Québec, H3C 3J7, Canada

1. Introduction

In recent years, ISO and CCITT have developed various languages for the description of communication protocols and services [Boch 90g]. On the one hand, so-called formal description techniques (FDT's) [Este 89, Loto 89, SDL 87] are intended for writing formal specifications for the OSI protocols and services to be used during the protocol development process [Boch 87c]. On the other hand, semi-formal methods, such as ASN.1 [ASN1], are used in many existing OSI standards and proposals. Another language, TTCN [ISO C3], is used for the description of OSI conformance test suites.

We consider in this paper the validation of TTCN test suites in respect to the corresponding protocol specification. A typical test suite for an OSI protocol consists of a large number of test cases, each defining a certain number of execution paths, in the following called scenarios. Each scenario contains a so-called verdict which indicates whether the implementation under test (IUT) "passes" or "fails" the test in question. A third verdict called "inconclusive" is also possible which indicates that the observed behavior of the IUT satisfies the rules of the specification, but that the particular behavior to be tested could not be observed.

Although certain methods for automatically developing a test suite have been described (see for instance [Sari 89c]), most test suites are developed manually. It can therefore be expected that proposed test suites contain certain errors which should be detected as soon as possible. In particular, the verdicts of the test cases should be consistent with the protocol specification. This means that any scenario with verdict pass or inconclusive should correspond to a sequence of observed input and output interactions which is valid according to the protocol specification, and the sequence of interactions corresponding to a scenario with verdict "fail" should not be valid according to the specification. This is indicated in Figure 1 by the arrow (1).

It is difficult to directly compare a test case written in TTCN with a protocol specification written in another formalism. In order to automate such a comparison, it is necessary that first, the protocol be specified formally, and second, that the language used to specify the test case, e.g. TTCN, be comparable with the language used for the formal protocol specification.

In the case that the protocol specification is written in LOTOS, as assumed in this paper, we have to relate TTCN and LOTOS. Two approaches are shown in Figure 1: In one approach, the TTCN test case is first translated into an equivalent LOTOS specification (arrow (2)), which is then compared with the protocol specification (arrow (3)). This approach is described in this paper. Another approach would be the execution of the test case (arrow (4)) followed by the comparison of the test results with the protocol specification (arrow (5)). This latter approach is useful for conformance testing, especially, if non-standard test cases are used which do not contain any verdicts [Boch 89j]. However, since it only considers a particular test trace with particular interaction parameters at a time, this approach can not be used for a complete validation of the test case definition.

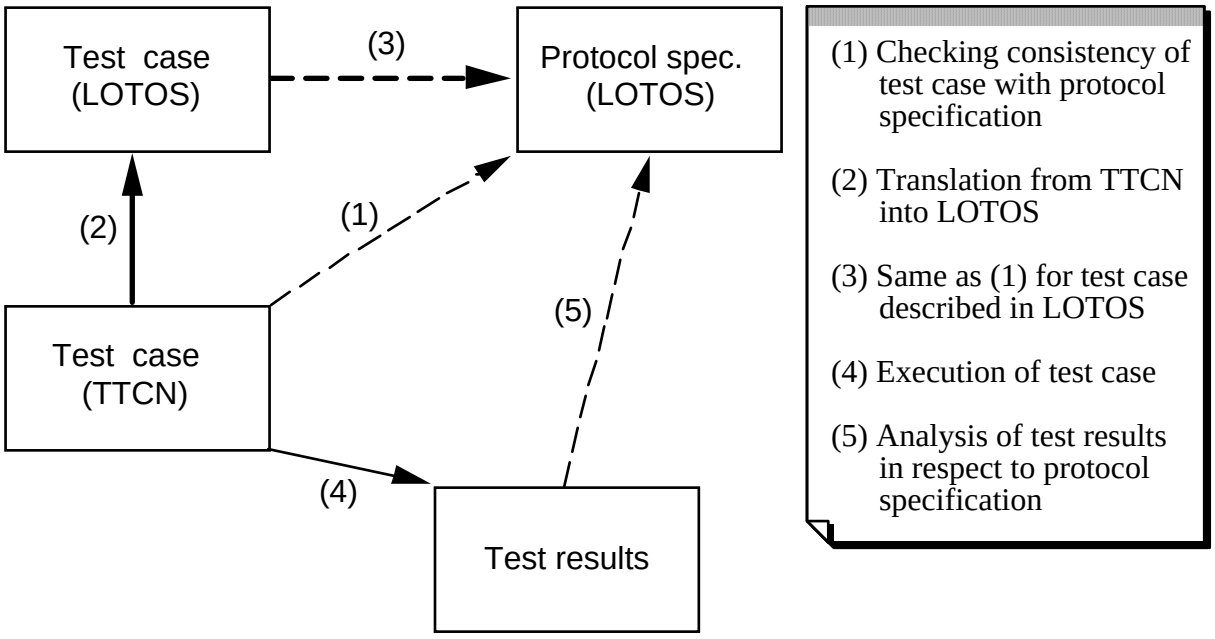

Figure 1. Validation steps

This paper describes an experiment with applying the first approach to a real OSI test suite, namely the LAP-B test suite for X.25 [ISO 8882]. This example was chosen since the test suite is relatively stable and a complete specification of the protocol in LOTOS [Guer 89a] was available. Section 2 describes our experience with the translation of the TTCN test suite into LOTOS. For checking the consistency of the test cases with the protocol specification (arrow (3) of Figure 1) we used an automated tool, called TETRA, which was originally developed for the analysis of test results in respect to a specification written in LOTOS, as described in [Boch 90h, Boch 89j]. It has been extended in order to be able to check the consistency of test cases with the protocol specification. Section 3 gives an overview of this tool.

The result of this experiment is threefold: (1) The translation of the test cases into LOTOS turned out to be relatively straightforward, as discussed in Section 2, (2) the TETRA tool was debugged and improved, and (3) a number of errors were detected, not only in the translated test cases in LOTOS, but also in the original TTCN test case definitions [ISO 8882] and the

formal protocol specification [Guer 89a], as discussed in Section 4. Some indications about the performance of the analysis method and tool is also given.

2. Methodology for translating TTCN into LOTOS

Our translation experience is limited to the ISO document [ISO 8882]. Eight different groups of test cases compose this document (see Table 1). We have translated five of these eight groups from TTCN to LOTOS (about two hundred fifty test cases), namely the disconnected phase, the link disconnection phase, the link set-up phase, the information transfer phase and the IUT busy condition. The remaining groups have a similar complexity.

Table 1 shows what we have accomplished to date in our translation/validation experiment. Note that the depth of a test case (i.e. maximum number of test steps in a branch) is a good measure of the complexity of the analysis. This depth is essentially proportional to the number of preambles that are executed before the test body. For example, the test cases of group DL5 have about the same complexity as the test cases of group DL6, and are more complex than those of the groups DL1 through DL4.

Group Name of TTCN Test Suite	Test Cases Refer to Preamble of Group(s)	Number of Test Cases	Number of Translated Test Cases	Number of Validated Test Cases
Disconnected (DL1)	DL1	37	37	10
Link Disconnection (DL2)	DL1 & DL2	58	58	2
Link Set-up (DL3)	DL1 & DL3	58	58	0
Information Transfer (DL4)	DL1 & DL4	59	59	2
Frame Reject condition (DL5)	DL1, DL4 & DL5	43	0	0
IUT Busy condition (DL6)	DL1, DL4 & DL6	49	49	0
Sent Reject condition (DL7)	DL1, DL4 & DL7	44	0	0
System Parameters and error recovery (DL8)	Same as those of groups DL1, DL2, DL3, DL4 or DL5	17	0	0

Table 1. Overview of LAP-B test cases

In this section, we explain the basic rules which we followed during the translation process. Our main goal is not to give a formal set of rules suitable for automatic translation, but to establish some informal schemes and show with concrete examples how the manual translation of test case can be performed. The reader may refer to Tables 7 to 10 which present two complete translation examples.

Most of our translation rules derive from an earlier proposals by Sarikaya and Gao [Sari 88f]. However, in certain cases we present different schemes more adapted to our goal of obtaining

LOTOS translations with a structure close to the original TTCN test cases. This simplifies the reading of the LOTOS test cases.

For expositional purposes, we divide the overall structure of a TTCN test case into four major parts: the test parameters, the global variables, the constraints and the dynamic behaviour. The test parameters and global variables are used in the dynamic behaviour. Global variables can also be used in the constraint part. The constraints are used in the dynamic behaviour. Each test step that acts on a PCO (Point of Control and Observation) may refer to a constraint. In the dynamic behaviour part, each test group has a specific preamble and postamble. Before the execution of the test body of a test case, the tester will execute the preamble which brings the IUT (Implementation Under Test) into the desired state (corresponding to the test group). When the test body is finished, the tester will execute the postamble corresponding to the expected final state of the IUT.

2.1 Test suite parameters (PICS, PIXIT)

Test parameters in the declaration part of a TTCN document can be translated into parameters of the LOTOS specification. In order to be able to refer to these parameters in the different processes of the specification, we must pass their values to the called processes. In the following example, the test suite parameters N1, N2 and k are used in the Test_Case process (Table 2).

```
process Test_Case[LowerTester](T1, N1, N2, k : Nat) : noexit :=
        Preamble[LowerTester](T1, N2) >>
        (* Test steps (some will refer to T1, N1, N2 and k) *)
        Postamble[LowerTester](T1) >>
        stop
endproc (* TestCase *)
```

Table 2. Test_Case process

2.2 Global variables

LOTOS is a functional language and does not support global variables directly. On the other hand, in TTCN, one can define and use global variables. In order to simulate this concept, we declare a recursive process which is able to store and retrieve values associated to a specific variable through the use of a gate. A set of gates synchronize this process with each test case that uses global variables.

An example of such a process representing the three variables V_S, V_R and PF_bit is shown in Table 3.

```
process Global_Variables[V_S, V_R, PF_bit](V_S, V_R : Nat, PF : Bit) : noexit :=
```

```
(
   V_S ! Read ! V_S;
   Global_Variables[V_S, V_R, PF_bit](V_S, V_R, PF)
[]
   V_S ! Write ? V_S : Nat;
   Global_Variables[V_S, V_R, PF_bit](V_S, V_R, PF)
[]
   V_R ! Read ! V_R;
   Global_Variables[V_S, V_R, PF_bit](V_S, V_R, PF)
[]
   V_R ! Write ? V_R : Nat;
   Global_Variables[V_S, V_R, PF_bit](V_S, V_R, PF)
[]
   PF_bit ! Read ! PF;
   Global_Variables[V_S, V_R, PF_bit](V_S, V_R, PF)
[]
   PF_bit ! Write ? PF : Bit;
   Global_Variables[V_S, V_R, PF_bit](V_S, V_R, PF)
)
endproc (* Global_Variables *)
```

Table 3. Global_variables process

Later on, in Section 2.4.2, we will see how this process is used in order to translate assignment clauses and boolean expressions.

The approach of [Sari 88f] is similar to ours except that it defines one process per global variable. Since in most cases all the global variables are used in each test case, our approach gives a shorter translation. Also, in Sarikaya's translation scheme, the initialization of each global variable requires a supplementary gate interaction, whereas in our scheme, they are initialized when the process Global_Variables is instantiated.

2.3 Constraints

In TTCN, the constraint declaration part describes the coding of the parameters of PDUs (Protocol Data Units) and ASPs (Abstract Service Primitives). Our LOTOS translation includes similar type definitions in which each TTCN constraint relates to a ACT ONE operator. Since all the coding is specified, we subsequently refer only to these operators in the test cases (no need of boolean clauses). Such a scheme allows us to automate the translation of the dynamic behaviour of the test cases by making the control part of the test cases independent of the type definitions used in the specification. The names of the constraints then serve as an interface between the constraint definitions and the dynamic behavior part of the test cases.

An example of such constraint definitions in LOTOS is represented in Table 4.

```
type Constraint is NaturalNumber, Boolean, ExtendedFrame, Frame, Packet,
                   Infor, Corrlength, Lesslength, Abortf, Mode
opns               DISC_S10, ... : -> ExtFrame
                   RR_S10, ... : Nat -> ExtFrame
                   I_S31, ... : Nat, Nat -> ExtFrame

eqns               forall N_S, N_R : Nat, I_Field : Packet,
                   ofsort ExtFrame

                   DISC_S10 = M(Make_DISC(Flagging, B, Ctl_UFrame(0), FCS, Flag),
                         NoInfoField, Correct, NotLess, NoAbort);
                   RR_S10(N_R) = M(Make_RR(Flag, B, Ctl_SFrame(0, N_R), FCS, Flag),
                         NoInfoField, Correct, NotLess, NoAbort);
                   I_S11(N_S, N_R) = M(Make_I(Flag, A, Ctl_IFrame(N_S, 1, N_R),
                         I_Field, FCS, Flag), InfoField, Correct, NotLess, NoAbort);
endtype (* Constraint *)
```

Table 4. Constraint type definition

2.4 Dynamic behaviour

2.4.1 Tree attachment

Each tree attachment of the form "+tree" is equivalent to a call to the LOTOS process "tree" followed by the enable operator (>>).

2.4.2 Assignment clauses and boolean expressions

Assignment clause and boolean expression make use of global variables. If we refer to our process Global_Variables (section 2.2), the gate interaction "V_S **!** Write **!** N_R" translates the TTCN assignment clause V_S := N_R. An interaction "V_S **!** Read **?** V_S : Nat" is introduced be used prior to the use of the value of variable V_S in an expression. Boolean expressions, when needed, are translated into LOTOS predicates.

2.4.3 Input (?) and output (!)

In TTCN, the event <PCO> **!** <ASP> means that <ASP> is sent through the PCO (output), while <PCO> **?** <ASP> means that the PCO waits for the reception of the proper <ASP> (input). In LOTOS, there is no direct representation of the input and output concepts. The symbols ! and ? are used to specify the exchange of values during an interaction at a gate. One can simulate input/output by synchronizing a "!" parameter with a "?" parameter in another process. However, synchronization can also occur between two "!" parameters or two "?" parameters. Therefore we may obtain some improper behaviour if we translate the TTCN

test step <PCO> **!** <ASP> into the LOTOS action <PCO> **!** <ASP> (as suggested in [Sari 88f]). Suppose that we have the process:

process Sample[A] : **noexit** :=
A **?** x : Nat; A **!** x; **stop**
endproc (* Sample *)

This process accepts the trace A **!** 0; A **?** x : Nat, but also accepts the traces A **!** 0; A **!** 0 and A **?** x : Nat; A **?** x : Nat.

In order to prevent such unwanted synchronization, we have to make sure that the input "channel" of a PCO is coupled to the output "channel" of the communicating PCO (and vice-versa). One could for instance translate <PCO> ! <ASP> into <PCO> ! Output ! <ASP> and <PCO> ? <ASP> into <PCO> ! Input ? <ASP> : ASPType, in order to avoid any confusion between input and output.

2.4.4 Alternatives

At a given level of indentation, TTCN alternatives define a number of possibilities of test steps that may take place. In the LOTOS translation, these different alternatives are separated by the choice operator "[]". The TTCN "?Otherwise" alternative indicates actions to be taken if another input is received. In the case of our specification, it can be translated with a predicate describing the complement of all the inputs of the current level of indentation. This is more practical than creating a type enumerating all other possible PDU's that could be received for the given ?Otherwise clause as suggested in [Sari 88f]

2.4.5 Labels

In order to translate the use of a label, we have to define a local process containing all the test steps from the definition of the label to the last test step refering to it. The first action of this process is the test step associated with the label. To jump to the label, we call this local process.

2.4.6 Repeat

A group of test steps repeated a certain number of times is translated into a local process within the main process of the test case. This process has a counter parameter which decrements before each recursive call. When it reaches 0, the recursion stops.

2.4.7 Verdicts

Verdict interpretation is not easily translated into LOTOS. One has to pass the verdict to the calling process by using **exit**(Verdict) and use the **accept** mechanism to interpret it afterwards. However, using such a method makes the translation rather complicated. As indicated in the Introduction, the verdicts PASS and INCONCLUSIVE are handled

identically. They are translated into **stop,** while the FAIL verdict is translated into the sequence of actions **fail; stop**. The **fail** gate is a reserved gate in TETRA which is used to identify branches that should not be accepted by the IUT.

2.4.8 Timers

Timers are rarely completely described in formal specifications. In most cases, the TIMEOUT event is the only event modeled. In TTCN, the definition of timers is quite complex. Four events (Start, Cancel, Suspend and Resume) and two pseudo-events (?Timeout and ?Elapse) may be involved.

To model the complete definition of timers as described in TTCN, one should declare a process Timer that would synchronize the test cases with the LOTOS specification. This sounds simple but it would make the validation a lot more difficult. The behavior tree would become enormous. Since the pseudo-events ?Timeout and ?Elapse are the only timer events found in the LAP-B test suite, we simply translate timer events into internal actions.

2.5 General structure of a LOTOS test case

Each TTCN test case is translated into a corresponding LOTOS process. This process communicates with its environment through gates as shown in Figure 2.

```
process Test_Case[L] : noexit :=
    hide V_S, V_R, PF_bit in
    (
        Preamble[L, V_S, V_R, PF_bit] >>
        (* Test body *)
        Postamble[L, V_S, V_R, PF_bit] >> stop
    )
    |[V_S, V_R, PF_bit]|
    Global_Variables[V_S, V_R, PF_bit](0 of Nat, 0 of Nat, 0 of Bit)
  endproc (* Test_Case *)
```

Table 5. Test Case synchronized with Global_Variables process

The syntax of the behaviour of a test case in LOTOS is shown in Table 5.

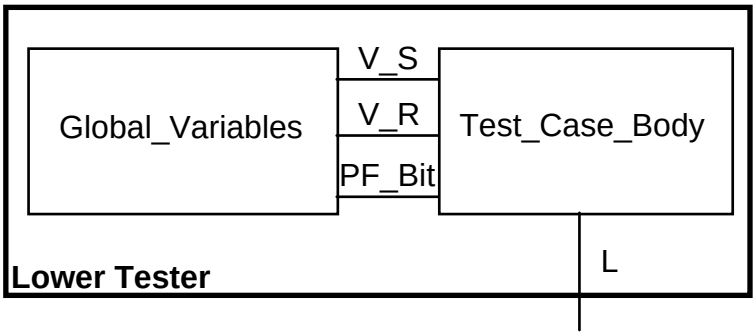

Figure 2. Test case processes synchronization

2.6 Translation experience

Following the basic rules exposed in the present section, we have found that the length and the structure of the LOTOS translation of test cases are close to its original form in TTCN. The translation is relatively obvious especially for the test body of the test cases which is relatively simple unlike its preamble and postamble which are generally more complex.

3. Overview of TETRA

We have modified an existing LOTOS interpreter [Logr 88] in such a way that it checks whether a given trace of observable interactions g_1, ..., g_n is valid in respect to a given reference specification S[g_1, ..., g_n]. The original interpreter requires interactions with the user for the selection of the next event to be interpreted, which may be an observable interaction at the gates g_1 through g_n, the internal event i, or an internal interaction at one of the gates hidden by the specification. Another version of the interpreter generates an overview of all possible sequences of observable interactions using backtracking over all possible internal events in case that several non-deterministic choices exist [Guim 89]. However, this latter version does not handle any interaction parameters. Our modified interpreter for test trace analysis, called TETRA, takes interaction parameters into account and uses backtracking to determine whether the tree of possible execution histories defined by the specification includes a history which gives rise to the trace T of observed interactions.

Backtracking is necessary for all those occasions where the reference specification allows for non-deterministic choices which are not directly visible. These choices relate to one of the following cases:

(a) Execution of internal events, i.e. event i or internal interactions on hidden gates, which appear as alternatives within a choice of subexpressions.

(b) Idem, when appearing within a **choice** statement relating to alternative gates.

(c) Selection of a data value associated with a **choice** statement.

(d) Expansion of recursive definitions in case of non well-guarded expressions, i.e. alternatives not beginning with a visible interaction.

In the cases (a) and (b), only a finite (usually small) number of alternatives are available, and they can be explored by the trace analysis interpreter, through backtracking, without excessive loss of efficiency. For case (c), however, the number of possible choices is sometimes not even bounded, as for instance in the case of the choice of a natural number. The trace analyzer has to determine whether a suitable choice of value exists which makes the given trace acceptable by the specification.This problem is in general undecidable.

TETRA also has an option for checking the consistency of a test case, written in LOTOS, with a reference specification. For this purpose, first the scenarios, or branches, of the test case are identified, and then each scenario is checked against the specification. In order to limit the number of scenarios in the case that the test case contains a loop, the loop is expanded only a limited number of times. Each scenario is checked like an interaction trace for conformance with the specification. The test case is consistent with the specification if each scenario with verdict "pass" or "inconclusive" conforms to the specification, and each scenario with verdict "fail" does not conform.

Since our initial experiments with TETRA [Boch 89j], the system has been improved in several respects (for more details, see [Boch 90h]. First, an on-line version has been built which analyses the trace of interactions one by one [Saba 90]. Second, certain options allow the processing of specifications with unlimited number of choices, as mentioned above, by arbitrarily limiting the depth of exploration of the tree of alternatives unless an observable interaction is involved.

In addition, the system has been modified in order to instantiate interaction parameters as late as possible. Input parameters of test case scenarios, as well as parameters of internal interactions, may not be defined directly. In this case, the analysis must be performed for all possible parameter values. The situation is similar for the variable representing the value selected by a **choice** statement. The improved system leaves such parameters or variables uninstantiated until their value can be determined from the constraints of the specification and/or subsequent observed input or output values. This avoids backtracking over all possible values, which would make the processing much more complex and inefficient.

Our experience with the analysis of the LAP-B test cases described in this paper and with the validation of a Transport protocol in respect to the OSI Transport service written in LOTOS [Saba 90] shows that the analysis with real-life specifications generates a large number of branches which are largely redundant or unfeasible. Therefore it is important to introduce certain optimizations to reduce the resulting inefficiencies (for more details, see [Boch 90h]).

Finally, we mention that TETRA provides error diagnostics in the case that an error has been found during trace analysis or test case consistency checking [Boch 89j]. During the optional error diagnostic phase, various fault hypothesis are checked for consistency with the given test trace (or scenario) and the specification. Each consistent hypothesis gives rise to a diagnostic message, which may be of the form "The second interaction is wrong and should be such and such", "The third interaction of the scenario should be absent", or "The verdict should be 'pass'". Each diagnostic message represents a possible interpretation for the reason of the problem.

4. Experimentation with TETRA

Recent optimizations allow us to use TETRA for the validation of test cases in respect to a complex specification. We are now able to validate test cases that have no more than seven test steps (i.e. external interactions) per scenario.

As shown in Figure 3, the test cases are executed under a simulated remote test architecture. We chose this architecture because the test cases of the ISO document were conceived to be executed within such an architecture. The test cases only describe the interactions with the lower tester (the upper tester is "empty"). The medium is a reliable full duplex queue.

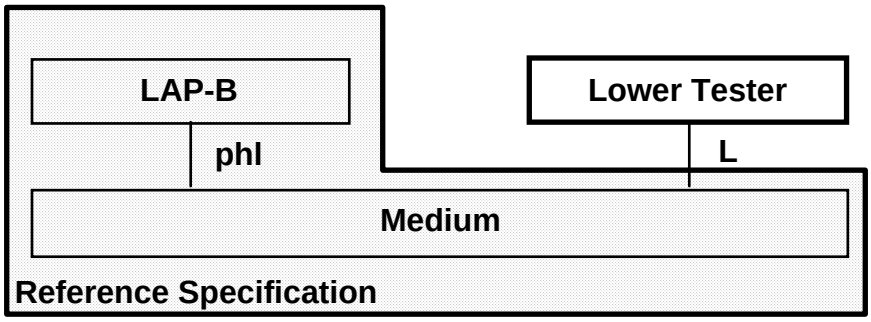

Figure 3. Remote Test Architecture

The complexity of the behaviour tree increases (dramatically) when we model the medium between the specification and the lower tester explicitly in the form of two FIFO queues in LOTOS. For simple test cases, the analysis time increases by a factor of up to ten. In more elaborated test cases, the analysis aborts due to a lack of memory. In order to obtain our results, we have bypassed the queue. Table 6 shows preliminary results for the validation time of some of the test cases. Note that TETRA is written in PROLOG and these results were obtained on a Sun 4/330 with 32 Mb of RAM. The following paragraphs discuss particular aspects of our results.

Test case	Number of branches validated	Maximum number of test steps in a branch	Validation time (in seconds)
DL1_101	8	7	522
DL1_207	9	7	567
DL2_101	38	9	3737

Table 6. Statistics on validation time

(a) Detected error in LAP-B test suite: We think that we have found an error in one of the test cases of the original TTCN document. Test case DL1_306 says that the trace:

```
L ! DISC (P:=1)
  L ? DM [F=1]
    L ! UA (F:=1)
      L ? DISC [P=1]
```

should have a fail verdict, but it is accepted by the specification. We found that this sequence of actions is valid with respect to the LAP-B standard. We believe that the last test step of the test case DL1_306 (L ?Otherwise) should have an inconclusive verdict instead of a fail verdict.

(b) Detected error in the specification: Test case DL1_207 indicates an error in the specification which does not include all details concerning error processing. The branch:

```
L ! DISC (P:=1)
  L ? DM [F=1]
    L ! Hex (string:='03F??'H)
      L ? Otherwise
```

has a fail verdict, but is accepted by the specification (string '03F??'H is a SABM/P=1 with an non-empty information field).

(c) Preambles for arbitrary initial states: We were surprised to see that TETRA rejected certain branches of preambles which are considered valid according to the test suite [ISO 8882]. For instance the branch:

```
L ! DISC [P:=1]
  L ? UA [F=1]
```

of the subtree DL1_STATE is not accepted by the LAP-B specification. Later we noticed that the ISO test cases do not necessarily assume that the IUT is initially in the disconnected state. For instance, the above branch is valid starting in the data transfer phase.

(d) Error diagnostics: As an option, TETRA provides diagnostics for locating the fault if an error is detected. While this facility works well for smaller specifications, we found that in our case the number of fault hypothesis (each indicated by a diagnostic message) was often too large to be useful. For instance, a fault in the first actions of a branch could lead to about hundred diagnostics messages.

5. Conclusions

Our work with the LAP-B protocol gives rise to conclusions concerning the translation from TTCN to LOTOS and concerning the use of TETRA. Concerning the translation of test cases into LOTOS, we found that in the case of the LAP-B test suite, the similar structure of all test cases largely simplified the translation process. In particular, the fact that the TTCN

constraints are referred to by name in the behavior description part leads to a clear separation between the part dependent on the data type definitions and the behavior part. The latter can be translated using the schemes described in Section 2, which could be easily automated. Further automation of the translation could be obtained in the case of OSI application layer protocols if an ASN.1-LOTOS translation scheme is adopted, such as described in [Boch 89h]. However, in order to keep consistency between the test cases and the protocol specification, the same ASN.1-LOTOS translation scheme must be applied in the development of the LOTOS protocol specification and for the translation of the test cases.

The structure of the translated test cases is relatively simple, at least compared with the complexity of a complete protocol specification. In fact, only few execution time related errors were found in the translated LOTOS test cases after they had been debugged through the syntax checks performed by the LOTOS compiler.

Our experience with TETRA shows that it is a practical tool for the validation of conformance test cases. Although it is still limited to not too complex test cases, it is possible to handle real-life protocol specifications which cover several thousand lines of LOTOS code. We hope that further optimizations of the analysis algorithm will lead to an improved tool which can handle all the test cases which are encountered in OSI conformance testing. The same optimizations are also useful for the application of TETRA to the analysis of test results, as discussed in [Boch 89j]. The diagnostics facility should also be improved.

The errors found during our experience indicate, as could be expected, that even well studied specifications still contain a few errors. This shows that the automatic checking of conformance test cases in respect to the corresponding protocol specification is a useful activity for increasing the confidence in the OSI specifications. It is important to note that the automation of this activity is only possible when a formal specification of the protocol is available. Unfortunately, at present, there are only few formal specifications of OSI protocols or services that have been generally recognized to faithfully represent the OSI standards.

Acknowledgements

The authors would like to thank L. Logrippo for helpful discussions in relation with the TTCN translation scheme, the development of TETRA, and the understanding of the LOTOS specification for LAP-B and D. Ouimet for technical support. This project was mainly funded under the IDACOM-NSERC-CWARC industrial research chair on communication protocols. Financial support from the Ministry for Education of Quebec is also gratefully acknowledged.

References

[ASN1] I. 8. ISO, Information Processing - Open systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).

[Boch 87c] G. v. Bochmann, Usage of protocol development tools: the results of a survey (invited paper), 7th IFIP Symposium on Protocol Specification, Testing and Verification, Zurich, May 1987, pp.139-161.

[Boch 89h] G. v. Bochmann and M. Deslauriers, Combining ASN.1 support with the LOTOS language, Proc. IFIP Symp. on Protocol Specification, Testing and Verification IX, North Holland Publ., 1989, pp. 176-186.

[Boch 89j] G. v. Bochmann and O. Bellal, Test result analysis in respect to formal specifications, Proc. 2nd Int. Workshop on Protocol Test Systems, Berlin, Oct. 1989.

[Boch 90g] G. v. Bochmann, Protocol specification for OSI, Computer Networks and ISDN Systems 18 (April 1990), pp. 167-184.

[Boch 90h] G. v. Bochmann, O. Bellal, F. Saba and M. Dubuc, Automatic test result analysis, in preparation.

[Este 89] ISO, IS9074 (1989), Estelle: A formal description technique based on an extended state transition model .

[Guer 89a] D. Gueraichi and L. Logrippo, Derivation of Test Cases for LAP-B from a LOTOS Specification. In: Vuong, S.T. (Ed.) Formal Description Techniques II. North-Holland, 1990, pp. 361-374.

[Guil 89] R. Guillemot and L. Logrippo, Derivation of useful execution trees from LOTOS by using an interpreter, In: K. J. Turner (Ed.) Formal Description Techniques, Proceedings FORTE '88, North Holland Publ., 1988, pp. 311-325.

[ISO 8882] ISO, X.25-DTE Data link layer conformance test suite, TC97/SC6, DIS8802 Part 2, 1989.

[Logr 88] L. Logrippo and al., An interpreter for LOTOS: A specification language for distributed systems, Software Practice and Experience, Vol. 18 (4), pp. 365-385, April 1988.

[Loto 89] ISO, IS8807 (1989), LOTOS: a formal description technique.

[OSI C3] ISO, DP 9646-3, Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN).

[Saba 90] F. Saba, Validation en ligne de traces d'exécution appliquée au protocole de Transport, M.Sc. thesis, Université de Montréal, fall 1990.

[Sari 88f] B. Sarikaya, Translation of test specifications in TTCN to Lotos, Proc. IFIP Symposium on Protocol Specification, Testing and Verification, Atlantic City, 1988.

[Sari 89c] B. Sarikaya, Conformance Testing: Architectures and Test Sequences, Computer Networks and ISDN Systems 17, 1989, pp. 111-126.

[SDL 87] S. X. CCITT, Recommendation Z.100, 1987.

BEHAVIOUR DESCRIPTION	LABEL	CONSTRAINT REFERENCES	VERDICT	COMMENTS
DL1_STATE				
L! DISC (P:=1)		DISC(S31)		§ 2.4.3
L? UA [F=1]	1	UA(S31)		§ 2.4.3, 2.4.5
L? DM [F=1]		DM(S31)		
L? DISC [P=1]		DISC(S11)		
L! UA (F:=1) -> 1		UA(S11)		
L? Otherwise -> 1				§ 2.4.4
?Elapse Td sec.			INCONC.	§ 2.4.7, 2.4.8

Table 7. First TTCN Test Case example

```
process DL1_STATE[L] : exit :=
   L ! Output ! DISC_S31; (* § 2.4.3 *)
   Label1[L, tm] >> exit

where
   process Label1[L] : exit := (* § 2.4.5 *)
      (
         L ! Input ? f : eframe [f = UA_S31];  (* § 2.4.3 *) exit
      [] (* § 2.4.4 *)
         L ! Input ? f : eframe [f = DM_S31]; exit
      []
         L ! Input ? f : eframe [f = DISC_S11];
         L ! Output ! UA_S11; (Label1[L, tm] >> exit) (* § 2.4.5 *)
      []
         L ! Input ? f : eframe [not(f == UA_S31) and (not(f == DM_S31) and
            not(f == DISC_S11))]; (* § 2.4.4 *)
         (Label1[L, tm] >> exit)
      []
         i; (* ?Elapse Td sec. *) (* § 2.4.8 *)
         exit (* Inconclusive *) (* § 2.4.7 *)
      )
   endproc (* Label1 *)
endproc (* DL1_STATE *)
```

Table 8. First TTCN Test Case translation

BEHAVIOUR DESCRIPTION	LABEL	CONSTRAINT REFERENCES	VERDICT	COMMENTS
DTE_SENDS_I_FRAMES(N,ACK)				
Repeat I_FRAME_EXCH(ACK) Step [Step=N]				§ 2.4.6
I_FRAME_EXCH(ACK)				
L! I (P:=0,N_S:=V_S,N_R:=V_R)		I(S30)		
(V_S:=V_S+1)				§ 2.4.2
L? I [P=0][N_S=V_R]		I(S10)		
[ACK](V_R:=V_R+1)				§ 2.4.2
L? Otherwise			FAIL	§ 2.4.7
?Elapse Td sec.			FAIL	§ 2.4.7

Table 9. Second TTCN Test Case example

```
process DTE_SENDS_I_FRAMES[L, V_S, V_R, fail](N : Nat, ACK : Nat) : exit :=
   (
      exit [N = 0 of Nat]
   []
      (
         I_FRAME_EXCH[L, V_S, V_R, fail](ACK) >>
         DTE_SENDS_I_FRAMES[L, V_S, V_R, fail](Pred(N), ACK) (* § 2.4.6 *) >>
         exit
      )
   )
where
   process I_FRAME_EXCH[L, V_S, V_R, fail] (ACK : Nat) : exit :=
         V_S ! Read ? V_S : Nat;
         V_R ! Read ? V_R : Nat;
         L ! Output ! I_S30(V_S, V_R);
         V_S ! Write ! Succ(V_S) mod 8; (* § 2.4.2 *)
         (
            L ! Input ? f : eframe [f = I_S10(V_R, ACK)];
            V_R ! Write ! Succ(V_R) mod 8; (* § 2.4.2 *) exit
         []
            L ! Input ? f : eframe [not(f == I_S10(V_R, ACK))];
            fail; stop (* § 2.4.7 *)
         []
            i; (* ?Elapse Td sec. *)
            fail; stop (* § 2.4.7 *)
         )
   endproc (* I_FRAME_EXCH *)
endproc (* DTE_SENDS_I_FRAMES *)
```

Table 10. Second TTCN Test Case translation