7

Principles for validation of abstract test suites specified in concurrent TTCN

M. Törő, K. Tarnay
KFKI Research Institute for Measurement and Computing Techniques
H-1525 Budapest, P.O.Box 49, HUNGARY,
Phone: (+36 1) 169 94 99, Fax: (+36 1) 160 12 90
E-mail: tmaria@sun60.mszki.kfki.hu, tarnay@sunserv.kfki.hu

Abstract

Concurrent TTCN defines the tester as a set of parallel test components. Similarly communication protocols described by specification languages are built up from system components running parallel. The combined test system that contains both the tester and the protocol is highly complex. Hence checking the correctness of conformance requirements specified in an abstract test suite is a great challenge.

The tester specification given in TTCN can be translated into SDL on the basis of the comparison of properties of SDL and concurrent TTCN. The combined test system specification can be validated.

The principles reflected in the paper are essentials for automatic validation of an abstract test suite against the SDL system specification.

Keywords

SDL-92, concurrent TTCN, abstract test suite, reference specification, validation

1 INTRODUCTION

Standardized specification languages allow one to describe communication systems of high complexity. These systems can have several interfaces that should be observed and controlled during conformance testing. To meet these requirements ISO has extended the Tree and Tabular Combined Notation (TTCN) with concurrency (ISO/IEC-9646, 1992; DA1 ISO/IEC-9646-3, 1993). Concurrent TTCN defines the tester as a set of test components in the same way that communication systems described by different specification languages are built up from several system components. In both cases components run parallel. This results in a highly complex combined test system. Moreover the basics of languages used to specify protocols and test suites are not the same. Manual validation of an abstract test suite (ATS) is impossible even for sequential TTCN because of the large number of test cases (Wuong, 1991; Bär, 1992), although it does not contain any non-determinism resulting from parallelism of test components. As a consequence, checking the correctness of conformance requirements specified in an ATS is a great challenge.

P. Dembiński et al. (eds.), *Protocol Specification, Testing and Verification XV*

In our paper we suggest a method that is based on the comparison of SDL-92 (Specification and Description Language) (Z.100, 1992; Belina, 1991) and concurrent TTCN. SDL was chosen because it is the specification language standardized and used by CCITT (now ITU-T); and TTCN is the only standardized notation for test suites used by organizations for standardization. The principles introduced can be used for automatic validation of an ATS against the SDL system specification. The validated system consists of the SDL reference specification of the protocol and the SDL specification of the tester translated from TTCN.

This paper is organized as follows: first a brief introduction of requirements for an ATS is given, then the basic features of SDL and TTCN are described. In the fourth section the main differences of the two languages are investigated more thoroughly, and suggestions are made to overcome them during the translation. In the fifth part the validation model and its derivation are presented. The sixth part of this paper describes how to use the validation model to prove the correctness of an ATS.

2 VALIDATION REQUIREMENTS FOR AN ATS

Requirements that an ATS should fulfil (Bär, 1992) can be grouped as follows:

1. Requirements of TTCN notation:
   - syntactic correctness;
   - semantic correctness.
2. Requirements against the relevant protocol standard:
   - static requirements;
   - dynamic requirements.
3. Requirements providing generation of test result.

2.1 Requirements of TTCN

The first group of requirements is given in the third part of the standard: ISO/IEC-9646 (1992) and in its extension (DA1 ISO/IEC-9646-3, 1993), where TTCN is defined.

Graphic TTCN is described informally in the standards. The grammar of machine processable TTCN is presented in Backus-Naur Form in Annex A of the standard; the operational semantics are given in Annex B in pseudo code. Although both static and operational semantics contain informal rules, they can be used to build a TTCN syntax and static semantics checker. Therefore these aspects will not be discussed further in this paper. Dynamic semantics of TTCN can be checked in runtime only.

2.2 Requirements of consistency with protocol specification

The second group of requirements checks the consistency of the protocol standard and the test suite (Bär, 1992). Protocol standards describe the **internal** structure and the behaviour of the system. Conformance testing in general is a black-box testing, hence an abstract test suite deals only with the **external** behaviour of the protocol. The consistency of these two aspects should be checked first statically then dynamically.

Static checking examines whether the protocol and the tester have the same set of interfaces, messages, variables, and parameters, etc. The protocol specification describes what interfaces an implementation has. The tester should use only these interfaces to control the test campaign and to observe the behaviour of the IUT (Implementation Under Test). Each interface of the protocol has a set of in-coming and out-going messages. From this aspect the tester should be complementary to the IUT, so it has to have the same set of in-coming messages as the set of out-going messages of the protocol, and vice versa. Parameters of messages of the tester and of the protocol should be the same type and range. So ought to be the observable variables.

During dynamic checking the dynamic part of the test suite is compared with the dynamic specification of the protocol. Test sequences described in the ATS should exist as traces of the protocol specification but only the observable part of them. On the other hand, ATS should cover the "important" part of the protocol, i.e. each conformance requirement defined by the standard should be checked. Standardization of MSC (Message Sequence Chart) language (Z.120, 1993) gives a possibility to formalize conformance requirements.

Another dynamic requirement is that every test sequence has to start from the "desirable stable testing state" of the protocol and return to it regardless whether the observed behaviour of the IUT is correct or not. In addition all reception events must be accepted and handled by the tester at any time of the test campaign. This means that the ATS should be complete for every reception. Conformance testing also checks the robustness of an IUT. Sequences aiming to test this aspect contain unexpected (not fair for a given state) or erroneous (with syntax error) messages sent to the IUT to examine whether it can return to the normal behaviour. A requirement could be formalized for this part of ATS: all messages should be sent to the IUT for all its states checked. However, no requirement was formalized for erroneous messages. Some hints can be gained from the system specification: each time an observable variable is checked at least one message with an out-of-range parameter should be sent.

2.3 Requirements ensuring resolution

To generate the test result two conditions must be fulfilled: the test suite should terminate, and a verdict should be assigned to each path of test cases of the ATS. This last group of requirements is also checked dynamically.

Termination is a protocol independent requirement. It holds if no unbounded loops are in the test suite. If the protocol allows unbounded loops the test designer should consider the situation and assign a verdict (*inconclusive*) after a certain number of repetitions. From the viewpoint of the test suite validation, *pass* and *inconclusive* verdicts cannot be distinguished. Both can be assigned to a test sequence derived from the protocol specification even for unexpected or erroneous messages. For all other cases that are present for the completeness of the ATS *fail* verdicts should be assigned.

3 CORE PART OF THE TWO FORMAL DESCRIPTION TECHNIQUES

3.1 SDL model of communication systems

SDL is based on the theory of extended finite-state machine (EFSM). A system specification in SDL is the definition of a set of EFSMs that communicate with each other and with the system environment by means of signals. Every EFSM, which is called process in SDL, has its own set of states, variables, and valid input signals. Processes are grouped together into blocks which compose the system. The system is surrounded by the environment with SDL-like behaviour. The behaviour of a process is given in the process body. It describes how the process of a given state reacts on a signal received, and what its next state will be. Actions taken by the process can be based on decisions. An action can be: value assignment, signal sending, timer management, creation of another process, or termination of the process itself. Processes of SDL run parallel. They are independent objects, and they influence each other's behaviour only by sending signals.

The communication is asynchronous in SDL. Communication paths are modelled as FIFO queues, one for each direction. A set of signals belongs to each queue that is conveyed by it. Paths connecting processes of the same block are called signal routes, and channels in any other case. Channels can delay signals non-deterministically whereas signal routes deliver the signal immediately. Every process has its own input queue to which signal routes leading to the process are connected.

A specification given in SDL describes **internal matters** (structure and behaviour) of a system. External characteristics should be derived from the behaviour of system components.

3.2 TTCN model of the tester with concurrent components

As was mentioned before, conformance testing is a black-box testing; this means that it manipulates only on observable objects. Therefore only observable and controllable features of the protocol are contained in an ATS. These are: interfaces used by the tester which are called points of control and observation (PCO), and messages (PDU - Protocol Data Units or ASP - Abstract Service Primitives) sent or received through them.

In concurrent TTCN, for every PCO on which concurrent events can happen a separate test component (TC) is described. The behaviour of TCs is given in separate test trees of a test case. Test trees are built-up from test events. An event can be message sending, message reception, or pseudo event. Test events can be grouped into test steps or defaults which have a role similar to the procedures of other languages. The configuration of TCs is given for each test case. Test components coordinate their actions by coordination messages (CM) sent to each other through coordination points (CP). Communications of both PCOs and CPs are asynchronous two-way communications, and modelled as two FIFO queues, one for each direction. One of the test components is special: the main test component (MTC) that creates parallel test components (PTC), manages CPs between itself and PTCs, and evaluates the test verdict based on the preliminary results of PTCs. When MTC terminates, all the PTCs still running are also stopped. However, a PTC can terminate earlier and assign a value to its preliminary result variable.

From this sketchy presentation of models used by SDL and by TTCN it is obvious how complicated a test system is that includes both the IUT and the tester, and how difficult and laborious it is to define an ATS, or even to check its correctness. The main idea of our method is that of transforming the TTCN specification of the ATS into an SDL description. Once this has been done the reference protocol specification is extended with the description of the tester and they are then validated together. However, certain points of the translation are not so clear as they seem to be. Some of the most problematic questions are investigated in the next section.

4 STRESSED POINTS OF THE TRANSLATION

4.1 Several PCOs for one input queue

An important difference between TTCN and SDL models is the following:

In SDL a process has only **one** input queue. All signal routes leading to the process and the timers put signals into this queue in their arrival sequence. The process consumes signals in this order. In general, the behaviour of the process is determined by the next signal in the queue. However, a signal can be saved in the input queue. The path by which the signal was delivered can be known only indirectly if the different paths have disjoint signal lists. Otherwise the sender of the signal will be known only after its consumption.

A TTCN test component has several input queues: one for each PCO or CP and one for the timers. The behaviour of the tester depends on the subsequent event prescribed by the ATS. On reception the tester will wait for the appropriate signal of the relevant queue if it is empty, or it checks input queues in the order specified in the test case. Queues are given by PCO and CP names. This difference between the SDL and TTCN should be solved during the translation of the tester.

To control the order of reception of signals in SDL in the way that TTCN does, save construct should be extensively used. At the translation of a test case into SDL all in-coming signals observed on PCOs or CPs, which are not mentioned (i.e. PCOs and CPs) at that reception level, have to be saved. However, this can be done only if PCOs and CPs can be distinguished

by signal names. If it cannot be done for each PCO and CP, signals should be renamed by attaching the PCO and CP names when the message names are translated.

4.2 Identification

In TTCN, test components are identified by their names whereas in SDL, processes have both names and identifiers, but only identifiers are unique. The problem with the process identifier is that it is generated by the SDL machine dynamically when the process is created. After creation the identifier is known by the creator (*offspring*) and by the process itself (*self*). On the other hand, a process knows the identifier of its creator (*parent*). Also each signal delivers the identifier of its originator process (*sender*). The variables *self* and *parent* are constant for the whole life of the process, but *offspring* and *sender* change from time to time when a new process is created or a new signal is received by the process, respectively.

In case of parallel test components they are all created by the main test component so it knows all identifiers and can store them. To make accessible these values for the whole system they can be viewed, exported, or sent to the particular process that needs it. The best way is directly exporting the values each time a change occurs, or indirectly though remote procedure call if operations should be done on these values. However, the problem of identification will probably not arise because SDL-92 allows addressing by communication paths that is appropriate for TTCN PCOs and CPs. The problem of the identification at reception should be resolved before signal consumption. Its solution by renaming signals was shown in the previous section (4.1).

4.3 Timer management

Regarding timers, the SDL model cannot be equivalent to the TTCN model because: SDL does not support declaration of time units and reading timers, both are applicable to TTCN. A timer of an SDL process puts its signal into the input queue of the process, whereas timers have own queue in TTCN. Finally, in TTCN all TCs have their own full set of timers declared in the ATS, however for an SDL process only its own timers are available.

Time units cannot be translated into SDL. If in the ATS a readtimer operation is required it should be considered and translated, for example, such as starting two or more timers of the same type with critical settings. By giving priority to timeout signals they will be handled by the process virtually as signals of another input queue.

TTCN allows one to start a timer by the MTC then to make fresh copies of it for PTCs created subsequently. In SDL this could be overcome by sending the list of active timers to the PTC that also starts them. However, this is only a rough approach because in SDL only the timer itself knows the remaining timing period. More sophisticated approach is to create a separate process which will handle all timers required and will distribute timeout signals.

4.4 Translation of constraints

The concrete parameters of messages at send and receive events are given by constraints in TTCN. Constraint declarations are listed in the Constraints Part of the ATS and they are referred to in test cases by names at send or receive events. There are different constructions used in the constraint part of TTCN. These are: flat, structured, chaining, parametrized, and modified. In constraints for receive events, different matching mechanisms are also defined.

Types of constraint

Flat constraints
In this case concrete values of parameters are given in the appropriate constraint declaration. For send events, flat constraints are translated as value assignments for each field of the message before the output of the signal that carries these values.

Example:

PDU Constraint Declaration		
Constraint Name: TCON_Class4_1 PDU Type: T_CONNECT1 Derivation Path: Comment:		
Field Name	Field Value	Comments
Source Destination T_Class UserData	TS_Par1 TS_Par2 4 "testing, testing"	

This TTCN constraint is translated as:

```
TASK  Source := TS_Par1,
      Destination := TS_Par2,
      T_Class := 4,
      UserData := 'testing, testing';
OUTPUT T_CONNECT1(Source, Destination, T_Class, UserData);
```

For receive events a constraint defines values which should be carried by the signal consumed. Examination of values is translated as decisions after the signal input:

```
INPUT T_CONNECT1(Source, Destination, T_Class, UserData);
DECISION ((Source = TS_Par1)
          AND (Destination = TS_Par2)
          AND (T_Class = 4)
          AND (UserData = 'testing, testing')
          );
      (TRUE): ...; /* continuation according to the test case */
      ELSE:  ...; /* join to the otherwise statement, or error message */
ENDDECISION;
```

Structured constraints, chaining of constraints and modified constraints

Structured constraints give the concrete values in different tables referred to by their names. During translation the referred part is inserted first then the constraint is handled as a flat constraint.

Chaining is very similar to structuring except that instead of a structured type constraint declaration, a PDU constraint declaration is referred to. This situation is handled in the same way as structured constraints.

In case of modification, those fields not mentioned in the modification are inserted from the modified constraint.

Parametrized constraints

For parametrized constraints formal parameters are given in the constraint declaration of the ATS, and actual parameters are given in the reference of the test case. Translation is executed in two steps: firstly, values of actual parameters are assigned to the formal parameters; secondly, the value assignments (send event) or checkings (receive event) are translated - as was shown for flat constraints.

Example:

PDU Constraint Declaration		
Constraint Name: TCON_1(class:INTEGER) PDU Type: T_CONNECT1 Derivation Path: Comment:		
Field Name	Field Value	Comments
Source Destination T_Class UserData	'1000'B ? class -	

The reference to this constraint may, for example, be: TCON_1(4). This is translated for the send event as:

```
TASK  class := 4;
TASK  Source := '1000',
      T_Class := class;
OUTPUT T_CONNECT1(Source, Destination, T_Class, );
```

and for the receive event as:

```
TASK class := 4;
INPUT T_CONNECT1(Source, Destination, T_Class, );
DECISION ((Source = '1000')
        AND (T_Class = class)
        );
      (TRUE): ...; /* continuation according to the test case */
      ELSE: ...; /* join to the otherwise statement, or error message */
ENDDECISION;
```

Matching mechanisms

AnyValue (?) and AnyOrOmit (*)

These wildcards are not translated, see the last example where "?" is used for the Destination field. For a send event this field should receive its value explicitly in the test case, or an error was found. For a reception this parameter is received if it exists, but it is not checked explicitly in a decision. It is checked only by the SDL machine dynamically if the value is appropriate for its type declaration.

Omit (-)

The UserData field is deleted in the last example. According to this it is not received, nor checked, i.e. it is not translated.

ValueList and Complement

In a value list acceptable values of a given type are listed. In contrast, after the keyword COMPLEMENT, values are listed which should not be received in the given field. Both are translated as a decision but the non-equality is checked in the second case.

Class COMPLEMENT(3, 4) is translated as:

```
DECISION ((Class /= 3) AND (Class /= 4)
        );
```

```
...
ENDDECISION;
```

Range

A *Range* is translated only for parameters of integer type as a range condition in the question part of the decision. The keyword INFINITY results in an open range, this part is not translated. Translation of conditions into the question part of the decision was chosen because in this case constraints related to other parameters can be attached to the question part.

```
   Class (0..4) and Sourece '1000'B are translated as:
DECISION ((0 <= Class) AND (Class <= 4)
         AND (Source = '1000')
         );
...
ENDDECISION;
```

SuperSet and SubSet

Parameters of type SET OF are declared in SDL through generator *powerset*. Hence *SuperSet* and *SubSet* are handled in a decision by the appropriate operators defined for the generator.

AnyOne (?) and AnyOrNone (*)

Both constraints can be used only for values of string, SEQUENCE OF and SET OF types. For all types, different operators were defined to handle *AnyOne*. These operators compare the received value with the constraint. The length of the variable that receives the value of the parameter is calculated from the constraint. In case of *AnyOrNone* only variables receiving the parameter are translated. Their values are checked dynamically by the SDL machine.

Permutation

Permutation is used for values of SEQUENCE OF type that is translated by using the *array* generator of SDL. For the permutated part an operator is defined that generates a set from the received parameters and checks its equality with the set derived from the constraint.

Length

Constraints on the length affect only the variable declaration part, where variables receiving values of signal parameters are declared according to the constraints. Afterwards, the SDL machine checks the length automatically.

IfPresent

For optional parameters of this kind, first their presence is checked in a decision in order not to receive dynamic error when their value is examined. To check their presence an operator was defined.

Good summaries on the use of ASN.1 with SDL are to be found in (prETS 300 414, 1994; Fischer, 1993); hints are given for the translation, but in future the translation of ASN.1 declarations into SDL can be omitted (Z.105, 1995).

5 VALIDATION MODEL

The combined test system consists of two parts: the protocol reference specification (PRS), and the tester specification derived from the ATS. Therefore the test system given in SDL at system level is built-up from two blocks. The PRS block as a block substructure contains the protocol specification.

The tester block will vary from test case to test case as long as concurrent TTCN declares a separate configuration of test components for every test case. This also means that only one test case can be validated at a time. However, all possible configurations of test components used during the test campaign, as well as TCs themselves, and the PCOs and CPs used by them are declared in the Declaration Part of the test suite. They can be examined during checking static requirements.

5.1 Steps of checking static requirements

The static requirements are checked during the translation of appropriate tables of the declaration part of ATS. Checking is executed as follows:

1. Establishing the relationship between PCOs and interfaces of PRS (channels of PRS leading to the environment);

   The introduction of non-delaying channels into SDL-92 gives a possibility to decide whether a PCO should be modelled as a non-delaying or delaying channel. For example, in different test methods the upper tester can be located within the test system or within the system under test. For the first case the delaying channel can be chosen, for the second one the non-delaying channel.
2. Comparison between:
   - signal lists of appropriate channels and PDU or ASP declarations of PCOs;
   - parameter types of signals and types of fields in declarations of the ATS;
   - ranges of parameter types and ranges in declarations;

   The basic criteria in comparisons are that the tester should accept any message sent by the protocol therefore all of them should be declared in the declaration part of the ATS (see matching mechanisms). On the other hand, the tester sends erroneous messages that are also declared in the ATS to check the robustness of the IUT. For them the relationship between signals declared in the protocol and messages declared in the ATS cannot be generalized and should be established manually.

   First comparison also determines whether signal renaming is necessary, i.e. two or more PCOs convey common signals. In this case processes extending the signal name with the PCO name are applied, as was described in 4.1.
3. Comparison of the external synonyms of the protocol specification and the test suite parameters;

   External synonyms allow parametrization of SDL systems. Test suite parameters have the same role in an ATS. For others: e.g. test suite, test case, and test component variables, constants, and parameters, no relationship can be established automatically. They are simply exported from the TTCN description into the SDL specification of the tester.

   Test suite parameters (or external synonyms) should also appear in documents PICS (Protocol Implementation Conformance Statement) and PIXIT (Protocol Implementation eXtra Information for Testing) to evaluate test selection expressions.
4. Setting of external synonyms of the system according to the test selection expression of a given test case;

   The last step should be executed after choosing a special test case because it is applicable only to the protocol of a given configuration, or option.
5. CPs and CMs are not linked with the protocol specification therefore they are simply translated from TTCN as signal routes between different test components of a given test case with appropriate signal lists. For CMs signal declarations are also generated. Signal names are compositions of the CP name and the message name.

5.2 Translation of the dynamic part of ATS into SDL

Introduction of object-orientation into SDL-92 allows one to generalize the main features of test components described by the standard and to put them together into a package. These generalized types will be specialized afterwards in accordance with the given test case of the ATS.

The generalized Tester block type consists of the MTC(1,1) process type that will be instantiated at system start, and the PTC(0,) process type that will be created by the MTC on the basis of its behaviour description. PTCs can be specialized as upper or as lower testers with appropriate gate constraints according to the test method used. Also during the translation the maximum number of PTCs, a list of their formal parameters, and the behaviour of all components will be derived.

All process types have a virtualized transition for *state* * and *input* * that generates test case error message if the test case is not complete for a reception. For both TC types a result variable is declared whose type is enumerated with the values *pass*, *inconclusive*, *fail*, and *none*, and which has the initial value *none*. The MTC process type also has a finalized transition for *state* * that evaluates the test result from preliminary test results sent by PTCs. Its algorithm is given in the amendment to the standard (DA1 ISO/IEC-9646-3, 1993).

We use static tree expansion for tree attachments and defaults before starting the translation of test cases into SDL. The translation of the behaviour results in a specialized set of processes of process types above. The tree name is used as the process name.

5.3 Rules for the translation of test trees

As long as the EFSM model is used in SDL, states must be introduced into the tester description:

- Before each level (nodes of reception events) of the TTCN tree a new state is inserted;
- Starting send, PTC creation and pseudo event(s) of the tree will compose the initial transition of the TC;

State transitions of the SDL description are derived as follows:

- Receive events are translated as *input*s with parameters:
  1. Firstly the reception is translated as an *input*;
  2. Then qualifiers independent of the reception are translated as *enabling condition* for the input;
  3. Qualifiers depending on the reception are translated as a *decision* after the input;
  4. Values of parameters received are checked against values given in relevant constraints, this is also translated as a *decision*;
  5. Value assignments are translated as *task*s after the input;
  6. All in-coming signals observed on PCOs or CPs that are not mentioned at that given level have to be saved.

An event line of TTCN is translated into the true branches of decisions as was shown in Section 4.4. False branches of these decisions should be derived from alternative event lines of given level and given message type, or they should join to the transition generated from the *OTHERWISE* statement if it exists. If no such transition exists that can be a test case error, there is no completeness with regard to signal parameters or predicates, and the else branch will generate error message.

The *OTHERWISE* statement is translated as an *input* with a signal list composed of signals conveyed by the relevant channel (PCO) and it has not been mentioned explicitly among the alternatives. If *OTHERWISE* refers to a CP that is a test case error, the translation will stop.

The receive event line

```
L ? N_DATAind (apdu:=N_DATAind.UserData)   NDind(PDU1)
```

is translated as:

```
STATE S;
INPUT L_NDATAind (Control, UserData);
                /* the PCO name attached to the original ASP name */
        DECISION ((Control = Control_constraint)
                AND (UserData = PDU1_constraint)
                );
              (TRUE): TASK apdu := UserData;
                      NEXTSTATE Sn;
              ELSE: JOIN Otherwise;
        ENDDECISION;
...
Otherwise: ... /* translated from OTHERWISE statement */
ENDSTATE S;
```

If the receive event above has an alternative of the same message type, it will be translated as a decision in the else part of the previous:

```
INPUT L_NDATAind (Control, UserData);
DECISION ((Control = Control_constraint)
          AND (UserData = PDU1_constraint)
          );
        (TRUE): TASK apdu := UserData;
          NEXTSTATE S2;
        ELSE:  DECISION ((Control = Control_constraint)
                         AND (UserData = PDU2_constraint)
                         );
                      (TRUE): TASK apdu := UserData;
                         NEXTSTATE S3;
                      ELSE: JOIN Otherwise;
               ENDDECISION;
ENDDECISION;
```

If the message type of the alternative event differs, it will compose another input of the given state.

- Send events are translated as *output*s with parameters as follows:
  1. If the send event has a qualifier, it is translated first as a *decision*;
  2. Then parameter values for the *output* are obtained from the constraint part of the ATS except for parameters that have explicit assignments in the event line; both cases are translated as value assignments;
  3. Finally the send event is translated as an *output* with a *via* path through a PCO or a CP given in the prefix of the send event. This way is applicable because of the relaxation of addressing in SDL-92.

The send event line

```
S_CP1 ! CM [cep = cep1]              CM1(apdu(cep1))
```

is translated as:

```
DECISION (cep = cep1);
        (TRUE):   TASK  apdu!cep := cep1,
                        apdu!etc := etc,/* as declared in the constraint */
                  ...
                  OUTPUT CM(apdu) via S_CP1;
        ELSE: JOIN Next;
ENDDECISION;
Next: ... /* continuation of the test case */
```

- Translation of pseudo events:
  1. Qualifiers are translated as *decisions* with true and false branches. If no alternative (false) branch exists that is a test case error, the translation will stop;

     [X=2] is translated as:

```
DECISION (X=2);
      (TRUE): ... /* subsequent events */
      ELSE: JOIN Alternative;
ENDDECISION;
Alternative: ...
```

  2. Assignments are translated as *tasks*;

     (X:=2) is translated as TASK X:=2;

  3. Timer management (see 4.3) is translated as *set*ting (start) or *reset*ting (cancel) of timers. Before resetting the state of the timer is checked and a warning message is generated if it is passive;

     START T0 (V2) is translated as SET (V2, T0);
     CANCEL T0 is translated as:

```
DECISION ACTIVE(T0);
      (TRUE): RESET (T0);
         JOIN Next;
      ELSE: /* Warning */
         JOIN Next;
ENDDECISION;
Next:...
```

- Labels and goto of TTCN have similar mechanisms (labels and join) in SDL;
- For the MTC, the creation of a PTC is translated as a process creation of the given type (tree name) while all test case variables are passed as formal parameters. The PTC identifier is used as a variable to keep the address of the process created, it receives the *offspring* value after creation of the PTC instance;
- For PTCs, *pass* and *inconclusive* verdict assignments are translated as sending to the MTC (the *parent* process) a signal with the preliminary result;
- For the MTC, *pass* and *inconclusive* verdict assignment is translated as the sending of a report message;
- For all TCs, *fail* verdict assignments are translated as sending test case error messages - as long as the reference protocol specification is considered to be correct and cannot make errors;
- For all leaf nodes the termination of the process is added.

In many cases it is reasonable to add an extra timer to the tester that guards situations when the protocol does not react on a test event. Handling of these conditions is not compulsory in TTCN although the absence of such kind of timers can cause a deadlock during testing. We control these circumstances on the level of the validation algorithm rather than change the tester specification itself to generate error messages.

Here, test suite operation definitions should also be mentioned that are rather informal in TTCN. Relaxation of abstract data type declaration in SDL-92 means that one can use any kind of operator declaration suitable for the simulator or validator used, e.g. functions written in C.

5.4 Changes in protocol specification

The main change is that the former system specification is used as a block substructure specification (the PRS block) during the validation. However, this difference does not lead to any changes in the specification except that the keyword *system* is substituted by *substructure* because everything declared at system level is appropriate at substructure level, as well.

It is worth mentioning that the protocol specification used to validate the ATS should contain error handling procedures. In protocol standards frequently only the correct behaviour is described. Although error conditions can be managed as implicit transitions (transition with no actions and leading to the same state) in SDL, signal routes and channels should convey erroneous signals, and this declaration must be done explicitly. These declarations could be derived from the ATS declaration part.

The other thing determined by the test suite is the appropriate configuration of the protocol. The best way to do it is receive values for external synonyms from the ATS, as was mentioned before. That is to say, they should be set according to the true value of the test selection expression of the test case chosen to be validated. Consequently these synonyms are no longer external but concrete values are assigned to them.

There are some useful but not necessary changes of the protocol specification: One of them is to make implicit transitions of SDL explicit. These transitions usually describe error handling and treating unexpected signals. If so they should be fired only in cases when such messages were sent by the tester, and they should not occur in pre- and postambles, and their repetition must be limited because they cause no progress in the testing campaign. For these transitions the protocol may send a warning message.

6 THE VALIDATION PROCESS

6.1 Application of the model

From the previous section one can see that for checking, some requirements are built into the model (Holzman, 1991). For these error messages or warnings are received from the system during its run. Other requirements should be checked by the validation algorithm.

The basic idea of the validation is to build the test reachability tree (TRT) **for the PRS** on the basis of signals sent by the tester. The algorithm can be based on Itoh (1983) and Arakawa (1992). The root node of the tree is the desirable stable testing state of the protocol to which every test case is applied. The nodes of the tree are global states of the protocol. A global state is represented by states and variables of processes and by the contents of queues in the system. A global state is stable if there is no executable input signal in any input queue.

In the introduction it was mentioned that every test case has to start from the desirable stable testing state of the protocol. However, this cannot be checked directly on the basis of the test suite because it is only an assumption used as a rule in ATSs that the protocol goes into this state after system start and remains there until an external event occurs. This stable state of the protocol can be defined as the first stable global state of the SDL system.

In SDL every process created at the system start executes its initial transition to reach the initial state. This can contain signal sending or process creation as well. If the addressee of the signal sent is a process of the protocol then it will either consume or save that signal. Similarly if a new process is created, then it will execute its own initial transition that can cause new transitions before the protocol reaches the initial stable state. Until this is done the tester should not influence the protocol according to the assumption mentioned above, i.e. it should send no messages to the protocol or, otherwise, these messages should be saved until the initial stable state reached.

Another requirement is that the protocol has to return to this stable state after finishing a test case. Therefore at the end of each test sequence generated from the test case it should be checked whether the final state is the desirable stable testing state. This will be a leaf node of

the test reachability tree. At this point, according to the introduced rules of the translation the main test component should send the verdict generated in a report message, and it should differ from the initial value *none*. If the SDL interpreter detects that a signal was sent to a TC which has already terminated a dynamic error will occur, it means the relevant test case is incomplete.

A requirement that must be checked: freeness of the test system from deadlocks. Deadlock in our model means that the tester side has not yet reached the end of the test case. There exist process instances of the tester which are waiting for a signal from the protocol, but there are no signals or running timers in the system that can cause any progress, hence, no other reachable state exists for the system.

An opposite requirement is the freeness of the system from livelocks. To check this any new state of the system should be compared to others in the TRT tree, and it should limit the number of times that any single one can be repeated. If this limitation is exceeded, a warning message is sent.

Before continuing the discussion of the validation process, in the following section non-deterministic features affecting testing are outlined.

6.2 Non-determinisms in testing

The main problem of testing - besides the complexity - is that the behaviour of the protocol in some cases is not determined. This can cause non-determinism in the observed behaviour and alternative responses should be taken into account in the ATS (Lee, 1991).

Basically there are two sources of non-determinism:

- The observed behaviour depends on internal decisions, actions or communication that cannot be controlled by the tester;
- The order of processing of signals cannot be determined.

These two sources may appear at the same time, as well, or they may be complicated by the situation that one or some interfaces of the IUT cannot be used as PCO (e.g. remote testing).

For cases when the behaviour of the protocol cannot be controlled by the tester because it has no appropriate interface or because it depends on internal decisions, actions or communication test purposes cannot be specified in the ATS. This type of behaviour should be covered by alternative events (e.g. OTHERWISE statement) or by defaults of test cases. A good example for this is an implementation that can send a busy signal at any time. However, the busy state cannot be controlled by the tester.

Cases when the order of processing signals cannot be determined are:

- Messages sent by the same or by different TCs through different PCOs are processed finally by the same process of the protocol which is observed by one test component.

The conveying channels and processing may delay signals differently on different paths, hence the order of reception by the same component is not determined. The "meeting point" of messages may also be the same test component. For example, a process serves multiple channels (multiplexing) and one of the channels asks for the restart of the system while others require other actions like establishing connection, data transfer, etc. The behaviour of the multiplexer observed will depend on the order of serving out the channels. In this case the tester should contain all possible alternatives with appropriate verdict assignments.

- A message sent to a component of the protocol can result in two or more outgoing messages. If they are observed by the same test component the reception is not determined. However, parallel test components solve this problem. An example of this situation is demultiplexing.
- Any concurrency hidden inside the protocol or by the underlying service provider can cause non-determinism. For example, the service provider uses several paths of delivery. In this case the service provider should be modelled appropriately.

In any case, preliminary results of PTCs are summarized by the main test component, i.e. MTC is a "meeting point" of parallel processes where the test result should be evaluated as far as possible independently of the order of reception. Similar care should be taken every time that parallel test components have some connection with each other, e.g. in order to coordinate their actions. TTCN ensures that the next message from the expected PCO or CP will be received if no alternative PCO or CP is defined at that level. Proper use of TTCN avoids these situations; however, there is no formal definition of "proper use" in this sense.

6.3 Checking non-deterministic features

The first reason of non-determinism requires no changes in the method of building the TRT tree. The second reason of non-determinism can be taken into account only with some limitations. By assigning a range of delay to each delaying channel and processing times to each process in the SDL model signals sent through these channels and processed by these processes should be considered with all possible times of delivery. The TRT tree should be built by applying all permutations of consumable signals arriving at the same time at processes of the protocol. This can result in alternative behaviour that should be handled by the tester. On the tester side, if the test component is specified properly, no unspecified reception should occur. To detect them in any case when an alternative is left out from the tester an error message is generated.

One of the requirements listed in the second section cannot be checked during the run of separate test cases: the completeness of the test suite in respect to sent messages. First of all for a protocol composed of several processes global stable states must be defined for which all possible messages should be sent. After that the combined test reachability tree composed from TRT trees generated for different test cases should be built on the basis of global states of the protocol. Then this combined tree is searched to check if all messages were sent for selected global states.

To check the completeness of the ATS regarding requirements given formally, e.g. in MSC, further investigation is needed because it is not obvious how to select subtrees defined by the MSC in the combined TRT tree. One possibility is to mark in the protocol specification which branches were derived from the MSC, then one need only check whether all these paths were crossed during the building of the TRT tree.

7 CONCLUSIONS

In this paper principles for validation of abstract test suites were discussed for protocols specified in SDL and ATSs given in concurrent TTCN.

The basic idea presented is to derive the SDL specification of the tester for every test case in the ATS, and to build the test reachability tree for the protocol reference specification in the combined test system. For this we investigated the differences between the two FDTs in order to deduce the SDL specification of the tester from its TTCN description. Then we described the combined test system composed of the protocol reference specification and the SDL specification of the tester. During translation, for checking some requirements an ATS should fulfil, sending error and warning messages are built into the model used for validation. For these, messages are received during the run of the combined system. Other conditions like freedom from deadlocks and livelocks are checked by the validation algorithm. For validation of sample test cases we used an SDL simulator.

The principles proposed can be used to develop tools for automatic validation of abstract test suites specified in concurrent TTCN.

8 REFERENCES

Arakawa,N. and Soneoka,T. (1992) A Test Case Generation Method for Concurrent Programs. *Protocol Test Systems, IV.* J.Kroon et al. ed., Elsevier Science Publishers (North-Holland), 95-106.

Bär,U. and Schneider,J.M. (1992) Automated Validation of TTCN Test Suites. *Protocol Specification, Testing and Verification, XII.* Linn,R.J. and Uyar,M.Ü. ed., Elsevier Science Publishers (North-Holland), 279-295.

Belina,F., Hogrefe,D., Sarma,A. (1991) *SDL with Applications from Protocol Specification.* Prentice Hall, Hertfordshire.

CCITT Recommendation Z.100 (1992) *CCITT Specification and Description Language (SDL).*

Draft Amendment 1 ISO/IEC-9646-3 (1993) *TTCN Extensions.*

Fischer,J. and Schröder,R. (1993) Combined Specification using SDL and ASN.1. *SDL'93 Using Objects.* Faergemand,O. et al ed, North-Holland, 293-305

Holzman,G.J. (1991) *Design and validation of computer protocols.* Prentice Hall, New Jersey.

ISO/IEC-9646 (1992) *Conformance Testing Methodology and Framework.*

Itoh,M. and Ichikawa,H. (1983) Protocol Verification Algorithm Using Reduced Reachability Analysis. *Transactions of IECE of Japan,* Vol. E 66. No 2.

ITU-T Recommendation Z.105 (1995) Specification and Description Language (SDL) combined with Abstract Syntax Notation One (ASN.1).

ITU-T Recommendation Z.120 (1993) Message Sequence Chart (MSC).

Lee,D.Y. and Lee,J.Y. (1991) A Well-Defined Estelle Specification for Automatic Test Generation. *IEEE Transactions on Computers,* Vol. 40, No. 4.

prETS 300 414 (1994) *MTS Use of SDL in European Telecommunication Standards.* Rules for testability and facilitating validation.

Wuong,R. (1991) LAPB Conformance Testing Using Trace Analysis, 11th *International IFIP WG6.1 Symposium on Protocol Specification, Testing, and Verification,* Stockholm.

9 BIOGRAPHY

Mária Törő is a computer scientist at the Protocol Laboratory of KFKI Research Institute for Measurement and Computing Techniques (Hungary). She is also a member of the Hungarian Committee of ETSI. Her research interest includes communication protocols and protocol engineering, mainly conformance testing and formal description techniques.

She graduated from the Institute of Engineer-Economics of Kharkov, Ukraina. She received her university doctorate (1992) and Ph.D. (1995) in computer science from the Technical University of Budapest.

Katie Tarnay received the Diploma degree in Electronic Engineering from the Technical University of Budapest and first the Ph.D. then the D.Sc. degree from the Hungarian Academy of Science.

She is currently the leader of the Protocol Laboratory of KFKI Research Institute for Measurement and Computing Techniques and a University professor of the Department of Telecommunication and Telematics at the Technical University of Budapest

Her major area of interest is communication protocols and conformance testing. Her book "Protocol Specification and Testing" was published by Plenum Press in New York (1991). She was the member of the Editorial Board of "Reseaux et informatique repartie".