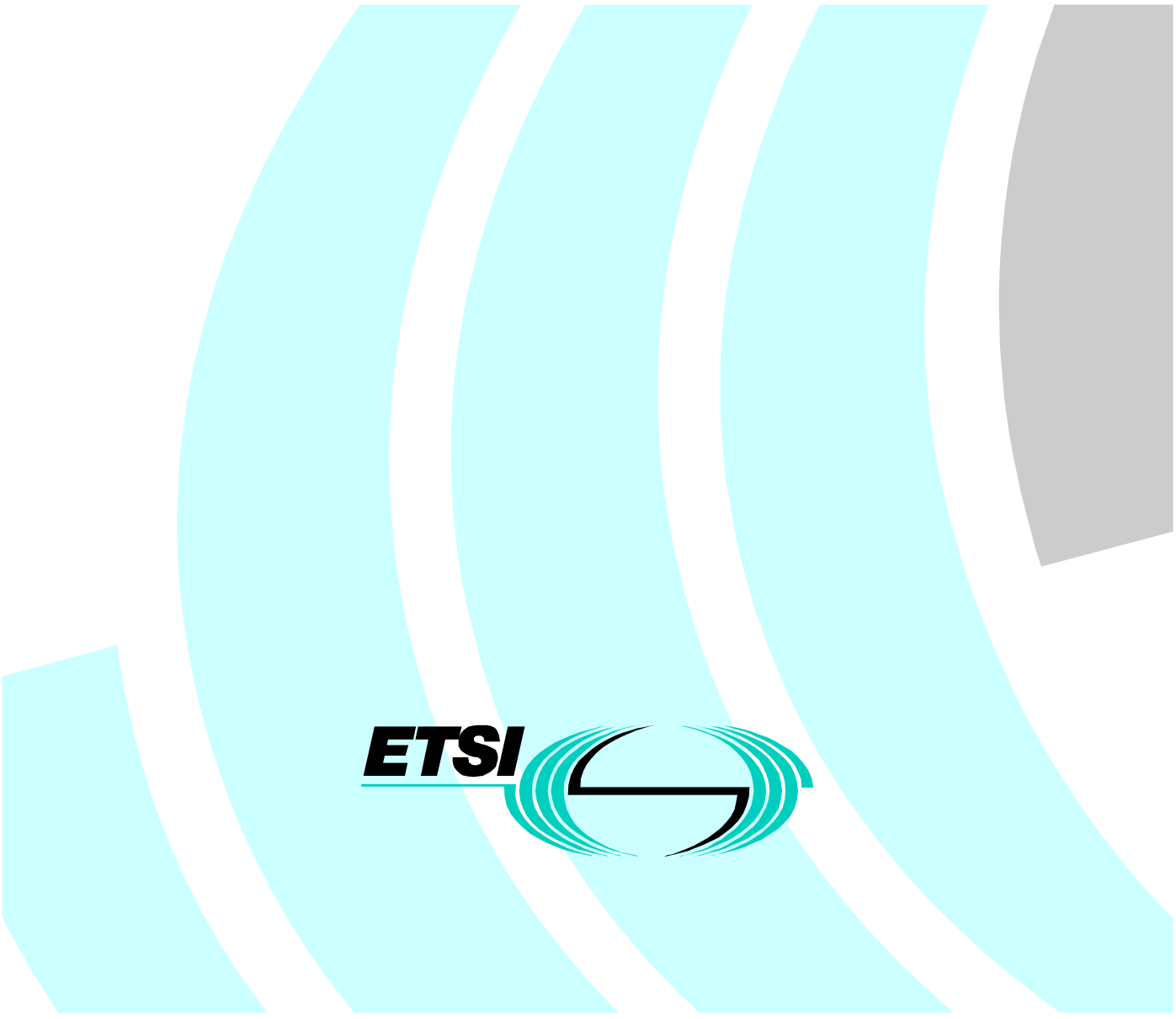
Draft EG 201 015 V1.1.1 (1997-02)

ETSI Guide

Methods for Testing and Specification (MTS); Specification of protocols and Services; Validation methodology for standards using SDL; Handbook



Reference
DEG/MTS-00031

Keywords
validation, SDL, ASN.1, MSC, methodology

ETSI Secretariat

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

X.400
c= fr; a=atlas; p=etsi; s=secretariat

Internet
secretariat@etsi.fr
http://www.etsi.fr

Contents

Intellectual Property Rights

ETSI has not been informed of the existence of any Intellectual Property Right (IPR) which could be, or could become essential to the present document. However, pursuant to the ETSI Interim IPR Policy, no investigation, including IPR searches, has been carried out. No guarantee can be given as to the existence of any IPRs which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Guide (EG) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI Membership Approval Procedure.

Introduction

Specifying a system with SDL is the same as programming in any structured language. The most efficient approach to development is to validate the specification to ensure that it contains no errors before it is implemented. Unlike concurrent programs which are intended to run in real environments, SDL systems run on abstract machines with unlimited resources. A system may be specified using SDL as well as a variety of complementary techniques such as natural language, MSC, ASN.1 and TTCN to produce a complete definition. The validation process should make sure that the specifications are consistent. It is important to note here that SDL specifications usually do not exist in isolation but in a context and, therefore, have to be validated within this context.

There are many techniques that can be used to analyse a formal model and these are described in ETR 184 [5]. Simply stated, static analysis techniques are those that do not require execution of the model and dynamic analysis techniques include both simulation and testing of a formal model, as well as specialized validation techniques. All of the dynamic analysis methods depend on the ability to execute the model. Static analysis can demonstrate that the form of a model is correct, but not that the functions it describes are correctly specified.

1 Scope

The present document provides rapporteurs and other writers of standards with a set of practical guidelines to a methodology for the validation of standards which use the ITU-T Specification and Description Language (SDL) to specify function. Although the validation of other types of specification, such as physical characteristics and transmission parameters, is not explicitly covered, the general approach may prove to be useful in the validation of such standards.

The present document is not intended to be a detailed description of the methodology itself. Rather, its objective is to simplify the use of formal validation techniques so that they become more widely used, thus increasing the overall technical quality of ETSI deliverables.

For a full description of the entire SDL specification process for ETSI standards, the present document should be read in conjunction with ETS 300 414 [1] and ETR 298 [6].

2 References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ETS 300 414 (1995): "Methods for Testing and Specification (MTS); Use of SDL in European Telecommunications Standards (Rules for testability and facilitating validation)".

[2] ETS 300 696 (1996): "Private Integrated Services Network (PISN); Inter-exchange signalling protocol; Cordless Terminal Incoming Call Additional Network Feature (ANF-CTMI)".

[3] GSM 03.93: "European digital cellular telecommunications system (Phase 2); Technical realization of Completion of Call to Busy Subscriber (CCBS)".

[4] DE/SPS 03038 1: "Intelligent Network (IN); Intelligent Network Capability Set 2 (CS2); Intelligent Network Application Protocol (INAP); Part 1: Protocol specification".

[5] ETR 184 (1995): "Methods for Testing and Specification (MTS); Overview of validation techniques for European Telecommunication Standards (ETSs) containing SDL".

[6] ETR 298 (1996): "Methods for Testing and Specification (MTS); Specification of protocols and Services; Handbook for SDL, ASN.1 and MSC development".

[7] MTS-TR 004 (1996): "Methods for Testing and Specification (MTS); Reports on Experiments in Validation Methodology; Part 1: GSM CCBS Supplementary Service".

[8] DTR/MTS-00030-2 (1996): "Methods for Testing and Specification (MTS); Reports on Experiments in Validation Methodology; Part 2: PISN Cordless Terminal Incoming Call Additional Network Feature (ANF CTMI)".

[9] ITU-T Recommendation I.130 (1988): "Method for the characterization of telecommunication services supported by an ISDN and network capabilities of an ISDN".

[10] ITU-T Recommendation Z.100 (1994): "CCITT Specification and Description Language (SDL)".

[11] ITU-T Recommendation Z.105 (1994): "SDL combined with ASN.1".

[12] ITU-T Recommendation Z.120 (1993): "Message Sequence Chart".

[13] ITU-T Recommendation X.208 (1988): "Specification of Abstract Syntax Notation One (ASN.1)".

[14] ITU-T Recommendation X.292: "OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications; The Tree and Tabular Combined Notation (TTCN)".

3 Definitions and abbreviations

3.1 Definitions

3.1.1 External definitions

The present document uses the following terms defined in other documents:

Abstract Syntax Notation One (ITU-T Recommendation X.208 [13]).
Message Sequence Chart (ITU-T Recommendation Z.120 [12]).
Specification and Description Language (ITU-T Recommendation Z.100 [10]).
Tree and Tabular and Combined Notation (ITU-T Recommendation X.292 [14]).

3.1.2 Internal definitions

For the purposes of the present document, the following definitions apply:

deadlock: A state of a system such that no progress in the form of external visible behaviour is possible.

formalization: The stepwise activity in which a formal SDL description of a system is produced.

formal SDL description: An SDL description which conforms to ITU-T Recommendation Z.100 [10], does not contain any SDL "informal text" and which, thus, can be interpreted by automatic tools.

implicit signal consumption: The receipt of a signal in a state where no processing is specified for the handling of the signal. It is, therefore, implicitly consumed which often means that it will be discarded (sometimes called unspecified reception).

informal SDL description: An SDL description which conforms to ITU-T Recommendation Z.100 [10] but which includes some SDL "informal text". As a result, it cannot be interpreted by automatic tools.

observers: A way to check temporal claims.

temporal claims: Assumptions on particular system states expressed formally.

validation: The process, with associated methods, procedures and tools, by which an evaluation is made that a standard can be fully implemented, conforms to rules for standards, and satisfies the purpose expressed in the record of requirements on which the standard is based; and that an implementation that conforms to the standard has the functionality expressed in the record of requirements on which the standard is based.

validation model: A detailed version of a specification, possibly including parts of its environment, that is used to perform formal validation.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ANF-CTMI	Cordless Terminal Mobility Incoming call Additional Network Feature
ASN.1	Abstract Syntax Notation One
CATG	Computer Aided Test Generation
CCBS	Completion of Call to a Busy Subscriber
CS2	Intelligent Networks Capability Set 2
GSM	Global System for Mobility
ICS	Implementation Conformance Statement
IN	Intelligent Network
INAP	Intelligent Network Application Protocol
LAN	Local Area Network
MSC	Message Sequence Chart
MTC	Master Test Component
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PID	SDL Process IDentifier
PTC	Parallel Test Component
SDL	Specification and Description Language
TTCN	Tree and Tabular Combined Notation

4 The validation model

4.1 The approach to validation

A protocol standard will normally use a mixture of techniques for describing the functions of the protocol. These may include informal prose as well as formal notations such as SDL, MSC and ASN.1. The specification may make reference to other standards without explicitly incorporating them into the specification. If this is the case, it may be necessary to develop more than one formal model in order to carry out the validation. Also, a protocol specification is likely to contain options and it is necessary to select a particular set of these before the model can be made executable.

Figure 1 illustrates a general approach to validation of SDL specifications.

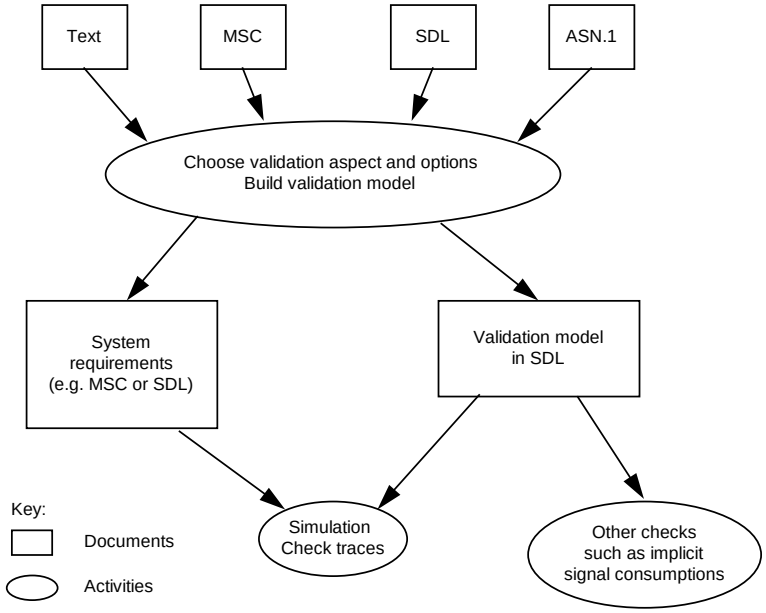


Figure 1: A general scheme for validation

4.2 Making a validation model

The validation model is a formal description of a system which is suitable for validation. It should reflect all aspects of the standardized specification which is likely to be expressed as free text, MSCs ASN.1 and semi-formal SDL. The impact of other relevant standardized specifications shall also be taken into account in the model.

The SDL specification presented in a standard may not be directly executable as it is likely to contain options and may not represent a complete working system. Specifications of supplementary services, for example, do not normally include the functions of the associated basic service. A validation model, however, must always be executable.

4.3 Identify the scope of the validation

In principle, all functions of a protocol could be validated in a single model and if the functionality is sufficiently small, this may be appropriate. However, most validation techniques suffer from the state space explosion problem and it is often better to make separate validation models for separate aspects of the validation, each being of a size that is manageable for tools.

EXAMPLE: Consider an imaginary connection oriented protocol which has three phases:

- Connection Setup;
- Data Transmission;
- Release.

The Data Transmission phase may also optionally include a window mechanism for flow control.

It would be appropriate in this case to establish two validation models; one for Connection Setup and Release without the window mechanism of the Data phase, and one for the Data phase with the window mechanism. Each validation model can then be validated separately although care shall be taken in considering possible interactions between the Connection/Release and Data phases. In general, the validation of individual components does not guarantee that the complete model behaves well.

Alternatively, incremental validation may be applicable when a large SDL specification is developed using an incremental strategy. First, the basic functions are specified and a first validation can take place using the exhaustive exploration technique. Then new functions of the system are added incrementally and for each increment an additional exploration is run to validate this feature in the SDL model. Thus, partial validations are combined to validation of the whole system.

4.4 Choose a particular set of implementation options

Many standards describe a range of possible implementation and functional options. As the validation model has to be executable, it is necessary for a particular set of options to be chosen. The valid sets of options are defined in the Implementation Conformance Statement (ICS) proforma.

Optional behaviour is often modelled in SDL using parameters that are declared to be external to the specification. Actual values of these parameters have to be chosen when the validation model is built and, according to the options chosen, different models may result. Each one shall be validated separately. A simple example of the use of external parameters is shown in figure 2.

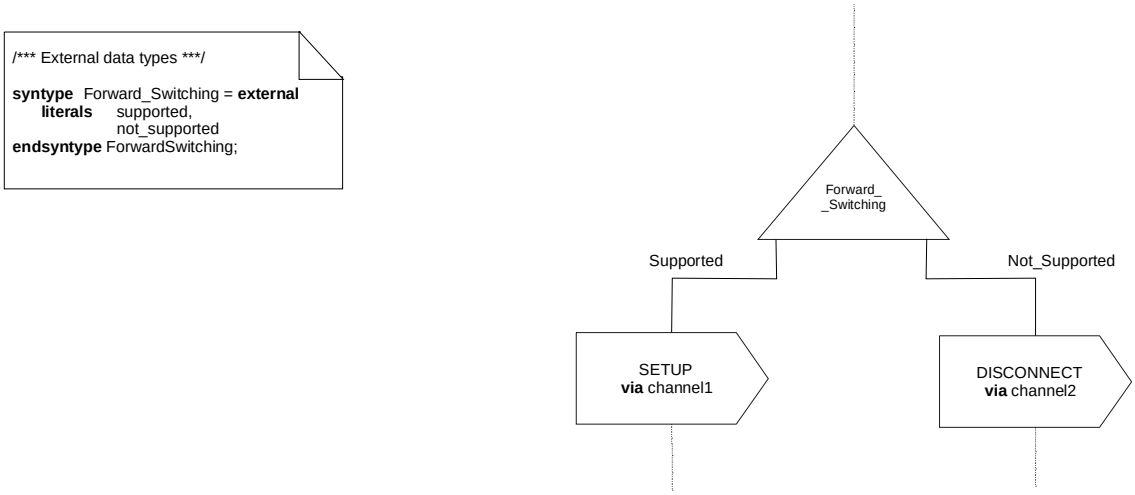


Figure 2: Example of the use of external data type to specify options

4.5 Make the system complete

In many standards, the specification of the standardized function is not complete because references are made to other specifications.

In order to build a complete and executable validation model, it is necessary to model these additional aspects. Such additions should be simplified as far as possible and should be restricted to those functions which are absolutely necessary to validate the protocol that uses it.

NOTE: A validation model specified in formal SDL should remain understandable for non SDL experts. Thus, the structure of the formal model should maintain, as far as possible, the implicit architecture that is underlying the original informal description of the system. However, the requirements for validation may impose changes in the SDL model. For example, to obtain an executable model, it is usually necessary to define some processing functions to describe the informal statements given in the initial specification. Apart from the resulting complexity of the final model, the changes should not affect the basic functionalities of the system for the validation to continue to be relevant to the initial model.

4.6 Optimize the model in terms of state space

The number of states that the system will run through during an execution is influenced by the stimuli it receives from the environment. If the environment is left completely open, i.e. nothing is specified about the environment, then any possible signal can be sent to the system at any time. This will result in a large number of states being validated without significantly increasing the symbol coverage of the SDL system.

In the example of the connection oriented protocol above (subclause 4.3), the environment could send a connection request at any time, even when awaiting confirmation from a previous request. In fact, the environment can send any number of connection requests without waiting for a confirmation. The connection requests need to be held in an input queue rather than being consumed immediately by the receiving process. For every additional connection request a new system state results and this is unlikely to be a valid scenario. It may, therefore, be worthwhile to specify the behaviour of parts of the environment. In the example, the environment should be specified in such a way that it does not send a second connection request before it receives a confirmation (or a disconnection, if that is part of the service).

For each instance of a state included in an analysis, a validation tool would need to store information such as:

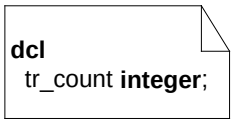
- the control state of each process;
- the contents of all input queues;
- the values of variables used in the processes;
- the state of the timers.

Thus, the fewer process instances exist, the less memory is needed. The behaviour of a connection oriented protocol supporting thousands of connections simultaneously may not be much different from one supporting only four. The validation model may, therefore, need to be artificially constrained to reduce the number of process instances as far as possible.

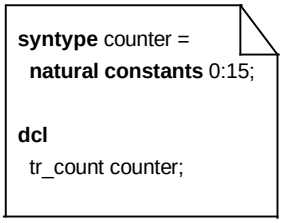
The following suggestions may help in limiting the size of the model:

- input queues in SDL are infinite by definition. Storage space may be saved by defining an upper limit for the queues in the validation model;
- the necessity of delaying channels should be investigated;
- there are often many ways to declare a variable that is needed for a specific purpose. A counting variable, for example, counting from 0 to 3, can be declared as an integer. This uses a lot of storage space (probably 32 bits). If it was declared as a range from 0 to 3, 2 bits may suffice to store it.

EXAMPLE: If a variable, *tr_count*, can only take values between 0 and 15, it can be specified in one of two ways:



or



The first approach will always use the predetermined number of bytes reserved for integer variables whereas, in the second case, the number of bytes used will depend on the range specified (in this example, only 1 byte).

These measures will tend to cause a differentiation between the validation model and the original specification. The designer of the validation model shall decide whether this is tolerable or not according to the validation purpose.

4.7 Model configuration

For validation purposes, it is not necessary to model a life-size configuration of the system. A "reduced model" may be sufficient. For example, if a transmission protocol contains 16 priority levels, it may be enough to represent just 2 or 3 of them in the model. Moreover, an SDL model intended for validation cannot always handle an exact representation of the real system and its environment.

As an example of this problem, consider the case of a LAN that can contain 1 000 stations. In order to validate the network, the first SDL validation model would only include 3 or 4 of them. The most complex model that can be validated would include up to 15 or 20 stations because the size of the state graph becomes too large above this number. Therefore an alternative approach is required to conduct the validation. Among the possibilities are:

- use the "Single Fault Hypothesis":
  - the failure of one station on the Local Area Network (LAN) would be considered at one time and recovery would be assumed to be complete before a new station failure occurs;
- derive properties by inference on the number of instances:
  - if it was not possible to validate a model with more than 10 stations on the LAN, it would be necessary to calculate by recurrence on the number of stations a given property for the complete network;
- create several validation models for different purposes:
  - the message loss caused by a medium error would be validated separately from the message loss caused by a station failure. The different models would then be combined in order to derive more general validation results involving the interdependency between the faults.

4.8 Dealing with the environment signals

In order to perform validation efficiently, it is generally assumed that the environment is reasonably fair [Hogrefe & Brömstrup 1989]. This means that the environment only sends signals to the system when no internal events can occur and in particular when the system has finished processing all events received during the previous transition. This 'reasonable environment' hypothesis leads to a substantial reduction of the state space. In the actual environment, this represents the general case where the timer delays and the execution speed of the environment are much longer than the internal delays of the system.

Putting the hypothesis to work implies the definition of some priorities between the events that are represented in the behaviour tree of the system. Such events can be classified as internal events, SDL timers, spontaneous transitions and inputs from the environment. The usual validation approach is to assign the highest priority to internal events, then to timers and spontaneous transitions and finally to inputs from the environment. Based on these priorities, validation tools will explore the state space in a more 'rational' way by examining which transitions to initiate according to their decreasing priorities. Consequently, external events will be offered only if there are no higher priority events to consider.

4.9 Validation results

In order to close the validation process, a validation report shall be provided that includes the following information presented in a structured manner:

- original models;
- identification of the validation tool used and its version;
- model configurations (e.g. regarding the environment);
- validation coverage and statistics;
- an explanation of why exploration was not exhaustive if non-exhaustive methods were used;
- limitations of the validation procedure;
- classification of the errors found (including the path that leads to the error);
- report logs, specific system states;
- parts of the specification that are never executed (dead code);
- parts of the specification that are incomplete (unspecified);
- over specification problems:
    - this can include such aspects as the provision of too much detail in parts of the specification or the multiple definition of a data type or behaviour using different names at each definition. In most cases, over specification problems can only be detected by inspection rather than with automatic tools;
- parameters, abstract data types, queues and timers testing (especially their upper bounds);
- reference scenarios to test specifiers (in the form of MSCs for example);
- quality of the model writing (use of constants, modularity, comments...);
- identification of SDL features which proved to be inadequate for validation or implementation.

A question that might be put at this stage is "*How do I determine that the validation has finished?*". The answer is easy when exhaustive exploration has been used and has always run to completion. If exhaustive exploration could not finish by itself or could not be applied, then termination rules should be found in the initial section that defines the validation context (subclause 6.1). In general, such aspects as the number of symbols covered, the number of parameters and timers tested and the number of functions checked provide the elements to answer the question. However looking for a 100 % symbol coverage should not always lead to changes in the model solely for the purpose of improving coverage.

5 Capabilities of SDL tools

This clause describes the capabilities that most of the tools offer.

5.1 Static validation of a specification

Static validation covers two aspects. First, the designer uses automatic tools to carry out a static analysis and to correct the model if necessary. Second, the model should be reviewed by other experts (STC/WG members, PT Experts, PEX or external specialists) to check its conformity against the specification rules (ETS 300 414 [1] and ETR 298 [6]) and the system objectives. These static analyses do not require the model to be executed. They only guarantee that the model is well formed and that it can be compiled to produce an executable model. Static validation covers:

- detection of syntax errors (e.g. invalid identifier);
- detection of semantic errors (e.g. type mismatch);
- completeness of the models;
- cross-reference checking;
- cross-reading of the specifications.

5.2 State space analysis

A formal SDL model builds a state space that starts at a root, the initial state of the system, and then expands as a tree, i.e. the root has some following states, the following states have followers and so on. A state in an SDL model is composed of all the states of the individual processes, the contents of the queues, the values of the variables used in the processes, the timers, and a few other components.

The aim of state space analysis is to search the state space in a systematic way for particular situations. It is, for example, very common to search systematically for implicit signal consumptions or deadlocks.

SDL analysis tools usually support several of the algorithms described in subclauses 5.2.1 to 5.2.6.

5.2.1 Exhaustive state space analysis by depth first

The state space of complex telecommunication systems is very large, sometimes even infinite. It is therefore feasible to consider the state space only down to a limited depth, e.g. 100. One way to search the state space is depth first, where all the visited states are stored in order to avoid visiting the same state twice. This requires a large amount of storage space.

All reachable states are generated this way.

5.2.2 Bitstate space analysis by depth first

Bitstate is a simulation of exhaustive state space analysis which avoids having to store the full state information [Holzman 1991]. The algorithm uses a hash function instead to calculate a value for each state. If the value has appeared before, the state is considered to be already visited. The probability that two different states map to the same hash value is very low.

Almost all reachable states are generated this way with only a small probability that a state is omitted.

5.2.3 State space analysis by depth first random walk

Another possibility is to search the state space in a random way [West 1992]. Starting from the root, one path down the tree is followed by random choice down to a predefined depth, e.g. 100. Afterwards, another path is followed by random selection again and this process is repeated until terminated by the operator. This algorithm does not require a large amount of memory, but there is no guarantee that all states will be visited. It is assumed that the results of an error in a specification do not only occur at a single place in the state space but at many places.

The number of the reachable states analysed in this way depends on the time allowed for analysis.

5.2.4 Exhaustive state space analysis breadth first

Breadth first analysis examines the state tree layer by layer. This is particularly useful in cases where errors are likely to occur in very early stages of the system development.

5.2.5 State space analysis by simulation

An alternative to random choice on the way down the state space is to allow choices to be controlled by the user of a tool. The system starts in some initial state. From there it is driven by the user through the state space. The user can influence the state transitions by signals input from the environment to the system. Another way to influence the course of events is to manipulate variables internal to the processes.

In this way, even non-reachable states of the system can be examined. The proportion of the states examined depends on the time the user spends with the simulation.

This form of analysis is also called debugging or testing of the formal model.

Test scenarios may be described using Message Sequence Charts (MSCs).

Simulation can be combined with performance analysis which evaluates system properties such as delay, throughput, and other timing related properties. A formal, functional model is a useful starting point for such an analysis, but shall be supplemented with timing and performance data.

5.2.6 Consistency checking between MSC and SDL

An MSC is produced to illustrate the execution of a particular part of a system. The MSC is then used as an input to the validation tool which checks the consistency between the MSC and the operation of the SDL.

6 A validation method

A validation method is only valuable if it is supported by tools. The details of the method therefore depend on the facilities of the tools. In this clause a general tool-supported methodology is described.

The following approach to validation has been proven useful:

- define the validation context, produce a validation model and formalize the requirement specification;
- carry out internal checks with both specifications to find deadlocks, implicit signal consumptions and other specification errors;
- define temporal claims and observers for both specifications and validate them;
- compare two specifications, e.g. verify MSCs against the SDL specification.

6.1 Define the validation context, produce a validation model and formalize the requirement specification according to figure 1

The production of a validation model is described in clause 4.

From the user perspective the validation model and the requirement specification should ideally have the same behaviour, although in practice this can seldom be achieved. The validation model is more implementation-oriented whereas the requirement specification is an abstract and exclusively user-oriented view. Usually both views can be specified with SDL but for the requirement specification view MSCs may be more appropriate. For an example, see GSM 03.93 [3].

For validation purposes it is very convenient to have more than one specification of the same behaviour which are compared with the support of automatic tools. Usually, during the comparison process, errors are found in both specifications. Discrepancies in the observable behaviours point to specification errors.

The requirement specification can also consist simply of some scenarios that the system is expected to be able to run through. MSCs are a particularly useful notation for expressing such scenarios.

Alternatively, the stage 2 and stage 3 specifications, as defined in ITU-T Recommendation I.130 [9], can be used for such a comparison if both are specified formally (with SDL and/or MSCs) and the stage 3 is produced in such a way that it shows a comparable observable behaviour to the stage 2.

6.2 Carry out internal checks with both specifications

In this subclause it is assumed that the usual syntax and static semantic checks have been made and the SDL specification can be compiled into an executable model. Now the measures as described in clause 5 can be taken. Here, for example, the issues of deadlock, implicit signal consumption and temporal claims are further detailed.

6.2.1 Deadlock

A deadlock is an ending state of the system. Tools that find deadlocks look for ending states. These states may have been reached on purpose or not. Usually in communication systems, an ending state is not desired. From any state the system should be able to perform some further action, e.g. a reset. Therefore an ending state is most likely a specification error, and it is worthwhile looking for it.

Unfortunately the state space of communication systems in many cases is too large to be able to visit every state during the state space exploration with a tool. Therefore there is no complete certainty that all deadlocks have been found after an exploration.

6.2.2 Implicit signal consumption

An SDL process consumes any signal that is in the valid signal input set of the process, even if there is no explicit state transition defined for a signal in a particular state. In this case there is an implicit transition that takes the signal from the input queue without changing the state of the process. The effect is as if the signal had been destroyed.

Although implicit signal consumption is not a semantic error in an SDL specification, it may point to an undesired behaviour. Tools can find implicit signal consumptions by simulation or state space exploration.

6.2.3 Define temporal claims and observers for both specifications and validate them

With temporal claims certain conditions can be defined that should hold true during the interpretation of a system, either permanently or at certain times. Temporal claims can exist in the form of anticipated values of specific variables or in terms of event sequences that should occur during the system interpretation.

6.3 Comparison of two specifications by MSC or TTCN

The two specifications mentioned in this subclause, the validation model and the requirement specification or protocol and service, should compare to each other in a well-defined way. For example, it could be required that the sequences of observable events are the same, i.e. the signals that enter and leave the system. Since the specifications here are usually quite complex, the observable behaviour is far from being obvious. It can usually only be determined by simulation.

SDL tools offer the possibility to generate MSC and TTCN from an SDL specification in order to capture the observations during a simulation run. This facility can be used for the comparison of two SDL specifications in the following way:

1) generate MSCs or TTCN from one specification;

2) run these MSCs or TTCN against the other specification.

With this approach the behaviour of the two SDL specifications can be compared for some user-defined situations.

The state of the art at the time of writing this document is that the tools cannot generate MSCs or TTCN in a highly automated way. For the moment the user has to guide the tools through each test case. This is a slow process and not suitable for a large scale event sequence based behaviour comparison. Prototype research tools such as SAMSTAG [Grabowski et al. 1995] and TVEDA [Phalippou 1994] show that the automation can be further increased.

7 The validation plan

Validation is an important procedure within the overall standards development process. It can help to ensure that published standards are of a consistently high quality and that the specified services and protocols can be implemented by manufacturers. Such quality cannot be achieved without any effort and the time and the manpower required for validation shall be planned into the schedule for the standard when it is first raised as a work item.

The stages that shall be considered when planning for validation are as follows:

Stage 0: Definition of the validation context.

A validation project must delimit precisely the environment within which the validation is being conducted.

Stage 1: Formal specification of the validation model and requirements.

A formal model of the standardized system shall be produced based on the methodology described in ETS 300 414 [1] and ETR 298 [6].

Stage 2: Validation of the formal model and the requirements.

Automatic tools are used to carry out analysis and to correct the models if necessary.

Stage 3: Validation results.

In order to close the validation process, a validation report shall be produced.

8 Experience

The validation methods described in this technical report are based on experience gained from a small number of experimentation projects using real standards for services and protocols. The subjects of the three experiments were chosen such that each one presented different problems and opportunities for the validation process. The standards used in the experiments were:

- GSM 03.93 [3]: European digital cellular telecommunications system (Phase 2); Technical realization of Completion of Call to Busy Subscriber (CCBS);
- ETS 300 696 [2]: Private Integrated Services Network (PISN); Inter-exchange signalling protocol; Cordless Terminal Incoming Call Additional Network Feature (ANF-CTMI);
- DE/SPS-03038-1 [4]: Intelligent Network (IN); Intelligent Network Capability Set 2 (CS2); Intelligent Network Application Protocol (INAP); Part 1: Protocol specification.

8.1 Results of validation experiments

8.1.1 GSM supplementary service, CCBS

The CCBS service for GSM was chosen because it is one of the most complex supplementary services and STC-SMG3 had already specified it quite fully in SDL. This specification was taken and transformed into a formal validation model maintaining both the service and protocol views used in the original. Once the model was complete and syntactically correct, it was processed by the validation tool and the reported errors were described and analysed within the experimentation report, MTS-TR 004 [7]. However, although the report suggests changes that would overcome the problems found in the model, none were tested and it is, therefore, not certain that all errors were found in this process.

The main lessons learned from this experiment were:

- when analysing the output of a validation tool, it is important that individuals with expertise in the service or protocol and in SDL itself are available (may be the same person);
- if a new state is introduced when building the validation model, all possible effects of this shall be considered. For example:
  - which timers can expire?
  - how should they be handled?
- if a protocol contains two views (a Service and a Protocol view) it is important that they describe the same behaviour. A major problem in the CCBS protocol was that the protocol view included a Queue-handling feature which was not present in the Service view;
- special care should be taken to avoid synchronization problems between the physical entities (processes) of the model. For example, all parts of the protocol should be in the "Idle" state before a new request is handled from the environment.

- much information can be found by looking at the Symbol Coverage reports. Symbols not executed at all are probably not needed or cannot be executed in the configuration chosen for that Validation.

- it must be remembered that the validation model is not part of the standard itself, although it should be included as an appendix.

8.1.2 QSIG additional network feature, CTMI

The QSIG Cordless Terminal Incoming Call Additional Network Feature was chosen because it is a relatively straightforward service performing the simple task of directing an incoming call for a user of a cordless terminal to the user's current registered location within the network. The protocol was already specified quite fully in ETS 300 696 [2] using extensive text as well as ASN.1, MSCs and simple, informal SDL. One of the objectives of this experiment was to produce formal SDL that remained almost as readable as the informal model found in the standard.

The ASN.1 was re-coded into SDL data following the guidelines in ETS 300 414 [1]and the SDL was formalized and extended to include a simple model of the QSIG basic call protocol. The validation tool revealed a number of errors in the model (all implicit signal consumptions) and after each run, the model was modified to remove the errors. In the early stages of validation, as each group of errors was remedied, further errors were revealed. As the exercise progressed, however, fewer and fewer errors were found until, ultimately, only four could be detected after many thousands of iterations by the tool. Each of these errors could be explained as either the result of an acceptable compromise in the modelling of the basic service or caused by a very unlikely sequence of events. A full report of this experiment can be found in DTR/MTS-00030-2 [8].

The additional lessons learned from this experiment were:

- before beginning the formalization of a supplementary service, it is necessary to reach agreement on the method to be used to incorporate the basic service into the validation model. There are a number of alternatives for this and it is not certain that the method chosen for ANF-CTMI is the best approach. Further study is needed in order to determine which are the best alternatives for the implementation of basic service;

- when preparing the validation model, move all activities that do not place requirements on implementors into procedures. Examples in the ANF-CTMI experiment are the Home Database and Visitor Database access procedures;

- once the formal model is coded, a step by step method should be followed for the validation of the model. In simple terms, the steps should be based on the following:

  a) analyse and correct the syntax of the SDL;

  b) analyse and correct the static semantics of the model;

  c) validate the model giving a high priority to internal signals. Correct any reported errors;

  d) validate the model giving a high priority to internal signals and timer events. Correct any reported errors;

  e) validate the model giving a high priority to internal signals, timer events and signals from the environment. Correct any reported errors and assess the symbol coverage to determine whether it could be improved by changes to the model.

- when evaluating errors reported by the validation tool, consider all of the errors together before making changes to the model. Taking one error at a time can lead to unnecessary coding and subsequent re-coding of the SDL. However, it is not advisable to allow the validation to continue until large numbers of errors have been detected;

- although validation tools report errors such as implicit signal consumptions and produce MSCs to indicate where in a sequence of messages the error occurred, it should be remembered that the reported error is usually only a symptom of a design problem in a transition which may have occurred much earlier in the sequence;

- the ANF-CTMI validation model did not attempt to specify simple behaviour in the environment. As a result, a number of extremely unlikely scenarios were evaluated. If testing a similar service, it would be beneficial to include specific behaviour of the environment in the model.

8.1.3 INAP Capability Set, CS2

At the time of writing this technical report, the validation experiment using the INAP CS2 package is incomplete. Many of the lessons learned so far are included in subclause 8.2.

8.2 Common errors made during validation

This subclause provides answers to some of the questions which have arisen when trying to resolve problems in the building of SDL validation models. In essence it is a list of "Most Frequently Made Mistakes" and covers the following aspects of modelling:

- configuration;
- reasonable environment;
- Process IDentity (PID);
- ASN.1 restrictions.

8.2.1 How do you configure your validation model?

A validation model is an executable SDL model that resembles to a large extent a real system. A specification usually describes components of this system, i.e. one switching system, that appears at various places in a real system. A validation model therefore may also need to have various copies of these components in order to get it operational, e.g. an experimental network.

8.2.2 Do you have a model of the environment of the system? Does it behave reasonably?

Tools for analysing validation models usually contain a function that examines all possible states that the system may run into. The validation model is driven by stimuli from the environment. The tools usually consider any possible input in any possible state of the system. In many cases these inputs are unreasonable; e.g. to send data in a connection oriented protocol before a connection has been established or to try to form a conference in an IN environment before the calls have been initiated. Some of this can be viewed as robustness checking but most of it just lets the state space explode. It is therefore most valuable to make a model of a fairly reasonable environment in order to facilitate validation [Hogrefe & Brömstrup 1989]. In particular different levels of priority can be assigned to the signals in order to avoid state space explosion. For example, in many cases it is worthwhile to give a lower priority to signals from the environment compared with signals internal to the system. This way the system has to come to a stable state with no internal signals waiting, before another signal from the environment is consumed.

8.2.3 Are you communicating PID values outside the system?

SDL allows the communication of PID values outside the system in order to enable the environment to address specific process instances inside the system. For validation purposes and test case generation this may cause problems. The PID values are assigned automatically by the SDL interpretation system. At each instance of execution they may be different. It is never possible to know the PID values beforehand. It is, therefore, not possible to verify an MSC specifying a sequence of message exchanges with the environment if these messages contain PID values.

Also test cases cannot be derived from a system that communicates PID values to the environment. Test cases are static. They are made once and forever. The SDL system interpretation is dynamic. In different instances of interpretation it assigns different PIDs to the processes involved.

8.2.4 Are you using data types that can be handled by both the SDL tool and the TTCN tool?

If SDL data types are used, you should make sure that they match to an ASN.1 construct. Most of the restrictions on SDL data can be found in Recommendation Z.105 [11]. Some additional restrictions are:

- do not use the same literals in different SDL types. Literals will be translated into enumerations in ASN.1, and they have to be disjoint in order to allow the SEQUENCE operator;
- do not use the "." in names of signals. This character cannot be handled by ASN.1.

Annex A (informative): Automatic Test Case Generation

This annex summarizes the possibilities for the automatic, or semi-automatic, generation of abstract test suites from SDL validation models. It is based on experience using the Telelogic ITEX Link and Verilog ObjectGEODE TTCgeN tools to produce test cases from the formal SDL for the Broadband ISDN (B-ISDN) protocol and the hypothetical INRES protocol.

A.1 Characteristics of a testing model

Generally a validation model will exhibit the complete functional characteristics of the system that is to be tested. It is self evident that if this model does not include all the functionality to be tested then this should be added. However, in most cases the dynamic properties of the validation model and those of the test model are expected to be very close, if not the same. There are some minor exceptions, noted below.

The main difference between the two kinds of model is in the way data is represented. In a validation model many of the data definitions (i.e. PDUs and their contents) will be in a partial or even abstracted form. If it is wished to generate tests from this kind of model then there is the choice of either :

1) using the model without modification; or

2) complementing the model with full data definitions.

In case 1), correct behavioural tests are derived but the complete PDU definitions and constraints need to be added to the TTCN manually. The generated constraints could be used as templates for complete constraints or they could be used as derived constraints with the base constraints being created manually.

In case 2) both the dynamic behaviour and the generated data would be complete. However, it should be noted that it is not simply a question of adding SDL data definitions. In order to have confidence in the generated tests it is necessary to ensure that all data relationships with the dynamic behaviour and all internal data relationships (e.g. dependent optional parameters) are *fully* modelled in the SDL. This can be a complex and time-consuming process which comes very close to producing an implementation of the system. It is arguable that if this additional modelling is solely for the purpose of generating tests then it is probably more efficient to put the effort into method 1); i.e. to do most of the data and constraint definitions manually.

Other dynamic aspects, such as the test method used (which may require additional SDL processes to take care of some interface interactions) or taking design decisions to simplify the model (e.g. removal of some options or the elimination of non-determinism) will also mean that the model may need to be changed. These changes, however, will usually be small compared to the initial effort of building the model and with respect to the potential time-savings in test suite production.

A.2 The feasibility of automatic test case generation for real test suites

Case-studies using the ITEX Link and ObjectGEODE TTCgeN tools indicate that the TTCN generated by different tools is likely to be quite similar. Certain information such as verdicts and comments shall be added manually. Also, it is unlikely that tools will be able to cope with such aspects as test suite parameters, selection expressions, structural grouping of the test suite and the generation of concurrent TTCN components (for example, PTCs). All these shall be added manually.

After these additions the resulting TTCN code should be directly compilable but it will contain much redundancy, obscure names etc. This kind of TTCN is not very readable. Extensive further manual editing is required if the TTCN is to have a style similar to that produced by a human programmer as would be expected in an ETSI standard. Typically, this would be the structuring of Test Case behaviour into Test Steps and the parameterization of Constraints. An example of the unmodified output and the output after manual processing is shown in table A.1 and below:

Table A.1: The raw TTCN output generated by a typical tool (B-ISDN Test Case)

Test Case Dynamic Behaviour

Test Case Name: TC1

Group:

Purpose:

Configuration:

Defaults: OtherwiseFail

Comments:

Nr	L	Behaviour Description	Constraint Reference	V	Comments
1		UNI! PREAMBLE	PR_0_0		
2		UNI! SETUP	SU_0		
3		N? Setup_ind	Setup_ind500		
4		N! CallProceeding_req	CP_req		
5		UNI? RELEASE	RELEASE500		
6		N? Release_ind	Release_ind500		
7		UNI? RELEASE	RELEASE501		
8		UNI? RELEASE	RELEASE502		
9		UNI? RELEASE	RELEASE503		
10		UNI? CALL_PROCEEDING	CALL_PROCEEDING500		
11		UNI! STATUS_ENQUIRY	SE_0		
12		UNI? STATUS	STATUS500		

The example above needs to be edited as follows in order to complete the test case:

- add the Test Group path and Test Purpose to the header;
- replace the dummy preamble in line 1with a real, parameterized Test Step (+Preamble);
- add the verdicts.

Other modifications which do not add information but which improve the style are:

- change the relative ordering of the receive statements in lines 6 - 10 to reflect that the *expected* input is CALL_PROCEEDING rather than RELEASE or Release_ind;
- collect all four RELEASEes (lines 5 and 7 - 9) into a single constraint;
- move lines 11 - 12 to a generic *CheckState* Test Step (parameterized for state N3);
- rename constraints according to agreed Test Suite conventions and add comments.

Table A.2: The Test Case after manual editing

Test Case Dynamic Behaviour					
Test Case Name:	TC1				
Group:	B-ISDN/Valid/				
Purpose:	Ensure that on receipt of a correct SETUP PDU the IUT responds with a CALL_PROCEEDING				
Configuration:					
Defaults:	OtherwiseFail				
Comments:					
Nr	**L**	**Behaviour Description**	**Constraint Reference**	**V**	**Comments**
1		+ Preamble (0, 0)			Start in N0
2		UNI! SETUP	SU_0		Correct SU
3		N? Setup_ind	SU_ind		
4		N! CallProceeding_req	CP_req		
5		UNI? CALL_PROCEEDING	CP_0	(P)	
6		+ CheckState (3)			Finish in N3
7		UNI? RELEASE	RL_0		
8		N? Release_ind	RL_ind	I	

NOTE: The CATG tools do not support concurrent TTCN. In practice, this test case would need to be split into an MTC and two PTCs, with one PTC for each PCO. The MTC would CREATE the PTCs and would assign the final verdict. The resulting Test Case is then very different from the original tool output of table A.1.

Given the possible extent of this manual intervention, it is legitimate to ask whether or not the tools offer any overall savings.

In the general case these tools will be of great assistance to less-experienced test engineers, or those not totally expert with the protocol to be tested. For experienced test case writers the benefits are less obvious.

Two possible scenarios have been identified:

1) the case where the test purposes require complex behaviour interactions but quite simple PDUs (e.g. INAP). Experience with the tools indicated that they included in the TTCN output possible interactions that even an experienced test engineer could well have missed. In this case the tools are definitely considered to be of assistance;

2) the case where the test purposes require simple behaviour interactions but fairly complex data (e.g. the tests are parameter variations over a limited set of simple behaviours). Unless the data is fully modelled in the SDL (which as stated earlier may not always be economical) the overall savings would be considerably less as the test suite writer could probably produce this behaviour quicker by hand.

In summary, tools could at least :

- aid the development and validation of test purposes;
- shorten the time needed to create data declarations and constraints;
- provide skeleton behaviour trees that are correct according to the protocol.

A.3 The future

In conclusion, the future for CATG techniques appears promising even though the tools on the market today are at an early, and crucial, stage of development. Tool makers are expected to improve their products and significant progress is likely in the near future.

In the meantime, the tools can be an important complement in the test case production process. As the tools become more mature, automated techniques will increasingly play an important and vital role in test suite development. It is unlikely, however, that the process will ever be completely automated. Writing tests is, after all, a complex programming activity and those elements of that activity which require deep human skill, knowledge and creativity will not be automated in the foreseeable future.

Annex B (informative): Bibliography

- West, C.: Protocol Validation, Computer Networks and ISDN Systems, No.24, 1992.
- Grabowski, J; D. Hogrefe; D. Toggweiler: Partial Order Simulation of SDL Specifications, SDL Forum 1995, North-Holland, 1995.
- Holzman, G.: Design and Validation of Computer Protocols, Prentice-Hall, 1991.
- Phalippou, M.: TVEDA, IWPTS; 1994.
- Hogrefe & Brömstrup TESDL: Experience with Generating Test Cases from SDL Specifications, Proceedings SDL'89: The Language at Work, Lisbon, Portugal 1989.
- ITU-T Recommendation Z.110 (1989): "Criteria for the use and applicability of formal description techniques".
- ITU-T Recommendation X.209 (1988): "Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)".

History

Document history		
V1.1.1	February 1997	Membership Approval Procedure MAP 9716: 1997-02-18 to 1997-04-18