— AUTOLINK—

A Tool for Automatic Test Generation from SDL Specifications

Beat Koch, Jens Grabowski, Dieter Hogrefe and Michael Schmitt

Institute for Telematics, University of Lübeck
Ratzeburger Allee 160, D-23538 Lübeck, Germany
eMail: {bkoch, jens, schmitt, hogrefe}@itm.mu-luebeck.de

Abstract

Due to an increasing interest in SDL, MSC and TTCN based tools for validation and test generation, Telelogic AB, Malmö, and the Institute for Telematics of the University of Lübeck are cooperating in a research and development project aiming at bringing new test generation facilities to the TAU tool set. For that purpose, a software component has been developed which supports the automatic generation of TTCN test suites based on SDL and MSC specifications. The project follows a pragmatic approach and is driven by practical experience. AUTOLINK has been used by the *European Telecommunications Standards Institute* (ETSI) to develop a test suite for Core INAP CS–2.

1 INTRODUCTION

The standardized *Specification and Description Language* (SDL) [1], *Message Sequence Chart* (MSC) [3] and the *Tree and Tabular Combined Notation* (TTCN) [5] are three of the most popular standardized languages for system specifications and test descriptions. They have been used successfully in industrial projects and for standardization purposes [6, 7, 8, 9, 10].

TAU from Telelogic AB is one of the major commercial SDL, MSC and TTCN tool sets. It provides a complete environment for the development of SDL specifications, MSC descriptions and TTCN test suites. TAU includes graphical editors, various browsers, analyzers, simulation tools and code generators. The combination of SDL with the *Abstract Syntax Notation One* (ASN.1) as recommended by ITU-T in [2] is also supported.

Currently, TTCN test suites are created manually by test development specialists. Graphical tools exist which allow to fill out TTCN tables and check the syntactical correctness of a test suite. But transforming test purpose descriptions into semantically consistent test cases is not supported efficiently by any tool. Therefore, the development of a test suite is a time-consuming and expensive task. Moreover, experience has shown that due to their complexity, the quality of manually written test suites is often insufficient.

AUTOLINK is a research and development project of the University of Lübeck and Telelogic AB. The goal of the AUTOLINK tool is to simplify the test generation process in order to get

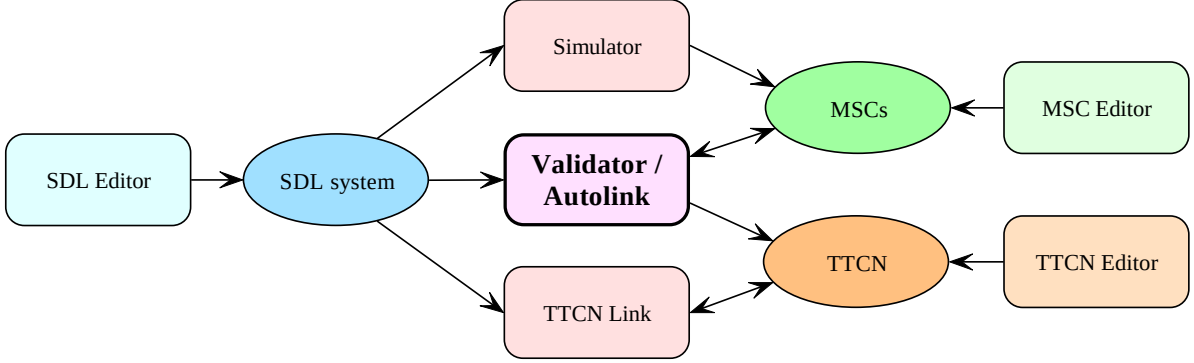

Figure 1: TAU tools used for automatic test generation

error-free test suites in less time. This is accomplished by the following main features:

- MSC diagrams are used to describe the interaction between the *System Under Test* (SUT) and the test equipment which is required to pass a test.
- AUTOLINK generates TTCN code which requires little manual post-processing. Therefore, the potential of inserting bugs into the test suite is reduced significantly.

AUTOLINK is available commercially as part of TAU version 3.2 since October 1997.

The paper is organized in the following manner. In Section 2, an overview of the TAU tools is given. The Validator in particular is introduced in Section 3. AUTOLINK is described in Section 4. In Section 5, practical experience of using AUTOLINK to generate a test suite for Core INAP CS–2 is presented. Future plans for AUTOLINK are listed in Section 6. Finally, a summary is given.

2 The TAU tool set

Telelogic's TAU package contains two tool sets: SDT on the one hand consists of SDL-related applications; ITEX on the other hand is used to work with TTCN test suites. The TTCN Link tool builds a bridge between SDT and ITEX. Figure 1 shows the tools which are important for automatic test generation, and the formal description languages which they use and support.

TAU contains *graphical editors* for SDL, MSC and TTCN. For all languages, the latest standards are implemented: The object-oriented features of SDL'92 are supported as well as MSC'96 with its high-level MSCs.

The *Simulator* for SDL specifications is equivalent to a debugger for a programming language. The user can stimulate the system by sending signals. The Simulator then allows to inspect the control flow, the exchange of signals between processes and the values of variables. Simulated paths through the SDL specification can be saved as MSCs. Such MSCs may be used as input for AUTOLINK.

The initial purpose of the *TTCN Link* tool has been the semi-automatic test generation. Within the TTCN editor of ITEX, the user can choose a send event which is appropriate for

the test purpose. Using a state space exploration, TTCN Link computes the corresponding receive events and adds them to the test case description. Then the user has to select the next send event.

AUTOLINK offers improved test generation functionality and is now used to generate the dynamic behavior and constraint tables of a test suite. However currently, TTCN Link is still needed to generate the declaration part. The following declaration tables are generated automatically:

- *Abstract Service Primitives* (ASP) type definitions. All SDL signals appearing on channels to the environment are translated into ASN.1 ASP definitions.
- *Points of Control and Observation* (PCO) declarations. All channels of an SDL specification to the system environment are considered to be PCOs.
- Structured SDL data definitions are translated into ASN.1 type definitions.

3 Validator

One of the main purposes of the Validator is to provide an automated fault detection mechanism which is able to detect dynamic and logical errors in an SDL system. Some of the potential problems are deadlocks, implicit signal consumptions and other dynamic errors like the sending of signals to non-existing processes.

The SDT Validator is based on state space exploration techniques (e.g. [12]). The state space of an SDL system is built up in form of a directed graph, called *reachability graph*. The reachability graph represents the behavior of the SDL system. Its nodes correspond to global system states, the edges represent the transitions between global system states.

Global SDL system states contain information about active process instances, variable values of all active processes, the SDL control flow state of all active processes, active procedures (with local variables), signals in the system's queues, active timers etc.

The edges of the reachability graph are annotated with SDL events. Depending on the configuration of the state space exploration, an edge may be annotated with a single SDL statement like a task, an input or an output, or with a list of SDL statements which may correspond to complete state transitions of SDL processes.

During validation, the reachability graph is analyzed. For example, a deadlock is found if a node in the graph does not have any outgoing edges.

MSC verification

Verification of a system against its requirements is another main purpose of the Validator [11]. The requirements can be expressed in form of MSC diagrams (see Figure 6 for an example of an MSC diagram). The Validator explores the state space and searches for a path in the reachability graph complying to the MSC which is checked. The MSC is "verified" if such a path exists.

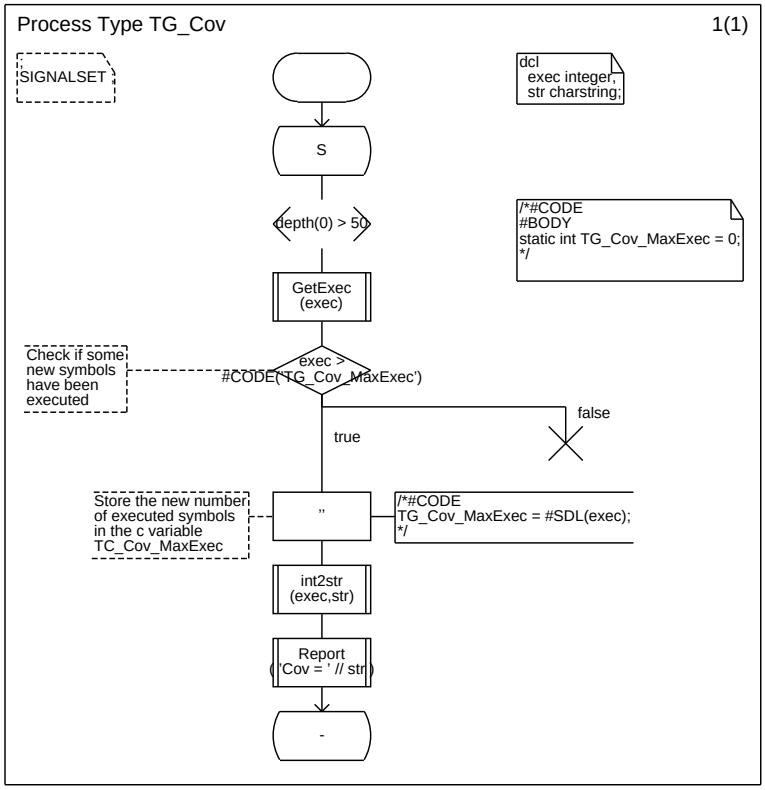

Figure 2: Observer process example

Observer processes

Observer processes allow to check more complex requirements on systems than can be expressed by MSCs. An observer process is a special kind of SDL process which is included in the SDL system. An observer process is able to inspect the SDL system without interfering with it. Several mechanisms are built into the Validator to accomplish this:

- The execution mode is changed if observer processes are defined: First, the SDL system executes one transition. Then all observer processes execute one transition during which they check the new system state.

- Through a special *Access abstract data type*, observer processes can examine the internal states of other processes in the system. Variable values, contents of queues etc. can be checked without any need to modify the observed processes.

During state space exploration, observer processes may generate *reports*. Reports contain a textual description and the path from the start state of the exploration to the state where the report was generated. This path can be stored as an MSC and used as input for AUTOLINK. A simple observer process type is shown in Figure 2. Each time a path with length 50 is found in the state space which covers an additional SDL symbol, a new report is generated. As shown in the example, some of the internal variables and functions of the Validator can be

accessed. That way, information about the current status of the state space exploration can be retrieved.

4 AUTOLINK

The objective of AUTOLINK is to provide an easy-to-use yet powerful tool to generate TTCN test suites from an SDL specification. Potential users are engineers who have a good understanding of the system they have built, but who cannot easily generate test cases because they have no detailed knowledge of TTCN.

Specialized test suite designers will also benefit from using AUTOLINK. They can concentrate on the correct description of test purposes while leaving the error-prone task of writing TTCN code to the tool.

4.1 Automatic test generation

AUTOLINK is part of the SDT Validator. It uses the state space exploration techniques and the MSC verification mechanism provided by the Validator.

The generation of a TTCN test case is based on a *path*. In the AUTOLINK context, a path is defined as a sequence of events which have to be performed in order to go from a start to an end state in the state space of the SDL specification. The externally visible events of a path describe the test sequence to which a TTCN *PASS* verdict is assigned.

Paths are stored as *system level MSCs*. A system level MSC shows the desired interaction between the *System Under Test* (SUT) and its environment during the execution of a test case. It consists of one instance for the SUT and one instance for every PCO, where every channel to the environment is considered to be a PCO. Using system level MSCs corresponds to the black box testing method, where the internals of the SUT are not known.

AUTOLINK uses a modified version of the MSC verification algorithm to compute all relevant transitions in the state space. Each transition is analyzed: Events which are visible at the environment are added to a special data structure (referred to as *Autolink tree*). If an event satisfies the MSC, it is added as a *PASS* event; if it violates the MSC, it is added as an *INCONCLUSIVE* event. Additionally, a constraint is created for every visible event. The AUTOLINK algorithm is divided into three distinct parts. These are:

1. *Initialization of data structures:* A new test case record is generated, the MSC is loaded etc.
2. *State space exploration:* The AUTOLINK tree is built and a list of constraints is constructed.
3. *Post-processing:* In some cases, the resulting AUTOLINK tree needs to be modified after the completion of the state space exploration. Furthermore, identical constraints are merged.

4.2 Constraint handling

Basically, a constraint with generic name is created automatically for every send and receive event in all test cases. From the point of readability of a test suite, this is far from optimal. Therefore, some special constraint handling mechanisms have been included in AUTOLINK.

Constraint merging

When processing several test cases consecutively, a lot of constraints are created. During post-processing of the test cases, each new constraint is compared with all previously created constraints. If there is a match, then the new constraint is removed and all references in the test case are updated. Usually, the number of constraints is reduced significantly through constraint merging.

Constraint description language

Early tests with AUTOLINK have shown that the assignment of generic names to constraints is not acceptable because the resulting test suite is hard to read.

AUTOLINK provides commands to rename constraints manually. But if a test suite has to be regenerated because of a change in the SDL specification, then the whole manual work is lost. Additionally, the number of similar constraints in a test suite can be quite large. Through constraint parameterization, this number can be reduced.

AUTOLINK includes a special *constraint description language*. By defining rules in a configuration file, the test designer can control the naming and parameterization of constraints. A typical constraint description rule may look like this:

```
TRANSLATE
  FROM       CONreq
  TO         "C_ConnectionRequest"
  PARAMETERS $1="Call_ID"
END
```

AUTOLINK translates *from* signals *to* constraints. The example above instructs AUTOLINK to assign the name *C_ConnectionRequest* to constraints which describe *CONreq* signals. Additionally, the first parameter of signal *CONreq* is used as a parameter of constraint *C_ConnectionRequest*.

The name of a constraint may depend on the textual description of signal parameters. In the following example, the generated constraint is called *C_EndReq_Normal* if the second parameter of signal *EndReq* is *norm*. It is called *C_EndReq_Exception* if the parameter equals *excp*.

```
TRANSLATE
  FROM       EndReq
  IF         $2 == "norm"
  TO         "C_" + $0 + "Normal"
  IF         $2 == "excp"
  TO         "C_" + $0 + "Exception"
END
```

Often, abbreviations are used for signal parameters. These abbreviations can be translated into more descriptive texts as shown in the example above. The constraint description language has been kept deliberately simple. Therefore, even an unexperienced user should be able to define rules quickly.

4.3 Direct MSC to TTCN translation

In order to use a state space exploration to generate test cases from MSCs, a complete SDL specification is required. However in the real world, only partial specifications exist for most systems; often there is no SDL specification at all. Standardized protocols like Core INAP CS–2 (see Section 5) cannot be specified completely, e.g. error handling has to remain unspecified. Nonetheless, to guarantee a uniform test suite development process, *all* test purposes should be formalized as MSCs.

AUTOLINK provides a function to translate MSCs directly into TTCN. Although it does not perform a state space exploration, AUTOLINK still needs information about the interface between the system and its environment. Therefore, a minimal SDL specification has to be written which defines at least the channels to the environment and the signals which are sent via these channels.

Obviously, there are some disadvantages if a direct MSC to TTCN translation is used. First, there is no guarantee that an MSC and the generated test case describe valid traces of the specification or the implementation respectively. Furthermore, events which lead to an INCONCLUSIVE test verdict cannot be computed.

Still, using the MSC to TTCN translation is much better than writing TTCN test cases manually:

- If several receive events are expected at different PCOs before the next send event, then alternative branches have to be written in TTCN for all permutations of these receive events.
- MSC to TTCN translated test cases are stored in the same intermediate format as test cases generated by a state space exploration. Therefore, constraint merging which is applied during post-processing (see Section 4.1) works identical for both kinds of test cases.

The resulting test suite is less likely to be incorrect, and it contains fewer constraints.

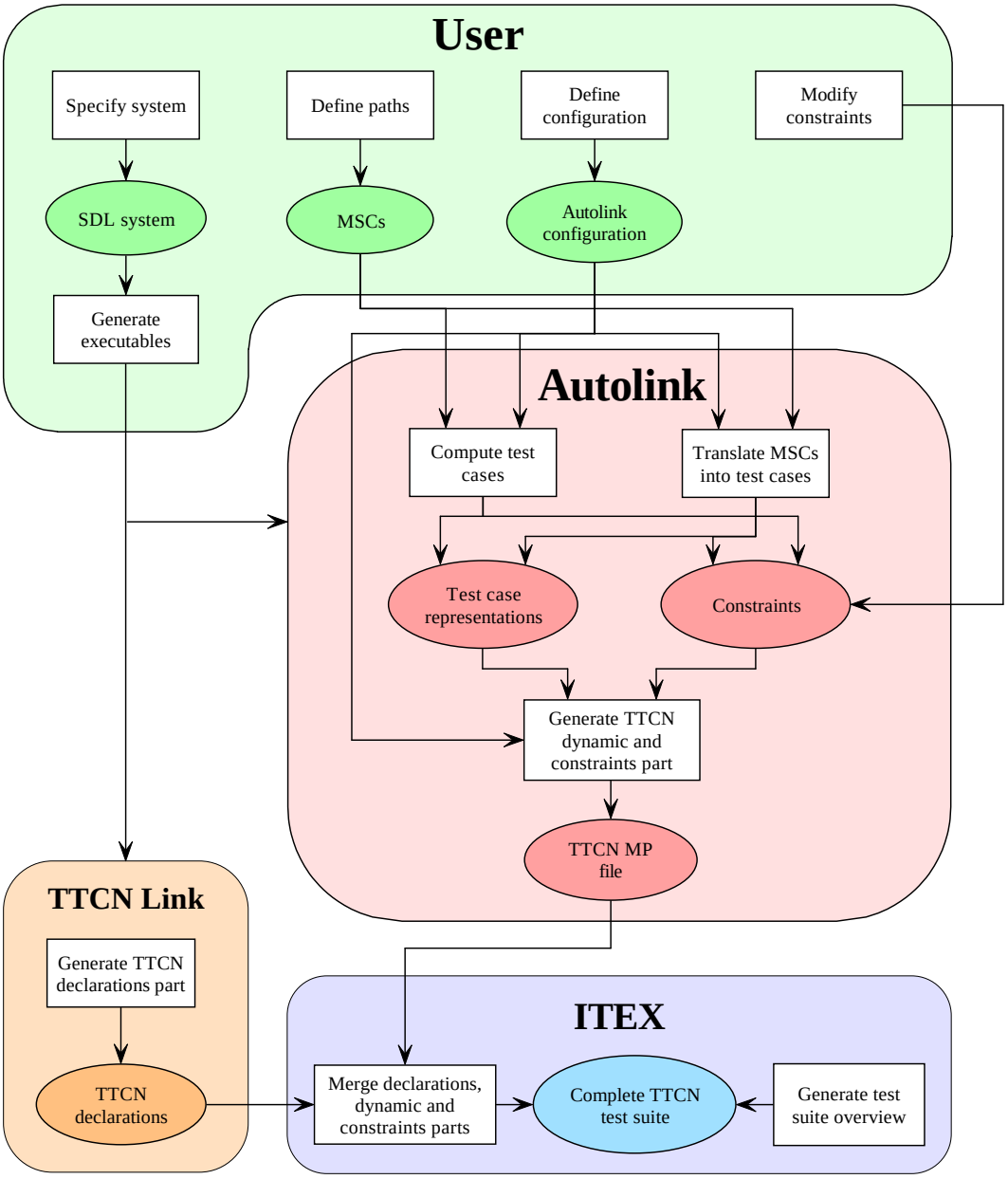

Figure 3: The test generation process

4.4 The test generation process

In this section, all steps are described which are necessary to create a TTCN test suite with the TAU tools. Figure 3 presents an overview of the test generation process.

SDL system specification

The starting point for automatic test generation is, of course, an SDL specification. When the specification is syntactically correct and also correct with respect to the static semantics, it can be converted into C code. This code in turn can be compiled and linked with separate kernel

libraries in order to generate a *Validator/Autolink* and a *TTCN Link* executable respectively. Whenever the SDL specification is changed, the executables have to be regenerated.

Path definition

A path describes the test purpose; it is stored as a system level MSC (see Section 4.1). There are several possibilities to define such an MSC:

- The path which results from a simulation run can be stored as an MSC. This is the most effective way to specify a path containing a given test purpose.
- A large number of MSCs can be generated automatically by using the Validator's random walk state space exploration in combination with observer processes (see Section 3). This method is used if a high coverage of system states is to be tested.
- With the Validator's navigation function, the state space can be explored manually. The resulting path can be saved as an MSC.
- An MSC can be drawn manually with the MSC editor.

The MSCs should be verified against the SDL system with the Validator before they are used as input for AUTOLINK.

Configuration

Test case generation is influenced by several options which have to be set before the processing is started. This can be done either by using the Validator graphical user interface or by writing an AUTOLINK *configuration file*. This configuration file may consist of up to three parts.

A problem which appears inevitably for complex, real-world SDL systems is the explosion of the state space. AUTOLINK provides a set of options which allow to reduce the state space during exploration:

- **Maximum search depth:** This should be restricted only if the SDL system can execute a large or unlimited sequence of internal events.
- **Channel queues:** Disabling internal channel queues strongly reduces the state space.
- **Priorities for classes of SDL events:** AUTOLINK distinguishes five classes of SDL events: Internal events, input from the environment, timeouts, channel outputs and spontaneous transitions. By assigning a higher priority to input events, the complexity of the state space can be reduced.
- **Process scheduling:** In every system state there is a queue of all processes ready to execute next. The process scheduling option defines if only the first or all processes in the queue are allowed to execute. Choosing the former results in the most effective reduction of the state space.

State space exploration options may be set in the first part of the AUTOLINK configuration file. The definition of constraint parameterization and naming conventions (see Section 4.2) may be provided in the second part. In a third part, the user may set options which control the format of the test suite:

- Constraints can be saved either as ASN.1 *Protocol Data Unit* (PDU) constraints or as ASN.1 ASP constraints.
- TTCN test steps can be stored either *globally* in the test steps library, as *local* trees in the test case dynamic behavior table, or they can be inserted directly into the test case (*inline* format).

Test case generation

When the SDL system, the MSCs describing the test purposes and the configuration file are created, then the test case generation can start. An MSC can either be used as input for test generation by state space exploration, or MSC to TTCN translation. No matter which method is chosen, the processed test cases are stored in a list of AUTOLINK trees. This list can be manipulated: More test cases can be added with subsequent test case generations; unwanted test cases can be removed. AUTOLINK trees can be previewed, saved in a file and reloaded.

The save and load functionality can be used to distribute test case processing on more than one computer: A simple shell script running on all machines determines the next unprocessed MSC. It then starts an AUTOLINK process with this MSC as input. The resulting test case is saved; all test cases can later be merged into a single test suite.

Constraint modification

During test case generation, a list of constraints is built automatically. Constraints can be added manually or removed if they are not referenced in a test case. Constraints can be renamed, stored in a file and loaded back.

Experience has shown that only little manipulation of constraints is necessary if constraint parameterization and naming rules are defined in the AUTOLINK configuration file.

TTCN generation

Using the information stored in the AUTOLINK tree and constraints lists, the dynamic behavior tables and the constraint tables of a test suite can be stored in a TTCN MP file. If desired, the saving of the test suite can be repeated with different formats for test steps and constraints.

Test suite completion

The TTCN MP file with the dynamic behavior and constraint parts can be imported in the ITEX tool. With a call of the TTCN link executable, the declarations part can be added. Finally, the test suite overview can be generated automatically.

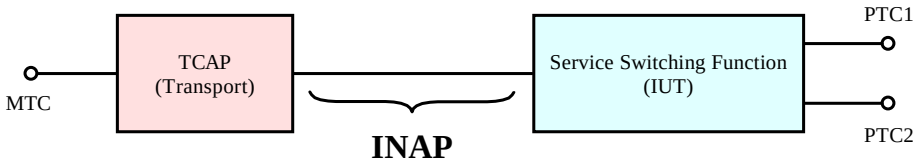

Figure 4: Core INAP CS–2 test architecture

5 Practical experience

The AUTOLINK approach and the AUTOLINK tool have proven to be usable for industrial applications by their extensive use within the European Telecommunications Standards Institute (ETSI) for the production of the conformance test suite for Core INAP CS–2[1][15]. The SDL specification of Core INAP CS–2, attached as annex A to [15], gives the normative requirements of the protocol behavior; it uses the ASN.1 definitions of the INAP PDUs. Therefore, it provides a complete specification of the protocol which is suitable for the application of AUTOLINK.

5.1 INAP, test architecture and interfaces

INAP enables communication between a *Service Control Function* (SCF) and a *Service Switching Function* (SSF) in a public network. [2] A test suite is provided only for the SSF, where the *Implementation Under Test* (IUT) is either a local exchange or a transit exchange. INAP is implemented on top of the *Transaction Capabilty Application Part* (TCAP) in *Signalling System No. 7* (SS7) [13]. Therefore according to ISO/IEC 9646 [4], the remote test method is used with a *Main Test Component* (MTC) serving the normative interface as depicted in Figure 4.

The SDL specificaton contains both TCAP and the SSF. The protocol operations of INAP state normative requirements on what the SSF should do in terms of signalling, e.g. establishing a call to a third party, releasing a call etc. Therefore in most cases, the PASS verdicts in the test suite must be based on the signalling events. However, INAP is defined independently of the particular signalling system (e.g., national variants of the *ISDN User Part* (ISUP) can be used). Therefore it is not possible to provide the definitions for the *Parallel Test Components* (PTC) PTC1 (incoming signalling) and PTC2 (outgoing signalling) in the test suite. Instead, the coordination messages between the PTCs and the MTC are provided, which are generated from the SDL specification of INAP.

5.2 Test purposes

The test purposes for the Core INAP CS–2 test suite are specified informally by using a textual description and formally with MSCs. Therefore, the development of the test purposes was done

[1] The abbreviation Core INAP CS–2 refers to the ETSI standard of the Intelligent Network Application Protocol (INAP) Capability Set 2 (CS–2) [15].

[2] For an introduction to intelligent networks and its architecture see, e. g., [14].

IN2_A_BASIC_AT_BV_01	
Purpose:	Test of **ActivityTest** in WaitForInstructions state
Requirement ref	
Preamble:	O_OS
Selection Cond.	
Test description	**ActivityTest** invoke sent by SCF to SSF with TCAP DialogueId of dialogue identical to the one used in the preamble
Pass criteria	**ActivityTest** result sent by SSF to SCF related to the existing dialogue
Postamble:	SigConA_Release .

Figure 5: Informal test purpose description

in the following two steps:

1. Based on the protocol requirements, the test purposes were identified manually and documented in tables which structure the informal text. As shown in Figure 5, the table entries may refer to pre- and postambles, describe the pass criteria and may provide further information.

2. By simulation of the Core INAP CS–2 SDL specification, MSCs were generated for all test purposes. The generated MSCs provided the input for the AUTOLINK tool and were also included in the test purpose document [16]. They give a more formal definition of the test purposes. The inclusion of the MSCs was a requirement from organisations which do not use TTCN for testing, but which still need a formal description of each test purpose.

Figure 6 shows an MSC test purpose example which formalizes the test purpose of Figure 5. The MSC refers to the preamble O_OS and the postamble SigConA_Release, which are also described by MSCs. The name of the test purpose MSC, IN2_A_BASIC_AT_BV_01, corresponds to the test case name; the names of pre- and postamble MSCs correspond to test step names.

An advantage of creating test purpose MSCs by simulation is that the consistency between the informally developed test purposes and the protocol is guaranteed. A number of errors in the manually developed test purposes were detected with this method.

5.3 Detailed description of the development of MSC test purposes

In this section, a more detailed view of the development of the MSC test purposes is presented. The development comprised some preparatory work and the MSC generation process.

Preparatory work

The MSC test purposes were generated by means of simulation. In order to facilitate their generation, the user interface of the SDT Simulator was adapted to the specific needs of the MSC test purpose generation for the Core INAP CS–2 protocol.

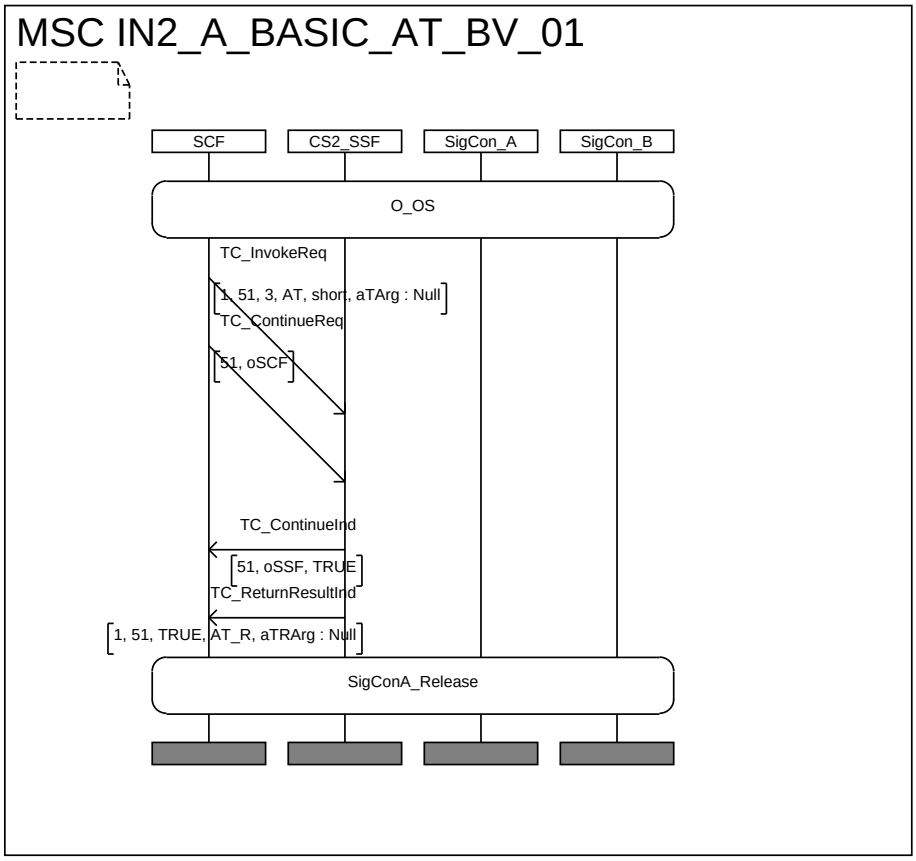

Figure 6: Formal MSC test purpose description

During the development of the MSC test purposes, the SDL specification of Core INAP CS–2 and the test purposes were also validated. For the validation, a more detailed view on the protocol behavior was needed than for the test purpose development itself. Easy switching between the different views was enabled by assigning the setting of corresponding Simulator options to a set of buttons.

Pre- and postambles were reused in several test cases. In order to ease their execution, the necessary actions were stored as command scripts and attached to a button. By pressing the button, the Simulator was forced to perform the actions to execute the corresponding pre- or postamble.

Before test generation started, the naming conventions and the parameterization of the PDU and coordination message constraints was defined in an AUTOLINK configuration file. For each message, a rule to determine the constraints name and a resonable set of message components to be parameterized in the constraints was identified.

MSC generation process

The generation of the MSC test purposes was a two phase process. It comprised a validation phase and a development phase. In the first phase, an MSC test purpose to be produced was executed, i.e. simulated, in order to validate the test purpose *and* the SDL specification. If

the expected behavior did not match with the validation run, then the test purpose, the SDL specification or both had to be changed or corrected. The validation step was repeated until the test purpose and the SDL model were in accordance.

In the second phase the test purpose was simulated again by performing the following steps:

1. Start the Simulator.
2. Set the Simulator options according to the needs of the MSC test purpose generation. This was done by using the appropriate predefined button.
3. If a preamble is required, execute it by using a predefined button.
4. Start the MSC trace. As a result, the rest of the simulation is displayed and recorded as an MSC.
5. If a preamble was performed, insert an MSC reference referring to a preamble MSC into the displayed MSC trace. This was done manually.
6. Simulate the test purpose.
7. If necessary, append an MSC reference referring to a postamble MSC to the displayed MSC trace. This was done manually.
8. Rename the MSC trace to the test case name.
9. Store the MSC trace.

The MSCs describing the pre- and postambles of the test cases (steps 5 and 7) were generated like normal MSC test purposes. But this was done only once for the whole test suite. The postambles only needed to be referred to, but not to be executed. During test case generation, their correct execution would be checked by the AUTOLINK tool. With some practice, it took the test developer only a few seconds to perform the actions 1–5 and 7–9. Step 6 took a little longer. Although the development phase of the MSC test purpose generation process appears to be more complicated, experience has shown that it takes less time than the validation phase.

5.4 Ensuring consistency between MSC test purposes and the SDL model

During the validation phase of the MSC test purpose generation process, the SDL specification had to be corrected and modified several times. This changed the behavior of the SDL specification and the already developed MSC test purposes became invalid. In order to detect invalid MSCs after each change of the SDL model, the MSC test purposes were revalidated against the SDL model. This was done automatically during night or weekends by using a UNIX script. For each MSC test purpose, the script started the TAU Validator in the command mode (without graphical user interface) and started an MSC verification. Due to the complexity of the SDL model, the validation of all MSCs took some time. To reduce it, MSC test purposes were validated in parallel on several computers.

5.5 Manual MSC test purposes

The SDL specification of the Core INAP CS–2 protocol does not include error handling and due to standardization politics, some of the protocol functions are only specified rudimentary. The MSC test purposes related to these protocol aspects were specified manually in order to apply the direct MSC to TTCN translation feature of AUTOLINK. These manual MSC test purposes look like the ones generated by state space exploration; they were also included in the test purpose document [16].

For these manual MSC test purposes, AUTOLINK could not validate the correct order of the exchanged PDUs and coordination messages. But it did check if their format and parameter values corresponded to the interface definition of the SDL specification.

5.6 TTCN test cases

State space exploration was used by AUTOLINK to generate TTCN test cases for the MSC test purposes created with simulation. The manual MSC test purposes were translated directly into TTCN code. Apart from the fact that the test cases related to the manual MSC test purposes do not include event sequences leading to an INCONCLUSIVE verdict, all test cases look very similar. Many constraints are shared by several test cases, because the merging of identical constraint definitions for different test cases was done automatically.

By using the MSC in Figure 6 as input, AUTOLINK generated the TTCN test case shown in Figure 7. A constraint for the *SetupReq* coordination message is shown in Figure 8.

5.7 Postprocessing of TTCN test cases

The postprocessing of the TTCN test cases involved replacing some values of the constraint parameters with *Protocol Implementation eXtra Information for Testing* (PIXIT) parameters and the addition of the timer *tSSF*. Both tasks were performed automatically with a UNIX script operating on the TTCN file; a test case after post-processing is shown in Figure 9. The result was manually inspected for consistency, especially concerning the timer handling, and necessary modifications were performed.

5.8 Remarks and expenses

This was the first time that ETSI used a test generation tool like AUTOLINK for the production of a normative conformance test suite. Therefore in the beginning, considerable resources were spent to install and refine the tools, the scripts and the methodology. Time was also spent to update the Core INAP CS–2 SDL specification and the already developed test purposes due to changes to the protocol made by the responsible ETSI technical body SPS3. It has to be noted that the development of the test suite started before the protocol was completed. This allowed the detection of errors during the test purpose development which could be corrected in the protocol before its approval.

At the time of writing this paper, the ETSI experts have developed 263 test purposes and generated the corresponding TTCN test cases. 188 test purposes were generated by simulation,

Test Case Dynamic Behaviour

Test Case Name : IN2_A_BASIC_AT_BV_01
Group :
Purpose :
Configuration :
Default : OtherwiseFail
Comments :

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+O_OS_2			
2		SCF ! TC_InvokeReq	CIR_ActivityTest_002(1 , 51)		
3		SCF ! TC_ContinueReq	C_TC_ContinueReq_001(51)		
4		SCF ? TC_ContinueInd	C_TC_ContinueInd_001(51)		
5		SCF ? TC_ReturnResultInd	CRR_ActivityTestResult(1 , 51)	(PASS)	
6		+SigConA_Release_2			
7		SigCon_B ? SetupReq	C_SetupReq({ callRef 2, calledPartyNumber '2000'H, callingPartyNumber '1000'H })	INCONC	
8		SCF ? TC_AbortInd	C_TC_AbortInd(51)	INCONC	
9		SigCon_B ? SetupReq	C_SetupReq({ callRef 2, calledPartyNumber '2000'H, callingPartyNumber '1000'H })	INCONC	

Detailed Comments :

Figure 7: Test case created by AUTOLINK

ASN.1 ASP Constraint Declaration

Constraint Name : C_SetupReq(callRef : SetupIRType)
ASP Type : SetupReq
Derivation Path :
Comments :

Constraint Value

{ setupIRType1 callRef }

Detailed Comments :

Figure 8: Constraint created by AUTOLINK

75 test purposes were developed manually. The test purpose and test case production for several Core INAP CS–2 protocol functions is not yet finished.

ETSI estimated an overall reduction of the expenses for the tool-assisted development of the first Core INAP CS–2 test suite of 20 % compared to manual test suite development [18]. This first test suite was delivered in April 1998 [17].

Test Case Dynamic Behaviour

Test Case Name : IN2_A_BASIC_AT_BV_01
Group :
Purpose :
Configuration : CS1_CONFIG
Default : OtherwiseFail
Comments :

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+O_OS			
2		SCF ! TC_InvokeReq	CIR_ActivityTest_002(PIX_InvokeId1 , Tsv_DialogId1)		
3		SCF ! TC_ContinueReq	C_TC_ContinueReq_001(Tsv_DialogId1)		
4		SCF ? TC_ContinueInd	C_TC_ContinueInd_001(Tsv_DialogId1)		
5		SCF ? TC_ReturnResultInd	CRR_ActivityTestResult(PIX_InvokeId1 , Tsv_DialogId1)	(PASS)	
6		+SigConA_Release_2({ callRef 2, calledPartyNumber PIX_CalledPartyNumber1_SetupInd , callingPartyNumber PIX_CallingPartyNumber1 })			
7		SigCon_B ? SetupReq	C_SetupReq({ callRef 2, calledPartyNumber PIX_CalledPartyNumber1_SetupInd, callingPartyNumber PIX_CallingPartyNumber1 })	INCONC	
8		SCF ? TC_AbortInd	C_TC_AbortInd(Tsv_DialogId1)	INCONC	
9		SigCon_B ? SetupReq	C_SetupReq({ callRef 2, calledPartyNumber PIX_CalledPartyNumber1_SetupInd, callingPartyNumber PIX_CallingPartyNumber1 })	INCONC	

Detailed Comments :

Figure 9: Test case after post-processing

6 Future plans

Future enhancements of AUTOLINK will focus primarily on the readability of the generated TTCN test suites. The goal is to reduce the amount of time needed for manual post-processing. The following problems will be addressed:

- **Support of concurrent TTCN:** A testing environment may consist of a number of Parallel Test Components (PTCs) which are controlled by a Main Test Component (MTC). In this case, test case dynamic behavior descriptions have to be split into separate descriptions for each PTC. Additionally, the test components have to be synchronized.

  AUTOLINK will allow to generate concurrent TTCN automatically. The user will have to provide only the information which cannot be derived from the SDL specification, e. g. the relation of PTCs to PCOs. Coordination messages which have to be exchanged between the test components may be computed by AUTOLINK using predefined rules,

but may also be defined explicitly in the MSCs.

- **Parameterization of test steps:** If a postamble is used in several test cases, it may result in a series of basically identical TTCN test steps differing only in the constraints of some receive events. Similar to constraints, test steps may be parameterized. This results in a reduced number of test steps. Possible parameters for test steps include signals, signal parameters and PCOs.
- **SDL-based test generation:** Normally, test cases start and end in a *stable testing state* where the system has to wait for a stimulus before it can continue. In a future version, AUTOLINK will assist the user in finding stable testing states and in defining preambles and postambles. It will also allow to produce MSC test case descriptions automatically based on the symbol coverage criterion.
- **Test suite structure:** Test cases are typically combined in a hierarchy of *test groups*. Each test group focuses on a specific aspect of the specification or the implementation respectively. Currently, the test suite structure has to be specified manually after test generation. However, the test suite structure may already be reflected in the names of the given MSCs. Therefore a mechanism will be implemented which groups test cases based on the names of their MSC descriptions.
- **Optimization of constraint parameterization:** In order to minimize the number of constraints and the size of the test suite, a mechanism to find the optimal constraint parameterization will be implemented. Since an automatic parameterization may not necessarily result in better readability, AUTOLINK may only make suggestions for constraint parameterization which have to be approved by the user.
- **PICS/PIXIT parameterization:** SDL does not support the use of symbolic parameters; during test generation, actual values have to be used for signal parameters. AUTOLINK will include a function which replaces concrete values with PICS/PIXIT parameters during post-processing. The user will be able to specify PICS/PIXIT parameters in the configuration file.
- **Support of timers:** There are three ways to use timers in a test case. First, a timer may be set at the beginning of a test case. It checks whether the complete test case can be executed within a given amount of time. Timers may also be set before each receive event in order to restrict the time allowed for a single signal. In both cases timers can be written automatically in a TTCN test suite. A more selective method is to define timers explicitly at the environment axes in the MSCs and to translate them into TTCN timers.
- **Measurement of complexity:** States space exploration options have a major impact on the computation time of test cases. In fact, complex SDL specifications can only be explored reasonably with certain restrictions. Currently, there is no help for the user to choose the optimal set of options.

  The complexity of a particular SDL specification can be measured by a number of sim-

ulations with different option settings. An automatic comparison of the results provides information about a reasonable choice of exploration options.

7 Summary

AUTOLINK is a tool for the automatic generation of TTCN test cases from an SDL specification and MSC test purpose definitions. It follows a pragmatic approach to test generation. Thanks to enhanced functions for the naming of constraints, the generated test suites require little manual post-processing. AUTOLINK has been used at ETSI to create a test suite for the INAP CS–2 protocol, where it has proven its efficiency. Feedback from practical experience and theoretical research are the basis for future development of AUTOLINK.

Acknowledgments

The authors like to thank Stefan Heymer for proof-reading.

References

[1] ITU-T Rec. Z.100 (1996). *Specification and Description Language (SDL)*. Geneva, 1996.

[2] ITU-T Rec. Z.105 (1995). *Specification and Description Language (SDL) combined with Abstract Syntax Notation One (ASN.1)*. Geneva, 1995.

[3] ITU-T Rec. Z.120 (1996). *Message Sequence Chart (MSC)*. Geneva, 1996.

[4] ISO/IEC. *Information Technology – OSI – Conformance Testing Methodology and Framework*. International ISO/IEC multipart standard No. 9646, 1994.

[5] ISO/IEC. *Information Technology – OSI – Conformance Testing Methodology and Framework – Part 3: The Tree and Tabular Combined Notation (TTCN)*. ISO/IEC IS 9646-3, 1996.

[6] O. Færgemand, M. M. Marques (editors). *SDL'89: The Language at work*. North-Holland, 1989.

[7] O. Færgemand, R. Reed (editors). *SDL'91: Evolving Methods*. North-Holland, 1991.

[8] O. Færgemand, A. Sarma (editors). *SDL'93: Using Objects*. North-Holland, 1993.

[9] R. Bræk, A. Sarma (editors). *SDL'95 with MSC in CASE*. North-Holland, 1995.

[10] A. Cavalli, A. Sarma (editors). *SDL'97 – Time for Testing – SDL, MSC and Trends*. Elsevier, 1997.

[11] A. Ek. *Verifying Message Sequence Charts with the SDT validator*. In [8].

[12] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International, Inc., 1991.

[13] ITU-T Rec. Q.771-775 (1993). *Signalling System No. 7 – Transaction Capabilities.* Geneva, 1993.

[14] J. Thörner. *Intelligen Networks.* Artech House, 1994.

[15] DEN 03038-1. *ETSI Core INAP CS–2; Part 1: Protocol Specification.* European Telecommunications Standards Institute, 1997.

[16] DEN 03038-3. *ETSI Core INAP CS–2; Part 3: Test Suite Structure and Test Purposes specification for Service Switching Function (SSF), Specialized Resource Function (SRF) and Service Control Function (SCF).* European Telecommunications Standards Institute, 1998.

[17] DEN 03038-4. *ETSI Core INAP CS–2; Part 4: Abstract Test Suite (ATS) for Service Switching Function (SSF), Specialized Resource Function (SRF) and Service Control Function (SCF).* European Telecommunications Standards Institute, 1998.

[18] ETSI TC MTS STF 99. *CATG Handbook. DTR/MTS-00030-3.* Temporary document for the ETSI TC MTS meeting in Sophia-Antipolis (France), March 1998, to be published as European Guide in 1998.