Autolink - Putting SDL-based test generation into practice

M. Schmitt[†], *A. Ek*[*], *J. Grabowski*[†], *D. Hogrefe*[†], *B. Koch*[†]
[†] *Medical University of Lübeck, Institute for Telematics*
Ratzeburger Allee 160, 23538 Lübeck, Germany
email: {schmitt, koch, grabowsk, hogrefe}@itm.mu-luebeck.de
[*] *Telelogic AB, P.O. Box 4128, S-20312 Malmö, Sweden*
email: anders.ek@telelogic.com

Abstract

AUTOLINK is a tool for automatic test generation. It allows to generate TTCN test suites based on a given SDL specification and MSC requirements. The first big challenge for AUTOLINK was the creation of a test suite for the *Intelligent Network Application Protocol* at ETSI. In this paper we discuss our experience in applying AUTOLINK to a real-life protocol and the improvements of AUTOLINK which were developed during this project. We also present future enhancements which will further ease the work of test suite developers.

Keywords

Autolink, SDL, MSC, TTCN, SDT, ITEX, Test generation

1 INTRODUCTION

In recent years, several formal methods have been developed for automatic test generation. However, when putting these methods into practice, many test generation tools fail due to implementation-specific restrictions. Only a few promising reports are presented in literature for real-life protocols, see e. g. [5, 6, 13, 17].

AUTOLINK is a research and development project which aims at tackling this problem. It has been started in 1996 by the Institute for Telematics in Lübeck (Germany) and Telelogic AB in Malmö (Sweden). AUTOLINK is part of Telelogic's TAU development environment. TAU provides tools for the design, analysis and compilation of systems and protocols specified in *SDL* (*Specification and Description Language*) [2, 8], *MSC* (*Message Sequence Chart*) [3] and *TTCN* (*Tree and Tabular Combined Notation*) [16]. It supports the object-oriented features of SDL'96 and also allows the combined use of SDL with *ASN.1* (*Abstract Syntax Notation One*) as defined in ITU-T Recommendation Z.105 [1].

An SDL specification which makes use of both the object oriented features and ASN.1 data descriptions was developed for the *Intelligent Network Application Protocol* (*INAP*). Based on the SDL specification and by using AUTOLINK, a TTCN test suite was created by a project team at the *European Telecommunications Standards Institute* (*ETSI*). Due to its complexity, INAP was a good example to demonstrate and verify the applicability of AUTOLINK to real-life systems. Feedback from the project has also directly influenced the development of AUTOLINK.

The rest of this paper is structured as follows: In Section 2, a short introduction is given to the general concepts of AUTOLINK and its embedding in the TAU tool environment. Sections 3 to 5 describe some aspects of AUTOLINK that have been of particular relevance for the construction of the INAP test suite. Section 3 discusses the influence of state space exploration heuristics on test case generation. A direct translation from MSC to TTCN which does not perform a state space search is motivated in Section 4. Section 5 introduces a language which allows to describe constraint naming conventions and parameterization. INAP and some test generation results are presented in Section 6. Finally, a summary and outlook is given in Section 7.

2 THE AUTOLINK TOOL

AUTOLINK is a tool which supports the automatic generation of TTCN test suites based on SDL specifications. Its basic concepts have already been documented in [7] and [18].

AUTOLINK has been influenced by the SAMSTAG method and tool [12, 19]. SAMSTAG was an experimental system, developed at the University of Berne together with Swisscom. It was applied successfully to large scale protocols, e. g. [13].

2.1 Integration into the Tau tool set

AUTOLINK is tightly integrated within the TAU tool family which comprises the well-known SDT and ITEX tools. AUTOLINK is a component of the SDT Validator. The Validator is based on state space exploration techniques and can be used to find dynamic errors and inconsistencies in SDL specifications. Additionally, it allows to verify an SDL system against requirements described by MSCs. AUTOLINK makes use of the core functionalities provided by the Validator and extends it with respect to test generation facilities. Besides AUTOLINK, some other TAU tools are involved in the generation of a complete TTCN test suite. Figure 1 shows the relations between all tools and information involved in the test generation process.

Based on an SDL system which is specified by the user (Task 1 in Figure 1)

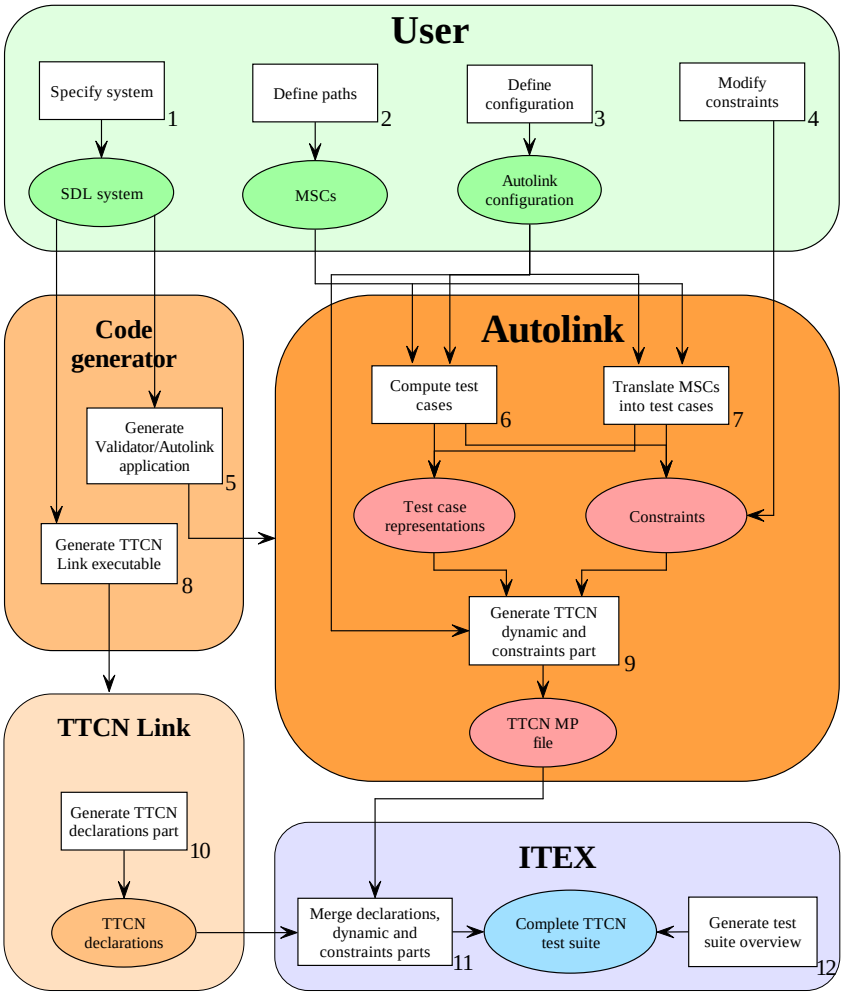

Figure 1 AUTOLINK and its integration into the TAU tool family

the code generator produces both an AUTOLINK/Validator and a TTCN Link application (Task 5 and Task 8).

Using AUTOLINK, a TTCN test suite can be generated which contains constraint and dynamic behavior tables (Task 9). This test suite can be completed and refined in ITEX, a development environment for test suites specified in TTCN. The TTCN Link application derived from the SDL specification is able to generate all static TTCN declarations (Task 10). These declarations can be merged with the AUTOLINK test suite (Task 11). Finally, the test suite overview can be generated automatically by ITEX (Task 12).

2.2 Developing a test suite with Autolink

The generation of a TTCN test suite with AUTOLINK involves several steps which are described below:

Define paths AUTOLINK derives test cases from *paths* which have to be provided by the user (Task 4 in Figure 1). A path is a sequence of SDL events which drive the system from a start state to an end state in the state space of the SDL system.

The SDT Validator provides several possibilities to define paths. For example, a path may be generated automatically by using observer processes. An observer process is a special kind of SDL process which is able to monitor the SDL system and guide a state space exploration. Observer processes can be used to define large sets of tests. Alternatively, the user may want to manually navigate in the state space and select single paths.

A path is stored as a Message Sequence Chart. Typically, an MSC used by AUTOLINK may only show the externally observable interaction of the SDL system with its environment. It consists of one instance axis representing the SDL system and one instance axis for each channel linked to the environment.

TTCN test cases can be logically structured into *test steps*, e. g. a preamble, a test body and a postamble. AUTOLINK represents test steps in MSCs by MSC references. A typical MSC for INAP is shown in Figure 2. It contains a preamble named *O_OS* and a postamble named *ReleaseCallAB_Cause_00*.

Define configuration A test suite produced by AUTOLINK depends on a number of options and settings. For example, the user may choose between several output formats for test steps. Test steps can be stored globally in the test step library, as local trees attached to a test case, or inline. In order to collect all relevant settings in one place, a test configuration file can be written (Task 3) which may include information about the output of test steps, search heuristics (Section 3), constraint naming and constraint parameterization (Section 5).

Process test cases Based on the MSCs and the configuration file provided by the user, AUTOLINK computes an internal representation for each single test case. The representations contain all sequences of send and receive events that lead to a *pass* or an *inconclusive* verdict. Additionally, it keeps track of the test case structure, i. e. the embedding of test steps.

There are two different approaches to generate test cases from MSCs. Normally, a state space exploration is started which simulates both a given MSC and the SDL system (Task 6). In this case, alternative receive events which violate the MSC but are valid according to the SDL specification are added to the test case representation with a TTCN *inconclusive* verdict. If a state space exploration is not applicable, MSCs have to be translated directly into test cases (Task 7; Section 4).

Send and receive events in a test case are associated with constraints cod-

ifying the signal parameters. Since constraints can be shared among several events in different test cases, they are stored separately from the test case representations. AUTOLINK merges identical constraints automatically and resolves naming conflicts. In addition, the user is allowed to define new constraints and to rename, merge and remove existing constraints (Task 4).

Generate TTCN test suite Based on the internal test case representations and the list of constraints, a TTCN test suite in MP format can be generated (Task 9). The appearance of the dynamic behavior and constraint tables can be controlled by various options defined in the configuration file. For example, constraints can either be stored as ASN.1 PDU or as ASN.1 ASP constraints. Before writing a test suite in a file, AUTOLINK checks the consistency of the test cases. For example, a postamble which is represented by an MSC and which is used for more than one test case description does not necessarily result in identical test steps. In this case, AUTOLINK has to distinguish the test steps by renaming them.

With respect to the *Framework of Formal Methods in Conformance Testing* [4], AUTOLINK uses trace preorder for relating a TTCN test suite to an SDL specification. This means that in the best case, AUTOLINK produces all possible traces from a specification and transforms them into test cases in TTCN. However, this is only an ideal scenario. In practice, the number of possible traces is much too large to be seriously considered. The MSCs are used to constrain the test cases to those which are considered relevant for testing the most important functions and to those which exhibit most likely an error.

3 THE STATE SPACE EXPLORATION

For test case generation, AUTOLINK performs a state space exploration based on the well-known bit-state algorithm [15]. Therefore it also has to cope with the state space explosion problem. In [14], Grabowski et al. list several *heuristics* to deal with the complexity of state space exploration algorithms. Heuristics make assumptions about the system. They avoid the analysis of system traces which do not comply to these assumptions.

The SDT Validator and hence also AUTOLINK allow to set several state space exploration options which can be related to heuristics. However, there are some options which definitely prevent AUTOLINK from generating all sequences of test events which lead to a *pass* verdict. Therefore some options are fixed, while others may be changed. Some relevant options are listed below:

- **Channel queues**
  Channel queues drastically contribute to the state space explosion. Therefore, deviating from the SDL standard, the SDT Validator and AUTOLINK allow to disable channel queues. However, queues should always be acti-

vated for all channels linked with the environment of the SDL system. Otherwise, most likely possible sequences of test events get lost.

- **Priorities of classes of SDL events**
  AUTOLINK allows to define priorities for five classes of SDL events: Internal events, input from the system environment, timeouts, channel outputs and spontaneous transitions. In the context of conformance testing, we assume that the tester is faster than the System Under Test (SUT). Therefore, when simulating the SDL system, input from the environment has highest priority. Since the number of inputs from the environment is limited by the MSC, this indeed reduces the complexity. Usually, SDL timers are used for exception handling. Therefore timeouts may be assigned a low priority at the risk of not finding all test events leading to an *inconclusive* verdict.
- **Process scheduling**
  In each system state, either all process instances in the ready queue are allowed to execute or only the first process instance. By using the second alternative, the state space is strongly reduced at the cost of not detecting any signal races.

Several other parameters can be adjusted, e. g. the maximum length of channel queues, the maximum search depth or the (in-)divisibility of SDL state transitions. Our experience with INAP has shown that it is essential to use the restricted process scheduling for complex SDL specifications. Additionally, internal channel queues have to be disabled.

4 TRANSLATING MSCS INTO TTCN TEST CASES

If a test purpose covers certain aspects of a protocol specification which are not represented in the corresponding SDL model, it is obviously not possible to generate a test case by starting a state space exploration. However, for a uniform test suite development process, it is desirable to formalize *all* test purposes as MSCs. Those MSCs which cannot be handled by a state space exploration should be converted directly into TTCN test cases.

AUTOLINK provides a function which performs a direct translation from MSC into TTCN. Figure 2 shows an MSC which has been constructed for INAP. The resulting TTCN test case generated by AUTOLINK is presented in Figure 3.

Although AUTOLINK does not need to perform a state space exploration, it requires some information about the interface of the specification. Therefore an SDL system has to be provided which at least defines the channels to the system environment and the signals sent via these channels. Using this SDL system, AUTOLINK can find out which MSC instances represent PCOs. Additionally, it can check whether the MSC is syntactically correct with regard to signals and signal parameters.

Direct translation of MSCs into TTCN test cases has to be applied with

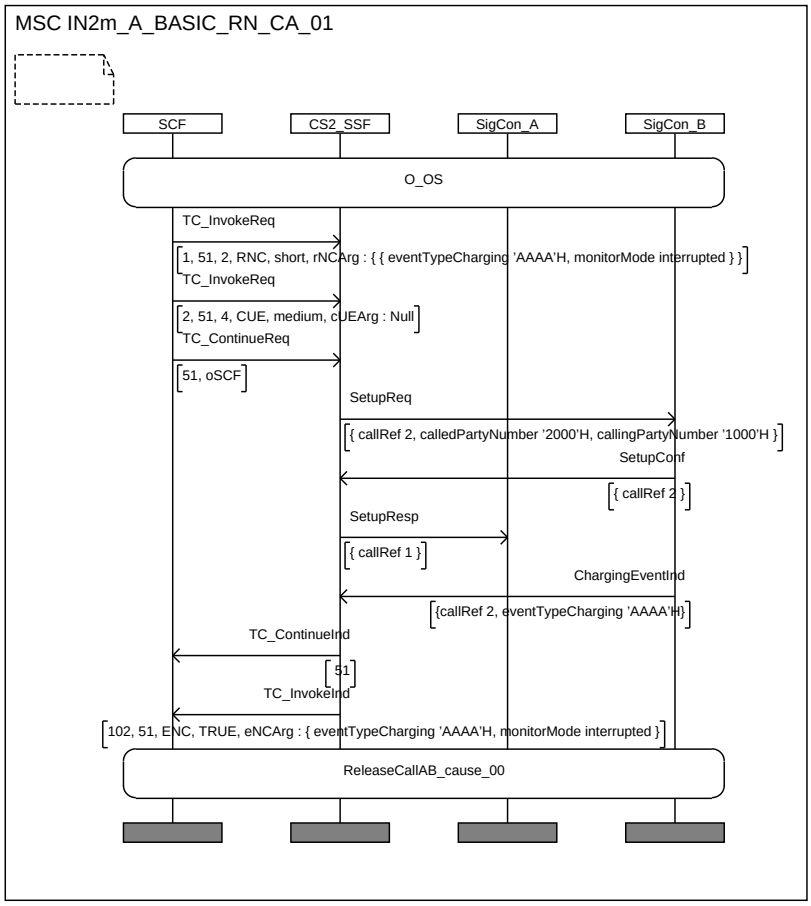

Figure 2 MSC *IN2m_A_BASIC_RN_CA_01* is translated ...

caution. There is no guarantee that the MSCs and hence the test cases describe valid traces of the specification or the implementation, respectively. Instead, AUTOLINK relies on the developer that the test cases are valid. Furthermore, is is not possible to compute test events which lead to an *inconclusive* verdict, meaning any deviation from the behavior described in the MSC is considered false.

On the other hand, there are good reasons to use MSCs instead of directly writing TTCN test cases. First, test cases typically span trees with several tree leaves because of the partial order of test events. For example, the test case in Figure 3 contains three valid sequences of test events. In MSCs the partial order is expressed inherently due to the semantics of MSC. While it is arduous for a test suite developer to write down a complete TTCN test case, AUTOLINK automatically computes all valid permutations of test events for a given MSC. Second, since AUTOLINK always translates MSCs into an intermediate internal

Test Case Dynamic Behaviour

Test Case Name : IN2m_A_BASIC_RN_CA_01
Group :
Purpose :
Configuration :
Default : OtherwiseFail
Comments :

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+O_OS			
2		SCF ! TC_InvokeReq	CIR_RequestNotificationCharging_002(1 , 51)		
3		SCF ! TC_InvokeReq	CIR_Continue_004(2 , 51)		
4		SCF ! TC_ContinueReq	C_TC_ContinueReq_001(51)		
5		SigCon_B ? SetupReq	C_SetupReq({ callRef 2, calledPartyNumber '2000'H, callingPartyNumber '1000'H })		
6		SigCon_B ! SetupConf	C_SetupConf({ callRef 2 })		
7		SigCon_B ! ChargingEventInd	C_ChargingEventInd_002		
8		SCF ? TC_ContinueInd	C_TC_ContinueInd_003(51)		
9		SCF ? TC_InvokeInd	CII_EventNotificationCharging_001(102 , 51)		
10		SigCon_A ? SetupResp	C_SetupResp({ callRef 1 })	(PASS)	
11		+ReleaseCallAB_cause_00			
12		SigCon_A ? SetupResp	C_SetupResp({ callRef 1 })		
13		SCF ? TC_InvokeInd	CII_EventNotificationCharging_001(102 , 51)	(PASS)	
14		+ReleaseCallAB_cause_00			
15		SigCon_A ? SetupResp	C_SetupResp({ callRef 1 })		
16		SCF ? TC_ContinueInd	C_TC_ContinueInd_003(51)		
17		SCF ? TC_InvokeInd	CII_EventNotificationCharging_001(102 , 51)	(PASS)	
18		+ReleaseCallAB_cause_00			

Detailed Comments:

Figure 3 ... into a TTCN test case.

test case representation, test cases generated by an MSC-TTCN translation can be merged with test cases generated by state space exploration. This leads to uniform and compact test suites with a reduced number of constraints.

5 CONSTRAINT RULES

Early tests with AUTOLINK have shown that the readability of automatically generated TTCN test suites is not very good. One particular problem is the naming of constraints. Due to the lack of information about the meaning of constraints, their names have to be created generically. For instance, a con-

straint may be named after its signal or the test case in which it is used. If there are different constraints with the same name, they might be distinguished by appending a sequence number.

In practice this naming scheme is not acceptable, even though AUTOLINK provides functions for the subsequent manipulation of constraints. Especially, if a test suite has to be regenerated due to a modification of the underlying SDL specification, a lot of manual work has to be repeated in order to assign meaningful constraint names.

Another important aspect is the parameterization of constraints. Without parameterization, a vast number of similar constraints is generated. This also makes the naming problem worse, since all these constraints have to get unique names.

For these reasons, AUTOLINK allows the user to specify rules which tell the tool how to map SDL signals onto TTCN constraints during the test generation process. Both the names of constraints and their parameterization can be controlled by these rules. The rules have to be provided in advance, i. e. before the test generation starts, as part of the configuration file.

A typical constraint rule looks like this:

Example 1

```
TRANSLATE
  FROM       TC_ContinueReq
  TO         "C_TC_ContinueReq"
  PARAMETERS $1="Dialog_ID"
END
```

Constraint rules can be considered as mapping rules: AUTOLINK translates *from* signals *to* constraints. Example 1 instructs AUTOLINK to map *TC_ContinueReq* signals onto constraints whose name is *C_TC_ContinueReq*. If more than one constraint is built, all constraints are distinguished by an additional sequence number. Moreover, the first parameter of each concrete signal (referred to by *$1*) becomes a parameter of the resulting constraint. The name of the formal parameter used in the constraint declaration table is *Dialog_ID*.

Constraint names may not only be composed of plain texts, they can also depend on signal parameters. However, in some cases it is not desirable to take the textual representation of a parameter value directly as part of a constraint name. E. g., a protocol engineer might use abbreviations as signal parameter values. But for the TTCN test suite, these abbreviations are intended to be mapped onto extended names.

In Example 2, the fourth parameter of signal *TC_InvokeReq* is taken as input for function *OpName*. Depending on its input value, the function returns a text which forms the second half of the constraint name. As a consequence, *TC_InvokeReq* signals with different fourth parameter are automat-

Example 2

```
TRANSLATE
  FROM       TC_InvokeReq
  TO         "CIR_" + OpName($4)
  PARAMETERS $1="Invoke_ID", $2="Dialog_ID"
END

FUNCTION OpName
    $1 == "ASF"   : "ActivateServiceFiltering"
 ...
  | $1 == "RC"    : "ReleaseCall"
 ...
  | $1 == "SL_R"  : "SplitLegResult"
  | TRUE          : "OperatorTypeNameUndefined"
END
```

ASN.1 ASP Constraint Declaration
Constraint Name: CIR_ReleaseCall(Invoke_ID : InvokeIDtype; Dialog_ID : DialogIDtype)
ASP Type : TC_InvokeReq
Derivation Path:
Comments :
Constraint Value
{ invokeIDtype1 Invoke_ID, dialogIDtype2 Dialog_ID, opClassType3 4, opCodeType4 RC, timeoutValType5 short, argType6 rCArg : initialCallSegment : '00'H }
Detailed Comments:

Figure 4 A parameterized constraint

ically mapped onto constraints with different names. A possible constraint declaration table for a *TC_InvokeReq* signal is shown in Figure 4.

Besides the constructs outlined above, AUTOLINK's constraint description language allows to define conditional rules. By using conditions in a `TRANSLATE` statement, constraints can be customized to specific requirements. For example, constraint parameterization can be guided by signal parameters. Moreover, it is possible to combine constraint rules for several signals by the use of regular expressions.

One goal for the design of the constraint description language was simplicity. Even an unexperienced user should be able to understand and define constraint rules. The syntax is not very strict in the sense that, for example, function parameters do not have to be declared. Instead, potential inconsistencies are checked and resolved at run-time.

The only data type used in the constraint description language is *text*. Any reference to a signal parameter returns a text. The same is true for function calls. Conditions are also evaluated on a textual basis.

Despite its simplicity, the language has proven to be sufficiently powerful. Nevertheless, it can be easily extended by additional built-in functions. One

restriction of the current implementation is that AUTOLINK can only refer to top-level signal parameters, i. e. it is not possible to address nested parameters. We plan to remove this limitation in a future release.

6 THE INAP EXAMPLE

The *Intelligent Network Application Protocol* [9] is the first protocol specified by ETSI for which a machine-processable SDL model is available. The SDL model was developed by ETSI Sub-Technical Committee SPS3 with support of the Protocol Expert Group and the Technical Committee 'Methods for Testing and Specification'.

The specification of INAP Capability Set 2 (CS–2) makes use of the object-oriented features of SDL'96 by inheriting CS–1. Data types are defined in ASN.1.

The SDL specification of ETSI's INAP CS–2 is voluminous. It comprises more than 450 pages in printed form. The phrase representation is about 1.6 MByte large (approximately 570 KByte without comments). When translating the specification into C with SDT's code generator, about 350 000 lines or 13.6 MByte of source code are generated.

6.1 Test suite generation

TTCN test suites for INAP CS–2 are developed by ETSI Specialists Task Force STF 100. A first test suite which covers the basic capability set, i. e. the CS–1 operations with CS–2 additions, has already been published in [11]. Another test suite covering the CS–2 operations is currently under development.

With respect to the CS–1 operations, test purposes were defined with textual descriptions and rough MSCs, first. Next, these test purposes were formalized as detailed MSCs using the SDT Simulator. In total, 126 test purposes were specified [10]. For 67 test purposes the MSCs could be simulated in order to produce the corresponding test cases. The remaining 59 test purposes had to be translated directly into TTCN due to unspecified parts in the SDL model.

The test suite resulted from a repetitive process of SDL/MSC refinements and modifications, MSC verifications and test generation runs. Whenever a modification of the SDL model was made, all MSCs were verified with the SDT Validator. If errors emerged, the SDL model or the MSCs were modified again until all MSCs passed the verification. Thereafter the test case generation using AUTOLINK was started.

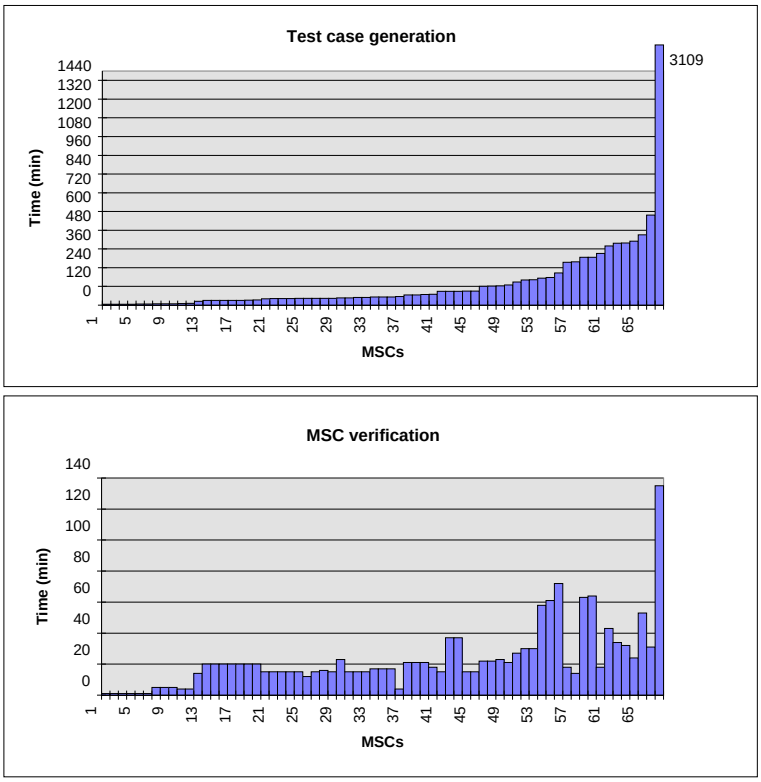

Figure 5 Computation time for MSC verifications and test generations

6.2 Statistics

Both the MSC verification and the test generation runs were executed at the Institute for Telematics in Lübeck. The test results discussed below were obtained on SUN ULTRA 2 workstations with 300 MHz processors.

Figure 5 shows the computation time for both the MSC verification and the test generation with AUTOLINK. The time needed for the verification of an MSC ranged from 1 min 24 sec to 2 h 15 min. It took between 6 min 44 sec and 51 h 49 min (= 3109 min) to generate a test case.

The larger amount of time needed for test generation is not surprising: During MSC verification, a path in the state space graph is truncated as soon as an event in an SDL transition conflicts with the MSC. On the other hand during test generation, the path needs to be extended until an observable event occurs.

Interestingly, there is no general correlation between the computation time for MSC verification and test generation. For example, MSC no. 57 in Figure 5 can be verified comparably fast, whereas its test case generation takes about 5 hours.

Due to the large number of data used in the SDL system, on average only 22 states per minute could be explored by AUTOLINK. Much time was spent for the computation of the hash values of each state needed for the bit-state algorithm.

6.3 Distributed test case processing

Verification of all MSCs on a single machine would have taken about a day; generation of all test cases would have taken about a week. Therefore, the processing of test purposes was distributed among up to fifteen workstations.

As described in Section 2, AUTOLINK does not directly write a generated test case into a TTCN MP file. Instead, it stores each test case in an internal representation in memory. This representation and the corresponding constraints can be saved on disk and reloaded later. This feature was used to compute each test case separately. After the computation finished, all test cases were reloaded and combined into a single test suite. Identical constraints were merged automatically during this process.

With the help of shell scripts, test generation runs were executed in batch mode, so no manual intervention was needed to start the generation of each single test case. This way, test cases could be generated overnight. In addition, information about previous test generation runs could be used in order to minimize computation time by placing time-intensive test cases on fast machines first.

6.4 Test suite post-processing

Even though AUTOLINK (in combination with TTCN Link) produced a complete, readable test suite, some manual steps were still needed to enhance the result:

1. Preambles which were generated during direct translation from MSC into TTCN were replaced by the ones generated with state space exploration.
2. Using parameterization on the level of test steps, the number of postambles was reduced significantly.
3. PIXIT information was added.
4. Test group information was added.
5. The test suite was converted to concurrent TTCN.

7 SUMMARY AND OUTLOOK

AUTOLINK makes it much easier to generate TTCN test suites based on an SDL specification. In particular, constraint rules save a lot of time. Additionally, the integration of MSCs for which no state space search can be performed allows a uniform development process. AUTOLINK is used by project STF 100 at ETSI whose goal is the development of test suites for INAP CS–2.

Due to the complexity of modern protocol specifications, it seems to be almost impossible to create correct test suites by hand. AUTOLINK allows to check several properties of the test suite which would otherwise have been overlooked. For instance, AUTOLINK checks whether a test step can be shared among several test cases.

The time needed for test generation depends both on the complexity of the SDL model, the size of the MSCs describing the test purposes and the heuristics for the state space search. While the first two factors cannot be altered, the state space options have to be chosen carefully in order to find a good compromise between computation time and the chance to find all events with *inconclusive* verdict. More tool assistance is needed by the user to choose appropriate options.

With regard to the whole development process, the time effort for the actual test generation is not relevant (if some restrictions of the state space are accepted). Most time is spent for refinements of the SDL specification and the test purposes. ETSI estimates that about 20% of the expenses for the development of the first INAP test suite could be saved by tool support in comparison with a manual test suite development.

However, experience has shown that the amount of time spent for manual post-processing of a generated test suite can be further decreased. Therefore, improvements of AUTOLINK will focus on the readability of the generated TTCN code. In particular, we plan to implement the following extensions:

- **Support of concurrent TTCN**
  In order to generate concurrent TTCN, AUTOLINK needs further information that cannot be automatically retrieved from the SDL specification. E. g. information is needed about the assignment of PCOs to channels and the relation between PTCs and PCOs. Therefore the user will have to provide this information as part of the configuration file. Several strategies for the coordination of PTCs will be implemented, e. g. a strict synchronization or synchronization only when indicated in the MSC.
- **Parameterization of test steps**
  MSC references can be used in test purpose descriptions to refer to test steps. If an MSC test step is used in several test cases, it may lead to several test steps which only differ in a few signal parameters. Parameterization of test steps can be handled similarly to constraint parameterization and should include parameterization of signals, PCOs and signal parameters.

- **Support of timer events**
  TTCN timers could be created automatically either once for a complete test case or for each event separately. There should also be a possibility to explicitly specify timers in the MSCs which are transformed to TTCN timer events.
- **Test suite structure**
  A test suite is typically structured into test groups, i. e. sets of test cases which test a specific aspect of the specification. Since the test suite structure is often reflected in the names of the test purposes, a mechanism will be implemented that groups test cases based on their names.
- **PICS/PIXIT parameterization**
  Since SDL does not allow the use of symbolic values, PICS/PIXIT parameters have to be encoded as concrete values for test generation and replaced by symbolic values in a post-processing step. This time-consuming task can be automatized.
- **Automatic constraint parameterization**
  Automatic constraint parameterization is a way to minimize the number of constraints. However, it is not yet clear whether it will also enhance the readability of a test suite.

REFERENCES

[1] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.105: Specification and Description Language (SDL) combined with Abstract Syntax Notation One (ASN.1). ITU, Geneva, 1995.

[2] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.100: Specification and Description Language (SDL). ITU, Geneva, 1996.

[3] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU, Geneva, 1996.

[4] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.500: Framework of Formal Methods in Conformance Testing. ITU, Geneva, 1998.

[5] R. Anido, A. Cavalli, T. Macavei, L. P. Lima, M. Clatin, and M. Phalippou. Testing a real protocol with the aid of verification techniques. In *XXIII Seminario Integrado de Software e Hardware (SEMISH'96)*, pages 237–248, Brazil, August 1996.

[6] A. Cavalli, B.-H. Lee, and T. Macavei. Test generation for the SSCOP-ATM networks protocol. In *SDL '97 Time for Testing – Proceedings of the Eighth SDL Forum*, Evry, France, September 1997.

[7] A. Ek, J. Grabowski, D. Hogrefe, R. Jerome, B. Koch, and M. Schmitt. Towards the Industrial Use of Validation Techniques and Automatic Test Generation Methods for SDL Specifications. In *SDL '97 Time for Testing – Proceedings of the Eighth SDL Forum*, Evry, France, Septem-

ber 1997.

[8] J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL – Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.

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

[10] European Telecommunications Standards Institute. *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)*, 1998.

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

[12] J. Grabowski, D. Hogrefe, and Nahm. R. Test Case Generation with Test Purpose Specification by MSCs. In *SDL '93 Using Objects – Proceedings of the Sixth SDL Forum*, Darmstadt, Germany, October 1993. North-Holland.

[13] J. Grabowski, D. Hogrefe, R. Scheurer, and Z. Dai. Applying SAMSTAG to the B-ISDN Protocol SSCOP - Technical Description and TTCN Testsuite. In *Testing of Communicating Systems*, volume 10, Cheju Islands, Korea, September 1997.

[14] J. Grabowski, R. Scheurer, D. Toggweiler, and D. Hogrefe. Dealing with the Complexity of State Space Exploration Algorithms. In *Proceedings of the Sixth GI/ITG technical meeting on 'Formal Description Techniques for Distributed Systems'*, University of Erlangen, June 1996.

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

[16] ISO/IEC JTC 1/SC21. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation. International Standard 9646-3, ISO/IEC, 1992.

[17] E. Perez, E. Algaba, and M. Monedero. A pragmatic approach to test generation. In *Testing of Communicating Systems*, volume 10, Cheju Islands, Korea, September 1997.

[18] M. Schmitt, B. Koch, J. Grabowski, and D. Hogrefe. Autolink - A Tool for the Automatic and Semi-Automatic Test Generation. In *Proceedings of the Seventh GI/ITG Technical Meeting on Formal Description Techniques for Distributed Systems*, Berlin, June 1997.

[19] D. Toggweiler, J. Grabowski, and D. Hogrefe. Partial order simulation of SDL specifications. In *SDL '95 with MSC in CASE – Proceedings of the Seventh SDL Forum*, Oslo, Norway, September 1995. Elsevier.

Publications concerning AUTOLINK can be downloaded from
http://www.itm.mu-luebeck.de/research/autolink/

8 BIOGRAPHY

Michael Schmitt studied computer science with focus on computational linguistics at the University of Koblenz, Germany, where he graduated with a diploma degree in September 1996. Since October 1996 Michael Schmitt is a research assistant at the Institute for Telematics at the Medical University of Lübeck, Germany.

Anders Ek studied computer science and engineering at Lund Institute of Technology in Sweden. After his graduation 1986 he started working at Telia, the national Swedish telecom operator. At Telia he did research on formal methods and associated development methodology. Since 1992 he has been working with development of tools and methods at Telelogic, a commercial CASE tool vendor providing tools based on formal description techniques.

Jens Grabowski studied computer science and chemistry at the University of Hamburg, Germany, where he graduated with a diploma degree. From 1990 to October 1995 he was a research scientist at the University of Berne, Switzerland, where he received his Ph.D. degree in 1994. Since October 1995 Jens Grabowski is a researcher and lecturer at the Institute for Telematics.

Dieter Hogrefe studied computer science and mathematics at the University of Hannover, Germany, where he graduated with a diploma degree and later received his PhD. From 1983 to 1995 he worked at various positions in the industry and at universities. Since 1996 he is director of the Institute for Telematics and full professor at the Medical University of Lübeck.

Beat Koch studied computer science at the University of Bern, Switzerland. After his graduation in 1994, he worked in the industry for two years. Since 1996, he is a research assistant at the Institute for Telematics.