Test Generation with AUTOLINK and TESTCOMPOSER

M. Schmitt, M. Ebner, J. Grabowski

Institute for Telematics, University of Lübeck,
Ratzeburger Allee 160, 23538 Lübeck, Germany,
{*schmitt,ebner,grabowsk*}*@itm.mu-luebeck.de*

Abstract

Testing of telecommunication systems is a major concern in industry and standardization. Therefore, the two major SDL tool vendors have integrated automatic test generation tools into their software development suites. While Telelogic has added AUTOLINK to the TAU tools, former Verilog has extended OBJECTGeode with TESTCOMPOSER.
Even though both tools are based on the same concepts, many features are realized differently and their focus is put onto different aspects of the test generation process. This paper introduces the major principles of SDL-based test generation and provides an overview of how both tools support test development.

Keywords: SDL, MSC, TTCN, Test generation, Autolink, TestComposer

1 Introduction

The complexity of modern telecommunication systems has increased significantly and the need for thorough and systematic testing is undisputed nowadays. However, testing is an extensive time-consuming task. Before concrete tests can be carried out on a system, much effort has to be spent on specifying what and how to test and on getting the test descriptions in a format that is accepted by the test equipment.

In many cases, a formal specification of the *System Under Test* (SUT) is given in the *Specification and Description Language* (SDL) [7]. SDL not only allows to describe the structure and behavior of a communicating system in a semi-graphical way; there also exist tools for dynamic analysis of SDL specifications by means of simulation and validation. Hence, a reasonable approach is to generate test cases automatically based on a given SDL specification. In addition to an increased efficiency in terms of both time and cost, automatic test generation ensures consistency between the formal specification and the test cases applied to an implementation.

For that reason, the two major SDL tool vendors Telelogic and former Verilog have integrated automatic test generation tools into their software development environments.[1]

[1]In December 1999, the two companies have merged. Nevertheless, their separate product lines are both kept.

Telelogic complemented its Tau tool suite with Autolink in 1997. Autolink has been developed at the Institute for Telematics in Lübeck [4] and is based on the former work of the SaMsTaG project [2]. In 1998, Verilog extended ObjectGeode with TestComposer. Similar to Autolink, it has its root in the research area as it is based on TGV and TVEDA which were developed at IRISA/Verimag and France Telecom/CNET [9].

Both tools share the same basic concepts. For example, they apply state space exploration techniques to search for suitable test sequences. In addition, they support the second edition of the standardized *Tree and Tabular Combined Notation* (TTCN) [6] as a common output language. Nevertheless, many concepts are realized differently in TestComposer and Autolink. Moreover, the two tools put their focus onto different steps of the test generation process. The strengths of TestComposer are in the flexible specification of test purposes whereas Autolink has its strong points when it comes to the customization of the generated TTCN test suites.

TestComposer and Autolink have been described separately in detail in former publications [1, 9, 10]. This paper is intended to give an overview of the general concepts and how these are realized in either tool. It is structured as follows:

In Section 2, a short introduction to the overall process of test generation is given in order to make the reader familiar with the general approach. Section 3 describes the mapping of SDL concepts onto corresponding TTCN concepts and the required test architecture. Section 4 presents several ways to specify test purposes. Different approaches to generate test cases based on test purpose descriptions are described in Section 5. The customization of TTCN test suites is discussed in Section 6. Finally, a summary is given in Section 7.

The use of TestComposer and Autolink is illustrated by examples based on a variant of the *Inres* SDL specification [3]. Inres is a simple protocol that is not intended to provide a profound evaluation of the tools. A complete case study with Autolink can be found in [12].

2 Overview

Autolink and TestComposer are tightly integrated into their corresponding development environments. TestComposer is built on top of the ObjectGeode Simulator; Autolink is part of the Tau Validator. In this way, the tools can make use of the functionalities of their underlying applications. The Simulator as well as the Validator are used to find dynamic errors and inconsistencies in SDL specifications. They provide roughly the same basic features with state space exploration as their fundamental concept.

Test generation with TestComposer and Autolink follows a three-stage process. An overview is given in Figure 1. In the diagram, actions are represented by rounded boxes. Data structures and files are depicted in rectangles. Finally, configuration scripts that influence the test generation are indicated by hexagons.

In a first step, the user has to specify a set of test purposes. Each test purpose defines a specific aspect of the behavior of the implementation that is intended to be tested. With regard to TestComposer and Autolink, a test purpose is considered to be a sequence of input and output events that are to be exchanged between the given SDL system and its environment. Test purposes are developed either manually by using, e.g., an MSC editor, interactively by stepwise simulation of the SDL system or fully automatically.

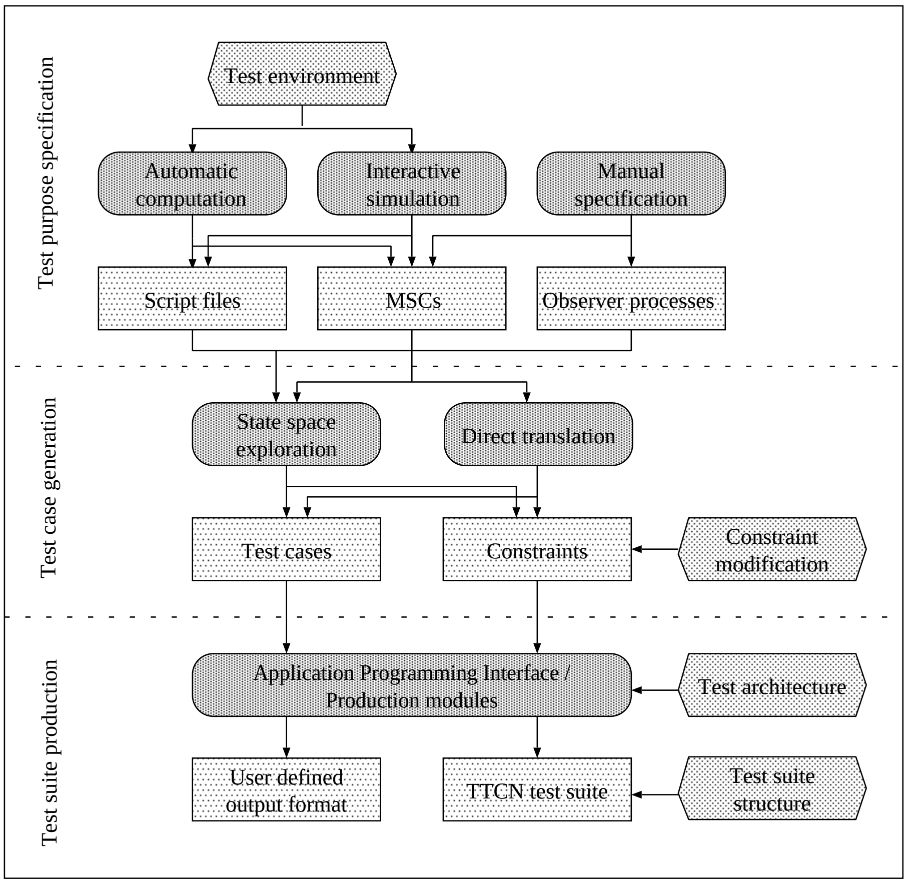


Figure 1: Test generation with TestComposer and Autolink

There are different representations for test purposes: Autolink uses Message Sequence Charts (MSCs) [8] as a uniform format. TestComposer uses MSCs as well but also creates scripts in a proprietary format that can be handled more efficiently by ObjectGeode than MSCs. Both tools support observer processes which are similar to regular SDL processes. They run in parallel with the actual SDL system and allow to inspect and control its simulation.

Based on a set of test purposes, test case generation takes place. Normally, a generation engine computes a test case based on state space exploration of the SDL system. By this, it can determine additional valid interactions between the tester and the SUT which are not already specified in the test purpose description. However, sometimes it is not possible to simulate a test purpose. For these cases, Autolink provides a way to translate test purposes directly into test cases.

All test case descriptions along with their *constraints*, i.e. the definitions of the data values exchanged between the tester and the SUT, are stored in an internal data structure. Autolink allows to save and reload generated test cases to disk such that the user can suspend and continue the generation of a full test suite.

In a final step, Autolink produces a test suite in second edition TTCN format. TestComposer provides a public Application Programming Interface (API) that allows cus-

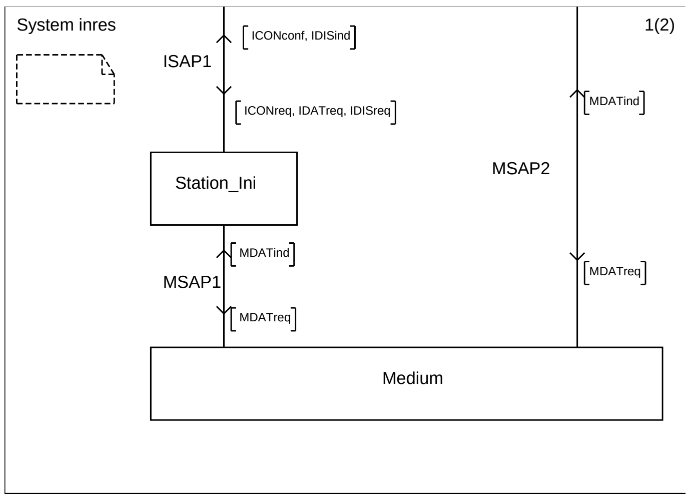


Figure 2: INRES system

tomers to adapt the tool to any arbitrary test specification language. In addition to the internal test case representations, the API provides access to general information about PCOs, timers, signal types etc. TestComposer already includes a module that produces test suites for second edition TTCN. In the following, only TTCN will be considered for output as many features of the tools are related closely to this notation.

Test generation with Autolink and TestComposer is influenced by a number of configuration settings. For example, when generating test purposes (semi-)automatically, the developer has to provide the simulator with information on the test environment of the system, i.e. reasonable input values. The look of the test suite can also be controlled by various options. In Autolink, constraints can be named and parameterized by user-defined rules. In addition, test cases can be combined in a hierarchy of test groups to express their relationships. Last but not least, the test architecture has a great impact on the final test descriptions. A test case that is executed on a monolithic tester will look differently from a test case that is designed for a distributed test system.

3 Test architecture

For test generation, it is necessary to define a mapping from a given SDL specification to the test architecture. Autolink always considers a complete SDL system to be the SUT. Each SDL channel that is connected to the system environment maps to a *Point of Control and Observation* (*PCO*) in TTCN. When generating test cases for the Inres system shown in Figure 2, the channels *ISAP1* and *MSAP2* are considered to be the interfaces of the test device. If only one block or process is supposed to be the SUT, this entity takes the role of an SDL system and must be simulated in a stand-alone manner.

In contrast to Autolink, TestComposer also allows to specify an arbitrary block *within* an SDL system to be the SUT. All channels connected to this block become PCOs regardless whether they link the block with the system environment or with another block.

For some test architectures, e.g. the remote test method of the *Conformance Testing*

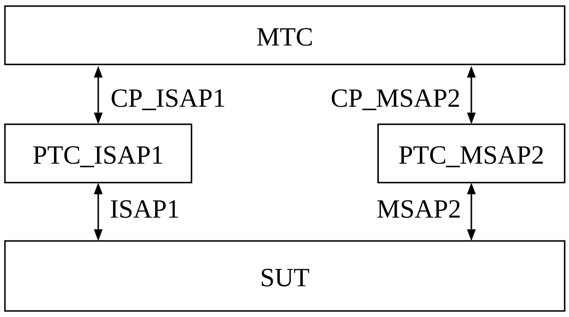


Figure 3: Distributed test architecture for the Inres system

Methodology and Framework [5], the controllability and observability of the IUT is restricted because some of its interfaces are not standardized or not accessible. For that reason, TestComposer allows to define for each PCO which signals are controllable and observable. All signals that cannot be observed are suppressed in the test case output whereas uncontrollable signals become *implicit send* events in TTCN.

In the same way as an SUT can be a distributed system, the tester itself may consist of a number of test components. While TestComposer only allows to generate tests for monolithic testers, Autolink also supports a generic architecture for distributed testers where each single PCO is associated exclusively with a *Parallel Test Component* (*PTC*). Coordination among PTCs is realized by exchanging *Coordination Messages* (*CMs*) via a *Main Test Component* (*MTC*). A distributed test architecture for Inres is presented in Figure 3.

Coordination messages cannot be computed automatically [1]. Therefore, the user has to state synchronization points explicitly in test purposes. In Figure 4, an MSC condition has been added in order to ensure that signal *IDISreq* is sent by *PTC_ISAP1* only after *PTC_MSAP2* has received *MDATind* and responded with *MDATreq*. During test generation, the MSC condition is resolved by a number of coordination messages. An excerpt of the corresponding TTCN test case including behavior descriptions for the MTC and *PTC_ISAP1* can be found in Figure 10 and Figure 11.

4 Test purpose specification

Test generation with Autolink and TestComposer starts with specifying a set of test purposes. Both tools provide several ways to define test purposes that differ in the degree of automation.

4.1 Automatic computation

Autolink and TestComposer allow to derive test purposes fully automatically from an SDL specification. Based on a state space exploration, a set of test sequences is computed whose execution shall result in a large structural *coverage* of the SDL specification. Each time a part of the SDL system is entered that has not been covered by a previous test purpose, a new one is generated. Sometimes, the test developer may only want to test

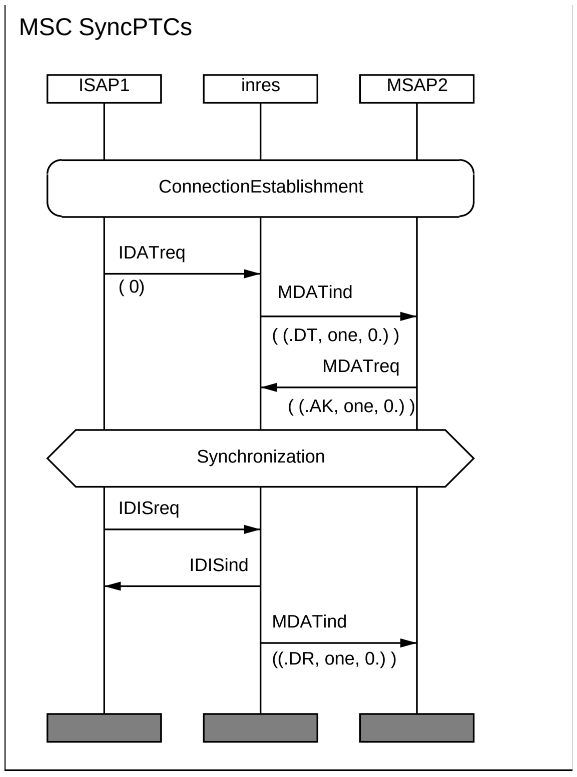


Figure 4: AUTOLINK test purpose with PTC synchronization

certain aspects of the system. Therefore, in TESTCOMPOSER transitions, processes or even whole blocks can be marked as covered, i.e. they are ignored during test purpose computation.

Basically, the two test generation tools have different notions of what a *coverage unit* is. AUTOLINK defines coverage on the basis of a single SDL symbol; for TESTCOMPOSER, the largest sequence of instructions that is not involved in branching is considered a unit. In any way, the execution of a coverage unit may not be observable when it comes to black-box testing, because it does not necessarily cause interaction of the SDL system with its environment. Therefore, generating a separate test purpose for each coverage unit leads to many identical test cases.

To circumvent the problem, both tools examine larger sequences of coverage units, called *observation s*, that lead from one stable state to another. A *stable state* is a state where the system waits silently for new input from its environment or the expiration of a timer. Each automatically generated test purpose includes at least one observation step. In most cases, an observation step includes a stimuli from the test environment and one or more corresponding responses from the system.

Unfortunately, due to nondeterminism one observation step can correspond to several different sequences of coverage units, i.e. an SUT may show the same external behavior but execute other statements internally than the ones that were supposed to be tested. The computation of Unique Input/Output (UIO) sequences such as, e.g., realized in SAMSTAG [11] would solve the problem. However, in practice it is not necessary to prove that a test includes an UIO sequences most of the time.

There is no universally applicable strategy to find paths through the reachability graph which results in a high coverage. Both tools offer depth-first and supertrace search algorithms to explore the state space; in addition, TestComposer includes a breadth-first search. However for large SDL systems, shallow exploration — as it is performed in breadth-first and iterative depth-first mode — results in poor coverage. Many relevant parts of a specification can only be covered after a long initialization sequence. For validation purposes, it is an adequate approach to explore the state space in a depth-first or even random walk manner. However, test purposes generated this way tend to include sequences of useless events which do not contribute to the primary test purpose. As a workaround to this problem, the user may first manually simulate the initialization sequence and then start an automatic exploration. Alternatively, Autolink provides a deterministic algorithm that makes repetitive local explorations with increasing depth at *various* places in the state space. The search algorithm is based on the heuristic that in a region of the reachability graph where an increase of coverage has been observed, it is more likely to detect transitions that result in a yet higher coverage.

Ideally, it should be possible to run a test suite without having to reset the SUT after each single test case. Under this premise, it is not sufficient to let a test purpose specification stop at the end of an observation step in an arbitrary stable state. Instead, a final test sequence, i.e. a *postamble*, must be computed that drives the system back into some *idle state*. However, an automatic test generation tool is not able to know what distinguishes an idle state. Therefore, TestComposer allows the user to specify a boolean expression that characterizes an idle state. It will then automatically search for a suitable postamble.

4.2 Interactive simulation

In order to produce test purposes automatically, the user has to define a set of reasonable input signals. Whenever the SDL system is in a stable state, the test tools continue the simulation with each possible input. For complex specifications, it may be difficult to predict a set of inputs in advance that result in a high coverage. In addition, due to the fact that each possible signal is tested in each stable state, the state space may grow excessively. For one of the test suites for Core INAP CS/2 [12], the SDL system had to be controlled by more than 130 different inputs! On the other hand, usually the user knows exactly which signal has to be sent at which time. Therefore, she may want to produce test purposes by a stepwise simulation. Both Autolink and TestComposer are build on top of tools that allow the user to navigate manually through the state space of the SDL specification.

4.3 Manual specification by observer processes

TestComposer and Autolink allow to use observer processes for test generation. An observer process is a special kind of SDL process which is able to monitor the SDL system. The representation of observer processes slightly differs in Autolink and TestComposer but in principle they have the same properties.

An observer process has direct access to all internal elements of the specification such as variables and timers. Moreover, it can also influence the simulator itself. An observer process may prune a path in the state space or create some report if a specific condition holds. For example, an observer process may check repetitively whether one of the SDL

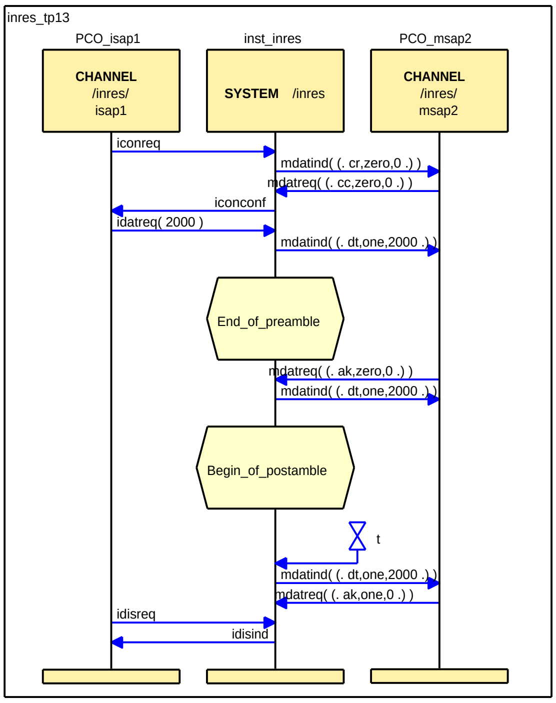

inres_tp13
PCO_isap1
inst_inres
PCO_msap2
CHANNEL /inres/ isap1
SYSTEM /inres
CHANNEL /inres/ msap2
iconreq
mdatind((. cr,zero,0 .))
mdatreq((. cc,zero,0 .))
iconconf
idatreq(2000)
mdatind((. dt,one,2000 .))
End_of_preamble
mdatreq((. ak,zero,0 .))
mdatind((. dt,one,2000 .))
Begin_of_postamble
t
mdatind((. dt,one,2000 .))
mdatreq((. ak,one,0 .))
idisreq
idisind

Figure 5: Test purpose generated automatically by TestComposer

processes has entered a particular state or the coverage of the system has increased by the last transition.

The test generation tools support observer processes at different stages: TestComposer can use an observer process directly as test purpose specification. On the other hand, Autolink uses observer processes for the *generation* of test purposes, i.e. an observer process has to be transformed into a set of MSCs first. The reason for this is that an observer process corresponds to an *abstract* test purpose. While TestComposer can handle such abstract specifications, Autolink requires complete MSCs as test purposes where all inputs and outputs of the SDL system are specified explicitly.

4.4 Structuring of test purposes

Test purposes can be structured into several parts, e.g. preamble, test body, and postamble. In Figure 5, an MSC test purpose is shown that has been generated automatically by TestComposer. The end of the preamble and the start of the postamble are denoted by MSC condition symbols. During test case generation, the information about the structuring of the test purpose is preserved in order to produce a user-friendly output (see also Section 6.2).

In contrast to TestComposer, Autolink does not structure test purposes automatically. Instead, it allows the user to split a test purpose into several MSCs. Test steps such as preambles and postambles can be stored as separate MSCs and reused for several test purposes with the help of MSC references (see Figure 4). In addition, Autolink supports High-level MSCs (HMSCs) and MSC expressions for stating the relation between distinct

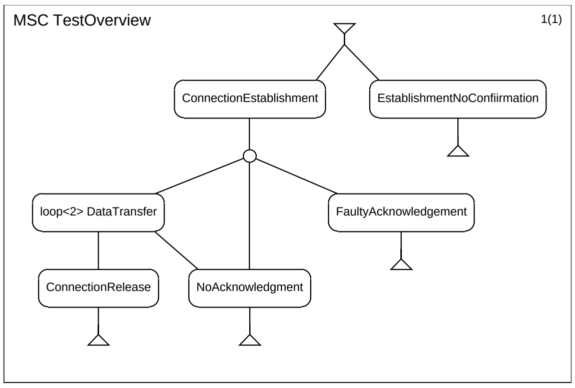


Figure 6: HMSC describing a set of test purposes

test purposes[2]. In Figure 6, a sample HMSC is given that defines a kind of roadmap for testing the Inres protocol. For each possible path through the HMSC, AUTOLINK creates a distinct test case.

5 Test case generation

After the specification of test purposes, test cases can be generated. There are two methods to do this: State space exploration and direct MSC to TTCN translation.

5.1 State Space Exploration

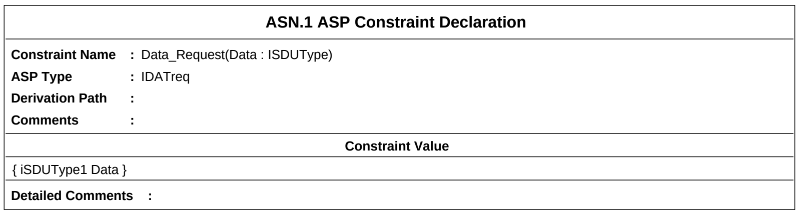
To generate a test case, paths through the SDL system have to be found which correspond to the test purpose. Both AUTOLINK and TESTCOMPOSER use state space exploration to find these paths. If alternative paths are found which violate the test purpose but are valid according to the SDL specification their externally visible events are added to the test case with a TTCN *inconclusive* verdict.

TESTCOMPOSER supports abstract test purposes which lack a full description of the interaction between the SDL system and its environment. Hence, missing events are added to the test case when exploring the state space of the SDL system.

Two different generation engines are provided by TESTCOMPOSER. The default engine is derived from the former TTCgeN tool. It computes the whole state space which corresponds to the test purpose and works very fast when dealing with complete test purposes.

The other engine is an industrial version of the TGV prototype. Like the AUTOLINK engine, it uses *on the fly generation* which is more efficient for incomplete test purposes [9, 13]. In addition, it allows TESTCOMPOSER to find postambles for paths leading to *pass* and *inconclusive* verdicts (see also section 4.1).

[2]HMSCs and MSC expressions can only be used for a direct MSC to TTCN translation as described in Section 5.2



ASN.1 ASP Constraint Declaration	
Constraint Name	: Data_Request(Data : ISDUType)
ASP Type	: IDATreq
Derivation Path	:
Comments	:
Constraint Value	
{ iSDUType1 Data }	
Detailed Comments	:

Figure 7: TTCN constraint generated by AUTOLINK

5.2 Direct translation of MSCs

If a test purpose defined as MSC covers certain aspects of a protocol specification which are not represented in the corresponding SDL model or if a SDL model is missing completely, it is obviously not possible to generate a test case by state space exploration. To handle these cases, AUTOLINK provides direct translation of MSCs into TTCN test cases with consistency checks regarding the SDL system interface definitions. Hence, an SDL system has to be provided which at least defines the channels to the system environment in order to identify the PCOs and the signals sent via these channels.

Since AUTOLINK always translates MSCs into an intermediate internal test case representation, test cases generated by an MSC to 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.

5.3 Constraints

Send and receive events in a test case are associated with constraints which denote the actual values of signal parameters. Since constraints can be shared among several events in different test cases, they are stored separately from the test case representations. AUTOLINK supports special handling of constraints like naming and parameterization based on user-defined rules. A typical constraint rule looks like this:

```
TRANSLATE "IDATreq"
 CONSTRAINT NAME "Data_Request"
            PARS $1="Data"
END
```

AUTOLINK translates signal names into constraint names. The example above instructs AUTOLINK to assign the name `Data_Request` to constraints which are related to `IDATreq` signals. Additionally, the first parameter of signal `IDATreq` becomes a parameter of constraint `Data_Request`. The corresponding TTCN constraint table can be found in Figure 7.

Test suite parameters and constants can be introduced by using the same configuration language. Furthermore, AUTOLINK supports replacement of message parameters by wildcards for irrelevant parameter values.

Test Case Dynamic Behaviour

Test Case Name	:	inres_13
Group	:	
Purpose	:	from state idle of msap_manager2, receive mdatreq, send () and go to state idle
Configuration	:	
Default	:	DEF_0
Comments	:	Generated by test oriented simulation of the test purpose .\inres_tp13.scn

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+PR_5			
2		msap2 !mdatreq START TAC	mdatreq_1		
3		msap2 ?mdatind CANCEL TAC	mdatind_8	(P)	
4		+PO_0			

Detailed Comments :

Figure 8: TTCN test case produced by TestComposer

6 Test suite production

A TTCN test suite contains four parts: The test suite overview, the declarations part, the constraints parts and the dynamic behavior part. A TTCN test suite can be customized by various options. In the following, a few features are presented that enhance readability and reduce the need for manual post-processing.

6.1 Test grouping

Typically, a test suite contains a hierarchy of test groups that subsume a set of related test cases. For example, all test cases that aim at testing the fundamental functionality of an SUT may be placed in a test group called *BasicCapability*.

In Autolink, test grouping is based on the test case names. The user can specify rules incorporating regular expressions which tell Autolink how to group test cases. TestComposer arranges test cases according to the state in which the event occurred that triggered the crucial observation step.

6.2 Test step format

When using TestComposer, the user may choose between two output formats for test steps. They are either printed directly in a test case or stored globally in the test step library. Autolink provides the same alternatives but, in addition, it allows to write test steps as local trees within a test case dynamic behavior table.

In Figure 9, two dynamic behavior tables are presented for preamble *PO_0* and postamble *PR_5*. Both test steps are referenced by test case *inres_13* shown in Figure 8.

When generating test cases for distributed test architectures, Autolink creates automatically separate test steps for each PTC in the test step library (see second table in Figure 11) and one test case for the MTC (Figure 10).

Test Step Dynamic Behaviour

Test Step Name	:	PO_0
Group	:	
Objective	:	
Default	:	DEF_0
Comments	:	Postamble

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		?TIMEOUT TEMPTY			Timeout of timer t of process initiator
2		msap2 ?mdatind	mdatind_6		
3		msap2 !mdatreq	mdatreq_4		
4		isap1 ?idisind		P	

Detailed Comments :

Test Step Dynamic Behaviour

Test Step Name	:	PR_5
Group	:	
Objective	:	
Default	:	DEF_0
Comments	:	Preamble

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		isap1 !iconreq START TAC			
2		msap2 ?mdatind CANCEL TAC	mdatind_6		
3		msap2 !mdatreq START TAC	mdatreq_3		
4		isap1 ?iconconf CANCEL TAC			
5		isap1 !idatreq START TAC	idatreq_7		
6		msap2 ?mdatind CANCEL TAC	mdatind_8		End of preamble

Detailed Comments :

Figure 9: TTCN test steps produced by TestComposer

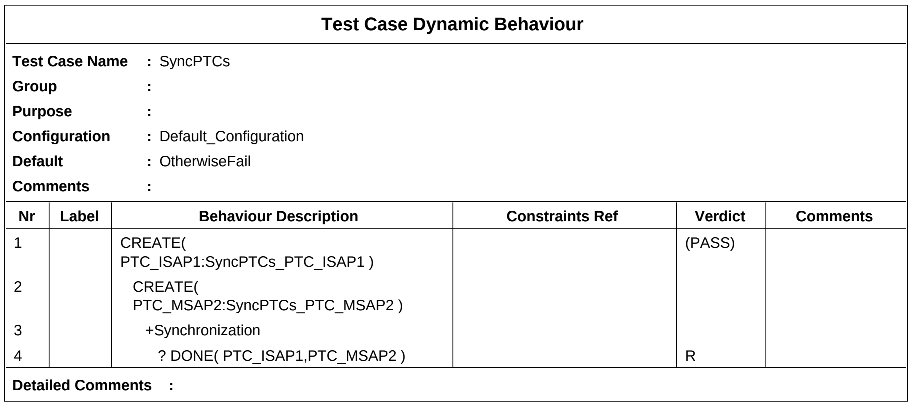
6.3 Test purpose comments

Comments in test cases support readability and maintenance. TestComposer automatically inserts a comment in a test case describing its test purpose. A typical comment is shown in the *Purpose* line of the *Test* Case *Dynamic Behaviour Table* in Figure 8. In case the user prefers another format for comments, she can specify a pattern. This pattern may involve pre-defined macros for test purpose related information such as the name of the signal which was received at the beginning of the central observation step.

6.4 Mapping SDL signals onto ASPs and PDUs

According to OSI conformance methodology, the tester and the SUT exchange two types of data: *Abstract Service Primitives* (*ASPs*) and *Protocol Data Units* (*PDUs*). In SDL, there exists only the *signal* concept. Autolink allows the user to specify for each signal individually whether it shall be mapped onto an ASP or PDU in the test suite.

In contrast, TestComposer allows to assign a role to each PCO. All signals that are exchanged via a PCO with role *Upper tester* become ASPs in the test suite. Signals corresponding to a *Lower Tester* are mapped onto PDUs.



Test Case Dynamic Behaviour

Test Case Name	: SyncPTCs
Group	:
Purpose	:
Configuration	: Default_Configuration
Default	: OtherwiseFail
Comments	:

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		CREATE(PTC_ISAP1:SyncPTCs_PTC_ISAP1)		(PASS)	
2		CREATE(PTC_MSAP2:SyncPTCs_PTC_MSAP2)			
3		+Synchronization			
4		? DONE(PTC_ISAP1,PTC_MSAP2)		R	

Detailed Comments :

Figure 10: TTCN test case produced by AUTOLINK for a distributed tester

6.5 Removal of signal definitions

One of the major drawbacks of SDL with regard to testing is the requirement that all communication is realized by signal exchange. In TTCN, there exists no corresponding concept. Instead the tester and the SUT are allowed to exchange values of arbitrary data type. Hence, if a PDU is defined in an external ASN.1 module, it needs to be wrapped up in a signal. Later, the test generation tools map each SDL signal onto an ASN.1 SEQUENCE which includes the PDU as its only parameter. Since the additional embedding causes some overhead and influences PDU encoding, AUTOLINK provides a command for stripping redundant signal definitions when generating a TTCN suite.

6.6 Timer

TESTCOMPOSER automatically generates four kinds of test timers in order (a) to guard expected SUT output events, (b) to check situations where no output is expected from the SUT for a specified period of time, (c) to check the maximum time to execute an implicit send and (d) to wait for the expiration of timers in the SUT.

In Figure 8, timer *TAC* is started after the sending of message *mdatreq* in order to check that *mdatind* is sent by the SUT within a prescribed duration. Once the response is received, *TAC* is canceled.

7 Summary

Even though both test generation tools are based on the same principles, many details are realized differently. The development of AUTOLINK and TESTCOMPOSER was driven by concrete needs of their respective users. Therefore, the tools address different kinds of test generation problems.

With regard to test purpose specification, a strong point for TESTCOMPOSER is its ability to compute postambles automatically. On the other hand, AUTOLINK supports MSC'96 which improves the manual and semi-automatic specification of test purposes.

Test Step Dynamic Behaviour

Test Step Name	: Synchronization
Group	:
Objective	:
Default	: OtherwiseFail
Comments	:

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		CP_ISAP1 ? CM	Ready_Indication		
2		CP_MSAP2 ? CM	Ready_Indication		
3		CP_ISAP1 ! CM	Proceed_Indication		
4		CP_MSAP2 ? CM	Ready_Indication		
5		CP_ISAP1 ? CM	Ready_Indication		
6		CP_ISAP1 ! CM	Proceed_Indication		

Detailed Comments :

Test Step Dynamic Behaviour

Test Step Name	: SyncPTCs_PTC_ISAP1
Group	:
Objective	:
Default	: OtherwiseFail
Comments	:

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+ConnectionEstablishment_PTC_ISAP1			
2		ISAP1 ! IDATreq	Data_Request(DataValue)		
3		CP_ISAP1 ! CM	Ready_Indication		
4		CP_ISAP1 ? CM	Proceed_Indication		
5		ISAP1 ! IDISreq	Disconnection_Request		
6		ISAP1 ? IDISind	Disconnection_Indication	PASS	

Detailed Comments :

Figure 11: TTCN test steps produced by AUTOLINK for a distributed tester

When it comes to test case generation based on state space exploration, TESTCOMPOSER is more flexible as it accepts incomplete test purpose descriptions. In addition, it takes several types of timers into account. AUTOLINK supports direct translation from MSC to TTCN which is useful if a given SDL specification is incomplete.

Finally, TESTCOMPOSER allows to save a test suite in any format by providing an API to its database, whereas AUTOLINK supports the generation of TTCN test suites for distributed test architectures.

In many cases, AUTOLINK and TESTCOMPOSER complement each other and a combination of the best features of both tools would lift the practical usability of test generation tools onto a new level.

References

[1] J. Grabowski, B. Koch, M. Schmitt, and D. Hogrefe. SDL and MSC Based Test Generation for Distributed Test Architectures. In Dssouli R, G. v. Bochmann, and Y. Lahav, editors, *SDL '99 The next Millenium – Proceedings of the Nineth SDL Forum*, Montreal, Canada, June 1999. Elsevier.

[2] Jens Grabowski, Rudolf Scheurer, Zhen Ru Dai, and Dieter Hogrefe. Applying SAM-STAG to the B-ISDN protocol SSCOP. *Testing of Communicating Systems*, 10, 1997.

[3] D. Hogrefe. *Estelle, LOTOS und SDL. Standard-Spezifikationssprachen für verteilte System.* Springer-Verlag, 1989.

[4] Institute for Telematics, University of Lübeck, Germany. `http://www.itm.mu-luebeck.de`, 2000.

[5] ISO/IEC. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework. International multipart standard 9646, ISO/IEC, 1994.

[6] ISO/IEC. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3 (second edition): The Tree and Tabular Combined Notation. International Standard 9646-3, ISO/IEC, 1997.

[7] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.100: Specification and Description Language (SDL). ITU, Geneva, 1999.

[8] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU, Geneva, 1999.

[9] A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from SDL specifications. In R. Dssouli, G. v. Bochmann, and Y. Lahav, editors, *SDL '99 The Next Millenium, Proceedings of the Ninth SDL Forum, Montréal, Québec, Canada, 21–25 Hune, 1999*, pages 135–151. Elsevier, June 1999.

[10] B. Koch, J. Grabowski, D. Hogrefe, and M. Schmitt. Autolink – A Tool for Automatic Test Generation from SDL Specifications. In *IEEE International Workshop on Industrial Strength Formal Specification Techniques (WIFT'98)*, Boca Raton, Florida, October 1998.

[11] R. Nahm. *Conformance Testing based on Formal Description Techniques and Message Sequence Charts.* PhD thesis, University of Berne, Institute for Informatics and Applied Mathematics, March 1995.

[12] M. Schmitt, A. Ek, J. Grabowski, D. Hogrefe, and B. Koch. Autolink – Putting SDL-based test generation into practice. In A. Petrenko and N. Yevtuschenko, editors, *Testing of Communicating Systems*, volume 11, Tomsk, Russia, June 1998. Kluwer Academic Publishers.

[13] Verilog SA. OBJECT*Geode* – TESTCOMPOSER *Reference Manual*, 4.1 edition, 1999.