— Autolink —
A Tool for the Automatic and Semi-Automatic Test Generation

Michael Schmitt, Beat Koch, Jens Grabowski and Dieter Hogrefe

University of Lübeck, Institute for Telematics, Ratzeburger Allee 160, D-23538 Lübeck, Germany,
e-mail: [schmitt,bkoch,jens,hogrefe]@itm.mu-luebeck.de, http://www.itm.mu-luebeck.de

Abstract

Due to increasing interest in validation and test generation tools, Telelogic AB, Malmö, and the Institute for Telematics of the University of Lübeck have started a research and development project which aims at bringing new test generation facilities to the Tau tool set. For that purpose, a software component is developed which supports the automatic and semi-automatic generation of TTCN test suites based on SDL specifications and MSC test cases. The project follows a pragmatic approach and is driven by practical experience. Autolink is currently used by the *Project Team 100* at the *European Telecommunications Standardisation Institute* (ETSI), where it has to prove its usefulness in everyday work.

1 Introduction

Autolink is part of Tau, a commercial product by Telelogic AB, Malmö. Tau combines the two well-known tool sets SDT and ITEX. It provides tools for the formal design, implementation and testing of communicating systems software. In particular, it supports the use of the specification languages SDL, MSC and TTCN. Tau includes graphical editors, analysers, simulation tools and application code generators.

One tool of Tau is the SDT Validator. The Validator is based on state space exploration techniques [3]. It builds up the state space of a given SDL specification and examines it with regard to some properties. In particular, the Validator provides

- an automated fault detection mechanism, which finds inconsistencies and problems like deadlocks, range errors and implicit signal consumption;
- an MSC verification mechanism, which allows to verify an SDL system against requirements described by MSCs.

Autolink is a component which uses and extends the SDT Validator's functions.

Figure 1 shows the Tau environment in which Autolink is embedded. SDT allows the user to construct SDL specifications and test them with the Validator, whereas ITEX supports the user in the development of TTCN test suites.

With Autolink one can specify test cases within the SDT Validator. The tool then automatically generates a TTCN test suite with constraints and dynamic behaviour tables. This test suite can be completed and refined in ITEX. Additionally a *Link executable*, which can be automatically derived from the SDL specification, allows to generate all static TTCN declarations which are not provided by Autolink.

The rest of this paper is structured as follows: In Section 2 we give a survey of the different steps of test suite generation. A short description of the search algorithm is given in Section 3. Section 4 contains some examples for the use of Autolink. Finally, in Section 5, plans for version 2 of Autolink are presented.

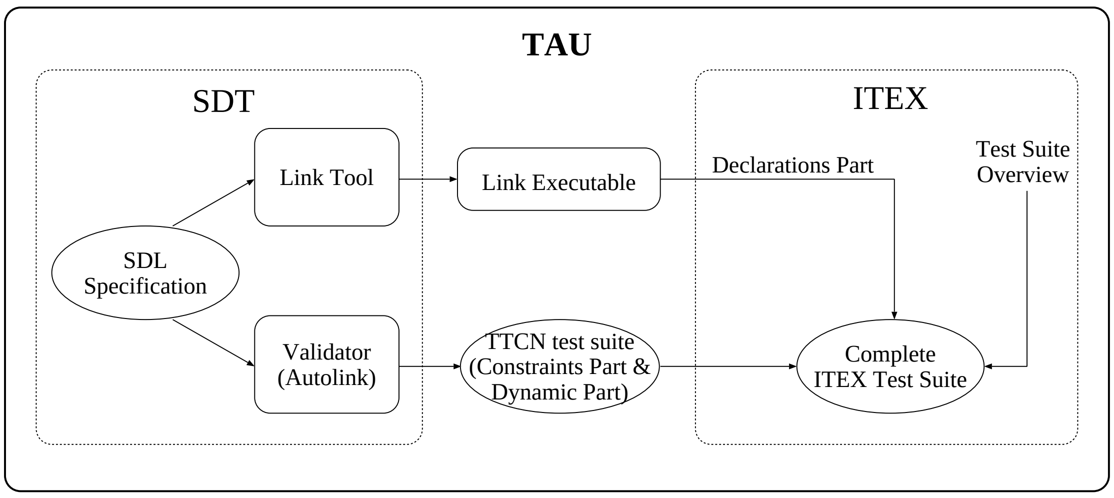


Figure 1: Test Generation – System Overview

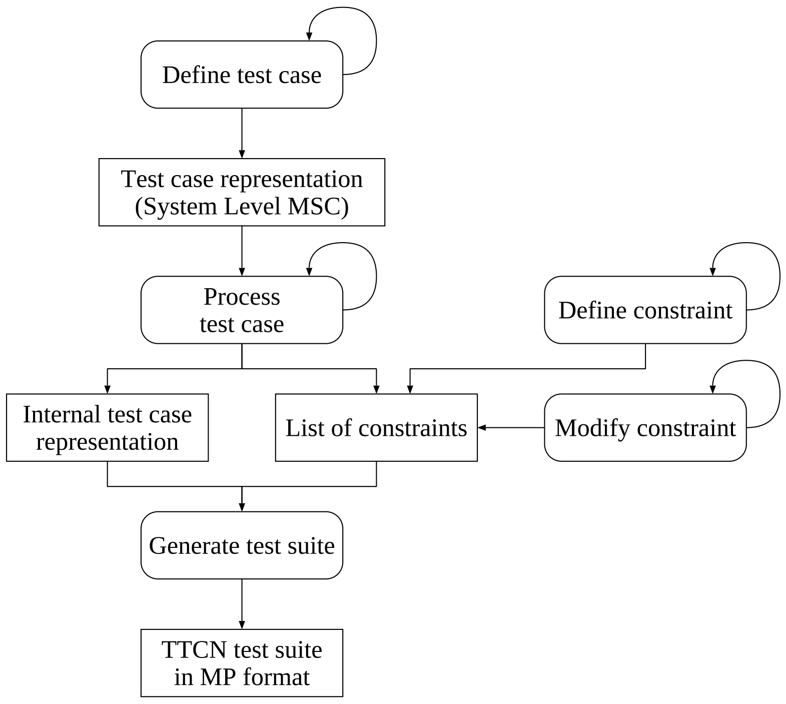


Figure 2: The Test Suite Generation

2 Autolink Version 1

The generation of a TTCN test suite with AUTOLINK proceeds in several steps. These steps are outlined in Figure 2. Most of the steps can be iterated as indicated by loops in the diagram.

Test case definition At first, test cases have to be defined for which a test suite is intended to be generated. In AUTOLINK a test case is derived from a *path*. A path is a sequence of events (like tasks, signal exchanges, etc.) that have to be performed in order to traverse from a start state in the state space of an SDL system to an end state. The Validator allows to specify paths in several different ways, including both selective and brute force strategies. Some examples are:

- **Manual navigation in the state space.** The SDT Validator provides a special window which allows users to manually explore the state space and thereby to define paths in an efficient way.
- **MSC verification.** The Validator searches for a path that satisfies a given MSC. This path is then used as a test case.
- **Observer processes.** An observer process is a special kind of process within an SDL system, which is able to observe other processes and the SDL system in general. Observer processes allow an automatic definition of a large set of tests for an SDL system. Each time a path is found by the Validator that satisfies the condition defined in an observer process, a report is generated. At the end of the state space exploration, the reports can be converted into MSC test cases. A typical field of application for observer processes is test generation based on SDL symbol coverage.

All three methods to define a path can be combined. The user may also use the SDT Simulator, a tool that works similar to the SDT Validator and helps to debug a specification. One advantage of using the SDT Simulator is its support of ASN.1 data types. The user may also manually create MSCs using the MSC editor.

The path is stored in the form of a system level MSC, i.e., an MSC with only one instance axis for the SDL system and one or more instances axes for the environment. A system level MSC shows only the visible interaction that is supposed to take place between an implementation and the test system. Note that a system level MSC is an abstraction of a path. There may exist several paths in the state space of the SDL system which show the same external behaviour and hence lead to the same system level MSC.

Usually test cases are logically structured into several parts, e.g. a preamble, a test body and a postamble. These parts are called *test steps*. AUTOLINK allows to incorporate test steps in test cases by using MSC references. Test steps again may refer to other test steps. During test case processing, AUTOLINK keeps track of the nested structure of test cases and test steps. Thus, test steps can be output in different ways in a test suite, e.g., as local trees in a test case description or globally in the test steps library. If a test case is structured into preamble, test body and postamble, AUTOLINK automatically assigns preliminary pass verdicts at the end of the test body.

Test case processing. The system level MSCs are taken as input for test case processing, which results in an internal representation for each test case. This representation consists of a sequence of send and receive events leading to a TTCN *pass* verdict. In addition, alternative receive events are looked up and added to the test case with a TTCN *inconclusive* verdict. Section 3 describes the search algorithm that builds the internal test case representation.

In addition to the internal test case representation a list of constraints is generated. The user is able to assign more reasonable names to the constraints generated by AUTOLINK before the test suite is saved. He may also manually define and remove signal constraints or merge two constraint definitions at any time. Merging constraints is useful if two constraints have the same meaning or if a signal parameter with different values in two constraints is irrelevant.

Test suite generation. Finally a TTCN test suite in MP format is generated based on the internal test case representation and the list of constraints. The TTCN test suite consists of four main sections, but only the constraints and dynamic behaviour parts are generated by AUTOLINK. Declarations can be generated automatically using the Link tool; the overview part can also be generated automatically by Itex.

Constraints are saved in ASN.1 format. Thereby a problem has to be solved that is caused by the different notations for data structures in SDL and ASN.1. Parameters of SDL signals have types, but no names, whereas ASN.1 always demands a name for the fields of a data structure. In order to generate a TTCN test suite that is well-formed, generic field names are created.

Dynamic behaviour tables can be generated by evaluating the internal representations for each test case. As mentioned above, AUTOLINK allows to output test steps in several ways.

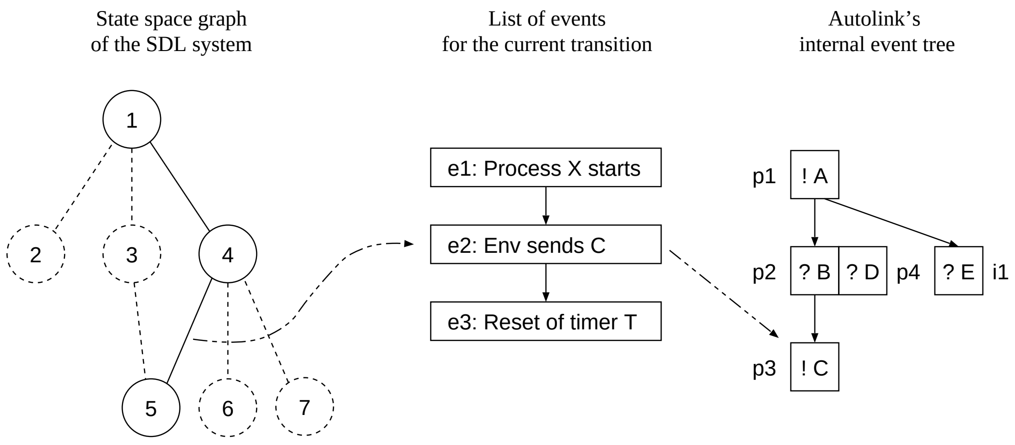

State space graph of the SDL system
1
2
3
4
5
6
7
List of events for the current transition
e1: Process X starts
e2: Env sends C
e3: Reset of timer T
Autolink's internal event tree
p1
! A
p2
? B
? D
p4
? E
i1
p3
! C

Figure 3: Test case processing algorithm

3 The Test Case Generation Algorithm

The processing of test cases consists of three distinctive parts. After some preparatory work, a state space exploration algorithm is used to build up the internal data structure which represents a test case. Finally, post-processing of the internal test case data structure is needed.

Three data structures are used during the state space exploration. They are shown in Figure 3.

1. *State space graph* Just one path through the state space of an SDL system is stored as a stack of states at any given time.

2. *Event list* Whenever a transition from one SDL system state to a successor state is computed, a list of events is generated.

3. *Autolink event tree* Each node of the Autolink event tree corresponds to a TTCN event. It contains type information about the event, a list of the event nodes with *pass* verdicts on the next level, and a list of events with TTCN *inconclusive* verdicts on the next level. All events on the same level will appear as alternatives in TTCN notation.

Here is a short description of the state space exploration algorithm. Let us assume that we are in State 4 in the state space and $p2$ is the current node in the Autolink event tree. Now State 5 is computed. This produces a list of events.

Event $e1$ of the list is checked against the system level MSC. The test shows that $e1$ is not relevant to the MSC, so $e2$ has to be checked next.

$e2$ is an observable event and it satisfies the MSC. Therefore it is appended as a *pass* event $p3$ to the current node in the Autolink event tree. $p3$ also becomes the new current node in the Autolink event tree.

$e3$ is not an observable event, therefore it can be ignored. It is also the last event in the list. This means that the transition is completed and State 5 in the state space graph has been reached. To continue the exploration, the first successor to State 5 has to be computed.

An *inconclusive* node is appended to the Autolink event tree if an event e in the list of events is observable but does not satisfy the MSC. To continue the exploration, the next successor of the previous state has to be computed. If we had such an event, e. g., in the transition from State 1 to State 2 in Figure 3, then the transition from State 1 to State 3 would be computed next.

For every state s in the stack, the current node $p(s)$ in the Autolink event tree is saved. If there is no successor to state s, then the next successor of state $s-1$ is computed and $p(s-1)$ becomes the current node in the Autolink event tree. The exploration ends if no more successors to the start state are found.

Post-processing removes unwanted events from the Autolink event tree. First, it is assumed that the environment always sends signals to the system as soon as possible, whereas receive events occur with an undefined delay. Therefore, alternative receive events to a send from the environment

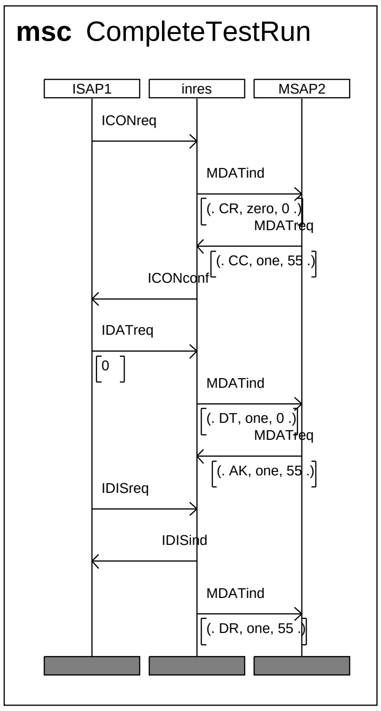


Figure 4: Complete MSC test run

are removed from the Autolink event tree. Second, incomplete pass paths may have been generated during the state space exploration. The first event in a branch which does not end with a *pass* verdict is reassigned as an event with *inconclusive* verdict, and the rest of the branch is discarded.

Test steps Test steps are represented in the Autolink event tree by two special nodes which indicate the beginning and the end of a test step. In order to generate correct test step descriptions, a change in the semantics of MSC references has been necessary: While ITU-T Recommendation Z.120 considers an MSC reference as some sort of macro, AUTOLINK requires that a test step is completely evaluated before the next test steps starts. This restriction is especially relevant for MSCs with more than two instances axes where there is no total ordering of the events of the environment.

4 Examples

In this section we present some examples based on the *Inres* protocol [2].

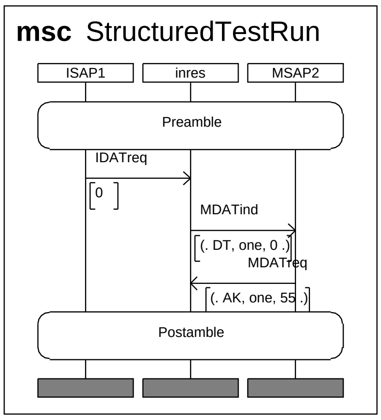
Figure 4 shows a complete test run which tests whether it is possible to establish a connection, send one packet of data and then release the connection.

AUTOLINK allows to use test steps in test cases. Therefore the test case above can be structured into a preamble (connection establishment), a test body and a postamble (disconnection) as shown in Figure 5 (a), (b) and (c).

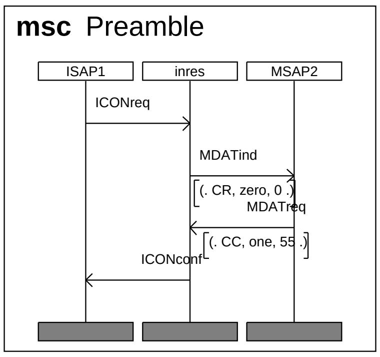
When processing the test case AUTOLINK builds an internal test case representation which is the basis for several possible TTCN outputs. Figure 6 presents one possible output format for *StructuredTestRun*. In this case the test steps are directly included in the test case description. Figure 7 shows another possible test case description derived from the same internal representation. Here, test steps are described by local trees.

5 Future plans

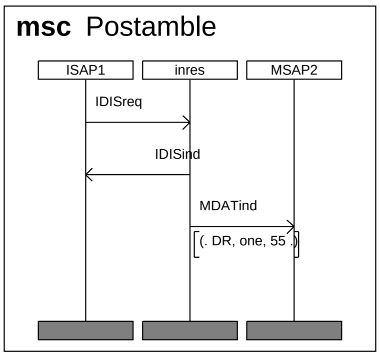
Version 1 of AUTOLINK [4] has already been shipped by Telelogic AB. It is used by a team working on Project 100 at the European Telecommunications Standardisation Institute (ETSI). The goal



(a) MSC test run with references



(b) Preamble of (a)



(c) Preamble of (a)

Figure 5: Structured MSC test description

Test Case Dynamic Behaviour

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

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		ISAP1 ! ICONreq	TestCase2_001		
2		MSAP2 ? MDATind	TestCase2_014		
3		MSAP2 ! MDATreq	TestCase2_003		
4		ISAP1 ? ICONconf	TestCase2_004		
5		ISAP1 ! IDATreq	TestCase2_005		
6		MSAP2 ? MDATind	TestCase2_006	(PASS)	
7		MSAP2 ! MDATreq	TestCase2_007		
8		ISAP1 ! IDISreq	TestCase2_008		
9		ISAP1 ? IDISind	TestCase2_016		
10		MSAP2 ? MDATind	TestCase2_011	PASS	
11		MSAP2 ? MDATind	TestCase2_011		
12		ISAP1 ? IDISind	TestCase2_016	PASS	
13		ISAP1 ? IDISind	TestCase2_016	INCONC	
14		MSAP2 ? MDATind	TestCase2_014	INCONC	
15		ISAP1 ? IDISind	TestCase2_016	INCONC	
16		ISAP1 ? IDISind	TestCase2_016	INCONC	

Detailed Comments:

Figure 6: TTCN test case description for *StructuredTestRun*

Test Case Dynamic Behaviour

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

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+Preamble			
2		ISAP1 ! IDATreq	TestCase2_005		
3		MSAP2 ? MDATind	TestCase2_006	(PASS)	
4		MSAP2 ! MDATreq	TestCase2_007		
5		+Postamble			
6		ISAP1 ? IDISind	TestCase2_016	INCONC	
		Preamble			
7		ISAP1 ! ICONreq	TestCase2_001		
8		MSAP2 ? MDATind	TestCase2_014		
9		MSAP2 ! MDATreq	TestCase2_003		
10		ISAP1 ? ICONconf	TestCase2_004		
11		MSAP2 ? MDATind	TestCase2_014	INCONC	
12		ISAP1 ? IDISind	TestCase2_016	INCONC	
13		ISAP1 ? IDISind	TestCase2_016	INCONC	
		Postamble			
14		ISAP1 ! IDISreq	TestCase2_008		
15		ISAP1 ? IDISind	TestCase2_016		
16		MSAP2 ? MDATind	TestCase2_011	PASS	
17		MSAP2 ? MDATind	TestCase2_011		
18		ISAP1 ? IDISind	TestCase2_016	PASS	

Detailed Comments:

Figure 7: Test steps stored as local trees

of this project is the development of a test suite for the INAP CS2 protocol using automated software tools. Before that, several parties have used Telelogic's Link tool for semi-automatic test suite generation [1].

Experiences with Link and AUTOLINK have shown that today the amount of time which has to be spent with manual post-processing of test suites is too high. It consumes too much of the time which has been won by the automatic generation in the first place. Therefore, improvements of the readability of the generated TTCN code are required with high priority. Later, methods to control the state space explosion during the exploration will have to be implemented. In particular, we plan the following extensions:

- **Detection of identical subtrees.** If there is a series of consecutive receive events at different PCOs, then all permutations of the order in which these events are observed have to be generated. Currently, the complete subsequent event tree is duplicated for every permutation. By detecting permutations of receive events and storing them as local trees, the readability of test cases will be improved a lot.
- **Support of concurrent TTCN.** Concurrent TTCN is not supported today, because not all of the necessary information (e.g., declaration of test components) can be deduced from the SDL system or the system level MSC. This information will have to be supplied by the user. Different strategies for the automatic generation of coordination messages will be implemented, e. g. synchronisation of all parallel test components before each send event.

- **Support of timers.** Timers on environment axes in the system level MSC will be translated into TTCN timers.
- **Support of test groups.** A test suite is normally structured into *test groups*. A test group is a set of test cases which focuses on testing a specific aspect of the specification. At the moment, this structuring has to be done in a post-processing step, i.e., after test case generation. In practice this is done before or during the test purpose definition. Therefore we intend to develop a mechanism which allows to relate paths, test purposes and test cases to test groups. This means that the next version of AUTOLINK will generate parts of the TTCN overview part automatically from the provided additional structuring information.
- **Definition of stable testing states.** Test cases should start and end in *stable testing states*. A stable testing state is a state where the system has to wait for an input from the environment before it is able to continue. Currently the user has to define the whole path from the start state of the SDL specification to a state from where the test purpose can be tested. But in practice, test cases do not necessarily begin in the start state of the SDL specification. Therefore we intend to provide a mechanism which allows to define and search for stable testing states as start states for the test case development.
- **Support of TTCN matching mechanisms.** AUTOLINK version 1 calculates the constraints for receive events from the selected path. The constraints include concrete values for the ASP and/or PDU parameters. In practice, during a test run often the concrete values of some parameters are not important, or a whole range of values is allowed. Currently, in such cases the TTCN output has to be modified manually, i.e., TTCN matching mechanisms have to be introduced. We intend to avoid this post-processing step by allowing to use TTCN matching mechanisms in MSC test descriptions.
- **Parameterisation of test steps.** Currently, a lot of basically identical test cases are generated, which only differ in the parameters of signals sent to and received from the implementation under test. By providing mechanisms for the parameterisation of test steps, the number of test cases in a test suite can be reduced significantly.
- **Structuring of test cases based on High Level MSCs (HMSCs).** In the next AUTOLINK version it will be possible to use HMSCs for test specification. The structuring information from HMSCs will be used to generate better readable test cases.

Further plans include the automatic generation of *Unique Input/Output (UIO) sequences, preambles and postambles*. It is generally acknowledged that the main problem of finding UIO sequences, preambles and postambles is the explosion of the state space during its exploration. Therefore our work will focus on developing strategies and mechanisms for dealing with this problem. Especially, we will start to implement tools for a *static* and a *dynamic pre-investigation* of SDL specifications and for performing *measurements of the test generation capabilities*.

Static pre-investigation will be based on a data flow analysis. It will provide information about all message parameters that influence the behaviour of the SDL specification. This information can be used to manually define the actual parameter values which have to be sent by the test equipment during a test run.

Dynamic pre-investigation will be based on symbolic execution of the SDL specification. It will allow to calculate "optimal input data" and to propose stable testing states.

While the number of states which can be examined by test case generators in a certain amount of time is almost constant, the complexity of the search within the state space of an SDL specification depends on the options and search heuristic chosen by the user. The complexity and thus the capabilities of a test case generator for a particular SDL specification can be measured by a number of simulation runs. An automatic comparison of the results will provide information for a reasonable choice of options and heuristics.

Acknowledgements The authors like to thank Anders Ek and Per-Olov Nilsson from Telelogic AB for their continuous support of the AUTOLINK project. Many thanks go to Stefan Heymer for proof-reading of this paper.

References

[1] 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. Schriftenreihe der Institute für Informatik\Mathematik Report A-97-03, Medizinische Universität zu Lübeck, Lübeck, January 1997.

[2] D. Hogrefe. *Estelle, LOTOS und SDL - Standard Spezifikationssprachen für verteilte Systeme.* Springer, 1989.

[3] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[4] B. Koch and M. Schmitt. AUTOLINK *Version 1 User Manual.* Telelogic AB, Malmö, March 1997.

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