Towards the Industrial Use of Validation Techniques and Automatic Test Generation Methods for SDL Specifications

Anders Ek[a], Jens Grabowski[b], Dieter Hogrefe[b], Richard Jerome[c], Beat Koch[b], and Michael Schmitt[b]

[a]Telelogic AB, P.O. Box 4128, S-20312 Malmo, Sweden, eMail: anders.ek@telelogic.se

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

[c]Ericsson Limited, Public Systems Division, Ericsson Way, Charles Avenue, Burgess Hill, West Sussex RH15 9UB, England, eMail: etlrdje@etlxdmx.ericsson.se

Due to increasing demands from companies and standardisation bodies, Telelogic AB and the University of Lübeck started a research and development project in October 1996 which aims at improving the validation and, especially, the automatic test generation facilities of the SDT/ITEX tool set. The project is driven by practical experiences and practical needs, but also takes care of research results. In this paper, we present two short experience reports and describe the project.

1. INTRODUCTION

The ITU-T Specification and Description Language (SDL) [16] is worldwide the most successful standardized formal description technique. SDL has been used successfully in industrial projects and for standardization purposes. Lots of SDL success stories can be found in the proceedings of the past SDL Forum conferences [1,6–8].

The SDT/ITEX tools from Telelogic AB are one of the most successful commercial SDL tool sets. They provide a complete environment for the development of SDL specifications, SDL based implementations and TTCN based test suites. For this, they include graphical editors, analysers, simulation tools, various browsers, application code generators adapted to different real time kernels and further support tools.[1] Besides SDL, SDT/ITEX support the specification languages MSC [17] and TTCN [14].

One tool of SDT/ITEX is the Validator [4]. The Validator is aimed at providing engineers with the possibility to increase the quality of their work and to automate time-consuming tasks. More specifically, the tool is designed to help engineers in three situations:

1. *During an incremental design*, by providing an automated fault detection mechanism that finds inconsistencies and problems in an early stage of development.

[1]An overall view of SDT/ITEX can be found in Section *'XII Demonstrations'* of [1].

2. *When verifying the system against requirements*, by providing an option for automatic MSC verification.
3. *When developing test cases*, by providing options for automatic test generation.

The items 1 and 2 are referred to by the term *validation*, whereas item 3 is referred to by *automatic test generation*. Validation and automatic test generation are closely related, because basically the same techniques can be used for both.

The Validator is based on the *state space exploration* technique (e.g., [12]), which is a well known technique for the automatic analysis of distributed systems. Using this technique a system property is validated by building up and examining the system's state space. Examples for properties which can be verified are freedom of deadlocks or the fulfilment of an MSC.

In general, for automatic test generation we are also interested in finding a property. Furthermore, we need to find a trace which can be used for testing the property in a system implementation. In this context a property is referred to by *test purpose*. Automatic test generation becomes a little bit more complex, because we also have to follow testing methodologies [13] and to transform the resulting test sequences into the TTCN format.

Validation and automatic test generation both have to deal with complexity. Due to the *state space explosion* problem it is often impossible to perform an exhaustive validation or to generate test cases automatically even for small systems.

Currently, we observe an increasing demand for advanced validation techniques and automatic test generation methods from SDT/ITEX customers. Research and case studies have shown that automatic test generation becomes feasible [2,9,10].

Therefore Telelogic AB and the University of Lübeck have set up a research and development project called AUTOLINK. AUTOLINK improves the validation and especially the test generation facilities of SDT/ITEX. The project is driven by practical experiences, but it also considers research results.

The paper continues as follows. In Section 2 the Validator is sketched. SDL validation experiences from Ericsson are presented in Section 3. The need for automatic test generation methods will be explained by an ETSI experience report in Section 4. The AUTOLINK project, its current status and the future plans are described in Section 5. Finally, the conclusion and outlook are given.

2. THE SDT VALIDATOR

The Validator is based on the state space exploration technique (e.g., [12]). In the SDL context this means that the state space of an SDL system is built up, stored in a directed graph and examined. The directed graph is referred to by *reachability graph* and represents the behaviour of the SDL system. The nodes of the reachability graph represent global SDL system states. The edges describe SDL events that can take the SDL system from one global system state to the next global system state. A global SDL system state, i.e., a node in the reachability graph, is characterized by:

- the active process instances,
- the variable values of all active processes,
- the SDL control flow state of all active process instances,
- the active procedures (with local variables),

- the signals (with parameters) that are present in the queues of the system,
- the active timers,
- etc.

The edges in a reachability graph define the atomic events of the SDL system. These can be SDL statements like tasks, inputs, outputs etc., but also complete SDL transitions depending on how the state space exploration is configured.

During validation the reachability graph is analysed. Properties of the reachability graph describe properties of the system behaviour. For example, a system is free of deadlocks if all nodes in the reachability graph have at least one outgoing edge.

2.1. MSC verification

The Validator provides two options to check an SDL specification against system requirements. The requirements have to be expressed in form of observer processes (see Section 2.2) or MSC diagrams. State space exploration techniques are used for both options. For an MSC requirement, the Validator explores the state space, i.e., it simulates the SDL specification, and searches for a path in the reachability graph which includes the MSC. If such a path exists the SDL system has the MSC property, i.e., the MSC is verified.

2.2. Observer processes

An observer process allows to check more complex requirements on systems than can be expressed by MSCs. The notion is to use a special kind of SDL processes, called *observer processes*, to describe the requirements that are to be checked. Then the observer processes are included in the SDL system. Typical application areas for observer processes are test case generation, feature interaction analysis, or the analysis of safety-critical systems.

To be useful, observer processes must be able to inspect the SDL system without interfering with it, and also to generate reports that convey the success or failure of whatever they are checking. To accomplish this, three features are included in the Validator:

1. *The observer process mechanism.* By defining processes to be observer processes, the Validator starts to execute in a two-step fashion. First, the rest of the SDL system executes one transition, and then all observer processes execute one transition and check the new system state.
2. *The assertion mechanism.* The assertion mechanism enables the observer processes to generate reports during state space exploration. These reports will show up in the list of generated reports in a special report viewer.
3. *The Access abstract data type.* The purpose of the Access abstract data type is to give the observer processes a possibility to examine the internal states of other processes in the system. Using the Access ADT it is possible to check variable values, contents of queues, etc., without any need to modify the observed processes.

There are two main characteristics of observer processes:

1. Continuous signals are used which check the internal state of other processes using the Access operator.

2. Assertions are used to report the result of the test.

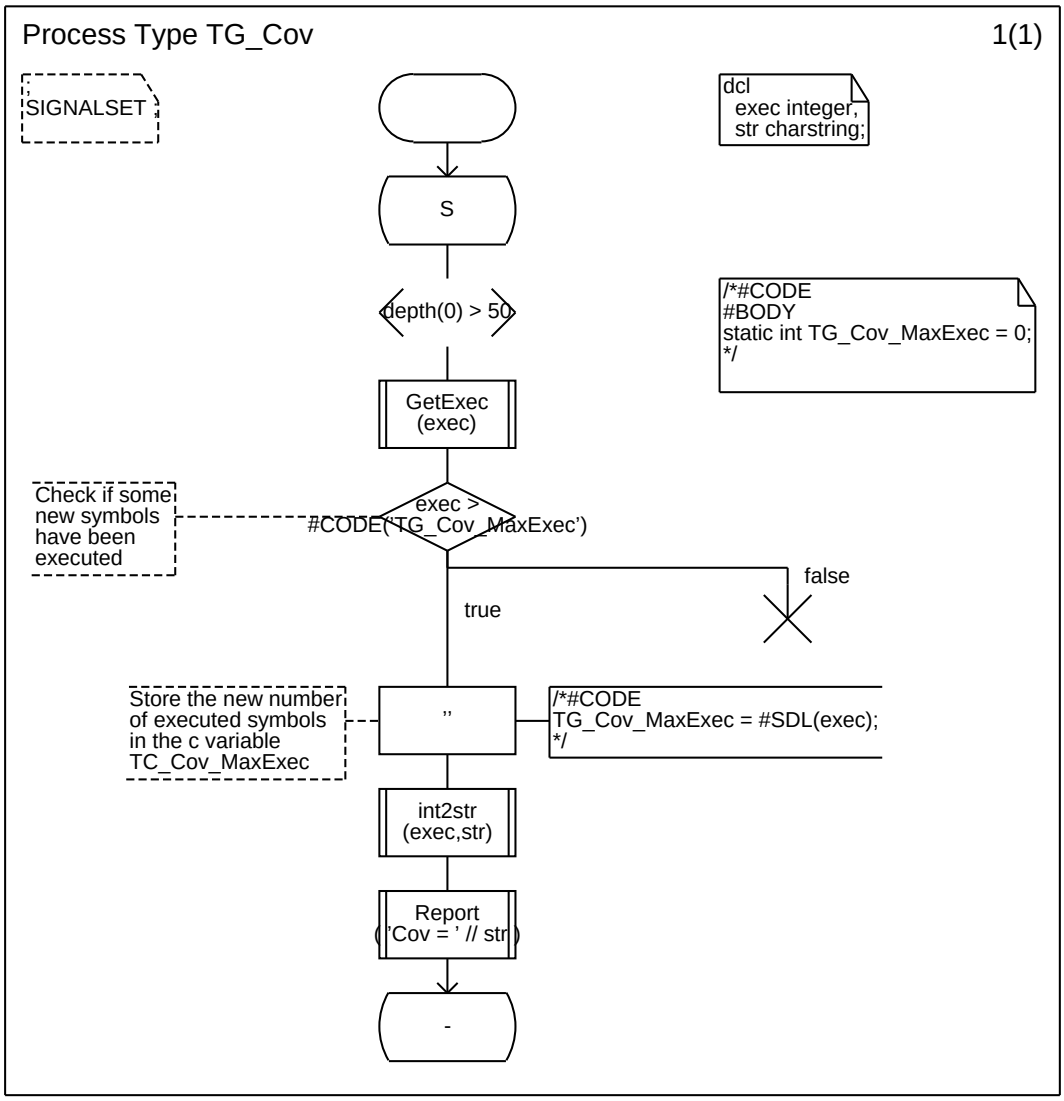

Figure 1. A simple observer process

A simple observer process type is shown in Figure 1. *TG_Cov* realizes a symbol coverage based strategy for the generation of reports. Each time a trace with length 50 is found that covers an additional SDL symbol, a new report is generated. SDT provides a mechanism for accessing variables and functions of the Validator's C Code in an SDL statement. This allows to retrieve information about the current status of the state space exploration. The observer process and its application for test generation will be described in detail in Section 5.2.

2.3. Using the Link tool for semi-automatic test generation

SDT/ITEX offers the option to link SDL specifications and TTCN descriptions by means of the Link tool. For this a *.link* file is generated from the SDL specification. This file is an executable program which in combination with ITEX supports the semi-automatic

(a) Existing system (b) Required system

Figure 2. Existing and required system

construction of TTCN test suites. The Link tool allows to generate the following parts of a test suite:

- ASP (Abstract Service Primitive) definitions are generated automatically. All SDL signals appearing on channels to the environment are translated to ASN.1 ASPs.
- PCOs (Points of Control and Observation) are generated automatically. All channels to the environment are considered to be PCOs.
- If the SDL signals are structured, then these structures are translated to ASN.1 type definitions.
- The test case behaviour can be constructed semi-automatically. The user chooses a *send* event appropriate for the test purpose, and the Link tool adds the following correct *receive* events to the test case description. Then the user has to select the next *send* event.
- A single default behaviour description for the test suite is generated.
- There is a limited generation of constraint definitions.

3. ERICSSON EXPERIENCES

Ericsson is a global telecommunications company that supplies a vast range of hard- and software solutions to network operators all around the world. For one of the current projects in the UK, Ericsson has used SDT/ITEX to design new application software for Ericsson AXE10 telephone exchanges. These exchanges are already in use and form an integral part of the UK switching network.

Customer requirements for such projects are normally provided in form of lengthy textual descriptions. These can prove imprecise when, for example, describing signalling systems or protocols. For the project in the UK there was an agreement to express the requirements in form of an SDL specification. Furthermore SDL was used for the system design.

3.1. The SDL requirements model

The project required to introduce a new signalling system into the existing network. As shown in Figure 2a, the *existing system* consists of the two in- and outputs A and B, with associated functionality X and Y, respectively. The *required system* (Figure 2b) consists of the three in- and outputs A, B and C, with associated functionality X, Y and Z, respectively.

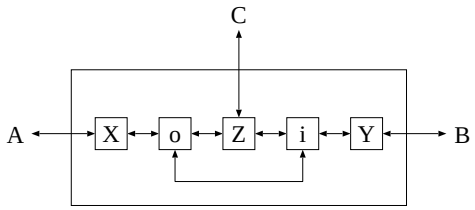

Figure 3. Design model

The system in Figure 2b can be seen as a black box model for the functional requirements to be fulfilled by the existing and the new functionality. The system has been described in SDL and served as functional requirements model (FRM) for validation purposes.

3.2. The SDL design model

The interfaces between the signalling in- and outputs A, B and C, and the functionality of X, Y and Z are clearly understood. The new components of the required system, i.e., the signalling system C with functionality Z already existed. Thus, single SDL specifications for A/X, B/Y and C/Z were developed.

For the required system design (Figure 3), the new functionality C/Z was incorporated in and interfaced with the existing system functionality by means of the additional software blocks o and i.[2] SDL models for o and i were produced and the (functional) SDL design model (FDM) completed. The FDM can be seen as a protocol conversion exercise, with o and i as conversion blocks.

3.3. Validation

The problem of software development is to detect faults as early as possible. For the Ericsson project, a small inter-work fault at FDM level might take an hour to fix, but it could take hundreds of hours if it is not found until the software starts to work on an AXE10 telephone exchange. Thus, the sooner the FDM can be validated against the FRM, the better.

The SDT Validator was used for the validation of FRM and FDM. Early thoughts were that this could just be used to ensure that the FDM did not have any inherent errors like deadlocks or infinite loops; that it described some kind of reliable and stable system. But as a result of the design process two SDL specifications FRM and FDM were developed which had identical external interfaces A, B and C. Therefore the Validator could be used to check if the FDM met the requirements provided by the FRM.

The validation was done in a two step procedure. In a first step all relevant execution traces of the FRM were generated. In a second step each trace was transformed into an MSC and loaded into the FDM. By using the MSC verification feature of SDT/ITEX the traces of the FRM were shown to exist in the FDM.

[2]It should be noted that this design includes abstraction and simplification. Depending on the call type, the system may have two instances of i and none of o. Furthermore, the system may be instantiated more than once. During operation a telephone exchange may maintain up to 64000 instances of the system.

Step 1: Recording FRM traces
Step 1 was performed by adding an observer process (Section 2.2) to the FRM. The observer process was used to control the exploration of the FRM state space. It recorded the path from the *idle* state to predefined global system states and determined the state-space coverage at these states. If the actual path increased the coverage, it was saved. If not, exploration continued. Further improvement of the result of step 1 was achieved by doing some fine tuning, e.g., considering only significant signal parameter values, i.e., the one causing branches in SDL processes.

After several days of execution, ~200 usable paths had been generated. The paths had a length of ~100 transitions and provided about 70% coverage of the system. Further 20% of coverage were achieved by manually navigating to the roots of unexplored branches in the state space and by implementing a rule which prevented to return to the *idle* state during the exploration.

Step 2: MSC verification
In step 2, the paths generated in step 1 were transformed into MSCs. Then each MSC was loaded into the FDM validator and, if possible, verified. An *unverified* MSC corresponded to an error in *either* of the models. Errors in the FDM model are easily rectified, but faults in the FRM cause a repetition of step 1, because MSCs generated from an erroneous specification may reflect the errors. In the AXE10 project the MSCs had to be regenerated only twice.

3.4. Results and discussion
It took three weeks for two people to (re)generate the MSCs and work through the faults. About 80 discrepancies between the FRM and FDM were detected and corrected. The majority of these were inter-working errors, usually the hardest of all errors to find. All errors were removed and the SDL specifications of FRM and FDM were corrected until all 260 MSCs were verified. It has been estimated that the 260 verified MSCs covered more than 60% of the FDM and approximately 90% of the FRM.

Quantifying the benefit of the Validator in this project is hard. Certainly without the validation process, errors would have slipped through to subsequent design phases. Inter-work errors are notoriously hard to find, and they cause large problems in software projects. It may be enough to say that the software has progressed from the SDT/ITEX environment through testing on AXE10 emulators and onto real AXE10 exchanges with no major difficulty. Normally projects of this magnitude prove rather more troublesome.

Subsequent demonstrations to the customer made a good impression. Test cases running on the SDT Simulator and on Ericsson's AXE10 emulators showed the required signalling interaction, which matched precisely in both environments. Without doubt, the SDT Validator will be used again. Additional options for automatic test case generation would be most useful, although most of the generated MSCs do not translate well into real-life test cases.

4. ETSI EXPERIENCES

Within Project Team 65 ETSI started an effort to experiment with test case generation methods [3]. Three different protocols were selected for experimentation: INRES [11],

INAP CS2 [5] and B-ISDN [15]. The protocols were specified with SDL. For each protocol one or two test purposes were specified as an example. Then the tools SDT Link from Telelogic and TTCgeN from Verilog were taken to generate test cases. For detailed results, see [3].

Some general observations can be derived from these experiments.

The automatic generation of the static declarations (PCOs, ASP types, ...) is very useful. The test case developer is freed from a time consuming task, all provided that the SDL specification already exists and is correct.

The dynamic test case generation is semi-automatic and performed in a step-by-step manner. In the case of SDT Link, a test purpose is performed by sending an appropriate signal to the SDL specification together with the constraint. The tool then generates all possible responses together with their constraints automatically. This is repeated until the test case is defined. The approach of TTCgeN is slightly different on first sight. It takes a complete MSC as input and then generates the TTCN in one go. The MSC has to be developed in a step-by-step manner with simulation. Therefore in effect the tools follow approximately the same strategy.

Most of the time is spent with defining the constraints. The first constraint has to be entered manually. Then SDT Link generates the outputs together with their constraints. In TTCgeN this is done on the MSC basis. Usually these automatically generated constraints can then be used, maybe in a slightly modified form, to serve as constraints for further inputs. Due to the tool support the user can speed up the constraint development considerably by reusing automatically generated constraints.

The use of the tools normally tempts into making rather large test cases compared to the single state transition test cases which are typically developed by hand. The resulting test suite then consists of a smaller number of large test cases compared to the traditional large number of small test cases. The reason for this is the fact that the SDL model allows to very easily perform a complete scenario in a simulative way. This has certain advantages and disadvantages over the state-by-state approach, which for some test case developers tends to be rather boring. The advantage is that the complete scenarios can be used as acceptance tests for a tested product. The test is performed from a user's perspective. The disadvantage is that the coverage may not be as complete as if every single state transition was tested.

The tool-supported test generation seems to be of particular benefit for those protocols which define a lot of states and transitions instead of data. These state oriented protocols can very naturally be defined in SDL (and they usually are nowadays). Therefore the effort to produce an SDL specification suitable for test case generation is acceptable. This was the case for INRES and INAP CS2. For data oriented protocols an SDL specification may not be so natural. Most of the protocol is defined in terms of parameter variations instead of dynamic behaviour. The effort of producing an SDL specification suitable for test case generation is considerable. This was the case for B-ISDN.

To sum up, the experience made by ETSI within PT65 was promising enough to continue one of the experiments in a follow-up project. A project team has been set up to generate a test suite for INAP CS2 from the SDL specification using software tools. The results from this first tool-supported test case generation with the aim of creating a complete test suite within ETSI will be available in early 1998.

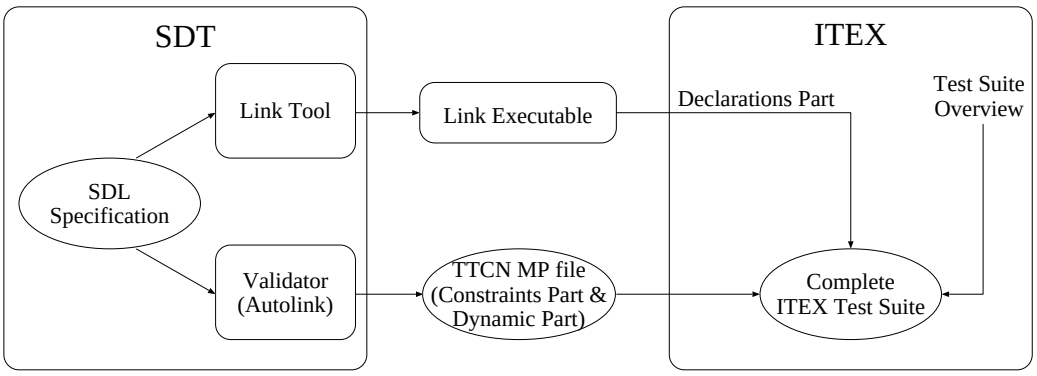

Figure 4. System overview

5. THE AUTOLINK PROJECT

The main objective of the AUTOLINK project is the development and implementation of an SDT component which supports the automatic and semi-automatic generation of TTCN test suites based on SDL specifications.

While SDT allows the user to construct an SDL specification and test it with the Validator, ITEX on the other hand supports the user in the development of TTCN tests suites. So far only the Link tool has served as a bridge between SDT and ITEX (Section 2.3).

With the AUTOLINK component the connection between SDT and ITEX becomes even tighter (Figure 4). Now, the user can specify the test cases within the SDT Validator and let the system generate a TTCN test suite with constraints and dynamic behaviour tables. This test suite can then be completed in ITEX with the declarations provided by the Link tool.

5.1. The first step: path based test generation

The generation of a TTCN test suite with the first AUTOLINK version proceeds in several steps. These are outlined in Figure 5. Most of them can be executed repeatedly as indicated by the loops. In the following we describe each step in more detail.

Define paths

The basis for the generation of a TTCN test case is a *path*. A path is meant to be a sequence of events that have to be performed in order to go from a start state to an end state. The Validator provides several mechanisms for selecting such a path in the state space of an SDL specification. They are sketched in Section 5.2. For AUTOLINK, the externally visible events of a path describe the test sequence to which a *pass* verdict is assigned.

All defined paths are transformed into and stored as *system level MSCs*. By doing this, we omit all internal events of the specification. A system level MSC shows the external interaction that is supposed to take place between an implementation and the test system when executing the test case.

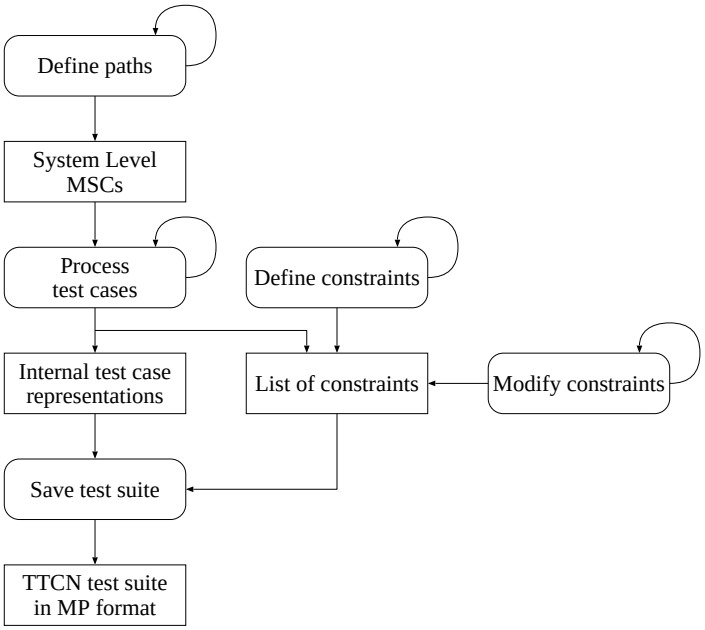

Figure 5. Test Suite Generation

Process test cases

During the processing of a test case, a sequence of send and receive events leading to a TTCN *pass* verdict is created. In addition, all alternative receive events are looked up and added to the test case with a TTCN *inconclusive* verdict.

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.

A modified version of the Validator's bit-state exploration algorithm is used to build up the internal data structure which represents the current test case. The data structure basically is a tree. Each node of that tree represents an event on a path to a TTCN *pass* verdict. It contains information about the current event, a list of the event nodes with *pass* verdict on the next level, and a list of events with TTCN *inconclusive* verdict on the next level. Here is a short description of the exploration algorithm. Figure 6 shows the data structures involved in the exploration. The state space is represented as a graph. Let us assume that we are in state 4 on level n and $p2$ is the current node in the Autolink event tree. Now state 5 on level $n+1$ 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 *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

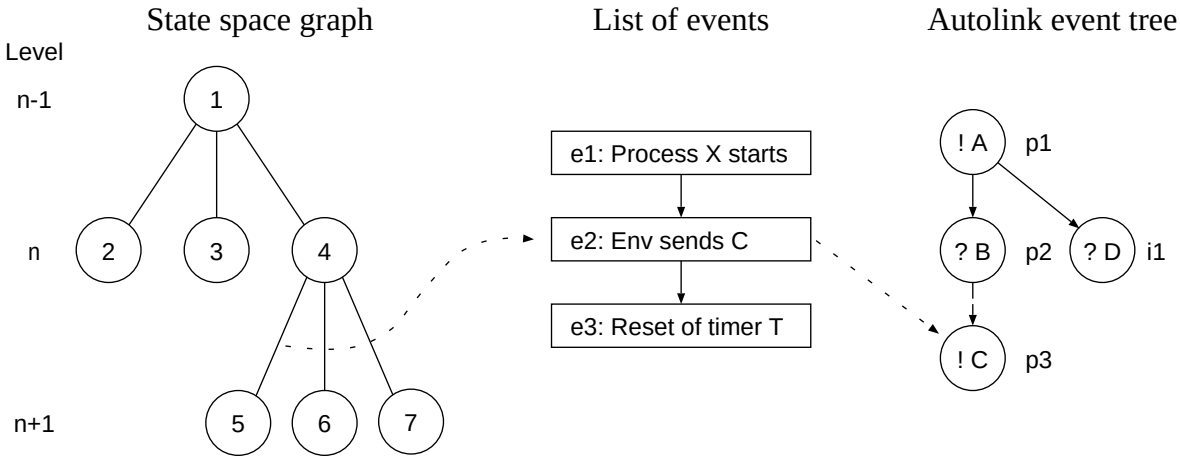

Figure 6. Test case processing algorithm

has been reached. To continue the exploration, the first state following state 5 on level $n + 2$ 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 state on the same level has to be computed. If we had such an event in the transition from state 1 to state 2 in Figure 6, e.g., then the transition from state 1 to state 3 would be computed next.

For every level n the current node $p(n)$ in the Autolink event tree is saved. If there is no next state on level n, then the next state on level $n - 1$ is computed and the saved $p(n - 1)$ becomes the current node in the Autolink event tree. The exploration ends if no more states are found below the start state.

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 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.

Define and modify constraints

Constraints are automatically created during test case processing. Additionally, the user can define and remove signal constraints at any time. It is also possible to assign more reasonable names to constraints generated by AUTOLINK and to merge two constraint definitions. This is useful if the constraints have the same meaning or if a signal parameter with different values in the two constraints is irrelevant.

Save test suite

The TTCN test suite generated by AUTOLINK is written in MP format. It mainly consists of two sections – the constraints part and the dynamic behaviour part.

Constraints are saved in ASN.1 format. Thereby a problem has to be solved that is caused by the different notation 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, the missing names are formed by the corresponding data type and the parameter number. For example, a constraint for an SDL signal with three integer parameters (55, 0, -55) will be mapped onto `SEQUENCE { integer1 -55, integer2 0, integer3 55 }`.

Dynamic behaviour tables can be created in a straight-forward fashion by recursively traversing the internal representation for each test case. By using the depth first search strategy, it is possible to output a dynamic behaviour table in one pass.

The TTCN format is a general purpose test description format that can be used in many testing situations; both when testing an implementation on a target platform and when testing SDL systems in a simulated environment. In SDT the support for target testing is given by the ITEX TTCN environment and SDL level testing is provided by a co-simulation possibility between the SDL and TTCN simulators available in the SDT/ITEX tool set.

5.2. State space navigation

The SDT Validator offers several different ways to specify paths (and thus test cases), including both selective and brute force strategies. Some examples are:

- Manual navigation/simulation in the state space
- Using MSCs as input
- Using observer processes for a brute force strategy

All three methods to define a path can also be combined.

Manual navigation

The SDT Validator provides a special window called the *Navigator* (Figure 7) that allows users to manually explore the state space and thereby to define paths in an efficient way.

MSC verification

MSCs provide another way to define paths. In this case the idea is to create an MSC, either manually or by using, for example, the SDT Simulator, and then make the Validator search for a path that satisfies the MSC. This path is then used as the test purpose.

Observer processes

Observer processes are sophisticated constructs that allow a completely automatic definition of a large set of tests for an SDL system.

The general idea when using observer processes for test generation is to encode a high-level test purpose in an observer process. Each time a path is found by the Validator which satisfies the condition defined in the observer process, a report is generated. At the end of the state space exploration, the reports can be converted into MSC test cases.

To simplify things for users a special SDL package called the *TestGen* package has been developed. This package includes a number of utilities including two observer process types that can directly be used in test generation applications:

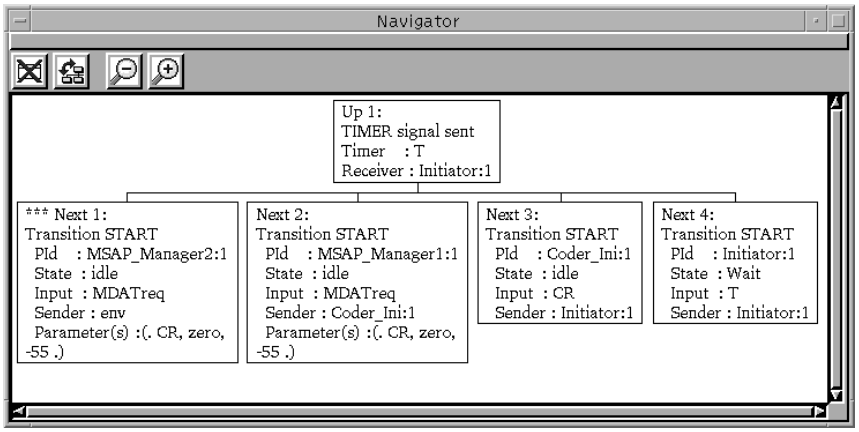

Figure 7. The SDT Navigator

- A process type for random test generation.
- A process type for test generation based on process graph coverage.

Both process types are delivered in source code and can be modified by users to suit the needs of special applications.

The random test generator process simply encodes the following rule:

> Whenever a path is found that contains more than a given number of transitions, a test is generated.

This observer process is intended to be used together with the *random walk algorithm* in the Validator. If it is included in an SDL system and a random walk through the state space is executed with a certain number of repetitions, one test case will be generated for each path that contains the given number of transitions.

The observer process is mainly included in the package in order to let users modify it with application specific test purpose conditions. If it is used as is, it will generate many equivalent test cases.

The second observer process is a very useful modification of the first one. It encodes the following test purpose condition:

> Whenever a path is found which contains more than a given number of transitions, and this particular path includes SDL process graph symbols that have *not* been covered by previously generated tests, then a test is to be generated.

This observer process is also intended to be used in combination with the random walk algorithm in the Validator. If it is included in an SDL system and a repeated random walk through the state space is executed with a certain number of repetitions, the following happens:

- The first random walk will generate a test case when the indicated depth is reached.
- All subsequent random walks will generate new test cases for all paths that cover new parts of the SDL process graph in the system.

Practical experiments have shown that this test generation strategy is a very efficient means to generate a large number of test cases automatically. Taken together, these two methods for test generation give a fairly good code coverage of the SDL system.

5.3. Future plans

The next version of AUTOLINK will provide more sophisticated mechanisms for the definition of test cases. At the moment the user has to provide complete SDL paths leading to a TTCN *pass* verdict. Via system level MSCs, these paths are transformed into TTCN notation. Additionally, all responses leading to an *inconclusive* verdict are computed. In the next version of AUTOLINK it will also be allowed to specify a test case by providing only a part of a trace leading to a *pass* verdict. Such parts are often referred to by the term *test purpose*. AUTOLINK will search for adequate completions, and pre- and postambles for a given test purpose. A test purpose may be given in form of an MSC, an observer process, or a piece of an SDL diagram which has to be executed during the test.

From research we know that the main problem of finding completions, pre- 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 of an SDL specification. A static pre-investigation may 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 helps to manually define concrete parameter values, which have to be provided by the test equipment during the test run.

Dynamic pre-investigation of an SDL specification. A dynamic pre-investigation may be based on a symbolic execution of the SDL specification. It allows to calculate the 'optimal input data' automatically and to propose stable testing states.

Measurement of the test generation capabilities. 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.

With test suite validation we recognized another important problem. Due to the different nature of TTCN and SDL it is nearly impossible to generate complete TTCN test descriptions from SDL specifications only. Therefore a generated test suite has to be refined manually. This on the other hand may introduce new errors or ambiguities into the test suite. Our work will also focus on providing tools for performing test suite validation against an SDL specification.

6. SUMMARY AND OUTLOOK

We have shown the current possibilities for tool assisted test case generation using the SDT Validator and ITEX. Practical use of these methods has indicated that an improvement of the test case generation facilities is needed and possible. Therefore, the AUTOLINK project has been set up. The first, basic version of the AUTOLINK tool has been described in this paper. In the future, AUTOLINK will be enhanced to include a pre-investigation of the SDL specifications, the measurement of the test case generation capabilities and additional test case generation mechanisms.

REFERENCES

1. R. Bræk, A. Sarma (editors). *SDL'95 with MSC in CASE*. North-Holland, 1995.
2. L. Doldi, V. Encontre, J.-C. Fernandez, T. Jéron, S. Le Briquir, N. Texier, M. Phalippou. *Assesment of Automatic Generation of Conformance Test Suites in an Industrial Context*. In 'Testing of Communicating Systems' (B. Baumgarten, H.-J. Burkhardt, A. Giessler, editors). Chapman & Hall, 1996.
3. ETSI MTS 10/96 TD42. *Report on Automatic Generation of TTCN from SDL*. ETSI, 1996.
4. A. Ek. *Verifying Message Sequence Charts with the SDT validator*. In [8].
5. ETSI STC SPS 3. *Intelligent Network (IN); IN Capability Set 2 (CS2); Scoping of Intelligent Network Application Protocol (INAP)*. DTR/SPS-03043, ETSI, 1996.
6. O. Færgemand, M. M. Marques (editors). *SDL'89: The Language at work*. North-Holland, 1989.
7. O. Færgemand, R. Reed (editors). *SDL'91: Evolving Methods*. North-Holland, 1991.
8. O. Færgemand, A. Sarma (editors). *SDL'93 - Using Objects*. North-Holland, 1993.
9. J. Grabowski, D. Hogrefe, R. Nahm. *Test Case Generation with Test Purpose Specification by MSCs*. In [8].
10. J. Grabowski, R. Scheurer, D. Hogrefe. *Applying SAMSTAG to the B-ISDN Protocol SSCOP*. Technical Report A-97-01, Medizinische Universität zu Lübeck, Schriftenreihe der Institute für Mathematik/Informatik, Lübeck, January 1997.
11. D. Hogrefe. *OSI Formal Specification Case Study: The Inres Protocol and Service* (revised). Technical Report IAM-91-012, University of Bern, Institute for Computer Science, May 1991, Update May 1992.
12. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International, Inc., 1991.
13. ISO/IEC. *Information Technology - OSI - Conformance Testing Methodology and Framework*. International ISO/IEC multipart standard No. 9646, 1994.
14. ISO/IEC. *Information Technology - OSI - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN)*. ISO/IEC IS 9646-3, 1996.
15. ITU-T Rec. Q.2931. *Broadband Integrated Services Digital Network (B-ISDN) - Digital Subscriber Signalling System No. two (DSS2) protocol*. Geneva, 1995.
16. ITU-T Rec. Z.100 (1996). *Specification and Description Language (SDL)*. Geneva, 1996.
17. ITU-T Rec. Z.120 (1996). *Message Sequence Chart (MSC)*. Geneva, 1996.