Test Case Generation with Test Purpose Specification by MSCs[*]

Jens Grabowski, Dieter Hogrefe, and Robert Nahm

Institut für Informatik, Universität Bern, Länggassstrasse 51, CH-3012 Bern

This paper presents a new test case generation method based on formal system specifications in SDL. The test purpose of a specific test case is specified formally by one or many Message Sequence Charts (MSCs). Based on the test purpose and the system specification a complete test case can be generated automatically in the TTCN format, including preamble, postamble and test body with all test verdicts.

1. Introduction

Formal description techniques (FDTs, i.e. LOTOS, Estelle and SDL) are frequently used within industry and standardization bodies to describe the functional properties of communication systems (e.g. OSI or ISDN). FDT descriptions can be simulated. Therefore, the possible interactions between a system and its environment can be generated automatically. Although test cases describe such interactions, the automatic generation of test cases from FDT descriptions is still an open problem. Furthermore, there exists a gap between research and practical testing.

Approaches coming from research like UIO [20] or the W-method [6] can handle systems with a small state space. They test every state transition exactly one time. From a successful test a behavioural equivalence between specification and implementation can be concluded. The problems of these methods are state explosion and infinite state spaces.

State explosions occur for example when state space exponentially grows with the number of processes, or with the size of buffers. Even small examples may cause problems for UIO or the W-method. None of the mentioned methods can be applied to systems with an infinite state space. Unfortunately, FDTs force the specification of systems with an infinite state space. Infinite signal queues of SDL processes or unlimited data descriptions are two examples for this. However, there cannot exist test methods which guarantee behavioural equivalence for systems with an infinite state space. Even finite state machines which communicate by means of unbounded FIFO buffers (i.e. the base model of SDL) are as powerful as Turing Machines [2] for which the behavioural equivalence is undecidable [14]. For testing, the situation is more complicated since there is in general no knowledge about the whole implementation. Only the interactions between an implementation and its environment are observed for a certain time.

Real systems are in general very complex. Therefore, approaches like UIO or the W-method cannot be applied. The present procedure of writing test cases is an intuitive and creative process which only is restricted by informal regulations. The intuition behind a

[*]This work was performed within the F & E project, no. 233, *'Conformance Testing - A Tool for the Generation of Test Cases'*, funded by Swiss PTT.

test case is reflected by the so-called *test purpose*. A test purpose is an informal statement. It denotes an *important part of a specification* which should be tested.

Our approach supports practical testing. It combines test purposes defined by Message Sequence Charts (MSCs) [5, 8] and a corresponding SDL description [4] in order to generate test cases. MSCs (cf. Figure 2) are widespread means for the graphical visualisation of selected system runs of communication systems [9]. A test purpose can be defined by an MSC in form of the required signal exchange[2]. An MSC does not define a complete test case. It does not describe the signal exchange which drives the implementation into a state from which the MSC can be performed (*preamble*). It does not define the stimuli which are necessary to drive the implementation back into an initial state after the MSC is observed (*postamble*). It does not define what to do if a signal is observed which is not defined in the MSC, and it does not describe the values of message parameters. The missing information can be provided by an additional FDT description. We choose SDL as FDT because SDL is more used within industry and standardization bodies than any other standardized FDT [12].

2. The ideas of the method

In the following the ideas of our approach are illustrated by means of an example which is taken from the behaviour of the Inres protocol [13].

2.1. Structure and behaviour of the Inres protocol

In the sequel the Inres protocol is briefly introduced. The architecture of the Inres protocol is shown in Figure 1. The Inres protocol renders a *connection-oriented service* for data transmission. It uses a *connectionless service*. Data are transported from an *Initiator* entity to a *Responder* entity. The used service is called *Medium*. Messages exchanged between Initiator, Initiator User, Responder, Responder User and Medium are called service primitives (SPs) and the information units exchanged between Initiator and Responder are called protocol data units (PDUs).

The Inres protocol works in three phases: *connection establishment*, *data transfer* and *disconnection* (cf. Figure 2). For a connection establishment the Initiator gets a connection request CONreq from its user, then sends a CR to the Responder and waits for a connection confirmation CC in return. After receiving CC the Initiator gives a CONconf to its user and the connection is established. If a CC does not arrive within some time limit, the Initiator will retransmit CR for three times. Afterwards, the Initiator indicates the failed connection establishment by a DISind. When the Responder receives a CR from the Initiator it gives a connection indication CONind to its user and waits for a response CONresp in return. Upon arrival of CONresp, the Responder sends a CC to the Initiator and waits for a first data package DT.

After connection establishment data can be transferred. The Initiator User gives a data request DATreq to the Initiator, which then sends a DT to the Responder and then waits for an acknowledgement AK. If the AK does not arrive within some time limit the

[2]It should be noted that some test purposes (e.g. time constraints, or reliability requirements) can not be expressed by MSCs. But the use of MSCs for describing the class of test purposes which can be expressed seems to be common industrial practice. Therefore, we concentrate on MSCs.

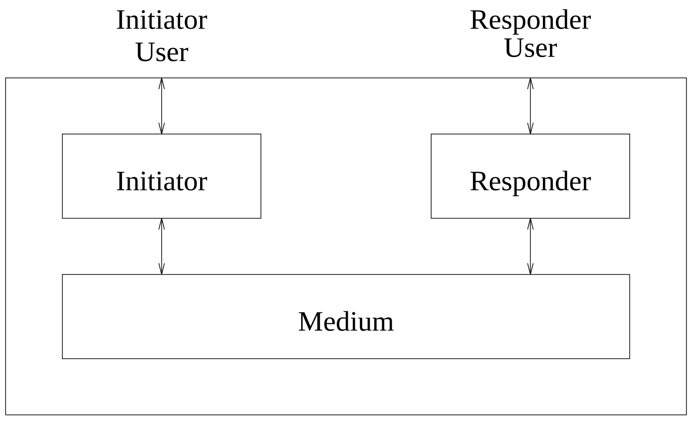

Figure 1: Architecture of the Inres protocol

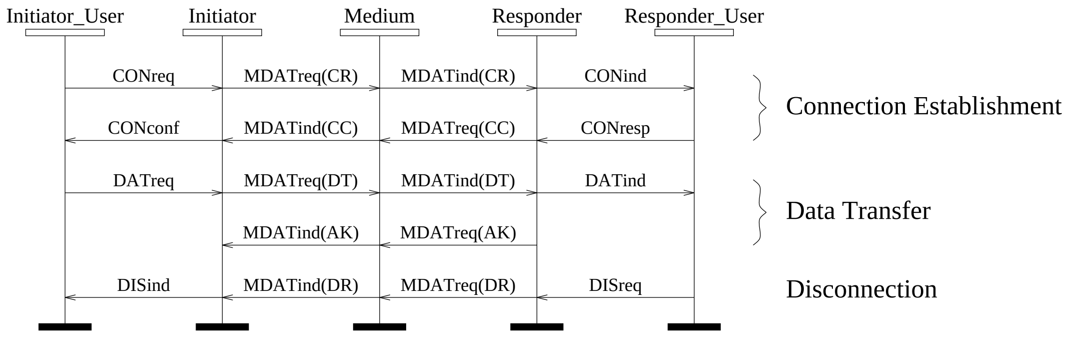

Figure 2: Complete system run of the Inres protocol

Initiator retransmits the DT for three times. Afterwards, the Initiator assumes that the connection is distroyed and indicates this by giving a DISind to its user. If the AK arrives in time, the next data package, if present, is sent. When the Responder gets a DT form the Initiator, it acknowledges the DT with an AK and gives a data indication DATind to its user. Afterwards the Responder waits for the next DT.

A disconnection can be initiated by a DISreq from the Responder User. Upon arrival of a DISreq the Responder sends a DR to the Initiator which then indicates the disconnection by an DISind to its user.

Initiator and Responder have to use the Medium service for their communication. The Medium service can be accessed by a data request MDATreq for transmission and by a data indication MDATind for reception. The PDUs CR, CC, AK, DT and DR can be considered as being parameters of MDATreq and MDATind. The MSC in Figure 2 shows a complete system run including connection establishment, data transfer and disconnection.

2.2. Testing the retransmission of the Initiator

A suitable test architecture for testing the Initiator entity of the Inres protocol might be the distributed test method [15] as sketched in Figure 3. The architecture of the Inres protocol (cf. Figure 1) can be adjusted to the distributed test method. The Responder is replaced by the lower tester (LT) and the upper tester (UT)[3] plays the role of the Initiator User. It is assumed that the test architecture is an SDL description which can be derived from the system specification. LT and UT are modeled as SDL processes which can send and receive any valid signal at any time[4]. A similar approach is used in [1]. The system under test (SUT) consists of an Initiator implementation which is the implementation under test (IUT) and a Medium implementation which is assumed to work correct.

We want to concentrate on testing a part of the retransmission property. In particular, we want to test whether it is possible to perform a correctly connection establishment after the third retransmission of the CR. The MSC in Figure 4 shows a scenario of which one may think about in the context of testing the retransmission property. The UT initiates a connection by CONreq. The LT waits for three CRs before it answers with CC which will then in return result in CONconf at the UT. Since the MSC in Figure 4 does not claim to define the entire scenario, it cannot be assumed that MSCs provides complete test information.

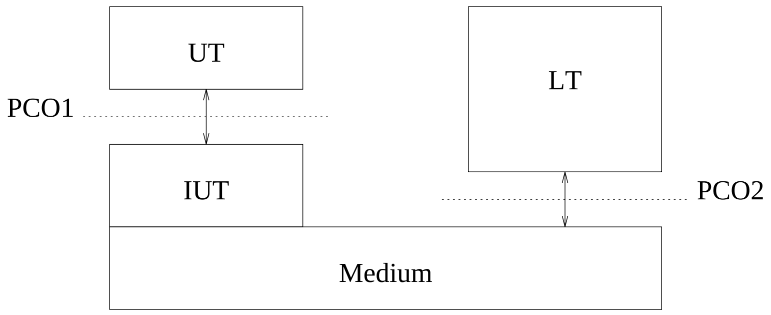

Figure 3: Distributed test method

2.3. The meaning and the representation of test cases

Our approach is based on the assumption, that an MSC defines a specific part of a test case, the so-called *test purpose*. For explaining this, the meaning of the terms *trace*, *observable* and *test case* have to be introduced, and the representation of test cases has to be described.

Traces and observables. A *trace* describes the ordering of events which are performed during a system run. A trace of an SDL description may include the events *tasks*, *inputs*, *outputs*, *decisions*, etc. of its processes. An MSC is a possible representation of an SDL trace. For testing, only inputs and outputs of LT and UT are interesting[5]. Therefore, we call a trace which only includes inputs and outputs of LT and UT an *observable*.

[3]UT and LT communicate via so-called points of control and observation (PCOs) with the IUT. For simplification the PCOs are not mentioned within the test case descriptions (e.g. Figure 4 and 5), but it is assumed that each tester serves its own PCO.

[4]For systems with a synchronous communication mechanism exists a simpler approach to define the behaviour of the tester: the inputs and outputs of the system which can be observed by its environment are inverted. Inputs become outputs and vice versa. Brinksma [3] uses this technique to define the *canonical tester*.

[5]SUT, LT and UT in general exchange SPs.

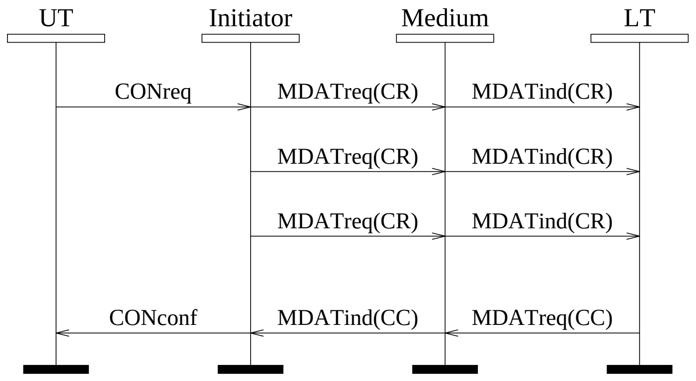

Figure 4: Connection establishment after the third retransmission of CR

An informal definition of test cases. A test case is defined in order to prove a specific *test purpose.* A test purpose might be a set of events which have to be performed, or a set of states which have to be reached by the IUT. A test case describes a set of observables. Each observable leads to a test verdict.

The test verdicts are PASS, INCONCLUSIVE and FAIL. PASS is given when the test purpose is reached, FAIL is assigned when the SUT behaves in an incorrect way and INCONCLUSIVE is given if neither FAIL nor PASS can be assigned.

A test case can be structured into three parts which are called *preamble*, *testbody* and *postamble.* The *testbody* describes observables which indicate that the IUT behaves according to the test purpose. The *preamble* drives the IUT from an initial state into a state from which the *testbody* can be performed. The *postamble* checks whether the testbody ends up in the correct state after it has been performed and drives the IUT back into an initial state from which the next test case can be applied.

The representation of test cases. Test cases for conformance tests are usually represented by the *Tree and Tabular Combined Notation* (TTCN) which is standardized by the ISO/IEC [16]. A TTCN test case for an Initiator implementation of the Inres protocol may look like the table in Figure 5. TTCN describes observables by means of a tree notation (cf. *Behaviour Description* in Figure 5). The tree structure is determined by the ordering and the indentation of the events. In general, the same indentation denotes a branching (e.g. lines Nr. 2 and Nr. 15 in Figure 5) and the next larger indentation denotes a succeeding event (e.g. lines Nr. 1 and Nr. 2 in Figure 5).

Events are characterized by the involved instance (i.e. LT or UT), by its kind (i.e. "!" denotes an output, "?" describes an input) and by the SP which has to be send or received. An example may clarify the notation. The statement *UT!CONreq* describes the sending of CONreq to the SUT by the UT. TTCN allows to specify events with arbitrary SPs by using the OTHERWISE statement (e.g. UT?OTHERWISE in Figure 6).

Test verdicts are defined within a verdict column of the TTCN table. The verdict column of Figure 5 only includes PASS and INCONCLUSIVE verdicts. In this example *FAIL* behaviour is specified by a *default behaviour description* which is shown in Figure 6. Such defaults have to be referenced in the test case header (cf. *Default* in Figure 5). TTCN offers much more facilities like *Constraints*, *Labels*, or *Timer* which are not relevant for the understanding of this paper. A tutorial on TTCN can be found in [17].

Test Case Dynamic Behaviour					
Test Case Name :	Test_Case_1				
Group :	Inres_Protocol/Initiator_Test/Connection_Establishment				
Purpose :	Connection Establishment after the third retransmission of a Connection Request				
Default :	Unexpected_Events				
Comments :					
Nr.	**Label**	**Behaviour Desription**	**Constraint Ref.**	**Verdict**	**Comments**
1		UT!CONreq			
2		LT?MDATind(CR)			
3		LT?MDATind(CR)			
4		LT?MDATind(CR)			
5		LT!MDATreq(CC)			
6		UT?CONconf			
7		LT!MDATreq(DR)			
8		UT?DISind		(PASS)	
9		LT?MDATreq(CR)		INCONC	
10		LT?MDATind(CR)		INCONC	
11		LT?MDATind(CR)		INCONC	
12		UT?DISind		INCONC	
13		UT?DISind		INCONC	
14		UT?DISind		INCONC	
15		UT?DISind		INCONC	
Detailed Comments :					

Figure 5: TTCN test case for the Inres protocol

Default Dynamic Behaviour					
Test Step Name :	Unexpected Events				
Group :	Inres_Protocol/Initiator_Test/Connection_Establishment				
Objective :	Handle unexpected Signals				
Comments :					
Nr.	**Label**	**Behaviour Desription**	**Constraint Ref.**	**Verdict**	**Comments**
1		UT?OTHERWISE		FAIL	
2		LT?OTHERWISE		FAIL	
Detailed Comments :					

Figure 6: Default behaviour for the TTCN test case in Figure 5

The role of MSCs and FDT descriptions for test case generation. It is assumed that an MSC defines the *test purpose* of a test case. This means that an MSC defines a signal exchange which has to be performed by the SUT to get a PASS[6]. An MSC does not describe the *pre-* and the *postamble* of the test case, responses of the SUT which lead to a FAIL or an INCONCLUSIVE, and the parameter values of the signals which are exchanged. In order to generate complete test cases the missing information has to be added. Therefore, an additional FDT description of the test architecture is necessary.

[6]From a theoretical point of view an MSC can be interpreted as a *liveness property* of the FDT description. It must be observable within a system run which leads from an initial state back to an initial state of the FDT description.

2.4. The observables of a test case

A test case consists of a set of observables. According to the test verdicts we distinguish between observables which lead to a PASS, observables which lead to an INCONCLUSIVE and observables which lead to a FAIL.

Possible pass observables. For generating a test case an observable has to be found which drives the SUT from an initial state back to an initial state, whereby the signal exchange defined within the MSC has to be performed without interrupts. We call an observable which fulfils these criteria a *possible pass observable*[7].

The observables which drive the SUT from an initial state to a state from which the MSC is applicable can be interpreted as the *preamble* of the test case and the observables which drive the SUT back into an initial state after the MSC has been applied can be interpreted as the *postamble*.

We explain this by means of our test case example. The connection establishment of the Inres protocol starts in an initial state. Therefore, no preamble has to be added and our test case starts with the observable defined by the MSC in Figure 4. The MSC ends in a state where the connection is established and data can be transferred. A possible postamble is a normal disconnection which starts with the sending of MDATreq(DR) by the LT and ends with the reception of a DISind by the UT. The MSC in Figure 7 shows the MSC in Figure 4 enhanced by the disconnection. The TTCN description in Figure 5 describes the observable which is defined by the MSC in Figure 7 within the lines Nr. 1 to Nr. 8. These lines also describe the *possible pass observable* of this example. The sketched postamble is specified within the lines Nr. 7 and Nr. 8.

Inconclusive observables. If a *possible pass observable* is found, observables which lead to an INCONCLUSIVE have to be generated. We call them *inconclusive observables.* An *inconclusive observable* has the same prefix as a *possible pass observable* but its last event is a response of the SUT which leads neither to a PASS nor to a FAIL. In our example interrupts of the connection establishment by DISind lead to an INCONCLUSIVE. Within Figure 5 these cases are shown in the lines Nr. 9 to Nr. 15.

Fail observables. *Fail observables* are added to the TTCN test case description by means of the OTHERWISE event and a default behaviour description (Figure 6).

Possible and unique pass observables. The *possible pass observable* of the TTCN test case in Figure 5 is shown in the lines Nr. 1 to Nr. 8. But this observable does not ensure that the MSC in Figure 7 has been performed during a test run. After the reception of DISind a test verdict is assigned and the test case is finished. But according to the SDL description of the Inres protocol a fourth MDATind(CR) may be on the way. In this case the MSC in Figure 8 would be performed.

Such problems arise because the SUT is treated as a black box and therefore, LT and UT only have an incomplete system view. For the tester the SUT behaves in an indeterministic way. In our example the indeterminism is caused by the asynchronous communication

[7]In general there can exist more than one *possible pass observable* for a test case.

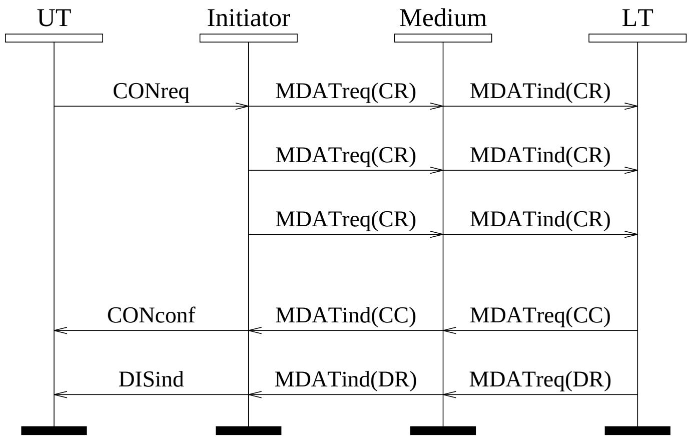

Figure 7: MSC of Figure 4 with a possible postamble

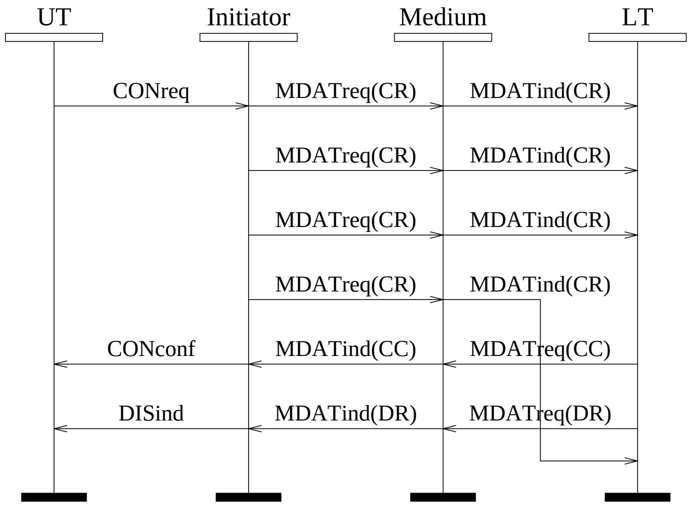

Figure 8: MSC describing a not expected system run

mechanism of SDL. Without seeing all input and output events of Initiator and Medium, we cannot make any assumption about an ordering, or a time relation between the DISind and a possible fourth MDATind(CR). However, if a fourth MDATind(CR) arrives, the *PASS* in Figure 5 has to be overwritten by an INCONCLUSIVE.

The LT does not know how long it should wait for a fourth MDATind(CR) after the reception of the DISind by the UT and before the assignment of a PASS. Therefore, a test run according to Figure 7 cannot be distinguished from test runs according to Figure 8 in all cases. A new postamble has to be found.

A correct postamble of our example is shown within Figure 9. Instead of MDATreq(CR), a data package DATreq[8] is transferred, but the reception by the LT is not acknowledged. The Initiator retransmits the data package DT three times, indicates afterwards the disconnection and goes back into a *disconnected* state. The FIFO property of queues and channels in SDL ensures that after the reception of the first MDATind(DT) no fourth

[8]Data is transported as a parameter of DATreq. Since this parameter does not influence the behaviour of the Inres protocol it is omitted.

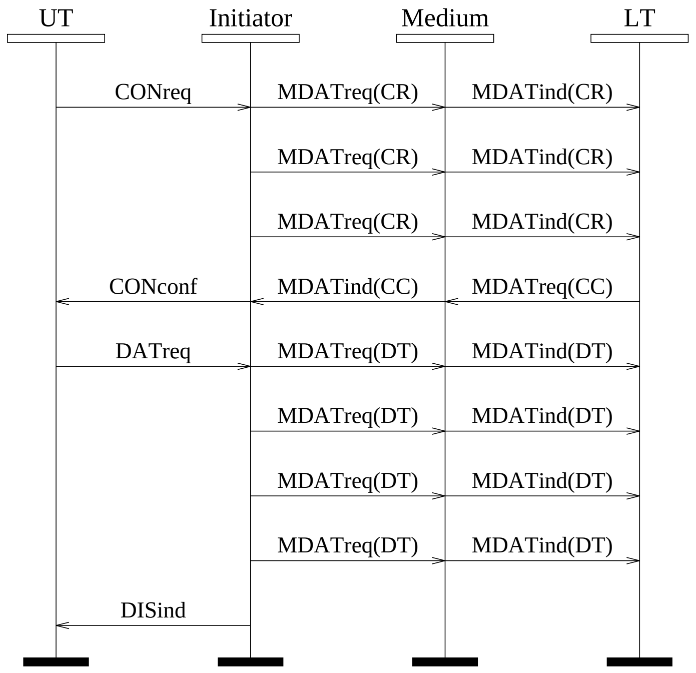

Figure 9: MSC of Figure 4 with a correct postamble

MDATind(CR) can be received. Thus, the reception of the DISind allows a unique assignment of a PASS.

We call a *possible pass observable* which uniquely ensures that the given MSC was performed a *unique pass observable*. The complete and correct TTCN test case which ensures that the test purpose given in Figure 4 was performed is shown in Figure 10. The *unique pass observable* of this example is described in the lines Nr. 1 to Nr. 12.

3. On the implementation of the presented method

Within the previous chapter our approach is presented on an intuitive level by means of an example. As a summary one can say that the approach is based on the calculation of four sets of observables: *possible pass*, *unique pass*, *inconclusive* and *fail observables*. In this chapter we explain how the observables can be calculated, how this is reflected in a tool architecture and how our method can be extended for using more than one MSC as test purpose.

3.1. The computation of the observables of a test case

The observables of a test case are generated in four steps. In a first step the *possible pass observables* of the test case are computed. In a second step we check for each *possible pass observable* whether it is a *unique pass observable* or not. Since we only need one *unique pass observable* to ensure the MSC test purpose, we select one of the shortest. For the chosen *unique pass observable* the corresponding *inconclusive observables* are generated within a third and the *fail observables* are defined within a fourth step.

The computation of possible pass observables. The computation of *possible pass observables* is a typical search problem. We have to find SDL traces which include the

Test Case Dynamic Behaviour

Test Case Name :	Test_Case_3
Group :	Inres_Protocol/Initiator_Test/Connection_Establishment
Purpose :	Connection Establishment
Default :	Unexpected Events
Comments :	

Nr.	Label	Behaviour Desription	Constraint Ref.	Verdict	Comments
1		UT!CONreq			
2		LT?MDATind(CR)			
3		LT?MDATind(CR)			
4		LT?MDATind(CR)			
5		LT!MDATreq(CC)			
6		UT?CONconf			
7		UT!DATreq			
8		LT?MDATind(DT)			
9		LT?MDATind(DT)			
10		LT?MDATind(DT)			
11		LT?MDATind(DT)			
12		UT?DISind		PASS	
13		LT?MDATind(CR)		INCONC	
14		LT?MDATind(CR)		INCONC	
15		UT?DISind		INCONC	
16		UT?DISind		INCONC	
17		UT?DISind		INCONC	
18		UT?DISind		INCONC	

Detailed Comments :

Figure 10: TTCN test case description which ensures the test purpose of Figure 4

events specified by the MSC and which lead the SDL system from its initial state back to its initial state. From such traces the *possible pass observables* are extracted. Unfortunately, we cannot ensure that we find *possible pass observables*, because this problem is equivalent to the reachability problem of turing machines [2] which is not decidable.

We search the required observables by simulating the SDL description and the MSC in parallel. There exist several search methods like depth and breadth search. Breadth search can not applied because it is impossible to store all possible states of the SDL system[9]. Depth search also is not usable since we can not guarantee termination. As a consequence we use a k-bounded depth search which evaluates all possible traces of length k. If no trace with required properties is found, the search can be repeated with a higher bound or stopped without results.

The computation of unique pass observables. For each *possible pass observable* we have to check if it is a *unique pass observable*. In general there can exist none or a whole set of *unique pass observables* for an MSC. For proving a test purpose defined by an MSC we only need one. We choose one of the shortest *unique pass observables* to be the *unique pass observable* of the generated test case.

[9]A state of an SDL system includes the control states of the processes, the contents of the queues and the values of the variables.

The computation of inconclusive observables. For the chosen *unique pass observable* the corresponding *inconclusive observables* have to be generated. Therefore, the SDL description is simulated according to the *pass observable.* The *inconclusive observables* are ending in a response of the SUT from which one can conclude that the required *unique pass observable* is not performed.

Fail observables. *Fail observables* are added by means of the TTCN constructs OTHERWISE and default behaviour. Therefore, they need not to be calculated.

3.2. On the optimization of the test case generation

The problem of generating test cases from SDL descriptions and MSCs is a search problem. The expense of the search heavily depends on the SDL description which represents the test architecture. The test architecture is derived from a system specification by omitting not tested parts and by adding the SDL processes for LT and UT. An open problem of the project is the optimal modelling of the test architecture, especially of LT and UT, since this may decrease the search expense. Presently, our test processes behave like the reasonable environment in [11]. A further optimization may include more restrictions on the sequences of signals which can be sent and received, and restrictions on the values of signal parameter.

3.3. The test case generation tool

Figure 11 presents the architecture of a tool which is developed at the University of Berne and which implements the presented approach. The tool is structured in the three parts *SDL simulator*, *MSC simulator* and *test case generator.* Both simulators consist of a *transformator* and an *interpreter.* The transformators read descriptions in phrase representation of SDL (SDL/PR) and MSC (MSC/PR) and transform them into internal representations. Afterwards the internal representations are simulated by the interpreters. The test case generator is structured in four modules:

- Calculation of *possible Pass observables.*
- Calculation of *unique Pass observables.*
- Calculation of *Inconclusive observables.*
- Generation of the corresponding TTCN/MP[10] code[11].

The tool is implemented on Sun workstations. Its inputs are MSC/PR and SDL/PR descriptions [5, 4], and its output is a TTCN/MP description [16]. Front- and backends of the tool are commercial SDL, MSC and TTCN editors.

3.4. Using more than one MSC as test purpose

In general, it is possible to use more than one MSC as test purpose. Therefore, the relation between the MSCs has to be specified. One can think about test purposes specified by a set of MSCs which are combined by arbitrary AND and OR relations. An AND relation of two MSCs means that both MSCs have to be performed to reach the specified test

[10]TTCN/MP denotes the machine processable form of TTCN.

[11]TTCN/MP has a standardized ASCII syntax. During test case generation it is not very efficient to use ASCII files as internal computer representation. As a consequence we have to translate our internal representation into TTCN/MP.

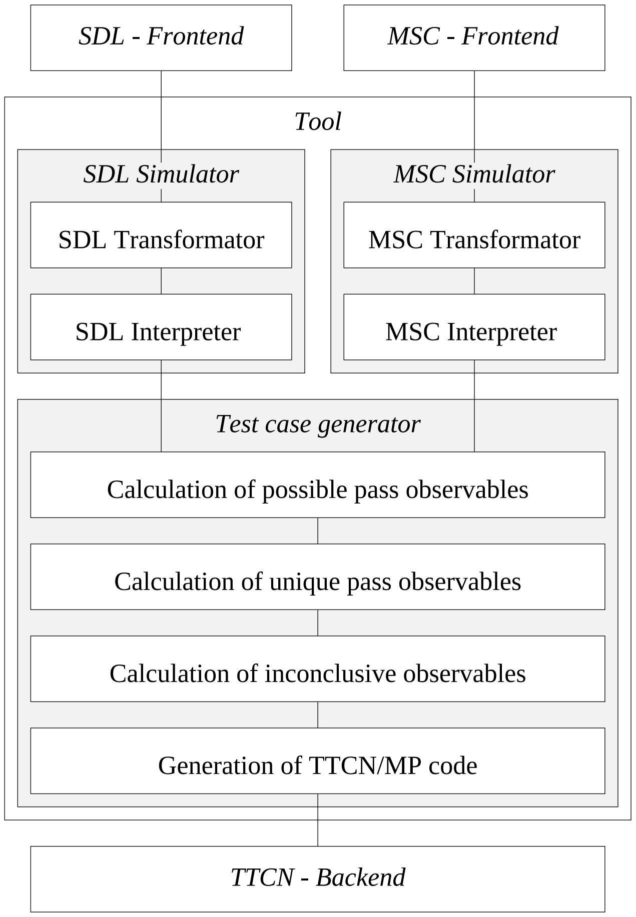

Figure 11: The tool architecture

purpose. An OR relation of two MSCs means that one of the two has to be performed to reach the test purpose. AND and OR can be realized by generating the observables for the involved MSCs and by implementing the operations on the observables.

4. Summary

A method for the generation of test cases based on SDL descriptions and MSCs is presented. The approach assumes that the purpose of a test case is given by one or more MSCs. Furthermore, the problem of assigning unique test verdicts is discussed and a solution by defining *unique pass observables* is presented. The method is implemented and its applicability for real systems will be proven by a following case study. More detailled information can be found in [7, 10, 18, 19].

References

[1] Bourget-Rouger, A.; Combes, P.: Exhaustive validation and Test Generation in Elvis, in SDL'89: The language at work - O. Faergemand and M.M. Marques (editors), North-Holland, 1989.

[2] Brand, D.; Zafiropulo, P.: On Communicating Finite State Machines, in Journal of the Association for Computing Machinery, April 1983.

[3] Brinksma, E.: On the Existence of Canonical Tests, Technical Report INF-87-5, University of Twente, Netherlands, 1987.

[4] CCITT Recommendation Z.100: Specification and Description Language (SDL), Geneva, 1992.

[5] CCITT Recommendation Z.120: Message Sequence Chart (MSC), Geneva, 1992.

[6] Chow, T.S.: Testing Software Design Modeled by Finite State Machines, IEEE-SE, 4(3):178–187, 1978.

[7] Grabowski, J.; Hogrefe, D.; Nahm, R.: Conformance Testing - ein Werkzeug zur Generierung von Testfällen, Ergänzung zum Zwischenbericht des F & E Projektes Kontraktnummer 233, finanziert durch die Schweizer PTT, 1992.

[8] Grabowski, J.; Rudolph, E.; Message Sequence Chart (MSC) - A Survey of the new CCITT Language for the Description of Traces within Communicating Systems, in Proceedings of the 2nd GI/ITG Workshop on Formal Description Techniques for Distributed Systems in Magdeburg (Germany), 1993.

[9] Grabowski, J.; Graubmann, P.; Rudolph, E.: The Standardization of Message Sequence Charts, Proceedings of the IEEE Software Engineering Standards Symposium 1993.

[10] Grabowski, J.; Hogrefe, D.; Ladkin, P.; Leue, S.; Nahm, R.: Conformance Testing - A Tool for the Generation of Test Cases, Interim Report of the F & E project contract no. 233, funded by Swiss PTT, Berne 1992.

[11] Hogrefe, D.: Automatic Generation of Test Cases from SDL-Specifications, in SDL-Newsletters No. 12, 1988.

[12] Hogrefe, D.: Conformance Testing of Communication Protocols in the Framework of Formal Description Techniques, Technical Report IAM-91-007, University of Berne, 1991.

[13] Hogrefe, D.: OSI Formal Specification Case Study: The INRES Protocol and Service, Technical Report IAM-91-012, University of Berne, 1991.

[14] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, Addison Wesley, 1979.

[15] ISO/IEC JTC 1/SC 21: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 1-5, IS 9646, 1991.

[16] ISO/IEC JTC 1/SC 21: Information technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation, IS 9646-3, 1991.

[17] Kroon, J.; Wiles, A.: A Tutorial on TTCN, in Protocol, Specification, Testing and Verification, volume 11, North-Holland, 1991.

[18] Nahm, R.: Semantics of Communicating Finite State Machines - Based on Graph Representation and Automata Interpretation, Technical Report, University of Berne, 1993.

[19] Nahm, R.: Semantics of Simple SDL, in Proceedings of the 2nd GI/ITG Workshop on Formal Description Techniques for Distributed Systems in Magdeburg (Germany), 1993.

[20] Wezeman, C.D.: Protocol Conformance Testing Using Multiple UIO-Sequences, in Protocol Specification, Testing and Verification, volume 9, North-Holland, 1990.