Improving the Quality of Test Suites for Conformance Tests by Using Message Sequence Charts

Jens Grabowski[a], Dieter Hogrefe[a], Iwan Nussbaumer[b], and Andreas Spichiger[a]

Abstract

The test of a communication system is a complex procedure which comprises several phases with various tasks. The quality of a test depends on the work in the phases and an easy and smooth transition between the phases. In the current test practice the different phases are mainly based on informal documents and they do not always fit together properly. Therefore we intend to improve the phases and the transition between the phases. This paper shows how the use of Message Sequence Charts (MSCs) may improve the test specification phase and facilitate the transition to the test implementation phase. We describe the different phases of testing, sketch the problems of practical testing, and explain the use of MSCs for test specification by the example of an ISDN switching system. We show how MSC based test specifications can be transformed into executable test cases, and present a tool set which supports the test specification and automates the test implementation.

[a] *Universität Bern, Institut für Informatik, Länggassstrasse 51, CH-3012 Bern, ph. +41 31 631 86 81, fax. +41 31 631 39 65*

[b] *Siemens-Albis AG, Öffentliche Vermittlungssysteme, Steinenschanze 2, CH-4051 Basel, ph. +41 61 276 71 11, fax. +41 61 276 76 71*

1 Introduction

Testing is performed to protect users and customers against insecure, inappropriate, or even erroneous soft- and hardware products. Furthermore, a thorough and comprehensive test gives an indication about the quality of a product. In the telecommunication area special tests, so-called *conformance tests*, are often demanded by the customers (mainly national PTTs). A telecommunication system is a distributed system and a soft- or hardware product may become a component of such a system. A conformance test should ensure the required functions of a component to interwork with other system components. These functions are defined within standards or recommendations provided by international standardization organizations (e.g. ITU-T[1], ISO/IEC, or ETSI) and by the customer which may require additional country specific functions.

A typical test environment of our application area is shown in Figure 1. We want to test the functions of the *layer 3 protocol Q.931* [CCI89] within a *Line Trunk Group* (LTG) of an ISDN[2] switching system. The Q.931 protocol is implemented within the LTG and there is no direct access to this implementation. Furthermore, each LTG has only one standardized interface which may be connected to an ISDN end system (e.g. a telephone). The interface of an LTG to the main processor is proprietary and not standardized. It is therefore not adequate for testing the conformance to standards. Consequently the test devices are connected directly to the standardized interfaces. The devices are controlled by a test manager which also records the test results.

[1] Until March 1993 the ITU Telecommunication Standards Sector (ITU-T) was called Comité Consultatif International Télégraphique et Téléphonique (CCITT).

[2] ISDN is an abbreviation for *'Integrated Services Digital Networks'*.

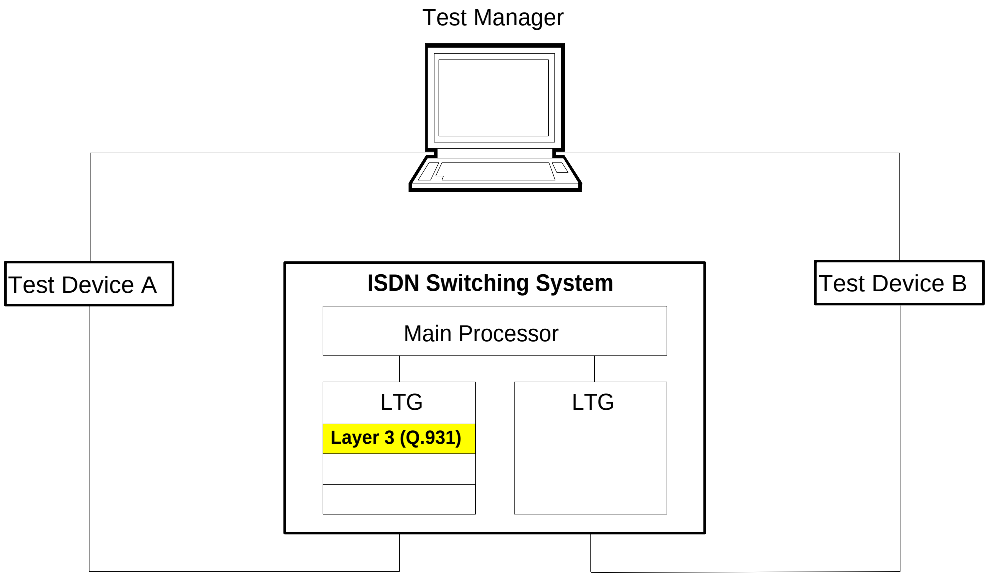


Figure 1: A test environment of an ISDN switching system

The analysis of the test results only proves the quality of a protocol implementation if the test itself has a high quality. Additionally, only tests of high quality may lead to an improvement of the protocol implementation itself. In the next section we will therefore explain the possibilities to improve the quality of conformance tests.

2 Testing in practice

Like the development of soft- and hardware systems, the procedure of testing can be divided into several phases. As indicated in Figure 2 the whole test procedure can be seen as a puzzle game. Each phase is a piece of this puzzle and should match with the other pieces.

2.1 The phases of testing

The test process can be divided into the five phases *analysis*, *specification*, *implementation*, *execution*, and *evaluation* (cf. Figure 2). A phase performs a certain task and delivers a document with the phase results as input to the next phase.

The objective of the *analysis phase* is to identify the test cases which are necessary to check the relevant requirements of the system. The purpose of each test case is described by an informal statement, the so-called *test purpose*. The result of this phase is a list of test cases together with the corresponding test purposes.

In the *specification phase* for each identified test case it is fixed how it must be realized. In general a test case consists of series or sequences of stimuli and foreseen responses. Stimuli and responses are called *test events*. They have to be executed, i.e. sent and observed, by the test devices (cf. Figure 1). However, the test specification phase concerns not only the test events related directly to the test purpose, but also the start states of system components necessary to interface the implementation which shall be tested, information of how these states can be reached, and other presuppositions which are required for achieving the test purpose. At the end of this phase a document with the exact specification of each test case is available.

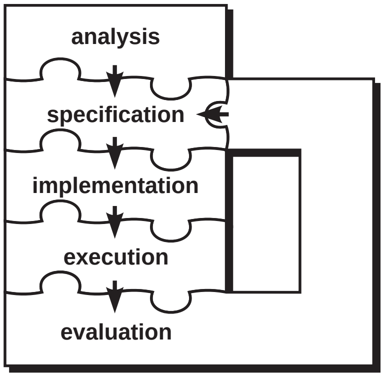


Figure 2: The ideal phase model of the test process

In the *implementation phase* the test case specifications are transformed into executable test programs, so-called *executable test cases*. Especially, the sequence of test events must be enriched with data descriptions, which have not been defined in the specification phase but which are necessary to execute the test case. Generally, test cases are implemented in a high level language, e.g. C, Pascal, Forth, or TTCN.

In the *execution phase* the executable test cases are run on a test system. A test verdict (pass, fail, or inconclusive) is assigned to each test run. The test runs, i.e. test verdicts and test events, are recorded in a *test log*.

In the *evaluation phase* the test verdicts from the execution phase are analyzed. In the case of a *pass* verdict the test case has been performed successfully, i.e. the test purpose is reached. In the case of a *fail* or *inconclusive* verdict it must be analyzed whether the test case or the tested implementation includes errors. The results of this phase are three lists of test cases. One list contains the test cases which have met the test purpose. All correctly implemented test cases with inconclusive and fail verdicts are put in the second list. This list is given to the product implementor for his own revision process. The third list includes all test cases, which have been realized incorrectly. These test cases must be revised and the list is therefore fed back to the specification phase.

2.2 Problems in implementing the test phase model

Usually, the phases of the test procedure do not accord with each other and a lot of resources, both manpower and time, are wasted for adapting the results of one phase to the requirements of the next one and for maintaining redundant information. In the field we find an implementation of the procedure described in the previous section with unharmonious phase transitions. Figure 3 indicates this situation, by showing a puzzle game with pieces which do not match.

In the analysis phase an informal text document is written. This document is the basis for the specification phase. The transition from the analysis phase to the specification phase is the simplest transition of all, because both phases use text based documents.

For test case specification, informal diagrams as shown at the bottom of Figure 4 are widely used and well accepted in the telecommunication area [GGR93]. They describe the message flow at the interfaces of the system which shall be tested. Often they are enriched with informal presuppositions. Figure 4 provides a complete example of such a test case description. The test case EDSAOUX is taken from the Layer 3 test suite for the LTG. It will be used throughout the rest of this article as an example. In the

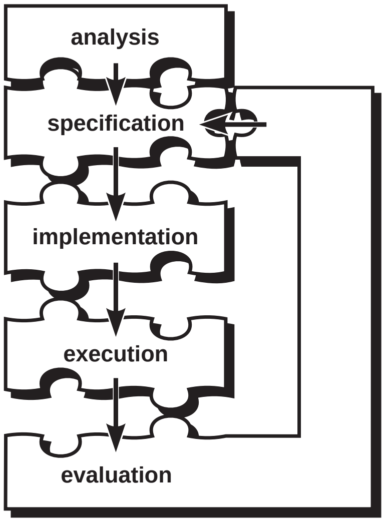


Figure 3: The phase model in practice

figure *Subscriber A* and *Subscriber B* indicate the behavior at the standardized interfaces. During the conformance test the roles of the subscribers are played by testers. A tester might be a software process, a hardware device, or a combination of both.

Informal test case specifications as shown in Figure 4 can not be used directly in the implementation phase. The informal information has to be formalized and the missing information has to be added. This means that test cases have to be implemented by hand. Usually, the implementation of test cases is rather error prone and requires a lot of resources. For example, the test suite for testing only the country specific functions for Switzerland of the Layer 3 protocol Q.931 within an LTG (cf. Section 1) includes more than 1700 test cases. Each of them is implemented by hand.

In the execution phase the executable test cases can be taken from the implementation phase as they are. Unfortunately, a test system as shown in Figure 1 is itself a distributed system. Therefore often the processes which control the devices at the different interfaces of one test case cannot be implemented in one executable. In such a case the coordination between the different processes must be made manually.

Also the evaluation of the test results has to be performed by hand.

2.3 Actions to improve the quality of testing

To improve the quality of the test cases and to reduce the time necessary to pass the test phases it is essential to have a good implementation of the test procedure. However, the previous section shows the incompatible phase transitions in the current practice. These transitions require a lot of manual actions, redundant steps have to be passed and all of them are the source for a lot of errors. In particular the implementation phase is very error susceptible. It demands a huge manual effort to implement all the necessary information for a complete executable test case and a lot of already specified items have to be rewritten in the high level language. Subsequently, we will present a method, which aims to automate the implementation phase and gives adequate tool support for the specification phase.

Test case identifier: EDSAOUX

Test purpose: The test shall ensure that after connection establishment Subscriber A receives at least three Information messages. The display parameter within the Information message shall have the format 'Fr. x.x0' (0 ≤ x ≤9).

Test configuration: Subscriber-A-SWITCH-Subscriber-B

Pre-conditions:
- The system is in its initial state n(0).
- The tax parameter ABS is not set.
- The tax units are set for time rates of 0,3 Rp/s.

Control: Observation of the tax display

General message flow:

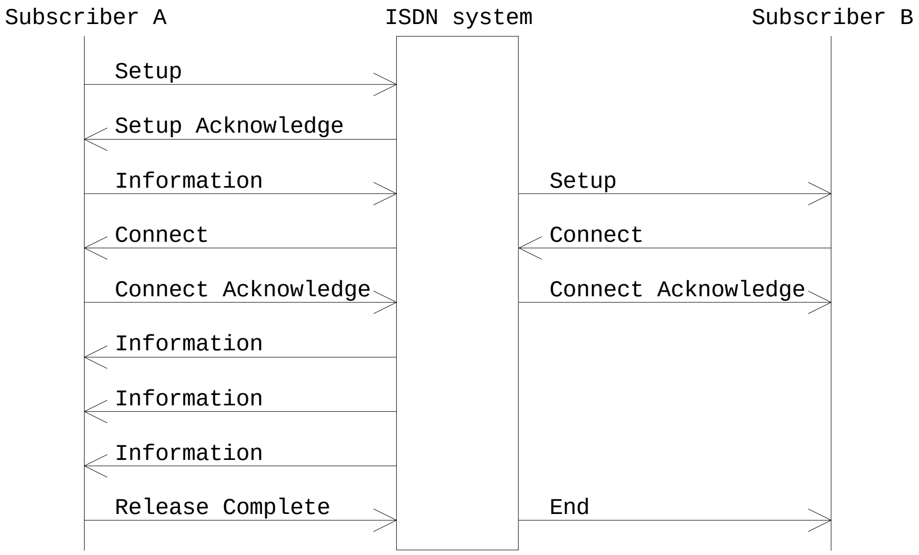


Figure 4: The specification of the test case EDSAOUX

3 Using MSC for test case specification

In our method we use the Message Sequence Chart (MSC) language for the specification of test cases. MSC has been standardized in [ITU92]. An MSC diagram is very similar to the notation already used for test case specification purposes. A comparison of the test specification in Figure 4 and the MSC[3] in Figure 6 shows the similarities and differences.

After the implementation phase test cases will be in TTCN (Tree and Tabular Combined Notation) which is a standardized notation for test cases [ISO92]. TTCN has become very popular in the telecommunications area. Although TTCN is intended to be used as a test case specification language, it has been recognized, that it can also be used as test case implementation language. As a consequence there already exist compilers for different test devices, e.g. [Sie93].

For complete test cases it is necessary to describe the parameter values of the exchanged messages. Since the standardized MSC language includes no possibility for data

[3] The term *MSC* is used for a diagram written in the MSC language and the language itself. Where necessary, we distinguish between both by using the terms *MSC language* and *MSC diagram*.

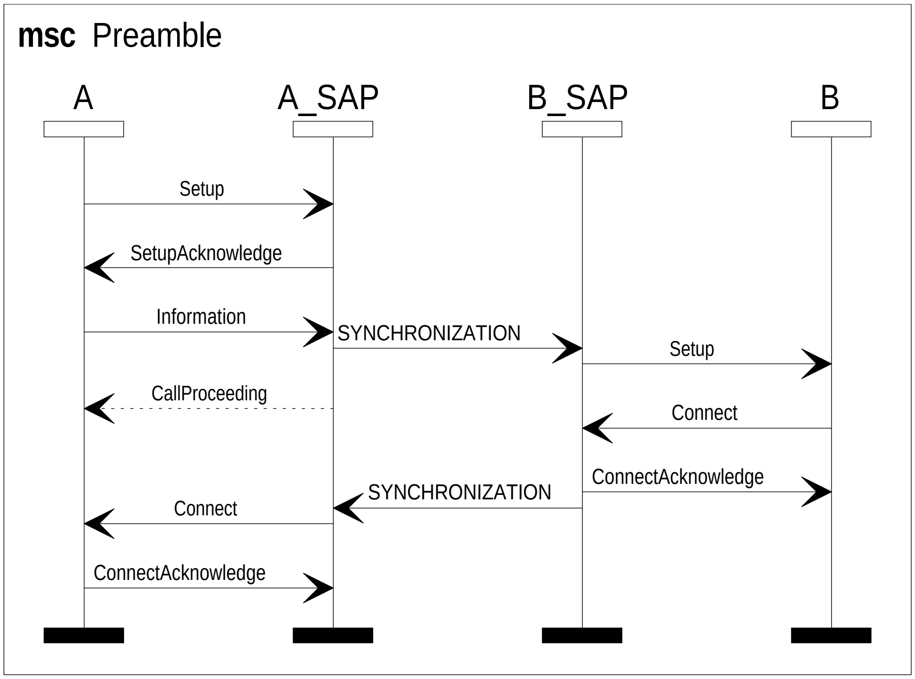


Figure 5: Preamble of the test case EDSAOUX

descriptions, we will explain how data can be introduced in MSC. We use a reference mechanism which relates TTCN data descriptions to the messages within an MSC. The reference to TTCN is optional. Other data description languages can be taken instead, using the same mechanism.

3.1 Adapting MSCs for the needs of testing

The MSC standard is provided by the ITU-T recommendation Z.120 [ITU92]. The MSC recommendation includes two syntactical forms: MSC/PR as pure textual and MSC/GR as graphical representation. An MSC in MSC/GR representation can be transformed automatically into a corresponding MSC/PR representation. We use the graphical form in the test case specification and base our algorithms on the MSC/PR form. Because of simplicity in this paper we only use the MSC/GR form.

Test cases describe sequences of test events which have to be performed by the tester. In the MSCs which form the test case EDSAOUX, the test events are those along the instances *A* and *B* (e.g. Figure 5). They represent *Subscriber A* and *Subscriber B* in Figure 4.

The automation of the test case implementation requires that the MSCs comprise all tester actions and further relevant information, e.g. information concerning the synchronization of the tester. Besides the sending and reception of messages, a tester may also supervise timer, or control the number of recurrences of a specific message.

The investigated examples show that the current MSC standard is in most cases sufficient for describing the message exchange of test cases. But, we also identified situations where additional language constructs might be helpful. Some of these constructs are shorthand notations, some are real extensions and some concern the combination of MSCs. In the following we introduce them briefly.

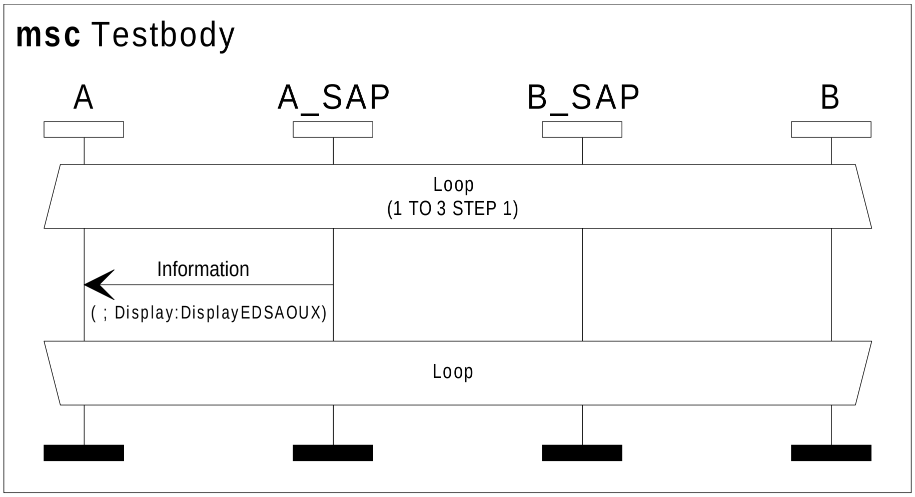


Figure 6: Test body of the test case EDSAOUX

3.1.1 Extensions of MSC

Optional messages. An *optional message* may occur in exactly one situation. *Call-Proceeding* in Figure 5 is an optional message which may occur immediately after the *Information* message sent by *A*. Whether the message occurs, depends on the configuration of the whole ISDN system.

Always messages. An *always message* is a message which always may occur from a certain point in time arbitrarily often. Within our tool the first use within the MSC defines the point from which it may occur.

Synchronization messages The MSC in Figure 5 includes messages which are inscribed with *SYNCHRONIZATION*. These *synchronization messages* are no real messages. They only express the order of send and receive actions on different instance axes. Synchronization messages may be used to synchronize the different testers.

Loops Several test cases require that a certain message or a specific part of a message exchange should occur repeatedly. The number of occurrences may be stated explicitly or determined by a time limit. Consequently, we introduced a *timer loop* and a *counter loop*. The graphical representation of both is the same. We use two trapeziums which enclose the recurrent message exchange. The termination criterion in the upper trapezium states whether the loop is controlled by a counter variable or a time limit. The MSC in Figure 6 describes the test purpose of the test case EDSAOUX by using a counter loop. The termination criterion *(1 TO 3 STEP 1)* states that the *Information* message within the loop should occur 3 times.

3.1.2 The combination of MSCs

The purpose of the test case EDSAOUX (cf. Figure 4) is to test the arrival of three *Information* messages[4]. The test case can be structured into a *preamble*, a *test body*, and a *postamble*. The *preamble* describes the message exchange from the initial state

[4] The test of the parameter values will be explained in Section 3.2.

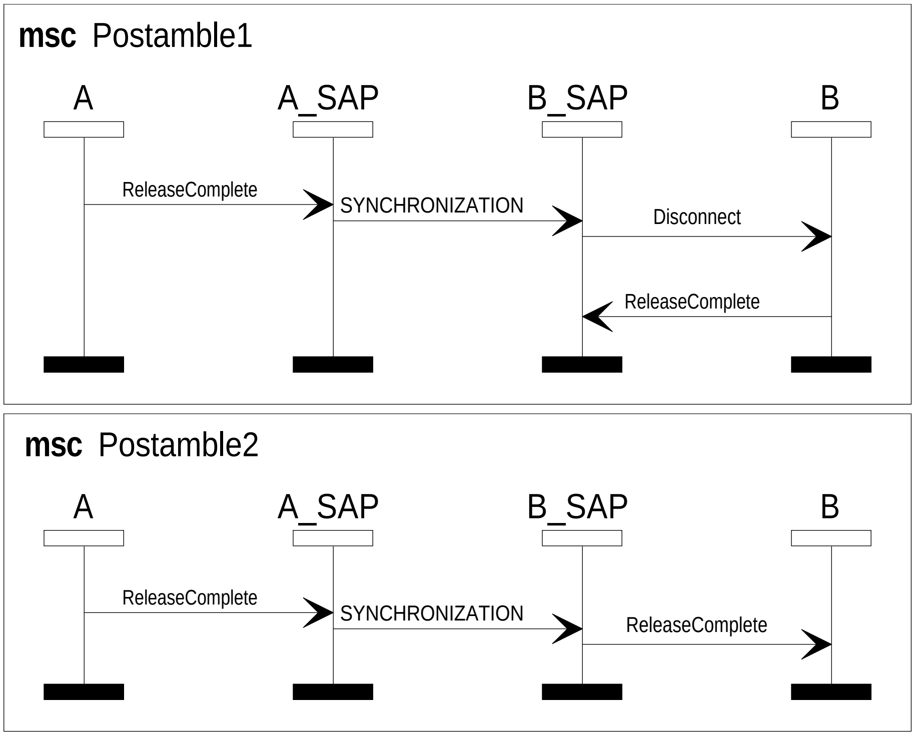


Figure 7: Alternative postambles of the test case EDSAOUX

(cf. *Pre-conditions* in Figure 4) into a state from which the *Information* messages are observable. This preamble is shown in Figure 5. The *test body* in Figure 6 comprises the observation of the three *Information* messages. The *postamble* includes the message exchange which is necessary to drive the tested system back into the initial state. For the test case EDSAOUX exist two alternative postambles (cf. Figure 7). The complete test case description should include both postambles.

The message flow of the test case EDSAOUX is described by the MSCs in the Figures 5, 6 and 7. Additionally to the MSCs we need a mechanism to specify how the MSCs should be combined. Such a description could be interpreted as a more general test case description since it abstracts from the message flow.

Figure 8 presents an example of the graphical notation we use. An arrow between two MSCs specifies a *sequence* of two MSCs. In Figure 8 the signal exchange of the MSC *Preamble* is followed by the MSC *Testbody*. A branching denotes *alternative* MSCs. Therefore the MSCs *Postamble1* and *Postamble2* in Figure 8 may happen alternatively. The *supernode* ellipsis only indicates the start of the test case description.

From our work we know that different test cases often check different aspects of the same, or at least of almost the same message flow. In such situations the test cases include identical parts, and it is advantageous to reuse parts of existing test case specifications. Due to the operators for combining MSCs structuring and therefore also reuse of MSCs is supported.

3.2 MSCs and data descriptions

In the previous section it is shown how test case specifications can be described by MSCs. In order to gain complete test cases the MSCs have to be related to data descriptions.

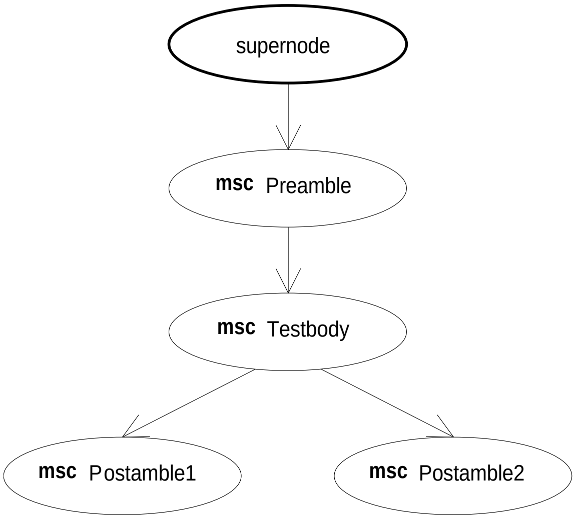


Figure 8: Combining the MSCs which form the test case EDSAOUX

3.2.1 Data descriptions in test cases

In test cases we are confronted with three kinds of data descriptions. These are: *data type definitions*, *default constraint definitions*, and *test case specific constraints*.

During the test the testers and the tested system exchange messages. The messages may have a specific format and parameters with values of a specific data type. The parameters my also be structured into further components. Roughly spoken the *data type definitions* specify the format and value range of all messages and message parameters used by the test cases. The data type definitions are valid for all test cases within a test suite and therefore they are defined globally. The test cases refer to them.

For most messages and message parameter values the protocol standard provides *default constraint definitions*. A default constraint is either a default value, or a restriction of the possible value range. Default constraints are valid for a whole test suite. Therefore they only need to be defined once. The test cases may use default constraints by referring to them, or test case specific constraints.

Test case specific constraints are constraints on messages or message parameter values which are adapted to the specific needs of a test case. They are important for two reasons. Sometimes, it is necessary to send specific message parameter values to drive the tested protocol into a state from which the test purpose can be checked, and test purposes often intend to check requirements on message parameter values. These requirements are expressed by test case specific constraints.

3.2.2 Data type and default constraint definitions

We assume that data type and default constraint definitions are given in form of TTCN data types and TTCN constraints. But this is only one possibility. Other data descriptions, e.g. ASN.1, can also be adapted to our method.

The relations between a message, the data type definitions, and the default constraints are defined implicitly by the message name. The message name refers to a type definition which itself includes, or refers to the type definitions of the message parameters. The message name also provides the relation between message and default constraint. We use name conventions to identify the correct default constraint of a message. But, it should be noted that default constraints are not available for all messages.

Although this implicit reference mechanism is trivial, it is necessary. TTCN test cases refer explicitly to default constraints and therefore we have to generate the references. We also like to mention that the implicit reference mechanism allows to focus on the test case specific requirements. The test case specifier shall not bother about default constraints.

3.2.3 Test case specific constraints

Test case specific constraints have to be defined explicitly when the test case is implemented. Their definition is based on the test purposes, the data type definitions, the relevant standards, and the additional requirements. Test case specific constraints have to be defined manually.

Since the definitions may become very complex, they cannot be included in an MSC. As a consequence we use TTCN constraint declarations for the constraints definition and refer to these definitions. For this purpose we developed a comfortable reference mechanism which

- allows to refer to self written test case specific constraints,
- provides possibilities to define test case specific constraints by modifying existing constraints, e.g. within a default constraint for a message several default constraints for parameter values can be replaced by test case specific constraints, and
- allows to define simple test case specific constraints within an MSC, e.g. if a test case specific constraint only comprises one concrete value.

Our reference mechanism is a reference language, in the following called RL, which can be used to specify the mentioned possibilities. Within an MSC the statements of RL are related to messages. They can be found in parentheses near the corresponding message name, or message arrow (cf. Figure 6). This is no extension of the MSC language, because the MSC standard [ITU92] proposes to use expressions in round brackets to assign parameter information to messages.

Based on an RL statement it is possible to automate the calculation of the constraint references which are necessary for the TTCN test cases, and to generate test case specific constraints which are based on existing constraints. The details of RL can be found in [Rüf94].

3.3 The algorithm for generating TTCN from MSCs

After having shown the use of MSCs for test case specification and the relations of data descriptions and MSCs, we will now sketch the algorithm which automates the generation of TTCN test cases from MSCs. Therefore we will at first describe the main principles of TTCN[5]

3.3.1 The main principles of TTCN

A TTCN test case describes the sequences of test events which may happen when the test case is executed. The sequences are arranged in a tree-like manner. A path from the

[5] A tutorial on TTCN can be found in [KW91].

root to a leaf represents one possible sequence of test events. The branches cope with alternative test events.

The test events of a TTCN test case are related to data type and constraint definitions. The relation of a test event to a data type definition is given by the name of the involved message. The relation to the constraints has to be defined explicitly.

For modularization purposes TTCN introduces the concept of test steps. A test step also describes a tree of test events and can be reused in different test cases.

3.3.2 MSCs and TTCN test steps

The test case specification comprises several MSCs which have been combined together to the test case. In general, each MSC is translated into one TTCN test step. The TTCN test case description combines the test steps by referring to them. The generation of the references is based on the diagram which defines the combination of the MSCs. For the test case EDSAOUX the combination has been defined in Figure 8.

3.3.3 Generating TTCN test steps from MSCs

For each MSC a TTCN test step has to be generated. A test step is characterized by the tree of test events, which is generated in five steps.

1. An MSC describes a partial ordered set of send and receive actions. The partial order is defined by the message arrows and by the order of actions along the instance axes. Based on this information we calculate all sequences of actions which include the actions of the MSC and which are consistent with the partial order defined by the MSC.

2. For the test case description only the actions of the testers, i.e. the test events, are of interest. Therefore in the second step from each sequence we remove all actions which are not performed by the testers.

3. MSC and TTCN are different languages with different semantics. For TTCN some of the sequences which have been generated in step 2 are redundant. During a test run they can not be distinguished. In this step the non-redundant sequences have to be selected.

4. The selected sequences are transformed into the TTCN notation.

5. Finally, the references to the default and test case specific constraints are generated and added to the TTCN test case description.

For the complete understanding of step 3 more knowledge concerning the TTCN semantics might be necessary. The details can be found in [ISO92] and [Sut94].

4 Tool support

The success of our method depends on various factors. To improve the acceptance by the users during the development of the method we try to be as close as possible to existing and well established procedures. The success also depends on the availability of tools which support the method. The choice of the standardized languages MSC

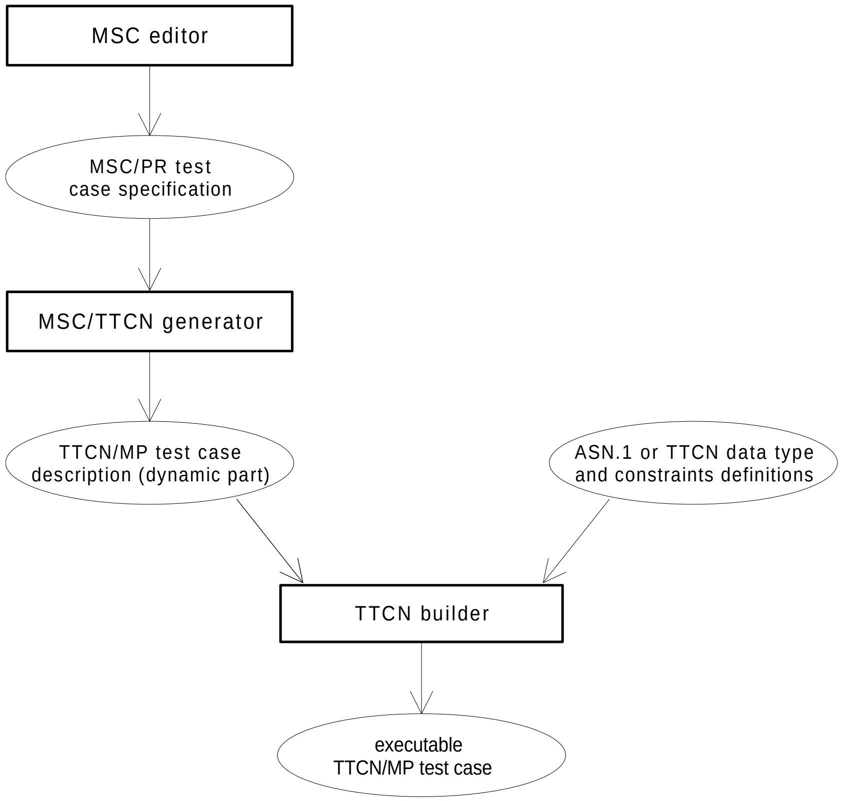

MSC editor
MSC/PR test case specification
MSC/TTCN generator
TTCN/MP test case description (dynamic part)
ASN.1 or TTCN data type and constraints definitions
TTCN builder
executable TTCN/MP test case

Figure 9: A set of prototype tools for test case specification and implementation

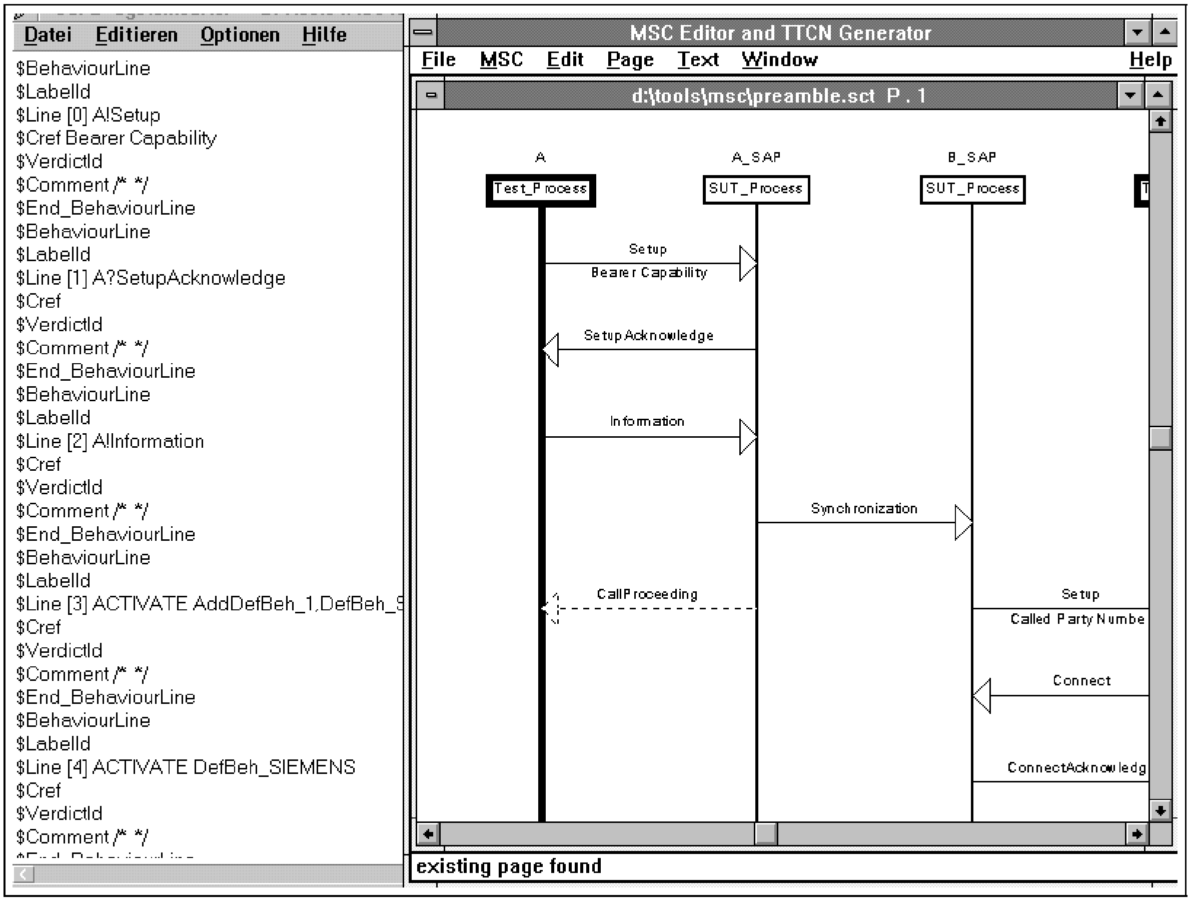
and TTCN allows to use commercial tools for test case specification and test execution. Furthermore, we developed a set of prototype tools which implement our method.

The tool set is shown schematically in Figure 9. The core of the tool set is a graphical MSC editor which can be used to specify MSCs, to refer to, or define test case specific constraints, and to combine MSCs to test case specifications. The editor transforms test case descriptions in the graphical MSC/GR form into the textual MSC/PR form. The MSC/PR files are the input for the *MSC/TTCN generator* which generates TTCN test case without data descriptions in TTCN/MP (Machine Processable) form. The *TTCN builder* combines the output of the MSC/TTCN generator, the data type definitions, and all constraint declarations to complete TTCN test cases.

All tools have been implemented on a PC in a Windows 3.1 environment. Figure 10 gives an impression of the tool interfaces. On the right hand side it presents the user interface of the MSC editor. The shown MSC is the preamble of the test case EDSAOUX. A part of the corresponding generated TTCN/MP code can be seen on the left hand side of the figure.

5 Summary and outlook

The presented method improves the test case specification and automates the test case implementation. With the MSC language a graphical notation is provided which is close to the informal diagrams which are already used for test case specification. MSCs have a standardized syntax and semantics and are therefore suitable for further processing. We introduced a reference mechanism which allows to relate MSCs to data descriptions. With the aid of a tool set it is possible to specify test cases by means of MSCs and to



```
$BehaviourLine
$LabelId
$Line [0] A!Setup
$Cref Bearer Capability
$VerdictId
$Comment /* */
$End_BehaviourLine
$BehaviourLine
$LabelId
$Line [1] A?SetupAcknowledge
$Cref
$VerdictId
$Comment /* */
$End_BehaviourLine
$BehaviourLine
$LabelId
$Line [2] A!Information
$Cref
$VerdictId
$Comment /* */
$End_BehaviourLine
$BehaviourLine
$LabelId
$Line [3] ACTIVATE AddDefBeh_1,DefBeh_S
$Cref
$VerdictId
$Comment /* */
$End_BehaviourLine
$BehaviourLine
$LabelId
$Line [4] ACTIVATE DefBeh_SIEMENS
$Cref
$VerdictId
$Comment /* */
$End_BehaviourLine
```

Figure 10: The user interface of the MSC editor

generate executable test cases in the TTCN notation.

For the application of our method in an industrial environment the interface to the reference mechanism for test case specific constraints should be improved. Complicated message constraints may lead to complex statements of the reference language RL. Furthermore, without detailed knowledge of the message structure the RL statements are not easy to read. But, an RL statement can be considered the minimum information to generate the references to test case specific constraints within the TTCN tables, and to define new constraints which are based on existing ones.

However, we believe that the reference mechanism should have no influence on the test case specification process. We have started to extend the MSC editor by a graphical interface for message constraints. With this interface the user will be able to check, define and modify the message constraints without any knowledge of the underlying reference mechanism.

Acknowledgements

The elaboration of this paper is partially funded by the KWF–Project No. 2555.1 *'Graphical Methods in the Test Process'*. The authors would like to thank F. Bosatta, Ch. Rüfenacht, Dr. R. Schönberger, S. Suter and Ch. Zehnder for their valuable comments and suggestions.

References

[CCI89] CCITT. Recommendations Q.930- Q.940: Digital Subscriber Signalling Systen No. 1 (DSS 1), Network Layer, User-Network Management. The International Telegraph and Telephone Consultative Committee (CCITT), Geneva, 1989.

[GGR93] J. Grabowski, P. Graubmann, and E. Rudolph. The Standardization of Message Sequence Charts. In *Proceedings of the IEEE Software Engineering Standards Symposium 1993*, September 1993.

[ISO92] ISO/IEC JTC 1/SC21. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation. International Standard 9646-3, ISO/IEC, 1992.

[ITU92] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU, Geneva, June 1992.

[KW91] J. Kroon and A. Wiles. A Tutorial on TTCN. In *Proceedings of the 11th International IFIP WG 6.1 Symposium on Protocol, Specification, Testing and Verification*, 1991.

[Rüf94] Ch. Rüfenacht. Extending MSCs with Data Information in order to Specify Test Cases. Diploma Thesis (written in German), University of Berne, Institute for Informatics, February 1994.

[Sie93] Siemens AG. Product Information K1197, K1103. Siemens AG Berlin, 1993.

[Sut94] S. Suter. The MSC Based Generation of the Dynamic Part of TTCN Test Cases. Diploma Thesis (written in German), University of Berne, Institute for Informatics, January 1994.