The Generation of TTCN Test Cases from MSCs

Jens Grabowski

IAM-94-004

May 1994

Abstract

In 1992 and 1993 the University of Berne cooperates with the Siemens-Albis AG Zürich in order to develop a method which allows to generate complete TTCN test cases from MSC descriptions. The goal is reached by extending the MSC language with a few new language constructs, relating MSCs and data descriptions, and developing the algorithms for the TTCN generation. The method is implemented by a set of prototype tools.

The paper starts with a short introduction (Section 1). Then the current procedure of conformance testing is examined (Section 2). The extensions of the standardized MSC language are described and the specification of test cases with MSCs is shown (Section 3). The algorithm for the generation of TTCN behavior descriptions is sketched (Section 4) and MSCs are related to data descriptions (Section 5). The whole method is summarized and a set of prototype tools which implements the method is presented (Section 6). Finally, a short outlook is given (Section 7).

CR Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: General; C.2.2 [Computer-Communication Networks]: Network Protocols; D.2.5 [Software Engineering:] Testing and Debugging

General Terms: Validation, Test Case Generation, Test Case Specification

1 Introduction

Testing is one of the most popular methods 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-TS[1], ISO/IEC, or ETSI) and by the customer which may require additional country specific functions.

The definition of test cases for conformance tests is a complex and error prone process. Therefore, it is necessary to define test cases in a clear and unambiguous way. For this purpose the ISO/IEC standardizes a special test case description language which is called *Tree and Tabular Combined Notation* (TTCN) [11]. TTCN seems to become very important since several standardized TTCN test suites are already available or are in preparation and since it is possible to generate the code which controls the test equipment directly from TTCN descriptions. Unfortunately, TTCN is not easy to read and the purpose of a test case is often hidden in the TTCN notation.

As a consequence we propose to specify the information exchange to fulfill the purpose of a test case in a more user-friendly way. We use Message Sequence Charts (MSCs) [13] which are a widespread and well accepted means for the graphical visualization of selected system runs within telecommunication systems [8]. With some additional assumptions and by extending the standardized MSC language with data references it is possible to generate complete TTCN test cases automatically.

There are several advantages for using MSCs as test case description language. The process of test case specification is facilitated since the MSC language is easier to use than TTCN. MSCs are easy to understand, and therefore a customer is able to make a critical test review without detailed TTCN knowledge. Furthermore, the customer can be enabled to define additional test cases which are documented in a clear, unambiguous and graphical way. As a summary one can say that the use of MSCs for test case specification will improve the quality of test suites.

2 The current procedure of conformance testing

In this section we describe the specification and implementation of test cases for conformance tests by means of an example. The related problems are explained and it is shown which of these problems are solved or improved with our method.

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

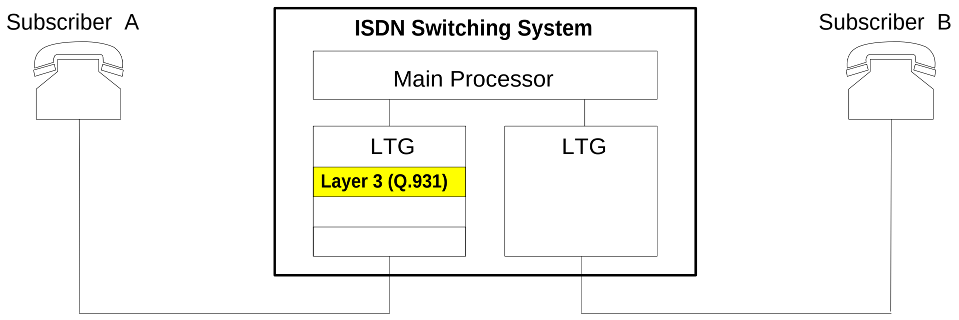


Figure 1: Part of an ISDN system

2.1 An environment for a protocol layer test

Suppose, we want to test a *layer 3 protocol* within a *Line Trunk Group* (LTG) of an ISDN[2] switching system as shown in Figure 1. The layer 3 protocol is given by the ITU-T Recommendation Q.931 [3].

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 is connected with a telephone. The interface of an LTG to the main processor is proprietary and not standardized. Since the conformance to standards shall be tested, proprietary interfaces are not adequate for conformance testing.

One possibility to interface the Q.931 protocol is to use the whole ISDN system and to connect the test equipment directly with the standardized interfaces.[3] Figure 2 shows such a test system. The telephones in Figure 1 are replaced by test devices and the devices are controlled by a test manager which also records the test results.

2.2 The manual specification and implementation of test cases

For the specification and implementation of test cases for an environment like the one in Figure 2 certain tasks have to be carried out. The whole procedure is shown in Figure 3. Within the figure, the rectangles represent actions and the ellipses describe data or documents which serve as input for, or are produced by the different actions.

All three tasks are based on standards and additional customer or country specific requirements. Since these documents mainly are written in plain text, all tasks have to be carried out manually by protocol specialists.

A test case description can be divided into a dynamic and a static part. The dynamic part is mainly given by a test case specification. The static part includes the specification of the data units which are exchanged between the test equipment and the system under test.

The definition of a *test case specification* (Task 1) is based on the relevant protocol standards and, in most cases, on additional country and customer specific requirements.[4]

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

[3]In practice the use of a whole ISDN switching system is very expensive. As a consequence for testing purposes, instead of a real system, parts of the ISDN system are often only simulated.

[4]Country specific requirements may for example be caused by the currency. A Swiss tax counter may

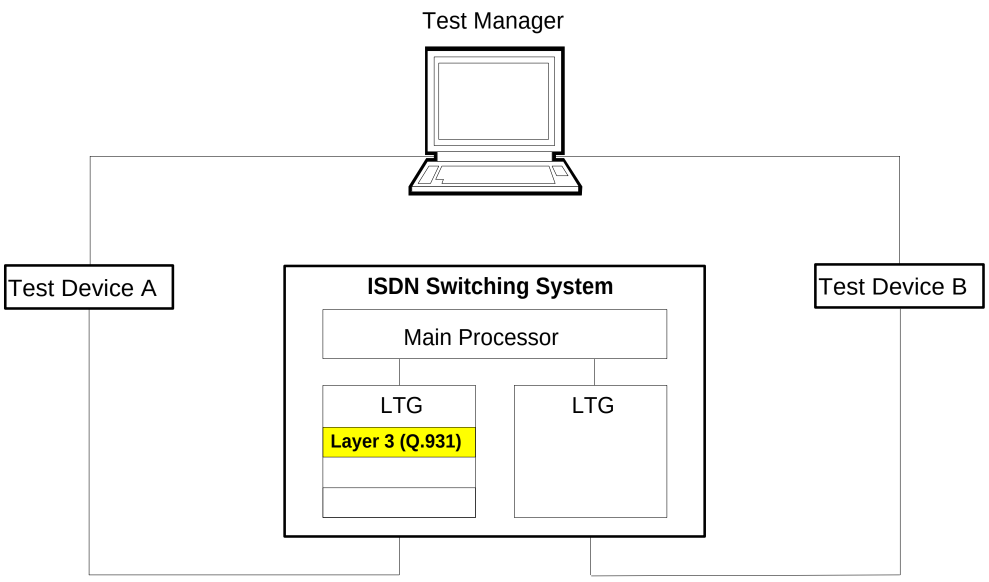


Figure 2: An ISDN test environment

A test case specification consists shall be independent from the concrete implementation and the test equipment. An example of a test case specification will be described in Section 2.4.1.

Task 2, the specification of data types and default constraints is based on the same documents as Task 1. The data type description comprises the definition of messages[5] and message parameters. For the parameters often *default constraints* exists. Such a constraint may define a concrete default value or restrict the range of parameter values. Default constraints also have to be specified formally. The output of Task 2 is a file which is interpretable by the tested system and the test equipment.

For the implementation of a test case (Task 3) the test case specification and the data type and default constraint specification have to be combined to an *executable test case*. An executable test case can be considered to be the program which controls the test equipment during the test case execution. It comprises the complete sequence of the exchanged messages, the corresponding parameter values, actions to synchronize the test devices and further information concerning specific characteristics of the test equipment. Often a test case specification is not sufficient to serve as implementation basis. Concrete parameter values may depend on test purpose or country specific requirements. Consequently, during test case implementation the standards and the additional requirements have to be consulted.

count in 10 Rappen units and a German one may use 10 Pfennig units.

[5]According to the OSI basic reference model [12] protocol entities exchange *protocol data units* (PDUs) and *service primitives*. Since this paper does not treat the OSI model, we use the more general term *message*. But, the abbreviations ASP and PDU occur in several TTCN tables.

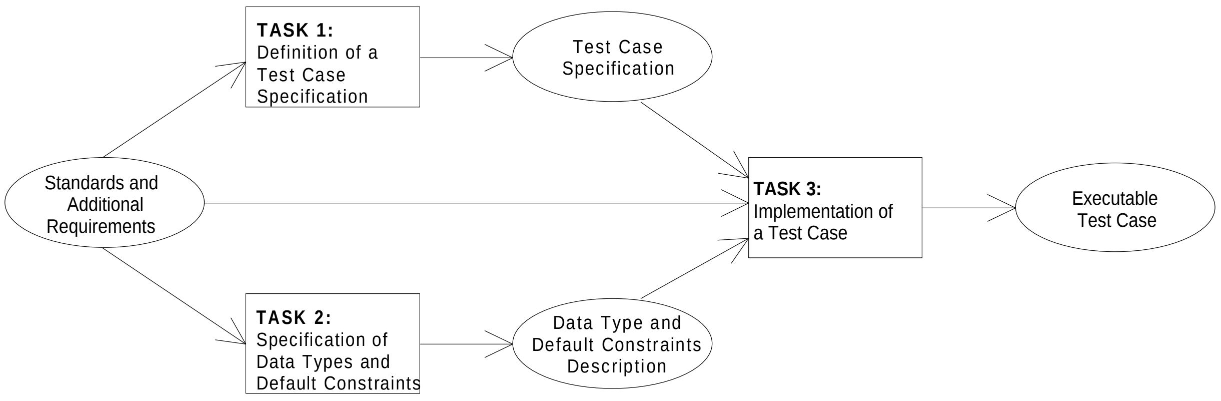

TASK 1:
Definition of a Test Case Specification
Test Case Specification
Standards and Additional Requirements
TASK 3:
Implementation of a Test Case
Executable Test Case
TASK 2:
Specification of Data Types and Default Constraints
Data Type and Default Constraints Description

Figure 3: The specification and implementation of test cases

2.3 The problems of the current procedure

The three tasks identified in the previous section are performed manually. This is the main problem of the whole procedure. Errors may be a result of misunderstanding or misinterpretation of the relevant standards or test case specifications. The intuition and experience of the persons which perform the tasks is a decisive factor for the quality of the test suite and the conformance test itself.

One possibility to improve the whole procedure is to increase the quality of the standards. Here, we have to distinguish between the description of the protocol behavior and the specification of the exchanged data units.

For the behavior description the use of standardized formal description techniques, i.e. LOTOS, Estelle and SDL [10], can help to avoid ambiguities and misinterpretations. The behavior description in existing protocol standards are mainly written in plain text which is enriched with more or less informal drawings. Furthermore, it is not always possible to specify all relevant properties with the standardized formal description techniques, e.g. time, probability and performance aspects cannot be described adequately.

For the data description the situation is more promising. The data description language ASN.1 [17, 4] is frequently used within protocol standards. One reason for the broad acceptance of this language is that there exist encoding rules which allow an automatic implementation of the data types and data values [5].

Because of the mentioned problems, the Tasks 1 and 3 in Figure 3 are the most critical parts in the test case specification and implementation procedure. Task 2 can considered to be easy if the standard includes a formal data specification, e.g. in the ASN.1 notation. The data description only has to be adapted to country and customer specific requirements.

For standards which mainly include descriptions in plain text, Task 1 only can be improved by education and perhaps by a critical review of the obtained test specifications. As we will see in Section 2.4, the output of Task 1 is a more or less informal document. This causes problems for Task 3, since in international telecommunication companies the Tasks 1 and 3 are often not carried out by the same people.

Task 3 is based on two informal documents and a data description. The test errors which can be produced during the realization of this task may be a result of

- misunderstanding and misinterpretation of standard and additional requirements,
- misunderstanding and misinterpretation of the test case specification,
- inconsistencies of the standard and the test case specification, and
- errors in the test case specification.

If during the conformance test the tested implementation behaves in an unexpected way, it is not automatically clear whether the implementation or the test case includes an error. Furthermore, it is possible that an implementation passes a test although it includes errors which should be detected with a correct test suite.

The output of Task 3 is an executable test suite, i.e. a set of executable test cases. Currently, most test case implementations are proprietary for a certain environment, i.e. the various manufacturers of test equipment for example use proprietary programming languages. However, the situation seems to change. The availability of TTCN as standardized test case description language [11], existing and forthcoming standardized TTCN test suites, e.g. [1, 7], and customer demands forces various manufacturers to develop TTCN compiler or interpreter for their test equipment, e.g. [2, 16]. The characteristics of TTCN will be explained in Section 4.1.

It is our goal to improve and to automate the implementation of the test cases, i.e. Task 3 in Figure 3. To motivate our solution we will examine the task in more detail.

2.4 The test case implementation

The implementation of a test case is based on a test case specification, the relevant standards, additional requirements, and the definitions of data types and default constraints. The executable test case is written in an implementation language.

2.4.1 The specification of test cases

In Figure 4 an example of a test case specification is presented. The test case shall prove a certain property of the ISDN system shown in Figure 1. The shown notation is very close to a notation which is used within the Siemens-Albis AG. But, it is not specific to Siemens-Albis, we know from several other telecommunication companies that they use very similar test case descriptions.

The test case specification in Figure 4 consists of two parts. A textual description and a diagram called *general message flow*. The textual description includes a *test case identifier*, a *test purpose*, a *test configuration*, *preconditions* which have to be satisfied before the test case can be applied to the tested system and a hint about further control of the test. The *general message flow* diagram gives some indication about the sequence of messages which shall be observed when the test case is executed. In the following we refer to this test case specification by using the the test case name *EDSAOUX*.

The analysis of the example shows that the test case description is informal and incomplete. The informal character of the description in Figure 4 is obvious, but the incompleteness shall be explained by means of two examples.

```
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:
```

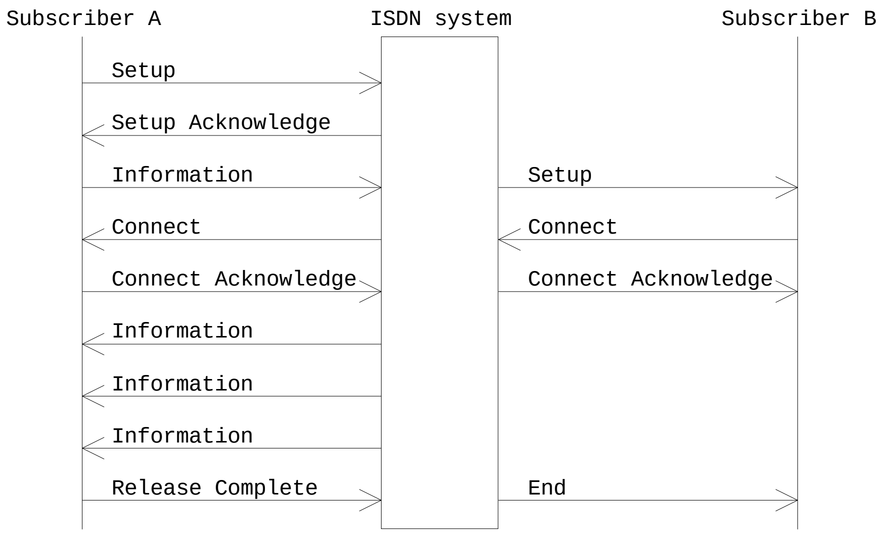


Figure 4: The specification of the test case *EDSAOUX*

1. The ISDN system is connected with the test devices at two different interfaces. In the OSI Reference Model [12] the interfaces are called *service access points* (SAPs). Within the *general message flow* the SAPs are described by the two vertical borders of the box representing the ISDN system. The sequence of messages exchanged at one SAP is described by the order of messages along the corresponding border. However, the protocol describes the message exchange between the users at different SAPs. As a consequence there exists an order between certain messages at different SAPs. For example, the message *Setup* can only be received by *Subscriber B* after the *Information* has been sent by *Subscriber A*. The *general message flow* does not describe such dependencies, although they can be important for the test case implementation, especially, in situations where the test devices have to be synchronized.

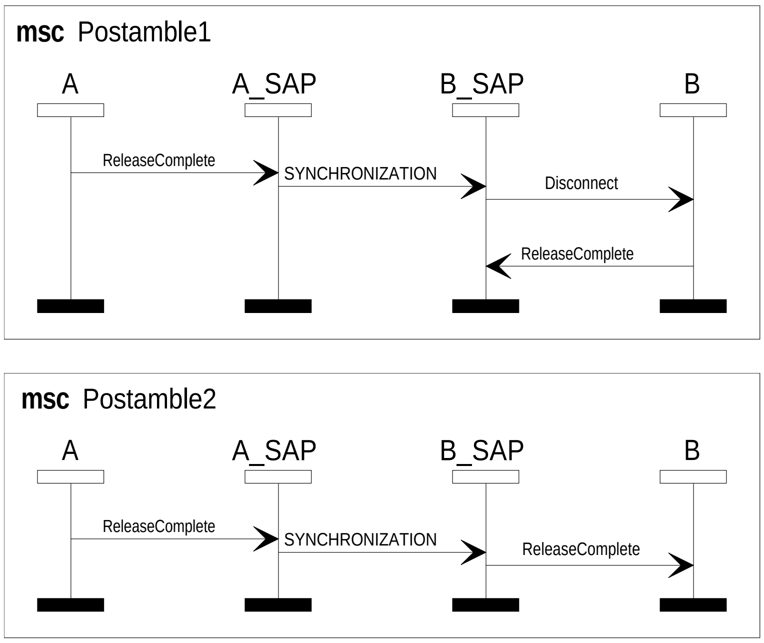


Figure 5: Alternative postambles of the test case *EDSAOUX*

2. The diagram with the *general message flow* includes the message *End*. This message is not defined in the corresponding protocol standard. After the reception of the *ReleaseComplete* message by the ISDN system the standard allows two alternative behaviors of the system. The system may send a *Release Complete* or a *Disconnect* message to the *Subscriber B*. In the case that *Subscriber B* receives a *Disconnect*, it has to answer with a *Release Complete* in order to release the connection correctly after the test case has been applied. The two alternatives are shown in Figure 5[6]. The notation differs slightly from the diagram in Figure 4. The ISDN system is represented by two vertical axes which describe the above mentioned SAPs and the names *Subscriber A* and *Subscriber B* are replaced by *A* and *B*. Furthermore, a special *SYNCHRONIZATION* message describes the ordering between messages at different SAPs.

Most of the missing information can be found in the relevant standards and the additional requirements. It has to be added when the test case is implemented. Therefore the influence of the standards and the additional requirements on the test case implementation may be reduced or even avoided by extending the test case specification. However, a test case specification as shown in Figure 4 can not serve as basis for the automatic generation of executable test cases. But, it defines the functions for the manual implementation of test cases. Since these are also the functions which shall be improved, we list them explicitly.

- The test case specification in Figure 4 indicates the control flow of the executable test case, i.e. the program which controls the test devices, by means of a diagram.

[6]The term *postamble* in the caption of the figure will be explained in Section 3.2.4.

ASN1 ASP Type Definition
ASP Name : Information **PCO Type :** A **Comments :** Reference 3.1.8, u<-->n, local
Type Definition
SEQUENCE {ProtocolDiscriminator [0] ProtocolDiscriminator_type, CallReference [1] CallReference_type, MessageType [2] MessageType_type, SendingComplete [3] SendingComplete_type OPTIONAL, Display [4] Display_type OPTIONAL, KeypadFacility [5] KeypadFacility_type OPTIONAL, CalledPartyNumber [6] CalledPartyNumber_type OPTIONAL}
Detailed Comments :

Figure 6: ASN.1 definition of the *Information* message in Figure 4

- It relates the data type and default constraint description to the test case specification. This is done by the names of messages and message parameters (within the informal text) which refer to type definitions.
- The test case specification describes the requirement which shall be tested. In our case the requirement consists of two properties for a specific message. One denotes the number of receptions by the test equipment. The other is a condition on the format of a message parameter.

2.4.2 The data type and default constraint description

The data type and default constraints description is derived from the relevant standards and the additional requirements. Since the description is necessary for all test cases, we can assume that it is given in a formal language, often even in the implementation language of the test case. Possible formalisms might be ASN.1, TTCN data types, or even C data types. Figure 6 presents an ASN.1 example. It defines the *Information* message which shall be checked by the test case EDSAOUX.

2.4.3 The executable test case

An executable test case can be considered to be the program which controls the test equipment and checks the required properties. For our test case example the executable test case has to simulate the functions of *Subscriber A* and *Subscriber B*. This simulator has to stimulate the ISDN system and to observe the corresponding responses. A response may be expected, not expected, correct, or even incorrect. A complete test case description should treat all different cases.

An executable test case is written in a language which can be interpreted by the test equipment. There are test devices controlled by programs written in C, C++, Forth, or even SDL. Furthermore, there exist a lot of proprietary solutions.

Currently, various manufacturers of test equipment start to interface their test equipment with the *Tree and Tabular Combined Notation* (TTCN) [11]. This development is forced by the demands of users and customers for a standardized interface, and by the availability and development of standardized TTCN test suites.

TTCN is developed as description language for *abstract test cases*, i.e. test case descriptions which are independent of the concrete test realization. Therefore standardized TTCN test suites, e.g. [1, 7], are abstract test suites, i.e. the test cases have to be adapted to the concrete test environment and the country specific requirements.

However, TTCN can be used on several levels of abstraction. The TTCN language can be applied for test case specification, although more intuitive descriptions as for example shown in Figure 4 might be more appropriate for this purpose. But, TTCN also offers the possibility to define test cases in all details. A detailed TTCN description can be executed by test devices [2, 16]. In this sense TTCN is a possible implementation language for executable test cases. In this chapter we also use TTCN for the representation of executable test cases and focus on the generation of TTCN descriptions.

2.5 The goal of our method

It is our goal to improve and to automate the implementation of the test cases, i.e. Task 3 in Figure 3. The advantages are obvious since most of the mentioned problems (cf. Section 2.3) can be avoided. To reach our goal we have to deal with five main problems:

1. The direct influence of standard and additional requirements on the test case implementation has to be suppressed.
2. A description language for test case specifications is needed.
3. The algorithms for generating executable test cases from data type descriptions and test case specifications have to be developed.
4. A mechanism which relates a test case specification to the data type and default constraint description has to be found.
5. We have to work out a mechanism to specify test case specific constraints.

The influence of the standard and the additional requirements on the implementation process can only be suppressed by putting more effort in Task 1, i.e. the missing information of the standard and the additional requirements have to be defined in the test case specifications. This increases the demands for the required test case description language. The formalism should be

- formal enough to support the generation of executable test cases,
- expressive enough to allow the comfortable specification of complete test cases,
- well accepted by the users, because the influence on the tasks 1 and 2 should be as small as possible,
- standardized to avoid proprietary and incomparable test suites, and
- tool supported, because this will facilitate the development, exchange and modification of test case specifications.

3 The specification of test cases with MSCs

There are several reasons for the popularity of test case descriptions as shown in Figure 4. One is of course the fact that all relevant information of a test case can be written on one page. Another reason is the use of informal diagrams (in our example it is called *general message flow*) which immediately give an intuitive understanding of the described behavior. As a consequence of these facts we searched for a graphical formalism which is almost as easy to use as the shown diagram, but which is formal enough to improve the test case implementation. We identified the Message Sequence Chart (MSC) language to be adequate for our purposes. MSC is standardized by the ITU-TS [13], its formal semantics definition is in preparation [8] and there exist tools which support the use of the language [9, 18, 20, 19].

In this section we introduce the MSC language (Section 3.1) and describe certain extensions which adopt the language to the specific needs of testing (Section 3.2).

3.1 The MSC language

The MSC standard is provided by the ITU-T recommendation Z.120. The MSC recommendation includes two syntactical forms: MSC/PR as pure textual and MSC/GR as graphical representation. An MSC[7] in MSC/GR representation can be transformed automatically into a corresponding MSC/PR representation. We profit by the graphical form in the test case specification and base our algorithms on the MSC/PR form. This gives us the flexibility to use several graphical tools for test case specification. Because of simplicity in this paper we only use the MSC/GR form.

Figure 7 presents an example of an MSC. The diagram describes the message flow between the instances *A*, *A_SAP*, *B_SAP* and *B*. The instances are represented by vertical axes. The messages are described by horizontal arrows. An arrow origin and the corresponding arrow head denote sending and consumption of a message. In addition to the message name, parameters may be assigned to a message. The send and receive actions along an instance axis are totally ordered. The order of events on different instance axes is mediated by the messages, i.e. a message must be sent before it can be received. The inscribed hexagon in Figure 7 which covers the instances *A_SAP* and *B_SAP* is a so-called *condition*. It denotes the state *n(0)* which the covered instances have in common.

Further constructs of the MSC language concern *instance actions*, *timer handling*, *instance creation*, *instance termination*, the order of events along an instance axis (*coregion*), and the refinement of instance axes by means of so-called *submscs*. A complete introduction to the MSC language can be found in [8].

The MSC in Figure 7 describes a part of the *general message flow* in Figure 4. The two SAPs of the ISDN system are represented by the individual axes *A_SAP* and *B_SAP*. The dependencies between events on different instance axes are defined explicitly by the *SYNCHRONIZATION* messages. The precondition concerning the initial state *n(0)* is added by means of the condition.

[7]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*.

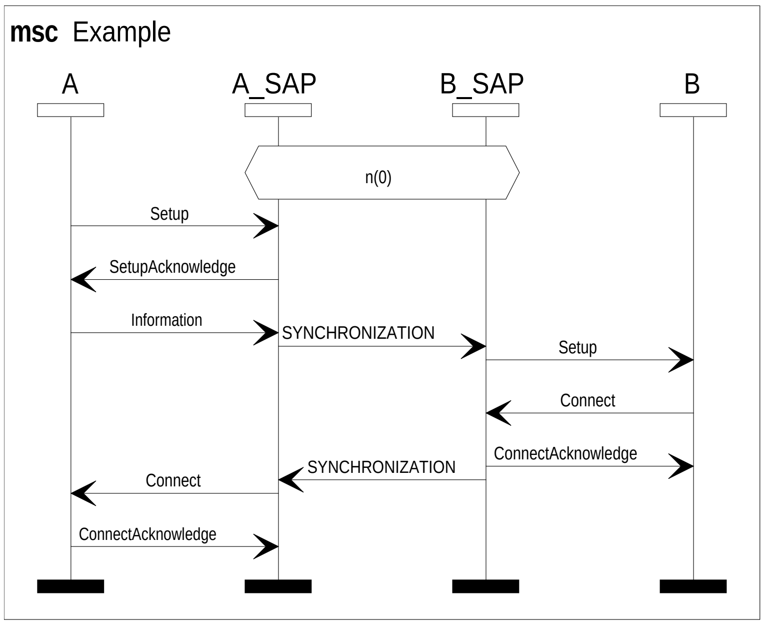


Figure 7: MSC describing a part of the *general message flow* in Figure 4

Obviously, the MSC in Figure 7 offers the same intuitive understanding of the required system behavior as the diagram in Figure 4. The example also shows that it is possible to suppress the influence of standards and additional requirements on the test case implementation (cf. Problem 1 on Page 9) by adding information to an MSC test case description. Furthermore, the MSC/PR form offers a standardized interface for tool supported test case implementation. These facts lead to the conclusion that the MSC language is appropriate for the specification of test cases.

3.2 Adopting MSCs for the needs of testing

The algorithms which automate the generation of TTCN test cases have to extract the actions of the testers[8] from the MSCs and to transform them into the TTCN notation. During the transformation process the different semantics of MSC and TTCN have to be taken into account. In the MSCs which later on will form the test case EDSAOUX the actions are those of the instances *A* and *B* (e.g. Figure 8). They represent *Subscriber A* and *Subscriber B* in Figure 4. The automation requires that the MSCs comprise all tester actions and further relevant information, e.g. information concerning the synchronization of the testers. Besides the sending and reception of messages, a tester may also supervise timers (e.g. Figure 12), or control the number of recurrences of a specific message.

The investigated examples show that the current MSC standard in most cases is 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.

[8]A tester can be a software process, a test device, or a combination of both.

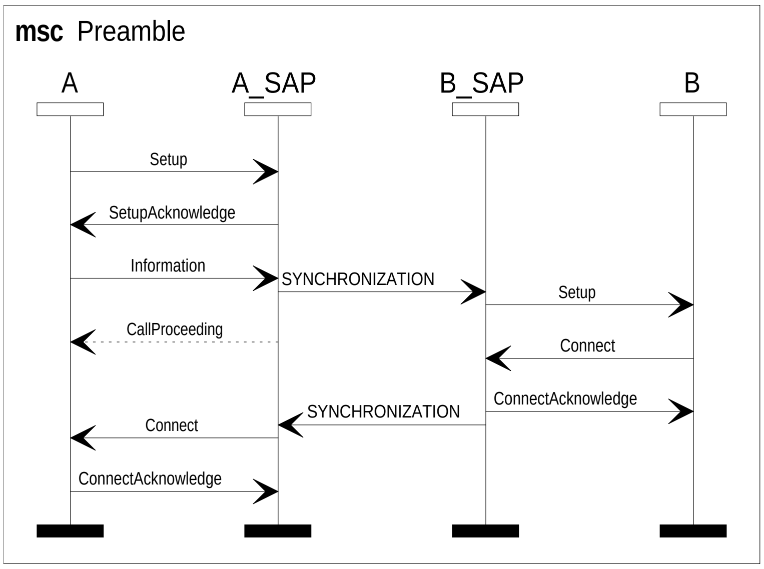


Figure 8: Preamble of the test case *EDSAOUX*

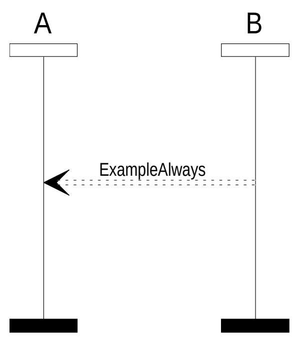
3.2.1 Optional messages and always messages

There are situations where a certain message may or may not occur, but has no further influence on the test run. The tester must be able to handle such messages and therefore they have to be specified. Within our example test cases we identified two sorts of messages and call them *optional messages* and *always messages*.

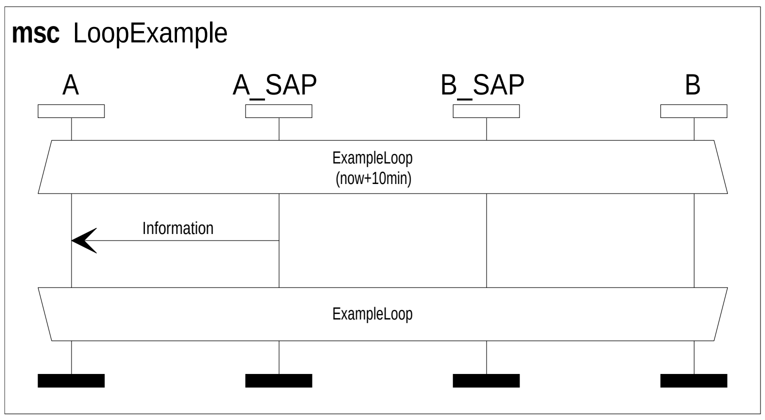
Optional messages. An *optional message* may occur in exactly one situation. The MSC in Figure 8[9] describes almost the same message flow as the MSC in Figure 7. Only the condition is omitted, and additionally it includes the message *CallProceeding*. *CallProceeding* is an optional message which may occur immediately after the *Information* message is sent by *A*. Whether the message occurs depends on the configuration of the whole ISDN system. Sometimes the tester has no influence on this configuration. By the use of the MSC standard the only way to express the possible occurrence of an optional message is to specify two alternative MSCs. Of course, this is awkward and not very user-friendly. As a consequence we introduced optional messages and represent them by dashed arrows.

Always messages. An *always message* is a message which from a certain point in time may always occur arbitrarily often. Within our tool the first use within the MSC defines the point from which it may occur. Figure 9 (a) shows an example for the graphical representation of an always message.

[9]The term *preamble* in the caption of the figure will be explained in Section 3.2.4.



(a) Always message



(b) Timer loop

Figure 9: MSC constructs for test case specification

3.2.2 Synchronization messages

The MSC in Figure 8 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. In certain cases such information is necessary to synchronize the different testers. In our examples we only use synchronization messages to describe the order of events at different SAP axes, but in our tool we also allow to describe synchronization explicitly, i.e. to specify synchronization messages between instance axes which represent testers.

3.2.3 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. Even the test purpose of the test case EDSAOUX requires that the *Information* message should at least occur 3 times. In this simple case it is possible to specify the three messages explicitly, but for more complicated message exchanges it is more appropriate to use a *loop* construct.

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. Figure 9 (b) presents an example. It specifies that the message *Information* should be observed for 10 minutes.

3.2.4 The combination of MSCs

The purpose of the test case EDSAOUX (cf. Figure 4) is to test the arrival of three *Information* messages.[10] The test case can be structured in a *preamble*, a *testbody*, and a *postamble*. The *preamble* describes the message exchange from the initial state *n(0)*

[10]The test of the parameter values will be explained in Section 5.

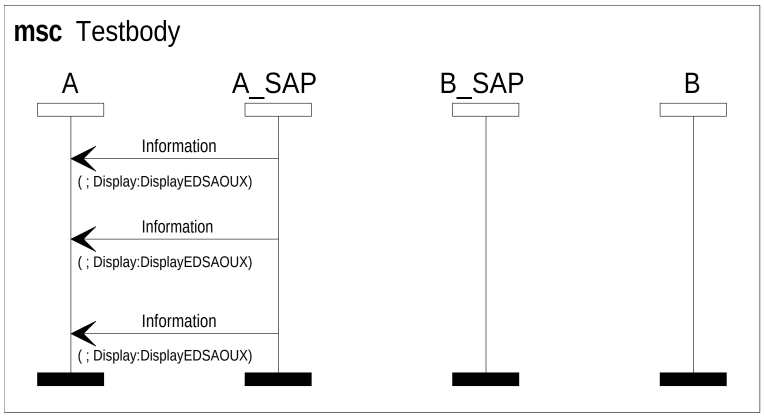

msc Testbody
A
A_SAP
B_SAP
B
Information
(; Display:DisplayEDSAOUX)
Information
(; Display:DisplayEDSAOUX)
Information
(; Display:DisplayEDSAOUX)

Figure 10: Testbody 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 8. The *testbody* comprises the observation of the three *Information* messages. It is shown in Figure 10. The *postamble* includes the message exchange which is necessary to drive the tested system back into the initial state. As stated in Section 2.4.1 for the test case EDSAOUX exist two alternative postambles. They are shown in Figure 5. 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, 8, and 10. Now, we need a mechanism to specify how the MSCs should be combined. Such a description can be looked at as a more general test case description since it abstracts from the message flow.

We use a graphical notation. Figure 11 presents an example. The MSCs are represented by inscribed ellipses. An arrow between two ellipses specifies a *sequence* of two MSCs, e.g. in Figure 11 the signal exchange of the MSC *Preamble* is followed by the MSC *Testbody*, a branching denotes *alternative* MSCs, e.g. the MSCs *Postamble1* and *Postamble2* may happen alternatively. The *supernode* ellipsis only indicates the start of the test case description.

Our graphical notation introduces a *sequence* and an *alternative* operator in the MSC language. The graphical notation can be used to specify recursion, i.e. an MSC is followed by itself. In this case the graph will include loops. The described behavior might be infinite. However, a test case should be finite, and therefore we only allow to specify tree structures. It is possible to define several other operators to combine MSCs, but for test case specification we only need the operators *sequence* and *alternative*. However, a discussion on MSC operators can be found in [6]. It is intended to include such operators in the MSC standard Z.120 [13].

One may argue that it is not necessary to structure test cases in a whole set of MSCs, but even the two possible postambles of our example show that it is necessary to include alternative MSCs in one test case description. Furthermore, we recognized 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. Structuring supports the reuse of existing

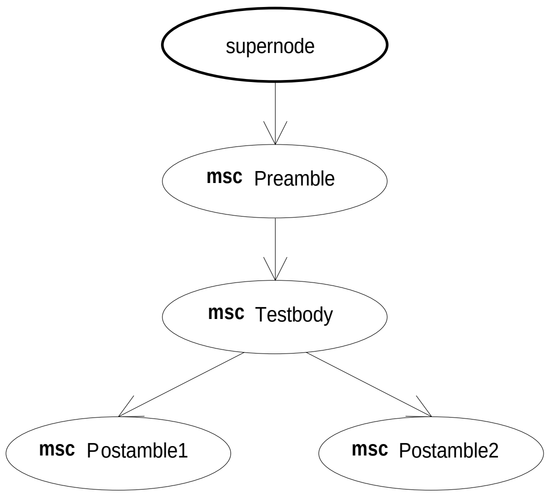


Figure 11: Combining the MSCs which form the test case *EDSAOUX*

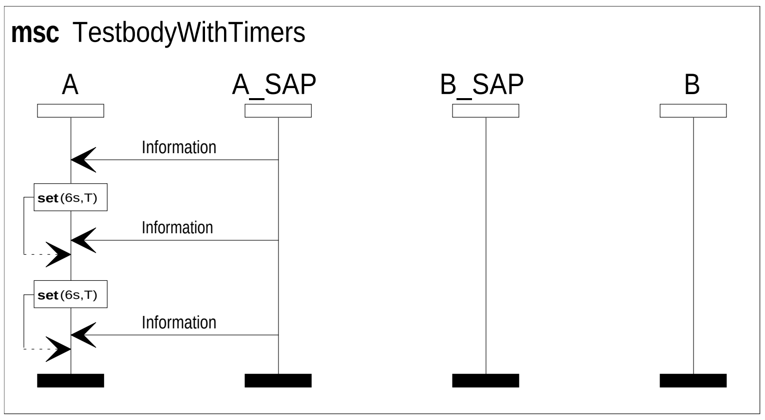


Figure 12: Alternative testbody

test specifications. An example may clarify this. The test purpose of our test case example (cf. Figure 4) is to check the arrival of three *Information* messages. Within the test suite which we investigated another test purpose states that the period of time between two *Information* messages should be less than 6 seconds. The test case descriptions of both only differ in the MSC which checks the *Information* messages. Figure 12 presents the new testbody. This MSC also shows the use of timer constructs in MSCs.

4 The generation of TTCN behavior descriptions

In previous sections we explained how a test case specification can be specified by means of MSCs. From such a specification it is possible to generate the *dynamic part* of a TTCN test case description automatically. The dynamic part of a TTCN test case description specifies the control flow of the executable test case. In order to understand the algorithm we have to introduce the TTCN language briefly.[11]

[11]A more complete tutorial on TTCN can be found in [14].

4.1 The TTCN language

A TTCN description specifies a whole test suite. It consists of

- a *test suite overview* which mainly is a contents list of the test suite,
- a *declarations part* which includes the message and data type definitions,
- a *constraints part* which consists of conditions on message parameters, i.e. default values or value ranges which should be tested, and
- a *dynamic part* which for each test case describes the sequence of exchanged messages.

TTCN has two syntactical forms: TTCN/MP (TTCN Machine Processible form) as pure textual representation and TTCN/GR (TTCN GRaphical form) which is a graphical representation. Both forms are equivalent and can be translated into each other. In this section only the TTCN/GR form is described. As indicated by the name *'Tree and Tabular Combined Notation'* (TTCN), a TTCN test suite is a collection of different tables. The Figures 15, 16, 17, 18, and 19 present several examples for TTCN tables. They are elements of the dynamic part and will be explained later on.

4.1.1 The declarations and constraints part

TTCN has its own data type and value assignment concept. It includes very powerful operators to express conditions on parameter values. For practical purposes TTCN also allows to use ASN.1 in the declarations and constraints part. E.g. the ASN.1 definition of the *Information* message in Figure 6 is written in a TTCN table. An ASN.1 constraint which checks the correct value format of the test case EDSAOUX is shown in Figure 13. For the sake of simplicity, we only like to mention that the fix letters of the format are represented by bit strings, e.g. the first letter F of the format *'FR. x.x.0'* $(0\leq x\leq 9)$ (cf. Figure 4) is represented by *'01000110'B.*

TTCN allows to structure constraints, i.e. a constraint may itself refer to other constraints. An example is shown in Figure 14. This constraint refers to the constraint shown in Figure 13

4.1.2 The dynamic part

A TTCN test case describes the sequences of events which should be performed by the testers. In general, these are send and receive events. The event sequence is specified by means of a tree notation. Figure 15 shows an example. The tree notation can be found in the *Behaviour Description* column.

The tree structure is determined by the ordering and the indentation of the specified events. In general, the same indentation denotes a branching (i.e. alternative events, e.g. lines Nr. 2 and 4) and the next larger indentation denotes a succeeding event (e.g. lines Nr. 1 and 2). Events are characterized by the involved entities (i.e. A and B), by its kind (i.e. "!" denotes a send event and "?" describes a receive event) and by the message

ASN1 Type Constraint Declaration
Constraint Name : DisplayEDSAOUX
ASN1 Type : Display_type
Derivation Path :
Comments : Test case specific constraint for Display values (Test case name: EDSAOUX)
Constraint Value
{d_id ’00101000’B, d_length ’08’H, d_info {’01000110’B, ’01110010’B, ’00101110’B, ’00100000’B, (’00110???’B, ’0011100?’B), ’00101110’B, (’00110???’B, ’0011100?’B), ’00110000’B}}
Detailed Comments :

Figure 13: ASN.1 constraint *DisplayEDSAOUX*

ASN1 ASP Constraint Declaration
Constraint Name : InformationEDSAOUX
ASP Type : Information
Derivation Path :
Comments : Test case specific constraint for Information messages (Test case name: EDSAOUX)
Constraint Value
{ProtocolDiscriminator ProtocolDiscriminatorDefRec, CallReference CallReferenceDefRec, MessageType ’01111011’B, SendingComplete SendingCompleteDefRec IF_PRESENT, Display DisplayEDSAOUX, KeypadFacility KeypadFacilityDefRec IF_PRESENT, CalledPartyNumber CalledPartyNumberDefRec IF_PRESENT}
Detailed Comments:

Figure 14: ASN.1 constraint *InformationEDSAOUX*

Test Step Dynamic Behaviour

Test Step Name : Postamble
Group :
Objective :
Default : UnexpectedEvents
Comments : Postamble of the test case EDSAOUX

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		A!ReleaseComplete	ReleaseCompleteDefSend		
2		B?Disconnect	DisconnectDefRec		
3		B!ReleaseComplete	ReleaseCompleteDefSend	PASS	
4		B?ReleaseComplete	ReleaseCompleteDefRec	PASS	

Detailed Comments :

Figure 15: TTCN test step *Postamble*

Default Dynamic Behaviour

Default Name : UnexpectedEvents
Group :
Objective :
Comments :

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		A?OTHERWISE		FAIL	
2		B?OTHERWISE		FAIL	

Detailed Comments : FAIL is assigned if something unexpected happens

Figure 16: TTCN default behavior description *UnexpectedEvents*

Test Case Dynamic Behaviour

Test Case Name : EDSAOUX
Group :
Purpose : The test shall ensure that after connection establishment Subscriber A receives at least three Information messages.
Default : UnexpectedEvents
Comments : The display parameter of an Information message shall have the format 'FR. x.x0' (0=<x=<9)

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+Preamble			
2		+Testbody			
3		+Postamble			

Detailed Comments :

Figure 17: TTCN test case description *EDSAOUX*

which should be sent or received. An example may clarify the notation. The statement *B?Disconnect* denotes the reception of the message *Disconnect* by the entity *B*.

The table in Figure 15 includes some further information. The entries in the *Constraints Ref.* column refer to a TTCN or ASN.1 constraint. An entry in the *Verdict* column assigns a so-called *test verdict* to a test run. The verdicts indicate the success of the test run. A *pass* verdict denotes that the test purpose is reached, a *fail* states that something *bad* happens and an *inconclusive* describes a situation where neither a *pass* nor a *fail* can be assigned.

The example in Figure 15 only includes *pass* verdicts. The *fail* cases are specified in a *default behavior description* which is referred in the test case header. The default is shown in Figure 16. It includes the special event *OTHERWISE* which represents arbitrary (correct or even incorrect) messages.

TTCN allows to structure test case descriptions. This is done by so-called *test steps*. A test step is a behavior tree which can be added to other behavior trees by means of a so-called *tree attachment*. Figure 17 presents an example. The test case attaches at first the test step *Preamble* (Figure 18), then the test step *Testbody* (Figure 19), and finally the test step *Postamble* (Figure 15).

The TTCN tables in the Figures 15, 16 17, 18, and 19 specify the whole message

Test Step Dynamic Behaviour

Test Step Name	**:**	Preamble
Group	**:**	
Objective	**:**	
Default	**:**	UnexpectedEvents
Comments	**:**	Preamble of the test case EDSAOUX

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		A!Setup	SetupDefSend		
2		A?SetupAcknowledge	SetupAcknowledgeDefRec		
3		A!Information	InformationDefSend		
4		B?Setup	SetupDefRec		
5		B!Connect	ConnectDefSend		
6		B?ConnectAcknowledge	ConnectAckDefRec		
7		A?CallProceeding	CallProceedingDefRec		
8		A?Connect	ConnectDefRec		
9		A!ConnectAcknowledge	ConnectAckDefSend		
10		A?Connect	ConnectDefRec		
11		A!ConnectAcknowledge	ConnectAckDefSend		

Detailed Comments :

Figure 18: TTCN test step *Preamble*

Test Step Dynamic Behaviour

Test Step Name	**:**	Testbody
Group	**:**	
Objective	**:**	
Default	**:**	UnexpectedEvents
Comments	**:**	Testbody of the test case EDSAOUX

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		A?Information	InformationEDSAOUX		
2		A?Information	InformationEDSAOUX		
3		A?Information	InformationEDSAOUX		

Detailed Comments : Checks the format of the display parameter in an Information message.

Figure 19: TTCN test step *Testbody*

exchange of the test case EDSAOUX. The main test case description is given by the table in Figure 17. It attaches several test steps and refers to a default behavior description.

4.2 The algorithm

In this section we sketch the algorithm which automates the generation of TTCN behavior descriptions from MSC specifications.

4.2.1 MSCs and TTCN test steps

The test case description comprises several MSCs. In general, each MSC is translated into one test step, i.e. in one TTCN table. The MSCs of the test case EDSAOUX can be found in the Figures 5, 8, and 10. The corresponding TTCN tables are shown in the Figures 15, 18, and 19. For the sake of simplicity the alternative postambles (cf. Figure 5) are described in one TTCN table (cf. Figure 15).

The TTCN test case description combines the test steps by means of tree attachments. The TTCN table which forms the test case EDSAOUX can be found in Figure 17. The generation of such a table is a based on the diagram which defines the combination of the MSCs. For the test case EDSAOUX the combination is defined in Figure 11.

4.2.2 Generating TTCN test steps from MSCs

For each MSC a TTCN test step has to be generated. The test step is specified mainly by the TTCN behavior tree in the *Behavior Description* column. The behavior tree is generated in four steps.

1. An MSC describes a partial ordered set of actions. The partial order is defined by the messages and by the order of actions along the instance axes (cf. Section 3.1). Based on this information we calculate the sequences of actions which include the actions of the MSC and which are consistent with the partial order defined by the MSC. For the MSC in Figure 8 for example 402 different sequences exist.

2. For the test case description only the actions of the testers are of interest. Therefore in the second step we remove all actions which are not performed by the testers from each sequence. For the MSC in Figure 8 we gain 13 different sequences. They are shown in Figure 20. The actions in the figure are defined in a TTCN like manner.

3. MSC and TTCN are different languages with different semantics. For TTCN some of the sequences which we generated in step 2 are redundant. During a test run they can not be distinguished. In other words, for TTCN several sequences are in the same equivalence class. In the third step we select one sequence of each equivalence class. For the sequences in Figure 20 only two different equivalence classes exist. One class includes the sequences (1) - (3), the other consists of the sequences (4) - (13). We select the sequences (1) and (10) for the test case description.

4. In the fourth step the selected sequences are transformed into the TTCN notation. The TTCN notation for the sequences (1) and (10) can be found in Figure 18.

For the complete understanding of step 3 more knowledge concerning the TTCN semantics might be necessary. But, in this chapter we only want to give an overall view. The details can be found in [11] and [18].

4.2.3 Assigning test verdicts

A TTCN test case assigns one of three test verdicts to a complete test run. We only assign the two verdicts *pass* and *fail*. This is due to the fact that industrial testing presupposes

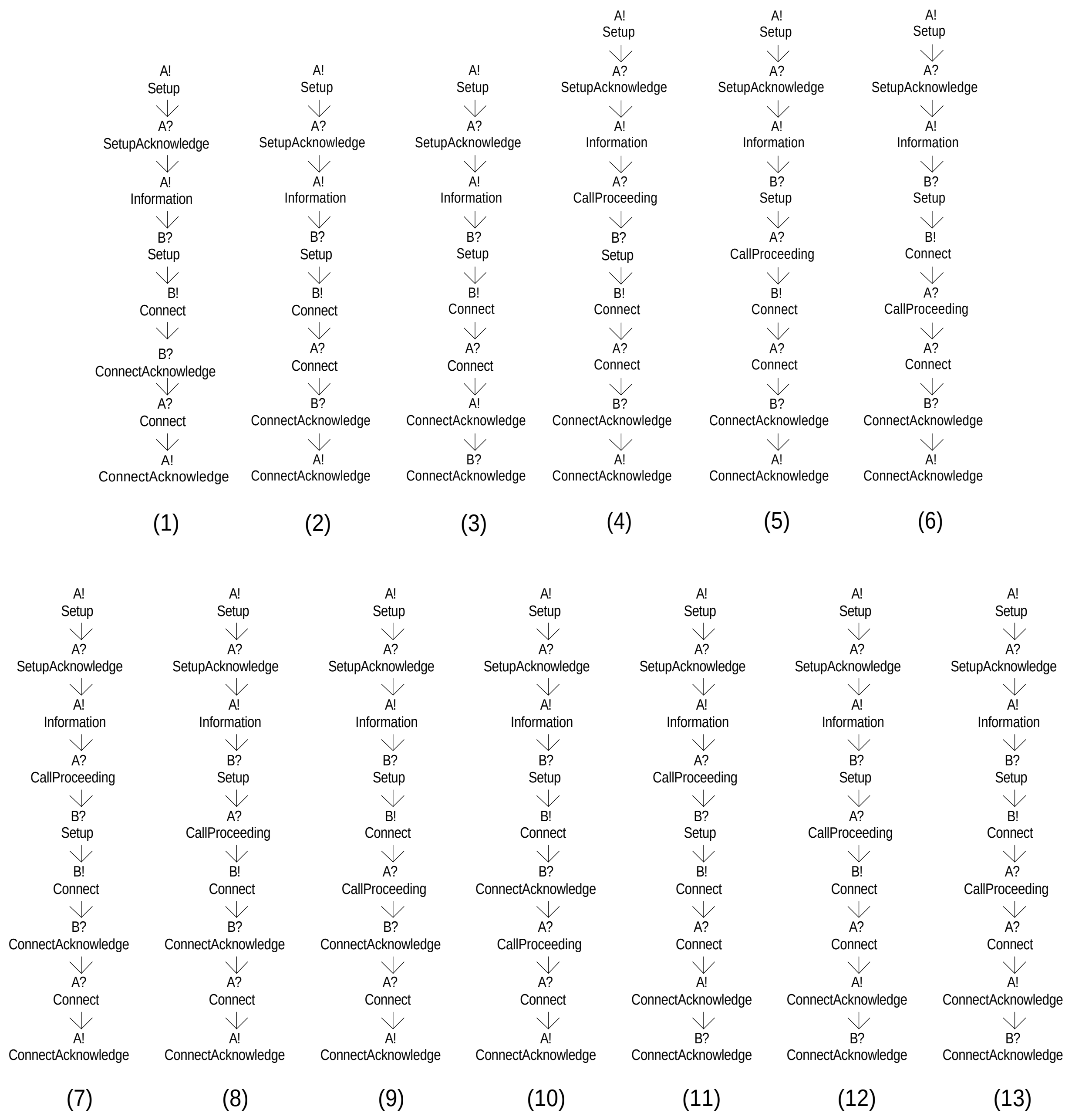


Figure 20: The different sequences of tester actions for the MSC in Figure 8

complete control over the implementation which should be tested and the test equipment. An *inconclusive* case should not happen.

We assume that the *pass* behavior is defined by the MSCs. A *fail* is assigned if something unexpected happens. In the test case EDSAOUX a *pass* is given if one of the two postambles is performed (cf. Figure 15). The *fail* cases are defined by means of a default behavior description (cf. Figure 16).

ASN1 Type Definition

Type Name : Display_type

Comments : Information element Display, Reference 4.5.15

Type Definition

```
SEQUENCE
{d_id [0]  OCTET STRING (SIZE (1)),
 d_length [1]  OCTET STRING (SIZE (1)),
 d_info [2]  OCTET STRING (SIZE (0..32)) OPTIONAL}
```

Detailed Comments :

Figure 21: ASN.1 type definition *Display_type*

5 MSCs and data descriptions

In the previous section it is shown how test case specifications can be described by MSCs and how the dynamic part of a TTCN test case can be generated automatically. In order to gain complete TTCN test cases the MSCs have to be related to data type and default constraints definitions (cf. Problem 4 on Page 9), and we have to find a mechanism to specify test case specific constraints (cf. Problem 5 on Page 9).

5.1 Data type and default constraints definitions

As described in Section 2.2 we can assume that data type and default constraints definitions are given in a form which can be interpreted by a machine. The relations between these definitions and the messages in an MSC 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. We explain this by means of the test case EDSAOUX.

The *Information* message in the testbody of the test case (cf. Figure 10) refers to the ASN.1 definition in Figure 6. The test case checks a part of the *Display* parameter format. The *Display* parameter has the type *Display_type*. The corresponding type definition is shown in Figure 21.

For most messages and message parameter values the protocol standard, and the additional user requirements provide default constraints, i.e. they define default values or restrict the value range. Also for the *Information* message which should be checked by the test case EDSAOUX a default constraint exists. It is shown in Figure 22. The constraint refers to the default constraints for the parameter values. Figure 23 presents the default constraint of the *Display* parameter. The value of *d_id* is completely defined by the bit string *'00101000'B*. The possible values of *d_length* are listed. Contrary to this, the question mark states that the value of *d_info* is not restricted. According to the type definition in Figure 21 it is an arbitrary string of octets with a maximal length of 32 (in hexadecimal form '20'H). We like to mention that the format which should be checked by the test case EDSAOUX is encoded in *d_info*. For the decoding of the message it is necessary to know that *d_length* describes the length of *d_info* in form of a hexadecimal string.

In the test case EDSAOUX the *Information* message occurs in two different situations.

ASN1 ASP Constraint Declaration	
Constraint Name	: InformationDefRec
ASP Type	: Information
Derivation Path	:
Comments	: Default constraint for Information messages which are received
Constraint Value	
{ProtocolDiscriminator ProtocolDiscriminatorDefRec, CallReference CallReferenceDefRec, MessageType '01111011'B, SendingComplete SendingCompleteDefRec IF_PRESENT, Display DisplayDefRec IF_PRESENT, KeypadFacility KeypadFacilityDefRec IF_PRESENT, CalledPartyNumber CalledPartyNumberDefRec IF_PRESENT}	
Detailed Comments	:

Figure 22: ASN.1 constraint *InformationDefRec*

ASN1 Type Constraint Declaration	
Constraint Name	: DisplayDefRec
ASN1 Type	: Display_type
Derivation Path	:
Comments	: Default constraint for Display values which are received
Constraint Value	
{d_id '00101000'B, d_length ('00'H, '01'H, '02'H, '03'H, '04'H, '05'H, '06'H, '07'H, '08'H, '09'H, '0A'H, '0B'H, '0C'H, '0D'H, '0E'H, '0F'H, '10'H, '11'H, '12'H, '13'H, '14'H, '15'H, '16'H, '17'H, '18'H, '19'H, '1A'H, '1B'H, '1C'H, '1D'H, '1E'H, '1F'H, '20'H), d_info ?}	
Detailed Comments	:

Figure 23: ASN.1 constraint *DisplayDefRec*

In the preamble it is sent by tester *A* (cf. Figure 8), and in the testbody it is received by tester *A* (cf. Figure 10). The message type definitions are identical, but the default constraints for both situations differ. A message which shall be sent by a tester must have concrete parameter values. Figure 24 presents the default constraint for an *Information* message which is send by a tester. The constraint refers to other parameter constraints. It has no *Display* parameter. The standard [3] requires that an *Information* message which is send by a subscriber has no *Display* parameter. Therefore the *Information* message definition declares *Display* as optional parameter.

For the default constraints the names of the messages are sufficient to generate the entries in the *Constraints Ref.* column of the TTCN test step tables, e.g. the TTCN table in Figure 18 (line Nr. 3) refers to the constraint in Figure 24. The MSC language need not to be extended. Different situations may require different default constraints for a message, or message parameter, but name conventions can be used to avoid ambiguities.

ASN1 ASP Constraint Declaration	
Constraint Name	: InformationDefSend
ASP Type	: Information
Derivation Path	:
Comments	: Default constraint for Information messages which are send
Constraint Value	
{ProtocolDiscriminator ProtocolDiscriminatorDefSend, var_CallReference CallReferenceDefSend, var_MessageType '01111011'B, var_CalledPartyNumber CalledPartyNumberDefSend}	
Detailed Comments :	

Figure 24: ASN.1 constraint *InformationDefSend*

5.2 Test case specific constraints

Test case specific constraints 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 proved, and test purposes often include constraints on message parameter values.

Also the test case EDSAOUX includes a test case specific value constraint. The test purpose requires to check the *Display* parameter format of the *Information* message. The format definition *'Fr. x.x0' (0 $\leq$ x $\leq$9)* (cf. Figure 4) is a constraint on the value range of the *Display* parameter.

Test case specific constraints have to be defined formally when the test case is implemented. Currently, the definition of the test case specific constraints is based on the informal test purpose description in the abstract test case, the data type definitions and the standards and additional requirements. We described this situation in Figure 3.

Since it is our goal to automate the test case implementation we have to suppress the influence of the additional requirements and standards. Our way to do this is to formalize the data aspects of test purposes, i.e. we include the test case specific constraints in the abstract test case description.

5.2.1 MSCs and test case specific constraints

We have two possibilities to introduce test case specific value constraints in MSCs. They can be explicitly defined in the MSCs, or they can be defined elsewhere and the MSCs refer to them.

The first possibility is problematic, because the constraints may become too big for the MSC. We explain this by means of the test case EDSAOUX. The ASN.1 constraint for the *Information* message which should be tested is shown in Figure 14. This constraint refers to another constraint which checks the format of the *Display* parameter. It is shown in Figure 13. However, the constraints of the *Information* message comprise two pages, and they should be valid for each *Information* message of the MSC in Figure 10. One will loose all clearness if the MSC and all constraints are defined in the same diagram.

The second possibility is problematic, because the principle of locality is violated.

A reference mechanism may lead to situations where the relevant parts of a test case description are defined at different locations.

Often, a test case specific constraint only differs slightly from an existing default constraint. In the test case EDSAOUX the default constraint and the test case specific constraint of the *Information* message are only different with respect to the *Display* parameter constraint (cf. Figures 22, 14). In such a case it is more appropriate to specify the difference to a default constraint within the MSC than to rewrite the whole default constraint.

5.2.2 A reference mechanism for test case specific constraints

As consequence of the discussion in the previous section we decided to develop 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 a test case specific constraint, and
- allows to define test case specific constraints within an MSC, e.g. if a test case specific constraint only comprises one concrete value.

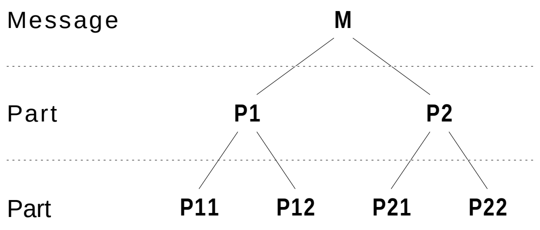
The reference mechanism is a reference language, in the following called RL, which can be used to specify the mentioned possibilities. Within an MSC the statements of RL are related to messages. They can be found in parenthesis near the corresponding message name, or message arrow (cf. Figure 10). This is no extension of the MSC language, because the MSC standard [13] proposes to use expressions in round brackets to assign parameter information to messages.

An RL statement consists of several *parts*. The parts are separated by semicolons. Each part may consist of several *subparts* which are separated by commas. The structure of an RL statement reflects the structure of a corresponding message.

A message has a hierarchical structure. A part of an RL statement represents a hierarchy level. The subparts describe elements within a hierarchy level. Figure 25 presents an example. The message *M* in (a) has the parameters *P1* and *P2*. *P1* is structured in *P11* and *P12*. *P2* comprises *P21* and *P22*. The statement in (b) shows how the different elements of *M* can be referred within an RL statement. The parts and subparts of an RL statement refer to, or are constraints for the corresponding message. In general, the omission of a part or subpart means that the default constraint is used.

Based on such an RL statement it is possible to automate the calculation of the references to test case specific constraints within the TTCN tables, and to generate test case specific constraints which are based on existing constraints. In this chapter we do not want to describe the details of RL, but an example shall give an impression of the reference mechanism. The details can be found in [15].

The MSC in Figure 26 is the testbody of the test case EDSAOUX (cf. Figure 10). The inscription of the *Information* messages *(; Display:DisplayEDSAOUX)* states that



(a) Message structure

(<M>; <P1>, <P2>; <P11>, <P12>, <P21>, <P22>)

(b) Structure of an RL statement

Figure 25: RL statements and the message structure

the constraint for these messages is a modification of the default constraint. The default constraint for the *Display* parameter shall be replaced by the test case specific constraint *DisplayEDSAOUX*. Figure 26 only indicates the replacement schematically. But, we already presented the concrete ASN.1 constraints. The default constraints for the *Information* message and the *Display* parameter are shown in the Figures 22 and 23. The test case specific constraint *DisplayEDSAOUX* is shown in Figure 13. The test case specific constraint for the *Information* message which can be generated automatically is shown in Figure 14.

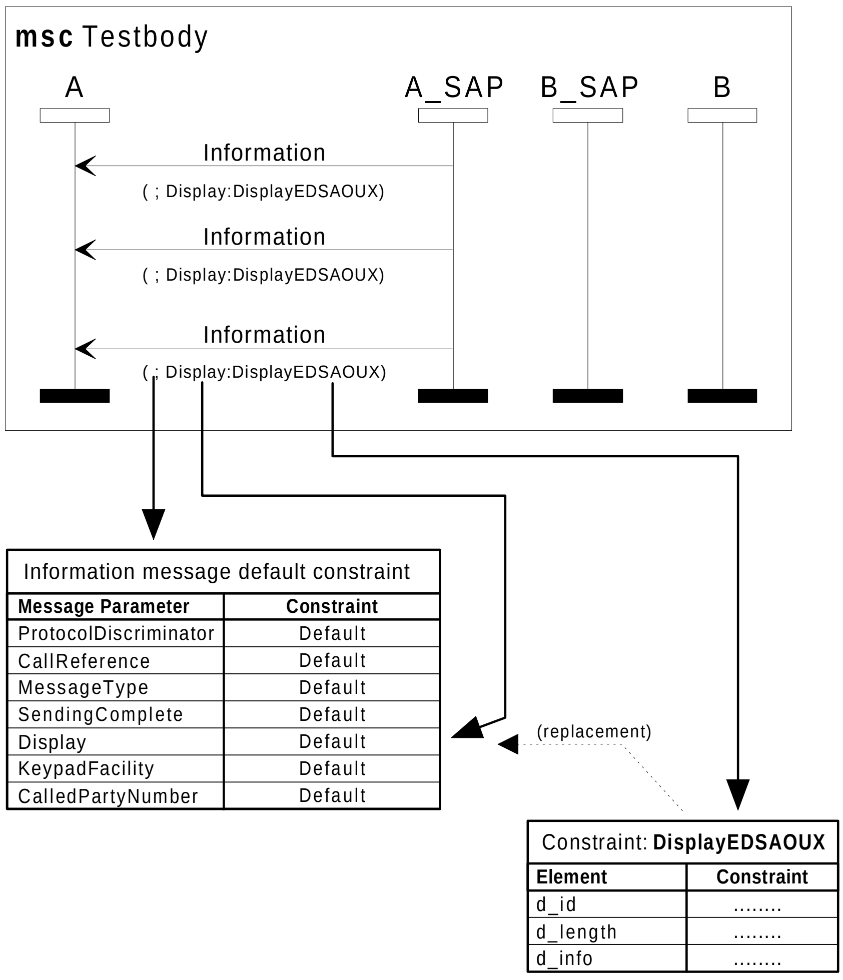
6 Summary and tool support

Within the previous sections we propose a method which automates the implementation of test cases. The influence of informal protocol standards and user requirements is the main problem of the current test case implementation procedure. We suppress this influence by extending and formalizing the description of test case specifications.

The MSC language plays the central role of the method, because it is the formalism used to describe test case specifications. The language is extended with a few constructs to meet the specific requirements of test case specification. It is shown how data type and default constraint definitions are related to MSCs and a comfortable reference mechanism for test case specific constraints is presented. We use TTCN as description language for executable test cases and sketch the algorithms which generate TTCN test cases from MSC test case descriptions. The algorithms presuppose that data type and default constraint definitions are specified in ASN.1 or TTCN.

The success of such a 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, TTCN and ASN.1 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 27. The tools are represented by rectangles and the interfaces between them by ellipses. 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 abstract test cases. The editor transforms test


msc Testbody
A
A_SAP
B_SAP
B
Information
(; Display:DisplayEDSAOUX)
Information
(; Display:DisplayEDSAOUX)
Information
(; Display:DisplayEDSAOUX)
(replacement)

Information message default constraint	
Message Parameter	**Constraint**
ProtocolDiscriminator	Default
CallReference	Default
MessageType	Default
SendingComplete	Default
Display	Default
KeypadFacility	Default
CalledPartyNumber	Default

Constraint: **DisplayEDSAOUX**	
Element	**Constraint**
d_id	....
d_length	....
d_info	....

Figure 26: The reference mechanism for test case specific constraints

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 the dynamic part of a TTCN test case in TTCN/MP form. The *TTCN builder* combines the output of the MSC/TTCN generator, and the data type and constraint definitions to complete TTCN test cases. The TTCN builder calculates the constraint references in the TTCN test step tables and generates additional test case specific constraints which are defined by our reference mechanism. All tools are implemented on a PC in a Windows 3.1 environment.

The Figures 28 and 29 shall give an impression of the tool interfaces. Figure 28 presents the user interface of the MSC editor. The shown MSC is the preamble of the test case EDSAOUX. The corresponding MSC/PR form and the generated TTCN/MP form can be found in Figure 29.

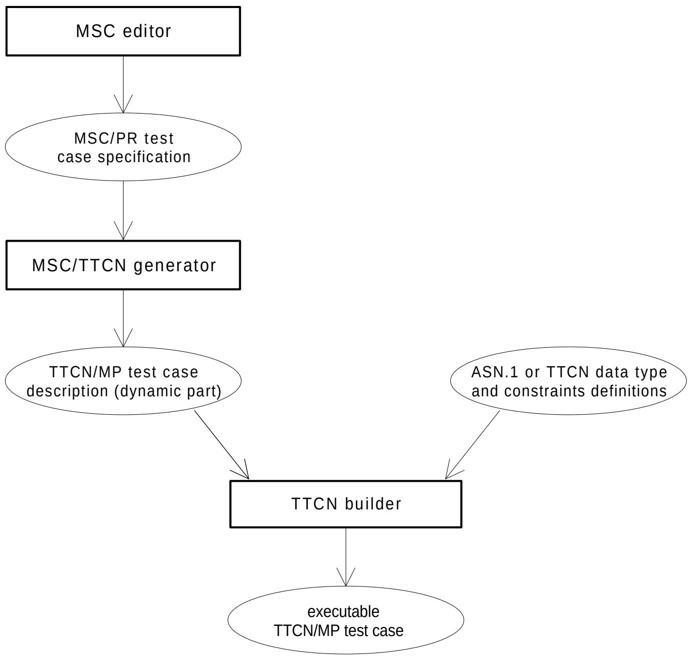
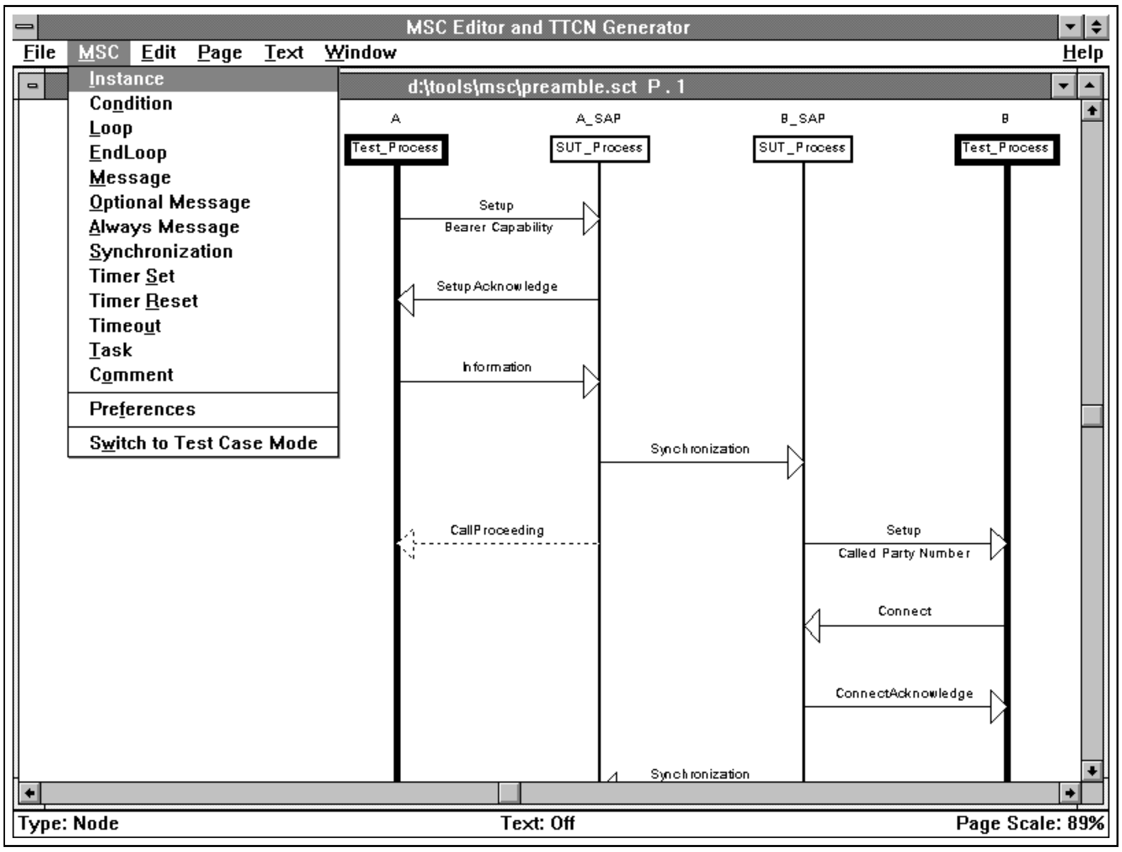
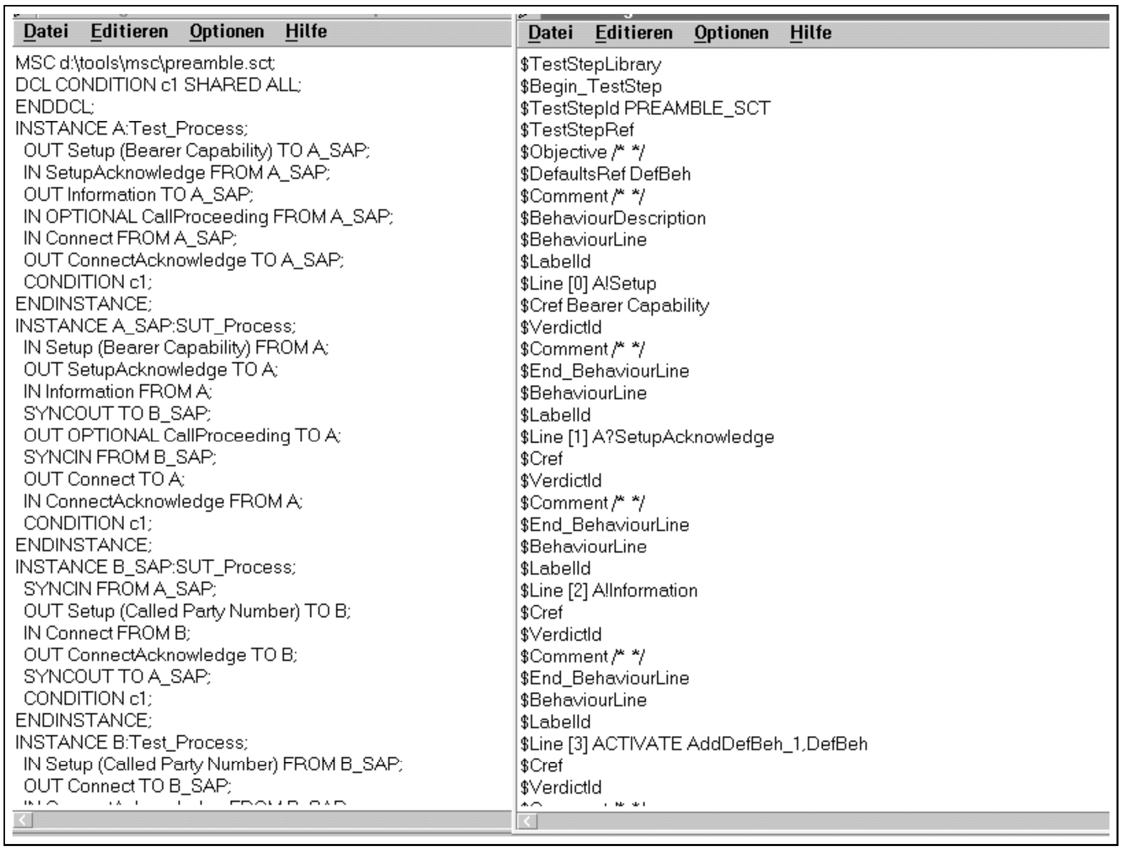

MSC editor
MSC/PR test case specification
MSC/TTCN generator
TTCN/MP test case description (dynamic part)
ASN.1 or TTCN data type and constraints definitions
TTCN builder
executable TTCN/MP test case

Figure 27: A set of prototype tools for test case specification and implementation



Type: Node   Text: Off   Page Scale: 89%

Figure 28: The user interface of the MSC editor



```
MSC d:\tools\msc\preamble.sct;
DCL CONDITION c1 SHARED ALL;
ENDDCL;
INSTANCE A:Test_Process;
 OUT Setup (Bearer Capability) TO A_SAP;
 IN SetupAcknowledge FROM A_SAP;
 OUT Information TO A_SAP;
 IN OPTIONAL CallProceeding FROM A_SAP;
 IN Connect FROM A_SAP;
 OUT ConnectAcknowledge TO A_SAP;
 CONDITION c1;
ENDINSTANCE;
INSTANCE A_SAP:SUT_Process;
 IN Setup (Bearer Capability) FROM A;
 OUT SetupAcknowledge TO A;
 IN Information FROM A;
 SYNCOUT TO B_SAP;
 OUT OPTIONAL CallProceeding TO A;
 SYNCIN FROM B_SAP;
 OUT Connect TO A;
 IN ConnectAcknowledge FROM A;
 CONDITION c1;
ENDINSTANCE;
INSTANCE B_SAP:SUT_Process;
 SYNCIN FROM A_SAP;
 OUT Setup (Called Party Number) TO B;
 IN Connect FROM B;
 OUT ConnectAcknowledge TO B;
 SYNCOUT TO A_SAP;
 CONDITION c1;
ENDINSTANCE;
INSTANCE B:Test_Process;
 IN Setup (Called Party Number) FROM B_SAP;
 OUT Connect TO B_SAP;
```

```
$TestStepLibrary
$Begin_TestStep
$TestStepId PREAMBLE_SCT
$TestStepRef
$Objective /* */
$DefaultsRef DefBeh
$Comment /* */
$BehaviourDescription
$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
$Cref
$VerdictId
```

Figure 29: MSC/PR and TTCN/MP

7 Outlook

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 considered to be 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 started to extend the MSC editor by a graphical interface for message constraints. The user should be enabled to check, define and modify the message constraints without knowledge of the underlying reference mechanism.

Acknowledgements

The elaboration of this report was funded by the KWF-Project No. 2555.1 *'Graphical Methods in the Test Process'*. The author would like to thank Beat Brändle, Dieter Hogrefe, Roger Schönberger, and Charles Zehnder for supporting this work, and Iwan Nussbaumer, Christoph Rüfenacht, Andreas Spichiger, and Stefan Suter for valuable comments and suggestions.

References

[1] Abstract Conformance Test Development Team for Layers 2/3 (ACT 23). LAPD Conformance Testing Abstract Test Suite. North American ISDN Users Forum, February 1990.

[2] Alcatel Network Systems. Alcatel 8650 - Conformance Test System - GSM. Product Information, Alcatel STR AG (Zürich), 1993.

[3] 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.

[4] CCITT. Recommendations X.208 (ISO/IEC IS 8824 and 8824/AD1): Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1) and Addenum 1: ASN.1 Extensions. The International Telegraph and Telephone Consultative Committee (CCITT), Geneva, November 1989.

[5] CCITT. Recommendations X.209 (ISO/IEC IS 8825 and 8825/AD1): Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) and Addenum 1: ASN.1 Extensions. The International Telegraph and Telephone Consultative Committee (CCITT), Geneva, November 1989.

[6] J. De Man. Towards a Formal Semantics of Message Sequence Charts. In O. Faergemand and A. Sarma, editors, *SDL'93 - Using Objects*. North-Holland, October 1993.

[7] ETSI SPS5. Integrated Services Digital Network (ISDN); Digital Subscriber Signal No. 1 (DSS1), Abstract Test Suite (ATS) for User of Data Link Layer Protocol for General Application. Draft prI-ETS 300 313, Reference: DE/SPS-5001, European Telecommunications Standards Institute, February 1993.

[8] 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.

[9] J. Grabowski, D. Hogrefe, and R. Nahm. Test Case Generation with Test Purpose Specification by MSCs. In O. Faergemand and A. Sarma, editors, *SDL'93 - Using Objects*. North-Holland, October 1993.

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

[11] 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.

[12] ISO/IEC TC97/SC21. Basic Reference Model. International Standard 7498, ISO/IEC, 1984.

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

[14] 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.

[15] Ch. Rüfenacht. Anreicherung von MSCs mit Dateninformationen zur Testfallspezifikation. Diploma Thesis, University of Berne, Institute for Informatics, February 1994.

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

[17] D. Steedman. *Abstract Syntax Notation One (ASN.1): The Tutorial and Reference.* Technology Appraisals, 1990.

[18] S. Suter. Die Erzeugung des dynamischen Teils von TTCN aus MSCs. Diploma Thesis, University of Berne, Institute for Informatics, January 1994.

[19] TeleLOGIC Malmö AB, Box 4128, S-203 12 Malmö (Sweden). *SDT 2.3*, 1993.

[20] D. Toggweiler. TTCN-Testfallgenerierung für mit Sequence Charts spezifizierte verteilte Systeme. Diploma thesis, University of Berne, March 1992.