Test Case Specification Based on MSCs and ASN.1

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

[a]University of Berne, Institut for Informatics, Neubrückstr. 10, 3012 Berne, Switzerland

[b]Siemens-Albis AG, Öffentl. Vermittlungssysteme, Steinenschanze 2, 4051 Basel, Switzerland

Informal test specifications are formalized by means of MSCs. Message definitions and constraints are included. For this purpose a new concept for the reference and modification of constraints is introduced. The formalized test specifications can be implemented automatically. Our approach is explained by means of a test case for a layer 3 ISDN protocol (ITU-T Rec. Q.931). The method is implemented in a set of prototype tools.

1. Introduction

For the specification and implementation of test cases for a telecommunication product certain tasks have to be carried out. The whole procedure is shown in Fig. 1. Within the figure, the rectangles represent tasks and the ellipses describe data or documents which serve as input for, or are produced by the different tasks.

All three tasks are based on international standards, e.g. ITU-T recommendations, and additional customer or country specific requirements, e.g. PTT requirement catalogs. Since these documents mainly are written in plain text, all tasks have to be carried out manually by protocol specialists.

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.[1] A test case specification shall be independent from the concrete implementation and the test equipment. Fig. 3 shows such a test case specification for a layer 3 protocol of an ISDN[2] switching system. Later on we will come back to this example.

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[3] and message parameters. For the parameters often *default constraints* exist. Such a *constraint* may define a concrete 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 test equipment.

---

[1]Country specific requirements may for example be caused by the currency. A Swiss tax counter may count in 10 Rappen units and a German one may use 10 Pfennig units.

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

[3]According to the OSI basic reference model [7] protocol entities exchange *protocol data units* (PDUs) and *abstract service primitives* (ASPs). Since this paper does not treat the OSI model, we use the more general term *message*. But the abbreviation ASP will occur in several data descriptions.

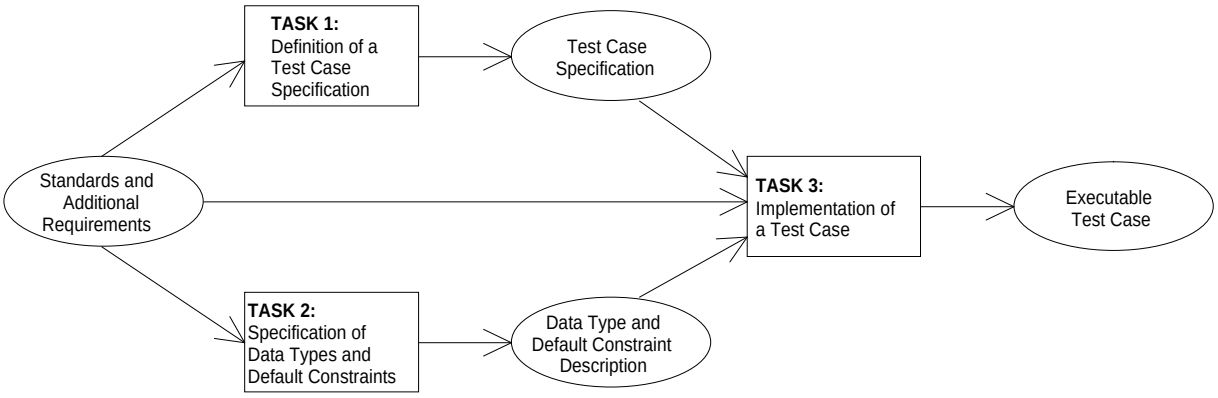


Figure 1. The specification and implementation of test cases

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 when the test case is executed. 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.

All tasks in Fig. 1 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 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 behaviour and the specification of the exchanged data units.

For the behaviour description the use of standardized formal description techniques, i.e. SDL [2], can help to avoid ambiguities and misinterpretations. Unfortunately, the behaviour descriptions in existing protocol standards are written mainly in plain text which is enriched by informal drawings.

For the data description the situation is more promising. The data description language ASN.1 [16] 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 [17]. For our application example (cf. Section 2) which is based on the ITU-T Recommendation Q.931 [11] the situation is not so good. Q.931 includes no formal data descriptions. Therefore the ASN.1 definitions in this paper are produced by hand.

Due to the mentioned problems, the Tasks 1 and 3 in Fig. 1 are the most critical parts in the test case specification and implementation procedure. Task 2 can be 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.

Task 1 is based completely on informal documents. It cannot be automated without improving the quality of the standards. This might be a goal for the future, but our current work is based mainly on already existing standards.

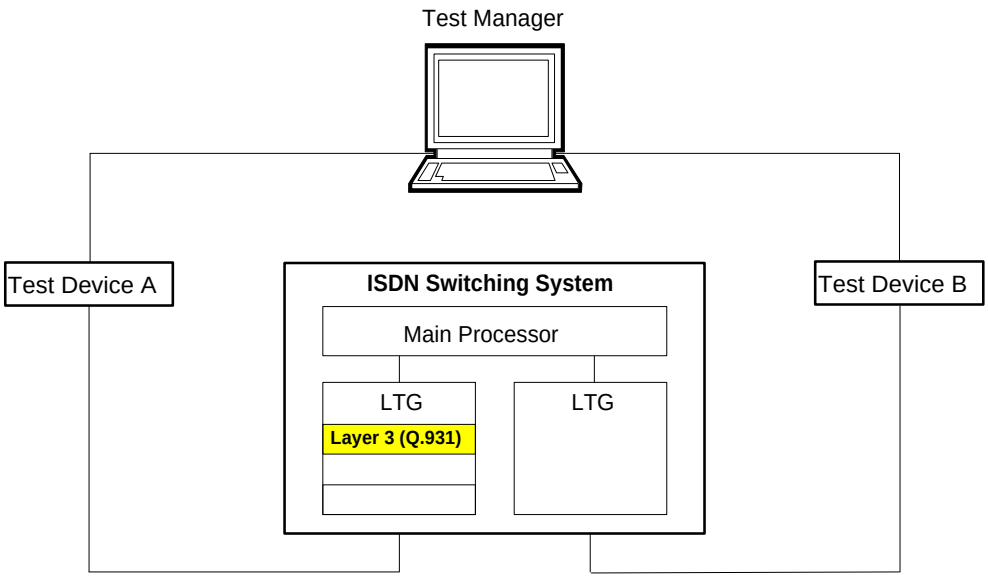


Figure 2. An ISDN test environment

It is our aim is to improve the implementation of a test case (Task 3). This is done by formalizing the test case specification (Task 1) and by avoiding the direct influence of the Standards and Additional Requirements on Task 3. The information of the Standards and Additional requirements go exclusively into Task 1.

The feasibility of this approach was shown in an application to the GSM[4] standard. The test case specifications (Task 1) were formulated in a machine processable form which is very close to the example given in Fig. 3. Tools were developed to interpret these specifications and translate them into executable code. With few additions this code could directly be used to control the test devices.

Experience from this GSM application showed that the overall effort on the way to executable test cases decreased, although specific tools had to be developed for the interpretation of the formalized test case specifications. But it also showed that a more general approach is needed to perform the same procedure on other applications. In particular the tools were of a rapid prototype nature, specifically created for the GSM application. The objective of the cooperation between the University of Bern and Siemens-Albis AG was to define a generally applicable method for the automation of Task 3. The method will be presented in this article. The description will focus on the data part of the method.

2. An application example

In this section we present an application example on which we base the description of our method for the automatic generation of executable test cases.

2.1. A test environment for an ISDN system

One of our applications is the test a *layer 3 protocol* within a *Line Trunk Group* (LTG) of an ISDN switching system (cf. Fig. 2). The protocol is given by the ITU-T Recommenda-

[4]GSM is an abbreviation for *'Groupe Speciale Mobile'*.

tion Q.931 [11]. The Q.931 protocol is implemented within the LTG and there is no direct access to this implementation. Each LTG has only standardized interfaces which may be connected with a telephone. The interface of an LTG to the main processor is proprietary and not standardized. A possibility to test the Q.931 protocol is to use the whole ISDN system as test environment.[5] Fig. 2 shows such a test system. The test devices access the LTG only via the standardized interfaces. The devices are controlled by a test manager which also records the test results.

This test method allows to use the same test cases for testing the LTG alone and for testing the integration of the LTG in the ISDN switching system. The whole procedure is also in line with the standardized *'Conformance Testing Methodology and Framework'* [8]. The use of standardized interfaces is a prerequisite to guarantee the interworking of telecommunication products from several manufacturers.

2.2. The informal specification of test cases

Fig. 3 presents an example of a test case specification for the test environment shown in Fig. 2. The test case specification is given by informal diagrams and plain text. 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. We also like to mention that there exist more than 1700 test case specification in the same form which only test the Swiss specific adaptations of the LTG. They are all specified and implemented by hand. The various possibilities to make errors force the need of methods for the automatic specification and implementation of test cases.

The test case specification in Fig. 3 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 test case name *EDSAOUX*.

There are several reasons for the popularity of test case descriptions as shown in Fig. 3. 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 gives an intuitive understanding of the described behaviour. 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.

3. Describing the test case behavior with MSCs

We identified the Message Sequence Chart (MSC) language to be adequate for our purposes. MSC is a graphical language, it is standardized by ITU-T [18], it has a formal semantics [19] and there exist tools which support the use of the language [5,14,15].

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

Test case identifier: EDSA0UX

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' ($0 \leq x \leq 9$).

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:

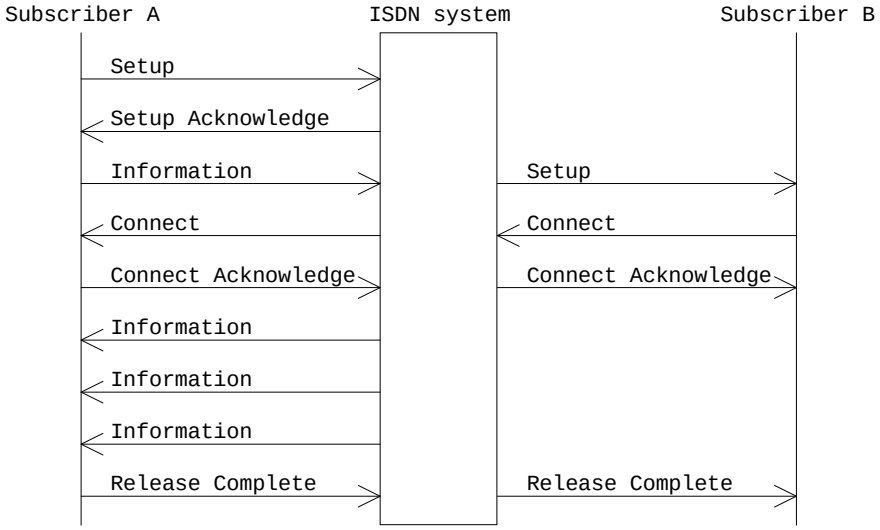


Figure 3. The specification of the test case *EDSAOUX*

3.1. Test case specification with MSC

The MSC standard Z.120 includes two syntactical forms: MSC/PR as pure textual and MSC/GR as graphical representation. An MSC[6] 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 present examples in the MSC/GR form.

Fig. 4 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

[6]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*.

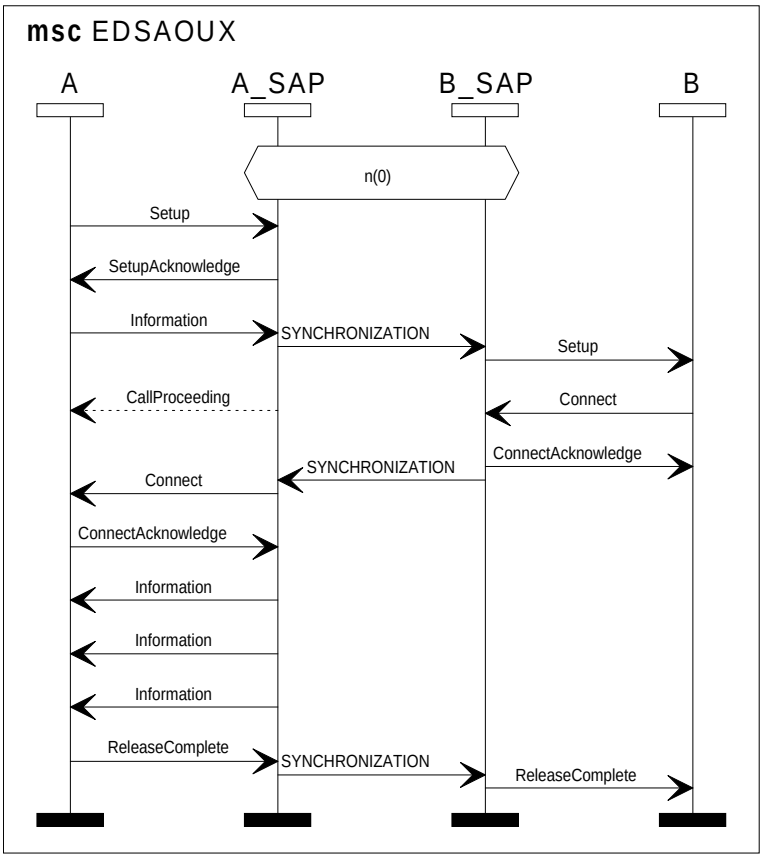


Figure 4. MSC describing the *general message flow* in Figure 3

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 ordered totally. 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 Fig. 4 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 [4].

The MSC in Fig. 4 describes the *general message flow* in Fig. 3. The two standardized service access points (SAPs) of the ISDN system and the two subscribers A and B are represented by the individual axes *A_SAP*, *B_SAP*, *A*, and *B*.

The initial state *n(0)* is not included in the *general message flow* of Fig. 3, but it is mentioned in the informal text. Since such preconditions might be relevant for test case implementation, it is added by means of a condition.

The *SYNCHRONIZATION* messages are also not part of the *general message flow* diagram. This is due to the fact that the protocol standard Q.931 states that there exist several dependencies between certain messages at the different SAPs. For test case implementation such information is relevant, because it might be necessary to synchronize

the test devices. In Fig. 1 the influence of the protocol standard on the test case implementation is described by an arrow. In order to automate the test case implementation this influence has to be suppressed. In our case this is done by adding information to the MSC test case specification. Therefore the dependencies between events on different instance axes are defined explicitly by the *SYNCHRONIZATION* messages.

The dashed arrow describes the optional message *CallProceeding*. *CallProceeding* may occur immediately after the *Information* message is sent by subscriber *A*. Whether the message occurs depends on the configuration of the whole ISDN system. Sometimes the tester has no influence on this configuration, although the possible arrival of *CallProceeding* has to be treated. The concept of optional messages is not part of the standardized MSC language. 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. We also introduced some further MSC constructs which facilitate the test case description. Like the optional message they can be seen as abbreviations for situations which are clumsy to express with the current MSC standard. A more detailed description of these extensions can be found in [6,14]

Obviously, the MSC in Fig. 4 offers the same intuitive understanding of the required system behaviour as the diagram in Fig. 3. The example also shows that it is possible to suppress the influence of standards on the test case implementation. 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. MSC and TTCN

The output of Task 3 is a set of executable test cases. For the automation of Task 3 we need a representation for executable test cases. Currently, most test case implementations are proprietary for a certain environment, e.g. the several manufacturers of test equipment use proprietary programming languages. However, the situation seems to change. The availability of TTCN[7] as standardized test case description language [9], existing and forthcoming standardized TTCN test suites, e.g. [10,3], and customer demands force manufacturers of test equipment to develop TTCN compiler or interpreter, e.g. [1,13]. Therefore we also have chosen TTCN for the representation of executable test cases.

A TTCN test case description can be divided into a static and a dynamic part. The static part includes the type and constraint definitions of the exchanged messages. The dynamic part defines the possible sequences of so-called *test events* which shall be performed by the test equipment when the test case is executed.

The event sequences are specified by means of a tree notation. Fig. 5 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. 6 and 15) 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 which should be sent

[7] TTCN is an abbreviation for *'Tree and Tabular Combined Notation'*.

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		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?Information	InformationEDSAOUX		
11		A?Information	InformationEDSAOUX		
12		A?Information	InformationEDSAOUX		
13		A!ReleaseComplete	ReleaseCompleteDefSend		
14		B?ReleaseComplete	ReleaseCompleteDefRec	PASS	
15		A?Connect	ConnectDefRec		
16		A!ConnectAcknowledge	ConnectAckDefSend		
17		A?Information	InformationEDSAOUX		
18		A?Information	InformationEDSAOUX		
19		A?Information	InformationEDSAOUX		
20		A!ReleaseComplete	ReleaseCompleteDefSend		
21		B?ReleaseComplete	ReleaseCompleteDefRec	PASS	

Detailed Comments :

Figure 5. TTCN test case EDSAOUX

or received. An example may clarify the notation. The statement *B?ReleaseComplete* denotes the reception of the message *ReleaseComplete* by the entity *B*.

The dynamic part of a TTCN test case can be computed automatically from an MSC. This is done be selecting the test events which shall be performed by the test devices within the MSC and by arranging them according to the time dependencies which are defined by the order along the instance axis or mediated by messages. The TTCN test case in Fig. 5 defines the test events of the test case EDSAOUX. It is generated from the MSC in Fig. 4. Although the algorithm seems to be simple, it should be noted that MSC and TTCN are different languages with different semantics. This has to be considered during the transformation. Since we want to focus on the data aspect of the test case description, we do not describe the details here. A more comprehensive discussion on the algorithms can be found in [6] and [14].

The TTCN table in Fig. 5 includes some further information. 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. The entries in the *Constraints Ref.* column refer to TTCN or ASN.1 constraints. In the following we describe a mechanism to generate constraint references automatically.

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 type definition *Information_type*

4. MSCs and data descriptions

In the previous section the test case specification with MSCs and the automatic generation of the dynamic part of a TTCN test case has been discussed. In order to gain complete TTCN test cases one of our major objective is to include the data description in the test case specification. For this purpose the MSCs have to be related to data type and constraint definitions. We distinguish between two kinds of constraints. *Default constraints* which are provided by the standards and country specific requirements and *test case specific constraints*. Test case specific constraints define value restrictions which are only valid in the context of the test case. They are often hidden in the purpose of the test case but have to be coded explicitly when the test case is implemented.

4.1. Data type and default constraint definitions

As described in Section 1 we can assume for Task 3 that data type and default constraint definitions are given in a form which can be interpreted by a machine[8]. 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* messages in the test case EDSAOUX (cf. Fig. 4) refers to the ASN.1 definition in Fig. 6. The test case checks a part of the *Display* parameter in the received information messages. The *Display* parameter has the type *Display_type*. The corresponding type definition is shown in Fig. 7.

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. The default constraint for the *Information* message is shown in Fig. 8. The constraint refers to the default constraints for the parameter values. Fig. 9 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

[8]In our example it is assumed to be in ASN.1.

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 7. ASN.1 type definition *Display_type*

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 8. 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 ('0?'H, '1?'H, '20'H), d_info ?}
Detailed Comments :

Figure 9. ASN.1 constraint *DisplayDefRec*

the type definition in Fig. 7 it is an arbitrary string of octets with a maximal length of 32 (in hexadecimal form '20'H). The format which should be checked by the test case EDSAOUX is encoded in *d_info*. *d_length* describes the length of *d_info* in form of a hexadecimal string. For the default constraints the names of the messages are sufficient to generate the corresponding entries of the TTCN test case description.

4.2. 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

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 10. ASN.1 constraint *InformationEDSAOUX*

the test purpose can be proved, and test purposes often include constraints on message parameter values. Also the test purpose of EDSAOUX includes a test case specific value constraint. It requires to check the *Display* parameter format of the *Information* message. The format definition *'Fr. x.x0' ($0 \leq x \leq 9$)* (cf. Fig. 3) is a constraint on the 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, the standards and the additional requirements. We described this situation in Fig. 1. 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 test case specification. 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.

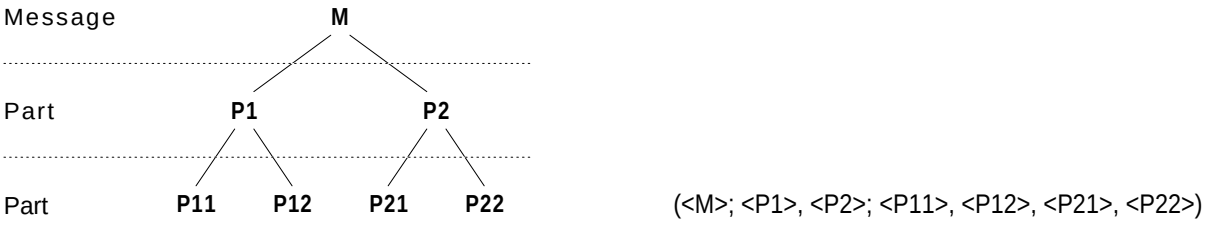
The first possibility is problematic, because the constraints may become too big for the MSC. For the test case EDSAOUX the ASN.1 constraint for the *Information* message is shown in Fig. 10. This constraint refers to another constraint which checks the format of the *Display* parameter (cf. Fig. 11). However, the constraints of the *Information* message comprise two pages, and they should be valid for each received *Information* message of the MSC in Fig. 4. 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. In the test case EDSAOUX, the test case specific constraint for the *Information* message would have be referred in the test case specification. The test purpose relevant constraint on the *Display* parameter itself would only be referred in the constraint of the *Information* message. Therefore there would be no direct indication of the test purpose in the test case specification.

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

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	: d_info describes the format: 'FR. x.x0' (0=<x=<9)

Figure 11. ASN.1 constraint *DisplayEDSAOUX*



(a) Message structure (b) Structure of an RL statement

Figure 12. RL statements and the message structure

constraint of the *Information* message are only different with respect to the *Display* parameter constraint (cf. Fig. 8, 10). In such a case it is more appropriate to specify the difference to a default constraint than to rewrite the whole default constraint.

4.3. 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, and which provides possibilities to define test case specific constraints by modifying existing constraints. Additionally, it allows to define test case specific constraints directly 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. Fig. 12). This is no extension of the MSC language, because the MSC standard [18] 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. Fig. 12 presents an example. The message *M* in (a) has the parameters *P1* and *P2*. *P1* is structured in *P11* and *P12*. *P2*

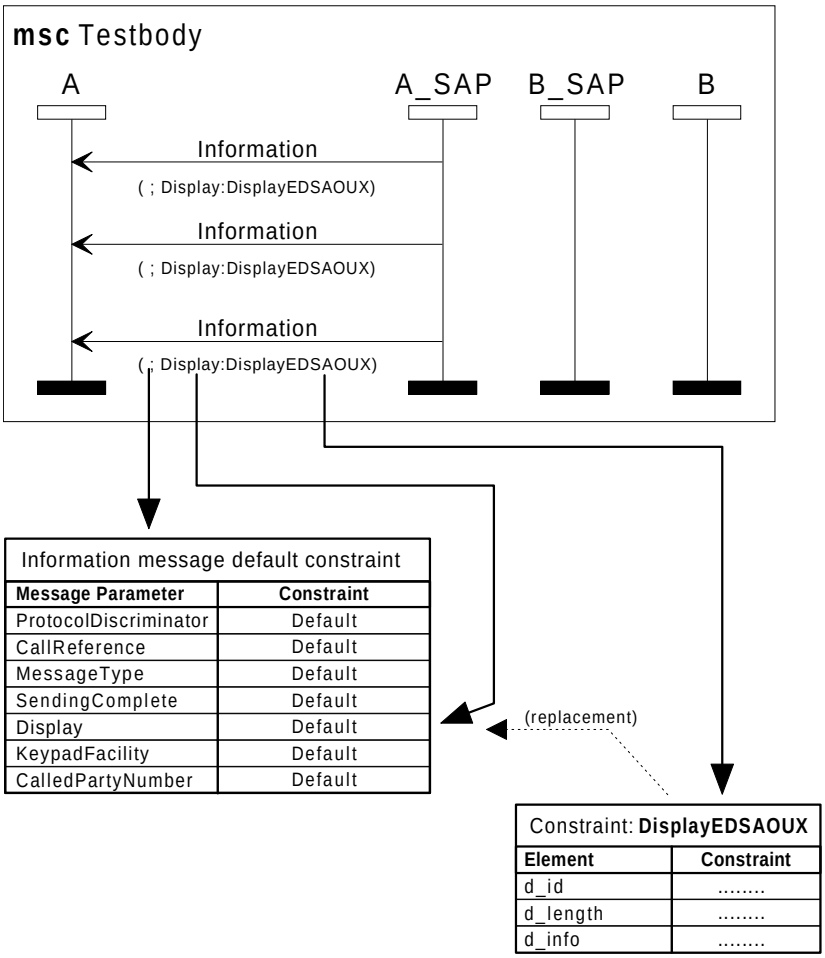


Figure 13. The reference mechanism for test case specific constraints

comprises *P21* and *P22*. The statement in (b) shows how the different elements of *M* can be referred within an RL statement. The statements indicate changes to the constraints for the corresponding message. Like this e.g. <*P1*> and <*P2*> change the constraint <*M*> and <*P11*> and <*P12*> change the constraint <*P1*>. In general, the omission of a part or subpart means that the corresponding constraint is not changed. Based on such an RL statement it is possible to automate the calculation of the references within the TTCN tables to test case specific constraints, and to generate test case specific constraints which are based on existing constraints. We will not give further details here, but an example shall give an impression of the reference mechanism. The details can be found in [12].

The MSC in Fig. 13 represents the test purpose of EDSAOUX. The inscription of the *Information* messages *(; Display:DisplayEDSAOUX)* states that 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*. Fig. 13 only indicates the replacement schematically. The concrete ASN.1 constraints have already been presented. The default constraints for the *Information* message and the *Display* parameter are shown in the Fig. 8 and Fig. 9. The test case specific constraint *DisplayEDSAOUX* is shown in Fig. 11. The test case specific constraint for the *Information* message which is automatically generated is given in Fig. 10.

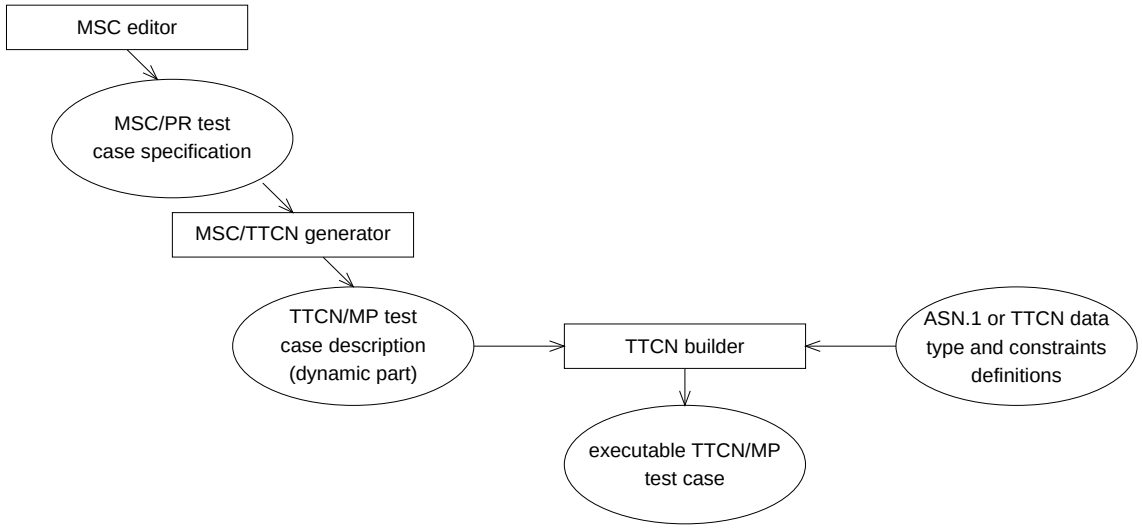


Figure 14. A set of prototype tools for test case specification and implementation

5. 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 generate TTCN test cases from MSC test case specifications. 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 Fig. 14[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 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

[9]The tools are represented by rectangles and the interfaces between them by ellipses.

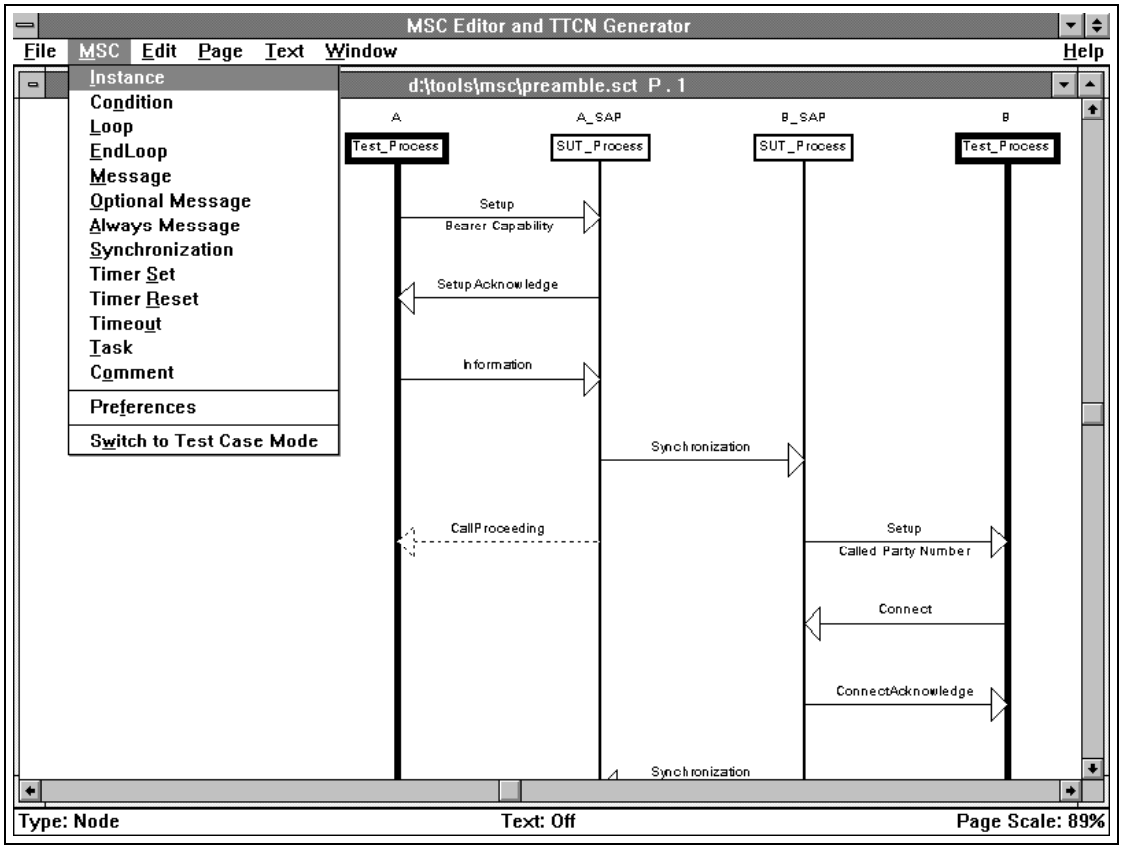


Figure 15. The user interface of the MSC editor

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 have been implemented on a PC in a Windows 3.1 environment. Fig. 15 presents the user interface of the MSC editor. The shown MSC is the first part of the test case EDSAOUX.

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

Acknowledgements

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

REFERENCES

1. Alcatel Network Systems. Alcatel 8650 - Conformance Test System - GSM. Product Information, Alcatel STR AG (Zürich), 1993.
2. F. Belina, D. Hogrefe, A. Sarma. SDL with Applications from Protocol Specification. The BCS Practitioner Series. Prentice Hall International, 1991.
3. ETSI SPS5. ISDN - DSS1: Abstract Test Suite for User of Data Link Layer Protocol for General Application. Draft prI-ETS 300 313, Ref.: DE/SPS-5001, ETSI, 1993.
4. J. Grabowski, P. Graubmann, E. Rudolph. The Standardization of Message Sequence Charts. Proceedings of the 'IEEE Software Engineering Standards Symposium 1993', Sept. 1993.
5. J. Grabowski, D. Hogrefe, R. Nahm. Test Case Generation with Test Purpose Specification by MSCs. In: SDL'93 - Using Objects. North-Holland, Oct. 1993.
6. J. Grabowski, D. Hogrefe, I. Nussbaumer, A. Spichiger. Improving the Quality of Test Suites for Conformance Tests by using Message Sequence Charts. Proceedings of the 'Fourth European Conference on Software Quality', Oct. 1994.
7. IS 7498 (1984). OSI - Basic Reference Model. International Standard, ISO/IEC, 1984.
8. IS 9646 (1992). Information Technology - OSI - Conformance Testing Methodology and Framework. International Multipart Standard, ISO/IEC, 1992.
9. IS 9646 Part 3 (1992). Information Technology - OSI - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation. International Standard 9646-3, ISO/IEC, 1992.
10. North American ISDN Users Forum (ACT23). LAPD Conformance Testing Abstract Test Suite. Feb. 1990.
11. Q.931 - Q.940 (1989). Digital Subscriber Signalling Systen No. 1 (DSS 1), Network Layer, User-Network Management. CCITT, 1989.
12. Ch. Rüfenacht. Extending MSCs with Data Information in order to Specify Test Cases. Diploma Thesis (written in German), University of Berne, Feb. 1994.
13. Siemens AG. Product Information K1197, K1103. Siemens AG Berlin, 1993.
14. S. Suter. The MSC Based Generation of the Dynamic Part of TTCN Test Cases. Diploma Thesis (written in German), University of Berne, Jan. 1994.
15. TeleLOGIC Malmö AB, Box 4128, S-203 12 Malmö (Sweden). SDT 2.3, 1993.
16. X.208 (1989). Information Processing Systems - OSI - Specification of Abstract Syntax Notation One (ASN.1) and Addenum 1: ASN.1 Extensions. CCITT, 1989.
17. X.209 (1989). Information Processing Systems - OSI - Specification of Basic Encoding Rules for ASN.1 and Addenum 1: ASN.1 Extensions. CCITT, 1989.
18. Z.120 (1993). Message Sequence Chart (MSC). ITU-T, Sept. 1994.
19. Z.120 B (1995). Message Sequence Chart Algebraic Semantics. ITU-T Publ. sched.: May 1995.