

## Conformance Testing with TTCN

By Ina Schieferdecker<sup>1</sup> and Jens Grabowski<sup>2</sup>

<sup>1</sup>GMD FOKUS Berlin, Germany, <sup>2</sup>Medical-Technical University of Lübeck, Germany

Published in Teletronikk (ISSN 0085-7130), 96 (4), 2000, pp. 85-95

Available also via

<http://www.telenor.no/fou/english/publisering/teletronikk.shtml>

### Abstract

The Tree and Tabular Combined Notation (TTCN) is a semi-formal notation which supports the specification of abstract test suites for protocol conformance testing. An abstract test suite is a collection of abstract test cases. As indicated by the name TTCN, test cases are described in form of behavior trees and different kinds of tables are used for the graphical representation of test suites.

*"Product testing is still seen as the only reliable way to assure that outsourced products meet the required specification and are suitable for inclusion in the live network."* Cited from Counting on IT, Issue 7 by National Physical Laboratory, UK, Summer 1998.

*This paper describes basic concepts of conformance testing and allows you to compare the previous versions of TTCN, i.e. TTCN-2, with the most recent 3<sup>rd</sup> edition of TTCN, i.e. TTCN-3, which Testing Technologies is promoting and deploying to the market.*

# Conformance Testing with TTCN

INA SCHIEFERDECKER AND JENS GRABOWSKI



Ina Schieferdecker (33) received her PhD from the Technical University Berlin in 1994. Since 1993 she has been a researcher at GMD Fokus, and a lecturer at Technical University Berlin since 1995. Her interests cover testing methods for distributed systems and formal methods for the design, validation and prototyping of distributed systems. She has been head of the Competence Center for Testing, Interoperability and Performance (TIP) since 1997 and is actively involved in several testing projects. She has published several papers on testing telecommunications systems and developing test systems, and is involved in the definition of MSC in ITU-T SG10 and of TTCN-3 in ETSI.

Schieferdecker@fokus.gmd.de



Jens Grabowski (38) graduated from the University of Hamburg with a diploma degree in Computer Science and Chemistry. He spent two years at SIEMENS AG in Munich focusing on protocol specification and protocol validation based on Petri Nets, SDL and MSC. 1990–1995 he was research scientist at the University of Berne, where he received his PhD in 1994. Since 1995 Grabowski has been researcher and lecturer at the Institute for Telematics at the Medical University in Lübeck; since 1996 he has also worked as expert in several ETSI standardization projects. He is a member of the ETSI experts team which develops the third edition of TTCN.

jens@itm.mu-luebeck.de

The Tree and Tabular Combined Notation (TTCN) is a semi-formal notation which supports the specification of abstract test suites for protocol conformance testing. An abstract test suite is a collection of abstract test cases<sup>1)</sup>. As indicated by the name TTCN, test cases are described in the form of behavior trees and different kinds of tables are used for the graphical representation of test suites.

*“Product testing is still seen as the only reliable way to assure that outsourced products meet the required specification and are suitable for inclusion in the live network.”*

Cited from Counting on IT, Issue 7 by National Physical Laboratory, UK, Summer 1998.

## 1 Introduction

TTCN [3] is the means of the Conformance Testing Methodology and Framework (CTMF) for the description of test suites for conformance testing. See terminology and explanations in Box 1. TTCN has two syntactical forms (Figure

1), called TTCN/gr (TTCN GRaphical form) and TTCN/mp (TTCN Machine Processable form). TTCN/gr is intended to be used by humans and TTCN/mp is developed for the exchange of documents between different computers and for further processing of TTCN test suites. A TTCN/gr description can be translated into an equivalent TTCN/mp representation and vice versa. In this paper only TTCN/gr examples are presented.

In the following, the different TTCN constructs are described by developing an example test suite<sup>2)</sup>. The system to be tested is a parcel service. A test case should check whether the parcel

Test Step Dynamic Behaviour					
<b>Test Step Name</b> : Preamble					
<b>Group</b> :					
<b>Objective</b> : To bring the SUT into the initial state					
<b>Default</b> :					
<b>Comments</b> :					
<b>Description</b> :					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		(PS_Init:= Reset_ParcelService())			
2		[PS_Init]		(P)	
3		[NOT PS_Init]		I	
<b>Detailed Comments:</b>					

```

...
$Begin_TestStep
$TestStepId Preamble
$TestStepRef Example_ATS/
$Objective /* To bring the SUT into the initial state */
$DefaultsRef
$BehaviourDescription
$BehaviourLine
$LabelId
$Line [0] (PS_Init:= Reset_ParcelService())
$Cref
$VerdictId
$End_BehaviourLine
$End_BehaviourDescription
$End_TestStep
$LabelId
$Line [1] [PS_Init]
$Cref
$VerdictId (P)
$End_BehaviourLine
$BehaviourLine
$LabelId
$Line [1] [NOT PS_Init]
$Cref
$VerdictId I
$End_BehaviourLine
$End_BehaviourDescription
$End_TestStep

```

Figure 1 The TTCN forms: TTCN/gr and corresponding TTCN/mp code below the table

<sup>1)</sup> CTMF and TTCN use the terms abstract and executable to distinguish between implementation-independent and implementation-dependent concepts, e.g. abstract test suite and executable test suite, abstract test case and executable test case or abstract service primitive. This paper introduces mainly implementation-independent CTMF and TTCN concepts. Qualifiers like abstract or executable will only be used in case of ambiguities.

## Conformance Testing Framework

Testing a system is performed in order to assess its quality and to find errors. An error is considered to be a discrepancy between observed or measured values provided by the system under test and the specified or theoretically correct values. Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements. It approves a quality level of a tested system.

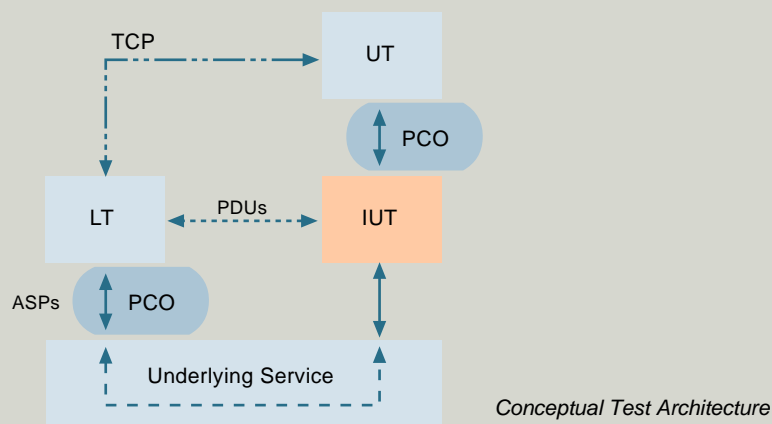
Conformance testing in particular is the process of testing the extent to which implementations of OSI protocol entities adhere to the requirements stated in the relevant standard or specification. Conformance testing is functional black-box testing. The term functional refers to the correct functional behavior of an Implementation Under Test (IUT), i.e. the correct input/output behavior in each state. Black-box testing means that the internal structure of the IUT remains hidden, i.e. it is a black box for the test developer.

The OSI conformance testing procedure is defined in the international ISO/IEC standard 9646 Conformance Testing Methodology and Framework (CTMF) [1]. CTMF consists of seven parts and covers the following aspects: concepts (part 1), test suite specification and test system architectures (part 2), test notation (part 3), test realization (part 4), means of testing and organizational

aspects (part 5, 6, and 7). The Tree and Tabular Combined Notation (TTCN) is defined in part 3 of CTMF. By definition, the target systems to be tested according to the CTMF principles are implementations of OSI protocol entities. However, CTMF and TTCN are applicable in a much wider scope than OSI-based systems. The CTMF principles and TTCN have also been applied successfully for conformance testing of ODP-, TINA-, CORBA- and IP-based systems, APIs and reactive systems in general<sup>3)</sup>.

In conformance testing, the IUT is an implementation of an OSI protocol entity. The IUT is part of an open system called System Under Test (SUT). The conceptual conformance test architecture is shown in the figure below.

The IUT has an upper and a lower interface through which it is tested. Conformance testing is done at standardized interfaces called Points of Control and Observation (PCOs)<sup>4)</sup>. Typically, the lower interface of an IUT is accessible only from remote. Therefore, the underlying service of the IUT is used to define an appropriate PCO on a remote site, i.e. the lower interface of the IUT is moved to the remote site. Communication is always meant to be asynchronous and therefore, a PCO is modeled by two FIFO queues, i.e. one queue for each direction.



CTMF distinguishes between an Upper Tester function (UT) and a Lower Tester function (LT). As indicated by the names, the upper interface of the IUT is controlled by the UT and the lower interface is controlled by the LT. During the test, the UT plays the role of a user that makes use of the service provided by the IUT and the LT plays the role of a peer entity of the IUT, i.e. the LT and the IUT communicate in order to provide the service to the UT.

IUT and UT communicate by means of Abstract Service Primitives (ASPs). Conceptually, IUT and LT provide their service by exchanging Protocol Data Units (PDUs). In practice, the PDUs are encoded in ASPs of the underlying service, i.e. PDUs will not be exchanged directly. However, CTMF allows to abstract from the

encoding of PDUs, i.e. allows to specify the exchange of PDUs in abstract test cases. Therefore, it is not necessary to distinguish between ASP and PDU explicitly, and hence, only the term PDU is used.

As shown in the figure, Test Coordination Procedures (TCP) can be used to coordinate the actions of LT and UT. This might be necessary if LT and UT are realized in separate tester processes. The figure presents the conceptual test architecture only. In practice, several variations of the conceptual test architecture are used. The test methods defined in CTMF are local, distributed, coordinated and remote test method. They differ in the possibilities to coordinate LT and UT and the ability to control and observe

<sup>2)</sup> Only a few TTCN tables can be presented in this paper, but the complete example test suite is available from the authors.

<sup>3)</sup> An overview of the use of conformance testing and TTCN is given in [9].

<sup>4)</sup> In most cases, a PCO maps to a Service Access Point (SAP) in the OSI basic reference model.

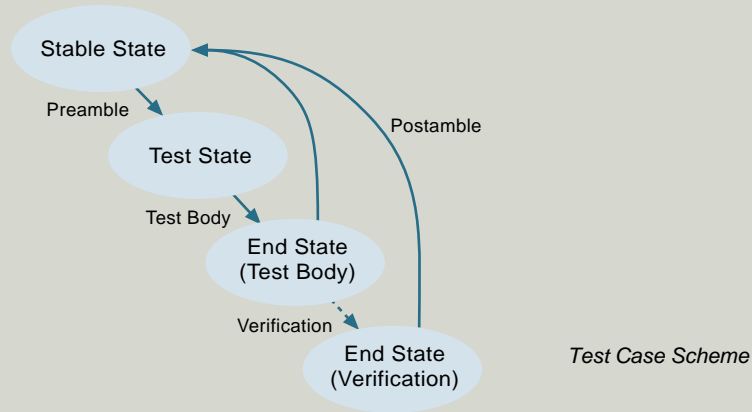
## Conformance Testing Framework, continued

the IUT. In addition, CTMF defines a multi-party context which allows to combine the different test architectures in order to specify tests with several UT and LT processes.

The test case development starts with the identification of test purposes. A test purpose is a prose description of a single requirement or a set of related requirements which should be tested. Test

purposes are identified based on the requirements in the specification of the IUT.

A test case is the implementation of a test purpose for a particular test architecture, i.e. a complete specification of the actions required to achieve a specific test purpose. The definition of a test case follows the schema shown below.



A test case starts and ends in stable testing states, which need not to be identical. It consists of a preamble, a test body, an optional verification step and a postamble. With the preamble, the IUT is driven from a stable testing state to the test state from which the test body is performed in order to check the test purpose. If the end state of the test body is not unique, it has to be checked by a verification step and then a postamble is used to drive the IUT into a stable testing state again. Otherwise, the IUT is put into a stable testing state immediately with the postamble.

Test cases developed according to the principles of CTMF are abstract. Executable test cases are derived from abstract ones by compilation and adaptation to the Means of Testing (MoT). The MoT is the combination of equipment and procedures that can perform the derivation, selection, parameterization and execution of test cases. It consists typically of dedicated test devices and facilities for the coordination of test devices and the observation of the IUT. These facilities may be installed inside the SUT.

service behaves as shown in Figure 2. A producer asks for a service offer and the parcel service indicates within a certain time frame that a 24h delivery service is available. Then, the goods are sent to the parcel service, which delivers them to the consumer. The consumer accepts the goods by sending an acknowledgement to the parcel service. The acknowledgement is forwarded as a confirmation to the producer. The confirmation is expected within 24h in accordance with the service assured by the parcel service. Not shown in Figure 2 is the possibility of the parcel service promoting new services to the producer by sending advertisements at any time.

The test architecture for testing the parcel service is shown in Figure 3. The IUT is the parcel service which is connected to the LT functions *Consumer* and *Producer* through the PCOs *LT\_Cons* and *LT\_Prod*. This test architecture can be seen as a combination of two remote test methods in a multi-party context, i.e. a special variant of the conceptual test architecture de-

scribed in Box 1. In this section, a non-concurrent test case will be developed, i.e. the behaviour of both LT functions will be implemented in one test component which controls and observes both PCOs.

## 2 Basics of TTCN

A TTCN test suite is composed of four parts: an overview part (Section 2.1), a declarations part (Section 2.2), a constraints part (Section 2.3) and a dynamic part (Section 2.4).

### 2.1 Overview Part and Test Suite Structure

The overview part of a TTCN test suite can be seen as a table of contents and provides all information needed for the general presentation and understanding of the test suite. It defines the test suite name and test architecture, describes the test suite structure, provides references to additional documents related to the test procedure and includes indexes for the test cases, test steps and default behaviour descriptions<sup>5)</sup>.

<sup>5)</sup> The meaning of test steps and default behaviour descriptions will be explained in Section 2.4.

Figure 2 Behaviour of a Parcel Service

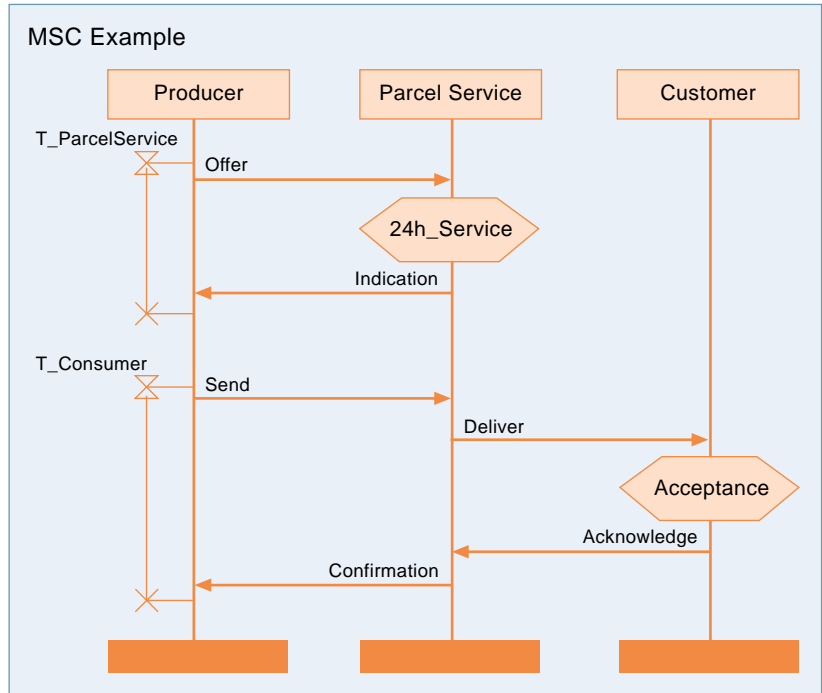
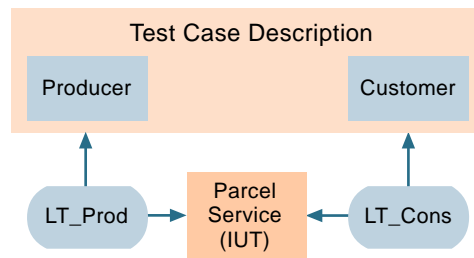


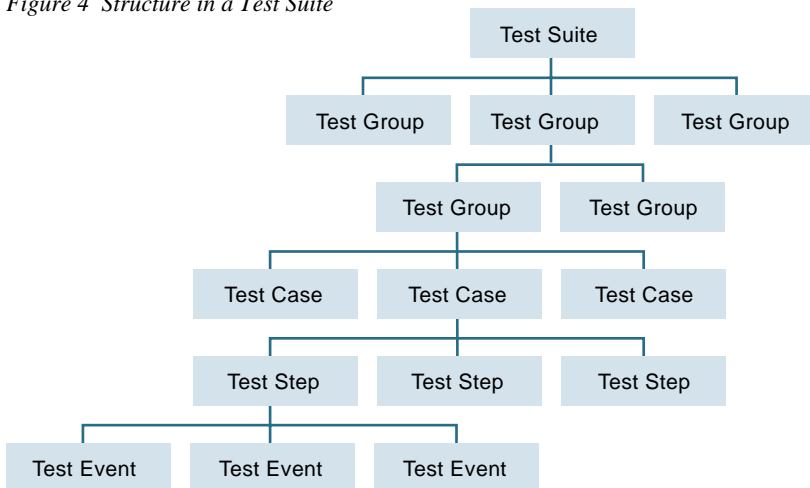
Figure 3 Test Architecture for the Parcel Service



The documents related to the test procedure are the specification on which the test suite is based, a PICS (Protocol Implementation Conformance Statement) document and a PIXIT (Protocol Implementation eXtra Information for Testing) document. In most cases, the referenced specification is a protocol standard. The PICS document is a questionnaire on mandatory and optional features of the IUT and the PIXIT document is a questionnaire on additional information required for the test execution such as address and timer information.

The different elements of a TTCN test suite appear in a predefined strict order. Only in the dynamic part it is possible to define a logical structure for test cases, test steps and default behaviour descriptions by putting them into groups and subgroups (Figure 4). Test events are the smallest elements and are explained in Section 2.4.2. The test suite for the parcel service example contains only one test group, which is specified in Figure 5.

Figure 4 Structure in a Test Suite



## 2.2 The Declarations Part

The declarations part provides definitions and declarations used and referenced in the subsequent parts of the test suite. Specifically, the declarations part defines and declares types, operations, selection expressions, test components, PCOs, timers, variables, constants and the encoding of ASPs and PDUs. In the following, the data types are explained and examples of operation definitions, test suite parameter declarations, variable declarations, timer declarations and PDU type definitions are given.

### 2.2.1 Data Types

TTCN has its own data type system and allows the usage and definition of ASN.1 data types.<sup>6)</sup> The TTCN type system includes the predefined data types INTEGER, BOOLEAN, BITSTRING, HEXSTRING, OCTETSTRING, and various character strings such as IA5String, Numeric-String and PrintableString. In addition, TTCN allows to define structured types which are comparable to C structures or ASN.1 sequences. For the usage of ASN.1, TTCN provides special tables, which include pure ASN.1 code.

### 2.2.2 Test Suite Operations

Test suite operations are comparable to functions in common programming languages like C or Pascal. They can be used to encapsulate any functionality relevant for the test execution such as setting up basic connections, resetting the IUT or just calculating a specific value. An example for the definition of a test suite operation is shown in Figure 6. The table header contains the name of the operation, a description of the input parameters and the type of the result. The table body includes the behaviour specification of the operation which may either be given in the form of a pseudo-code like procedural definition language or in the form of an informal textual description. For simplicity, the behaviour specification of the operation in Figure 6 has been omitted.

### 2.2.3 Test Suite Parameters

Test suite parameters are global parameters of the test suite. Typically, they are derived from the PICS and PIXIT documents and are constant during test execution. Test suite parameters serve as a basis for test case selection and for the parameterization of test cases. The declaration of the test suite parameter *Duration\_T\_ParcelService* of the parcel service example is shown in Figure 7. The parameter is used to set the duration of timer *T\_ParcelService* in Figure 9.

### 2.2.4 Variables

TTCN supports two types of variables: test suite variables and test case variables. Test suite variables are defined globally and retain their values throughout the whole test campaign. They are used to pass information from one test case to another. Test case variables are also declared globally, but their scope is local to a test case. Each test case receives a fresh copy of all test case variables when it is started. The declaration of the test case variable *PS\_Init* with the initial value *TRUE* is shown in Figure 8. In Figure 1, *PS\_Init* is used to store the result of the test suite operation *Reset\_ParcelService* given in Figure 6.

Test Suite Structure			
<b>Suite Name</b> : Example_ATS			
<b>Standards Ref</b> : None			
<b>PICS Ref</b> : None			
<b>PIXIT Ref</b> : None			
<b>Test Method(s)</b> : Remote Test Method			
<b>Comments</b> : This is an ATS for explaining selected TTCN constructs.			
Test Group Reference	Selection Ref	Test Group Objective	Page Nr
Valid/		Test the valid behaviour of the Parcel Service.	18
<b>Detailed Comments:</b>			

Figure 5 Test Suite Structure

Test Suite Operation Definition	
<b>Operation Name</b>	: Reset_ParcelService
<b>Result type</b>	: BOOLEAN
<b>Comments</b>	: This is used to initialize the Parcel Service, e.g. the storage capacity should be reset.
Description	
...	
<b>Detailed Comments:</b>	

Figure 6 Definition of a Test Suite Operation

Test Suite Parameter Declarations			
Parameter name	Type	PICS/PIXIT Ref	Comments
Duration_T_ParcelService	INTEGER		This is the timeout period for the timer to watchdog the indication process of the Parcel Service.
<b>Detailed Comments:</b>			

Figure 7 Declaration of a Test Suite Parameter

Test Case Variable Declaration			
Variable Name	Type	Value	Comments
PS_Init	BOOLEAN	TRUE	This is to store the result of the Reset_ParcelService TSO.
<b>Detailed Comments:</b>			

Figure 8 Declaration of a Test Case Variable

<sup>6)</sup> For further information on ASN.1, please refer to [7] and to the paper on ASN.1 in this issue of the journal.

Timer Declaration			
Timer Name	Duration	Unit	Comments
T_ParcelService	Duration_T_ParcelService	min	Timer between Offer and Indication
T_Consumer	24*60	min	Timer between Send and Confirmation
Detailed Comments:			

Figure 9 Timer Declaration

ASN.1 PDU Type Definition	
<b>PDU Name</b>	: Offer
<b>PCO Type</b>	: LT_PCO
<b>Encoding Rule Name</b>	:
<b>Encoding Variation</b>	:
<b>Comments</b>	: Ask for an offer to deliver certain goods by the Parcel Service
Type Definition	
SEQUENCE {type_of_good OBJECT IDENTIFIER, amount_of_good INTEGER}	
Detailed Comments:	

Figure 10 A PDU Type Declaration

ASN.1 PDU Constraint Declaration	
<b>Constraint Name</b>	: Offer_Large
<b>PDU Type</b>	: Offer
<b>Derivation Path</b>	:
<b>Encoding Rule Name</b>	:
<b>Encoding Variation</b>	:
<b>Comments</b>	: Sending a question for an offer to the Parcel Service
Constraint Value	
{type_of_good {1 2 3 4 5}, amount_of_good 100}	
Detailed Comments:	

Figure 11 Sending Constraint for an Offer PDU

### 2.2.5 Timer Declarations

As shown in Figure 9, timers are declared with their name, an optional default duration and a timeout period in the range from pico second (ps) up to minute (min). The two timers *T\_ParcelService* and *T\_Consumer* of the parcel service example (Figure 4) are declared in Figure 9. The default duration of *T\_ParcelService*

is set to the test suite parameter *Duration\_T\_ParcelService* (Figure 7). The default duration of *T\_Consumer* is given by an expression of type INTEGER reflecting the  $24h=24*60min$  duration for the 24h service.

### 2.2.6 PDU Definitions

Instances of PDU types are (either directly or embedded in ASPs) sent to or received from the IUT at PCOs. As presented in Figure 10, the definition of a PDU type consists of its name, the PCO type associated with the PDU type, and a list of PDU fields. Each PDU field is defined by its name and its type. The encoding of PDU fields follows the relevant protocol specification unless encoding information is included in the test suite. Figure 10 defines the *Offer* PDU of the parcel service example (Figure 2). The definition is given in the form of an ASN.1 PDU type definition and shows the usage of ASN.1 in TTCN. The table body includes a pure ASN.1 type definition. The Offer PDU type uses a unique object identifier referring to the type of goods and an INTEGER value referring to the amount of goods that should be delivered.

### 2.3 The Constraints Part

The constraints part of a TTCN test suite provides the values of the PDUs (and ASPs) to be sent to or received from the IUT. This is done by means of PDU (and ASP) constraints. A PDU constraint is related to a PDU type and describes a concrete value or value ranges of the PDU type. Constraints are referenced in the dynamic part of a test suite (Section 2.4) in order to describe the PDU exchange in the different test cases. A constraint specification will follow the structure of the corresponding PDU type and can be specified either in tabular form or in the form of the ASN.1 value notation.

A constraint for a PDU which should be sent to the IUT, has to provide concrete values for all PDU fields. An example constraint for the *Offer* PDU type (Figure 10) is presented in Figure 11. It defines the value of the *type\_of\_good* field to be {1 2 3 4 5} and the value of the *amount\_of\_good* field to be 100.

A constraint for a PDU that is received from the IUT may define value ranges for the PDU fields. TTCN provides powerful matching mechanisms to specify specific values (if concrete values are expected) and to specify value ranges (if several values are expected). Value ranges can be specified by referring to any value of a given type, by listing specific values, by complementing specific values or by providing value patterns. Value patterns are described by using wildcards such as '\*', '-' or '?'.

An example for a constraint of the *Indication* PDU which should be received from the IUT of the parcel service example is shown in Figure 12. The *Indication* PDU which matches the constraint must have information about the delivered goods, i.e. the kind and amount of goods must be identical to the information contained in the preceding *Offer* PDU (Figure 11). In addition, an order number and an indication of the 24h service has to be received, but these can have any values (indicated by '?'). The comment field can be omitted or have any value (indicated by '\*').

## 2.4 The Dynamic Part

The dynamic part describes the dynamic behaviour of the tester processes by test cases, test steps and default behaviour descriptions.

### 2.4.1 Test Cases, Test Steps and Default Behaviour Descriptions

A test case is a complete program, which has to be executed in order to judge whether a test purpose (cf. Box 1) is fulfilled or not. Test cases can be structured into test steps and default behaviour descriptions.

A test step can be seen as a procedure definition which can be called in test cases by means of an ATTACH operation. Figure 13 presents a TTCN test case description. In lines 1 and 4 of the table body, the test steps *Preamble* and *Postamble* are attached to the test case behaviour. The corresponding TTCN specifications can be found in Figure 1 and Figure 14.

A default behaviour description is a special test step and copes with exceptional test situations where the IUT does not behave in an expected manner. In contrast to a test step, a default behaviour description is not used inside a test case or test step behaviour description. Instead, it is referenced in the table header. Figure 15 presents the default behaviour description *OtherwiseFail*. This is the default for the parcel service example and referenced by the test case shown in Figure 13.

The specification of the test behaviour is identical for test cases, test steps and default behaviour descriptions and can be found in the body of the corresponding tables (Figures 1, 13, 14, 15). The body consists of columns and rows. The Nr. column includes row numbers. The *Label* column allows to specify labels for the TTCN statements defined in the *Behaviour Description* column. The *Constraints Ref.* column provides references to constraints (Section 2.3). The *Verdict* column includes verdict assignments to indicate the success or failure of a test run with respect to the sequence of statements that have been performed. In the follow-

ASN.1 PDU Constraint Declaration	
<b>Constraint Name</b>	: Indication_24h
<b>PDU Type</b>	: Indication
<b>Derivation Path</b>	:
<b>Encoding Rule Name</b>	:
<b>Encoding Variation</b>	:
<b>Comments</b>	: Receive an offer from the Parcel Service.
Constraint Value	
{goods {type_of_good {1 2 3 4 5}, amount_of_good 100}, order ?, delivery_time 24, comments*}	
<b>Detailed Comments:</b>	

Figure 12 Receiving Constraint for an Indication PDU

Test Case Dynamic Behaviour					
<b>Test Case Name</b> : Indication_1					
<b>Group</b> : Valid/					
<b>Purpose</b> :					
<b>Configuration</b> :					
<b>Default</b> : OtherwiseFail					
<b>Comments</b> :					
<b>Selection Ref</b> :					
<b>Description</b> : Test the offer and indication sequence of behaviour.					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+Preamble			
2		LT_Prod !Offer START T_ParcelService	Offer_Large		
3	L1	LT_Prod ?Indication CANCEL T_ParcelService	Indication_24h	(P)	
4		+Postamble			
5		LT_Prod ?Advertisement	Advertisement_Any		Ignore advertisement
6		GOTO L1			
<b>Detailed Comments:</b>					

Figure 13 Test Case Description

Test Step Dynamic Behaviour					
<b>Test Step Name</b> : Postamble					
<b>Group</b> :					
<b>Objective</b> : To reset the test system and to assign the final verdict.					
<b>Default</b> :					
<b>Comments</b> :					
<b>Description</b> :					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		CANCEL			
2		[TRUE]		R	
<b>Detailed Comments:</b>					

Figure 14 Test Step Description

Dynamic Dynamic Behaviour					
<b>Default Name</b> : OtherwiseFail					
<b>Group</b> :					
<b>Objective</b> : Cover all unexpected reactions from the IUT.					
<b>Comments</b> :					
<b>Description</b> :					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		LT_Prod?OTHERWISE		F	
2		LT_Cons?OTHERWISE		F	
3		?TIMEOUT		F	
<b>Detailed Comments:</b>					

Figure 15 Default Behaviour Description

ing sections, all TTCN statements are introduced, the execution of behaviour descriptions is explained and the assignment of test verdicts is described.

### 2.4.2 TTCN Statements

The *Behaviour Description* column includes TTCN statements. TTCN statements can be grouped into test events, control constructs and pseudo events.

Test events are SEND, IMPLICIT SEND, RECEIVE, OTHERWISE and TIMEOUT. SEND and IMPLICIT SEND specify the sending of PDUs. RECEIVE and OTHERWISE denote the processing of received PDUs. OTHERWISE is the mechanism for dealing with unforeseen test events and denotes that the test system shall accept any incoming PDU. TIMEOUT events check for the expiration of timers. Test events may be qualified and/or followed by assignments and timer operations. Instead of keywords, TTCN uses ‘!’ to describe send events and ‘?’ to denote receive events. For example, the statement *LT\_Prod ! Offer* (Figure 13, line 2) describes the sending of PDU *Offer* via PCO *LT\_Prod* to the IUT and the statement *LT\_Prod ? Advertisement* (Figure 13, line 5) denotes the reception of PDU *Advertisement* at PCO *LT\_Prod* from the IUT.

Control constructs are ATTACH, GOTO and REPEAT. The ATTACH construct allows to attach test steps. GOTO transfers control to a statement identified by a label in the *Label* column and REPEAT is used for the specification of loops.

Pseudo events are qualifiers (i.e. boolean expressions), timer operations (i.e. SET, READTIMER and RESET) and assignments.

The TTCN statements in a behaviour description can be grouped into statement sequences and sets of alternatives. Statement sequences are represented one after the other on separate lines,

being indented from left to right. The statements on lines 1 to 4 in Figure 13 constitute a statement sequence. Statements on the same level of indentation and identical predecessor form a set of alternatives. In Figure 13, the statements on lines 3 and 5 form a set of alternatives. They are on the same level of indentation and their common predecessor is the statement on line 2.

### 2.4.3 Behaviour Execution

The execution of a behaviour description will be explained by means of Figure 13. Execution starts with the first level of indentation (line 1) and proceeds towards the last level of indentation (lines 4 and 6). Only one alternative out of a set of alternatives at the current level of indentation is executed, and execution proceeds with the next level of indentation relative to the executed alternative. For example, the statements on lines 3 and 5 are alternatives. If the statement on line 3 is executed, processing continues with the statement on line 4. Execution of a behaviour description stops if the last level of indentation has been visited, a final test verdict has been assigned (see below), or a test case error has occurred.

Before a set of alternatives is evaluated, a snapshot is taken. This means that the state of the test component, the state of all PCOs and all expired timer lists related to the test case are updated and frozen until the set of alternatives has been evaluated. This guarantees that the evaluation of a set of alternatives is an atomic and deterministic action.

Alternatives are evaluated in the order of their specification. The first alternative with successful evaluation is executed, i.e. all conditions of that alternative are fulfilled. Execution then proceeds with the set of alternatives on the next level of indentation. If no alternative can be evaluated successfully, a new snapshot is taken and the evaluation of the set of alternatives is started again.

### 2.4.4 Verdict Assignment

Test verdicts are assigned in the *Verdict* column of test cases, test steps and default behaviour descriptions. TTCN supports three different verdicts: PASS to indicate that the test behaviour gives evidence for conformance, FAIL to describe that the specification has been violated, and INCONCLUSIVE for cases where neither a PASS nor a FAIL can be given.

TTCN distinguishes between preliminary and final test verdicts. Preliminary verdicts are given in parentheses, e.g. the preliminary PASS in line 3 of Figure 13. Final verdict assignments are specified without parentheses, e.g. the three final FAIL verdicts in Figure 15. The difference

between a preliminary and a final test verdict is that the assignment of a final test verdict terminates the test case execution, i.e. it can be considered to be a combination of a verdict assignment and a subsequent stop operation.

For the handling of test verdicts, each test case has a predefined variable *R*. Variable *R* stores the current preliminary verdict of a test case and its value becomes the final verdict if the test case ends without the assignment of a final verdict. In other words, the assignment of a test verdict in the *Verdict* column of a behaviour description is an assignment to variable *R*. As shown in Figure 14, variable *R* can also be used to calculate the final verdict of a test case. The entry *R* in the *Verdict* column indicates that the test case ends and that the actual value of *R* will be the final verdict.

There are special rules for the assignment of verdicts during the execution of a test case. They are shown in Figure 16 and can be summarized as: “A verdict can only become worse”. For example, if the value of *R* is (FAIL), then the assignment of (PASS) or (INCONCLUSIVE) will have no effect on *R*. Please note that the value *none* in Figure 16 describes the situation where *R* has not been initialized, i.e. no preliminary verdict has been assigned to *R*.

### 2.4.5 The Example Test Case

The test case *Indication\_1* in Figure 13 should be read as follows: the test case starts with the execution of test step *Preamble* in order to initialise the parcel service. Afterwards, an *Offer* is sent to the parcel service at PCO *LT\_Prod* and the timer *T\_ParcelService* is started. Then, two alternative events are expected: Either, an *Indication* or an *Advertisement* is received. If an *Indication* is received (line 3), the timer *T\_ParcelService* is cancelled, a preliminary *PASS* verdict is assigned and *Postamble* is executed in order to reset the test system. If an *Advertisement* is received (line 5), a GOTO statement is used (line 6) to put the test case control back to the set of alternatives at label *LI* in order to await the expected *Indication* PDU.

The *Preamble* (Figure 1) executes the test suite operation *Reset\_ParcelService* and stores the result in the test case variable *PS\_Init*. In case of a successful initialisation, i.e. the value of *PS\_Init* is TRUE, a preliminary *PASS* verdict is assigned (line 2) and the test case proceeds with the execution. If the initialisation is not successful, i.e. *PS\_Init* has the value FALSE, a final *INCONCLUSIVE* verdict is assigned (line 3) which terminates the test execution.

The *Postamble* (Figure 14) resets the test system by cancelling all running timers (line 1). Finally,

Current value of R	Entry in verdict column		
	(PASS)	(INCONC)	(FAIL)
none	pass	inconc	fail
pass	pass	inconc	fail
inconc	inconc	inconc	fail
fail	fail	fail	fail

Figure 16 Handling of Test Verdicts

it assigns the final verdict by referring to the value of the special verdict variable *R* (line 2).

The default behaviour *OtherwiseFail* (Figure 15) defines that the reception of any other PDU at *LT\_Prod* (line 1) or *LT\_Cons* (line 2) will lead to the assignment of a FAIL verdict and the termination of the test case. In addition, the occurrence of a timeout (line 3) will also terminate the test case with the final test verdict FAIL.

## 3 Concurrency in TTCN

The term concurrent TTCN refers to TTCN language constructs and concepts for the description of concurrent test cases. In concurrent TTCN, each test case consists of several test components that execute independently and in parallel. A Main Test Component (MTC) controls the test case execution and creates Parallel Test Components (PTCs). The MTC cannot stop PTCs but has the possibility to check their termination by means of a DONE statement. A test case always ends when the MTC ends. Each test component controls its own local verdict. The final verdict of a test case is calculated according to the rules described in Figure 16 by the MoT.

Test components can coordinate themselves by exchanging Coordination Messages (CMs) at Coordination Points (CPs). CPs connect test components and are similar to PCOs. CMs are similar to PDUs, but they are used for the information exchange among test components only.

A concurrent test configuration for the parcel service example is given in Figure 17. It uses *PTC\_Main* as MTC, which creates the PTCs *PTC\_Prod* and *PTC\_Cons*. The PTCs control

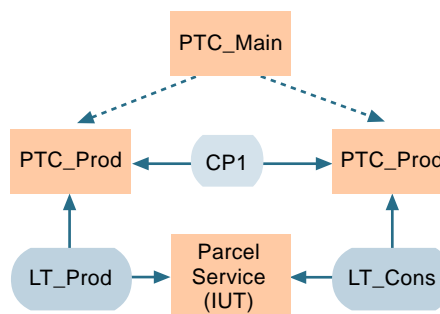


Figure 17 Example of a Concurrent Test Architecture

Test Case Dynamic Behaviour					
<b>Test Case Name</b> : Deliver_1					
<b>Group</b> : Valid/					
<b>Purpose</b> :					
<b>Configuration</b> : Example_Conc_Conf					
<b>Default</b> :					
<b>Comments</b> :					
<b>Selection Ref</b> :					
<b>Description</b> : Test the offer and indication sequence of behaviour.					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+Preamble			
2		CREATE (PTC_Prod: Deliver_1_Prod, PTC_Cons: Deliver_1_Cons)			
3		?DONE(PTC_Prod, PTC_Cons)		(P)	
4		+Postamble			
<b>Detailed Comments:</b>					

Figure 18 Description of PTC\_Main

Test Step Dynamic Behaviour					
<b>Test Step Name</b> : Deliver_1_Cons					
<b>Group</b> : Concurrent/Version/					
<b>Objective</b> :					
<b>Default</b> : OtherwiseFail_LT_Cons					
<b>Comments</b> :					
<b>Description</b> :					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		CP1 ?Continue(Order_No:= Continue.order)	Continue_Order_Recv		
2		LT_Cons ?Deliver	Deliver_Large (Order_No)	(P)	
3		LT_Cons !Acknowledge	Acknowledge_Large (Order_No)		
<b>Detailed Comments:</b>					

Figure 19 Description of PTC\_Cons

```

testcase Indication_1() runs on MTCType {
// Test the offer and indication sequence of behaviour

    activate(OtherwiseFail)
    Preamble();
    LT_Prod.send(Offer_Large);
    T_ParcelService.set;
    alt {
        [] LT_Prod.receive(Indication_24h) {
            T_ParcelService.stop;
            verdict.set(pass);
            Postamble()
        }
        [] LT_Prod.receive(Advertisement_Any); {
            goto alt;
        }
    }
}

```

Figure 20 TTCN-3 Description of Figure 13

and observe the IUT via the PCOs *LT\_Prod* and *LT\_Cons*. They coordinate themselves through the coordination point *CPI*.

The definition of the MTC *PTC\_Main* for an example test case is shown in Figure 18. After the *Preamble* (line 1), *PTC\_Main* creates *PTC\_Prod* and *PTC\_Cons* (line 2). The behaviour of the PTCs is given by the test step descriptions *Deliver\_1\_Prod* and *Deliver\_1\_Cons*. The test behaviour can be related to Figure 2: *Deliver\_1\_Prod* covers the testing of the sequence from *Offer* to *Confirmation* at PCO *LT\_Prod*, while *Deliver\_1\_Cons* covers the *Delivery* and *Acknowledgment* events at PCO *LT\_Cons*. *PTC\_Main* waits for the termination of the two PTCs through a *DONE* statement. The MTC assumes their successful termination by assigning a preliminary pass verdict<sup>7)</sup> (line 3). Finally, the *Postamble* finishes the test case.

The use of a CM is shown in Figure 19. The CM Continue on line 1 enables the PTC to proceed with the test execution by expecting to receive a *Delivery* PDU at PCO *LT\_Cons* (line 2). The *Delivery* should be parameterized with the correct order number which was sent to *PTC\_Cons* as a parameter of the CM. This number is used as a parameter to the constraint for *Delivery*. At the end, *PTC\_Cons* initiates a proper *Acknowledgment* and terminates.

## 4 Outlook: Next Version of TTCN

Currently, the third edition of TTCN (TTCN-3) is in work at ETSI [4, 10]. TTCN-3 is a text-based language for the specification of tests for reactive systems. TTCN-3 is on a syntactical (and methodological) level a drastic change compared to the previous TTCN versions. However, the main concepts of TTCN have been retained and improved and new concepts have been included, so that TTCN-3 will be applicable for a broader class of systems. New concepts are e.g. a test execution control program to describe relations between test cases such as sequences, repetitions and dependencies on test outcomes, dynamic concurrent test configurations, and test behaviour in asynchronous and synchronous communication environments. Further improved concepts are, e.g. the integration of ASN.1, the module and grouping concepts to improve the test suite structure, and the test component concepts to describe concurrent test setups.

<sup>7)</sup> Please note that any worse verdict returned by one of the test components will overrule this assignment according to the table given in Figure 16.

The top-level unit of a TTCN-3 test suite is the module which can import definitions from other modules. A module consists of a definitions part and a control part. The definitions part of a module covers definitions for test components, their communication interfaces, type definitions, test data templates (previously known as constraints), functions, and test cases. The control part of a module calls the test cases and describes the test campaign. For this, control statements similar to statements in other programming languages (e.g. if-then-else and while loops) are supported. They can be used to specify the selection and execution order of individual test cases. TTCN-3 provides a variety of constructs to describe test behaviour within a test case such as the alternative reception of communication events and their interleaving. Moreover, default behaviour can be covered, e.g. unexpected reactions from the system under test. In addition to the automatic test verdict assignment, more powerful logging mechanisms are provided, e.g. for detailed tracing. An example of a TTCN-3 test case definition is shown in Figure 20. It is the TTCN-3 representation of the TTCN test case in Figure 13.

In addition to the pure textual format, TTCN-3 will define at least two presentation formats: A tabular conformance testing presentation format [5] that resembles the tabular form of TTCN and a graphical presentation format [6, 8] that supports the presentation and also the development of TTCN-3 test cases, as Message Sequence Charts (MSC).

## References

- 1 ISO. *Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework. – Seven Parts Standard*. Geneva, 1991–1999 (includes [2] and [3]). (ISO/IEC 9646.)
- 2 ISO. *Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework – Part 2: Abstract test suite specification*. Geneva, 1991. (ISO/IEC 9646-2.)
- 3 ISO. *Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework – Part 3: The Tree and Tabular Combined Notation (TTCN)*. 2nd ed. Geneva, 1998. (ISO/IEC 9646-3.)
- 4 ETSI. *TTCN-3 – Core Language. European Norm (EN) 00063-1 (provisional)*. Sophia-Antipolis, 2000. (ETSI TC MTS.)
- 5 ETSI. *TTCN-3 – Tabular Presentation Format*. EN00063-2 (provisional). Sophia-Antipolis, 2000. (ETSI TC MTS.)
- 6 ETSI. *TTCN-3 – MSC Presentation Format*. EN00063-3 (provisional). Sophia-Antipolis, 2000. (ETSI TC MTS.)
- 7 ITU. *Information Technology – Abstract Syntax Notation One (ASN.1)*. Geneva, 1994. (ITU-T Recommendations X.680-683.)
- 8 ITU. *Message Sequence Chart (MSC)*. Geneva, 2000. (ITU-T Recommendation Z.120.)
- 9 Walter, T, Schieferdecker, I, Grabowski, J. Test Architectures for Distributed Systems – State of the Art and Beyond (Invited Paper). In: *Testing of Communicating Systems*. Petrenko, A, Yevtuschenko, N (eds.). Dordrecht, Kluwer, 1998, 149–174. (Volume 11.)
- 10 Grabowski, J et al. On the Design of the new Testing Language TTCN-3. In: *Testing of Communicating Systems*. Ural, H, Probert, R L, von Bochmann, G (eds.). Dordrecht, Kluwer, 2000, 161–176. (Volume 13.)