EG 202 103 V1.1.1 (1999-05)

ETSI Guide

Methods for Testing and Specification (MTS); Guide for the use of the second edition of TTCN

ETSI

Reference

REG/MTS-00056 (jh000icq.PDF)

Keywords

TTCN, testing, methodology, performance

Contents

Intellectual Property Rights

Foreword

This ETSI Guide (EG) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

1 Scope

The present document provides an introduction to the new features defined in the second edition of the Tree and Tabular Combined notation (TTCN) as defined in ISO/IEC 9646-3 [1]. The present document is a revision of EG 201 148 [2]. The present document is intended to be used as introductory reference material for those wishing to gain an understanding of the new TTCN features and on the application of TTCN to areas other than conformance testing.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ISO/IEC 9646-3 (1998): "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)".

[2] EG 201 148 (V1.1): "Methods for Testing and Specification (MTS); Guide for the use of the second edition of TTCN".

[3] ITU-T Recommendation X.680 Amendment 1 (1994): "Abstract Syntax Notation One (ASN.1): Specification of basic notation: Rules of extensibility".

[4] TR 101 295 (V1.1): "Methods for Testing and Specification (MTS), Rules for the transformation of ASN.1 definitions using X.681, X.682 and X.683 to equivalent X.680 constructs".

[5] ETS 300 806-2 (1998): "Private Integrated Services Network (PISN); Inter-exchange signalling protocol; Generic functional protocol for the support of supplementary services; Part 2: Abstract Test Suite (ATS) specification".

[6] EN 300 286-6 (V1.2): "Integrated Services Digital Network (ISDN); User-to-User Signalling (UUS) supplementary service; Digital Subscriber Signalling System No. one (DSS1) protocol; Part 6: Abstract Test Suite (ATS) and partial Protocol Implementation eXtra Information for Testing (PIXIT) proforma specification for the network".

[7] ISO/IEC 8825-1: "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[8] EN 300 130-4: "Integrated Services Digital Network (ISDN); Malicious Call Identification (MCID) supplementary service; Digital Subscriber Signalling System No. one (DSS1) protocol; Part 4: Abstract Test Suite (ATS) and partial Protocol Implementation eXtra Information for Testing (PIXIT) proforma specification for the user".

3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
ASP	Abstract Service Primitive
ATM	Asynchronous Transmission Mode
ATS	Abstract Test Suite
BER	Basic Encoding Rules
CC	Control of Component
CLR	Cell Loss Ratio
CORBA	Common Object Request Broker Architecture
CL	Communication Link
CM	Co-ordination Message
CoP	Communication Point
CP	Co-ordination Point
EET	Earliest Execution Time
FILO	First-In-Last-Out
IC	Interface Component
IDL	Interface Description Language
INAP	IN Application Part
IUT	Implementation Under Test
LET	Latest Execution Time
LILO	Last-In-Last-Out
MFBS	Maximum Frame Burst Size
MIMO	Message-In-Message-Out
MTC	Main Test Component
NFOT	Nominal Frame Output Time
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
PTC	Parallel Test Component
QoS	Quality of Service
SUT	System Under Test
TCAP	Transaction Capabilities Application Part
TINA	Telecommunications Information Networking Architecture
TSP	Test Suite Parameter
TTCN	Tree and Tabular Combined Notation

4 Introduction

The present document summarizes and gives examples of the additional features and capabilities of TTCN version 2 that are considered to be of most interest when specifying Abstract Test Suites for telecommunications protocols and services. The major new features are:

- concurrency, which allows the parallel execution of different dynamic behaviours;
- encoding definitions, which allows the definition of encoding of PDU types and PDU Constraints (or even individual Constraint fields);
- modularity, which allows the reuse of parts of existing ATS and the specification of modules;
- use of ASN.1 1994 (X.680 series).

Less extensive, but nonetheless important, changes are:

- grouping of constraints and other tables;
- possibility to pass matching symbols to Constraints;
- PDUs need not contain any fields;
- the RETURN statement to exit Test Steps and Defaults;
- no longer mandatory that an OTHERWISE in a Default should lead to a fail verdict;
- Collective Comments in some tables;
- predefined type (R_Type) for verdicts;
- use of Active Defaults to switch on/off default behaviour;
- Test Suite Operations may be specified as procedures rather than informal text;
- ability to declare Test Suite Constants by reference.

5 Concurrency

5.1 Introduction to Concurrent TTCN

Unlike TTCN version 1, concurrent TTCN allows test suites:

- to use more than two Points of Control and Observation (PCO);
- to use more than one underlying service provider;
- to have dynamic behaviours executing in parallel;
- to specify co-ordination between concurrently executing test components.

These capabilities have introduced additional concepts, proformas, statements and verdict mechanisms to TTCN.

- Additional concepts:
    - Test Components;
    - Test Component Configurations;
    - Co-ordination Points (CP);
    - Co-ordination Messages (CM).
- Additional proformas:
    - Test Component Declarations;
    - Test Component Configuration Declaration(s);
    - CP Declarations;
    - CM Declarations (Tabular and ASN.1);
    - CM Constraints (Tabular and ASN.1).

- Additional constructs and statements:
  - CREATE;
  - DONE.
- Additional verdict mechanisms:
  - local result variables;
  - global result variable.

5.2 Test Components

The building-blocks of Concurrent TTCN are called test components. A test component can be either a Main Test Component (MTC) or a Parallel Test Component (PTC). All test components are declared in a single Test Components Declaration table. In a Test Suite this table comes after the Timer Declarations table.

Table 1: Declaring Test Components

Test Component Declarations				
Component Name	**Component Role**	**Nr of PCOs**	**Nr of CPs**	**Comments**
MTC1	MTC	0	2	
PTC1	PTC	1	1	
PTC2	PTC	1	1	

Each test component should have a name that is unique within the test suite and be assigned the role of either **MTC** or **PTC**. This table should contain at least one MTC and at least one PTC (see also subclause 8.1.1).

NOTE 1: **MTC** and **PTC** are now reserved words in TTCN.

At this stage we need only define placeholders for the actual PCOs and Co-ordination Points that will later be associated with the test components in a particular configuration. This is done by simply stating the number of PCOs and/or CPs that may be associated with each test component.

NOTE 2: A test component may have neither PCOs nor CPs. In fact, it is quite feasible that this could apply even to an MTC. The MTC could CREATE the PTCs and achieve co-ordination, albeit limited, using the DONE statement and the implicit verdict mechanism.

5.3 Co-ordination of test components

Explicit co-ordination between test components is achieved using Co-ordination Messages (CM) exchanged at Co-ordination Points (CP).

5.3.1 Co-ordination points

CPs are very similar to PCOs. They allow asynchronous communication between test components (i.e., between exactly two PTCs or between one PTC and the MTC). In other words, a CP may not be shared between more than two test components.

A CP may not be connected to the IUT, either directly or indirectly via a service provider.

In a test suite the Co-ordination Point declarations come after the PCO Declarations.

Table 2: Declaration of Co-ordination Points

Coordination Point Declarations	
CP Name	**Comments**
CP1	
CP2	

There is a predefined type in TTCN called **CP**. This is useful when parameterizing PTCs.

NOTE: CP is a reserved word in TTCN.

5.3.2 Co-ordination Messages

CMs are very similar to Abstract Service Primitives (ASP) except that they occur at CPs only. For the definition of a CM either the tabular form or Abstract Syntax Notation One (ASN.1) as defined in ITU-T Recommendation X.680 [3] can be used.

NOTE 1: CMs are treated like ASPs and not as PDUs because TTCN is not concerned with the encoding of Co-ordination messages. That is a matter for the implementers of the test suite.

A CM parameter may be of any TTCN type including structured types and the metatype **PDU**. Generally, though, it is recommended that CMs are kept as simple as possible. In many cases a CM will not even have parameters, the name itself will be adequate (e.g., STOP, WAIT, etc.).

There are no predefined TTCN Co-ordination Messages.

CMs may be declared using either TTCN tables or ASN.1. In a test suite, CM Declarations come after the PDU Declarations, as illustrated in table 3.

Table 3: Definition of a Co-ordination Message

CM Type Definition		
CM Name : CM_ERROR Comments :		
Parameter Name	**Parameter Type**	**Comments**
Error	INTEGER	

CM constraints are similar to ASP constraints. In a test suite, CM Constraints come after the PDU Constraints.

Table 4: A Co-ordination Message constraint

CM Constraint Declaration		
Constraint Name : ERR (err_num:INTEGER) CM Type : CM_ERROR Derivation Path : Comments :		
Parameter Name	**Parameter Value**	**Comments**
Error	err_num	

NOTE 2: When a CM has no parameters it is not necessary to define a constraint for it. This also means that an entry in the constraints' reference column of a dynamic behaviour is not required.

NOTE 3: In TTCN Version 2 this point also applies to ASPs and PDUs generally.

5.4 Defining different test configurations

An actual abstract test architecture is defined by connecting together a number of test components in what is called a Test Component Configuration Declaration. In the present document we will sometimes use the term configuration for short.

A configuration will consist of exactly one MTC and zero or more PTCs.

In most practical applications a single ATS will make use of more than one configuration. Each configuration is defined in a separate Test Component Configuration Declaration Table.

In a test suite, Test Component Configuration Declarations come after the Test Component Declarations.

Table 5: A typical configuration

Test Component Configuration Declaration			
Configuration Name : CONFIG1 Comments :			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
MTC1 PTC1 PTC2	 PCO1 PCO2	CP1, CP2 CP1 CP2	

5.4.1 Connection to the IUT

The PCOs Used column (see table 5) lists the actual PCOs (if any) associated with the test components. The relation of the PCOs to the IUT, either directly (as in the case of a PTC that is part of an Upper Tester) or indirectly via an underlying service provider is indicated, as usual, in the PCO Declarations table and also in the PCO Type Declarations table.

Note the following rules:

- each entry in this column is a list of zero or more PCOs;
- for each MTC and each PTC the number of entries in this list should be the same as the corresponding number of PCOs stated in the declaration of the test component;
- no PCO may be used more than once in a single configuration (i.e., test components cannot share PCOs).

5.4.2 Connecting MTCs and PTCs

The CPs Used column lists the actual CPs (if any) associated with the test components. These entries are used to interconnect test components.

Note the following rules:

- each entry in this column is a list of zero or more CPs;
- for each PTC the number of entries in this list should be the same as the corresponding number of CPs stated in the declaration of the test component;
- for each MTC the number of entries may be the same or less but not more. This allows for flexibility when using the same MTC in various configurations;
- no CP name is allowed to appear more than once in a single list;
- each CP name that is in one list should appear in exactly one other list. In this manner connected pairs of test components can be specified.

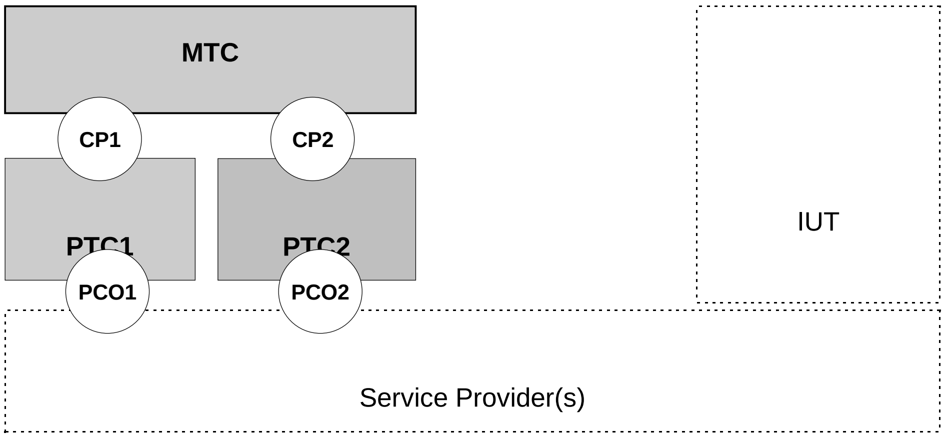

Figure 1: The configuration specified in table 5

5.4.3 The Master and Parallel Test Components

Each test configuration shall have one and only one MTC. Furthermore, the MTC shall be located on the Lower Tester side of the architecture. The MTC is responsible for creating the PTCs, for overall co-ordination of the Test Case and for assigning the final verdict. This means that the Test Case *is* the MTC. The variables, constants, timers, etc. of the Test Case are the variables, constants, timers, etc. of the MTC.

A particular configuration is associated to a Test Case by the (new) entry in the Test Case Dynamic Behaviour header, as illustrated in table 6.

Table 6: Associating a configuration with a Test Case

Test Case Dynamic Behaviour					
Test Case Name :					
Group :					
Purpose :					
Configuration : CONFIG1					
Defaults :					
Comments :					
Nr	**L**	**Behaviour Description**	**Constraint Ref**	**V**	**Comments**

5.4.4 The CREATE Construct

PTC behaviour is specified using Test Steps which may either reside in the Test Step library or be local trees. The MTC binds dynamic behaviour to the PTCs using the CREATE construct. This construct also has the effect of starting execution of the named PTC or PTCs. As each created PTC is a separate executing entity the same Test Step may be used, if wished, to define the behaviour of different PTCs.

Table 7: The CREATE construct

7		:			
8		CREATE (PTC1:Step1 (CP1), PTC2:Step2(CP2))			
9		:			

In table 7 the Test Step Step1 is bound to PTC1 and the Test Step Step2 is bound to PTC2. Note that parameters may be passed to the PTCs, in this case the Co-ordination Points CP1 and CP2.

Test Steps that are invoked from a Test Case in the normal manner (i.e., + TestStepName) are part of the MTC and should not be considered as PTCs, only Test Steps that are invoked from a Test Case using the CREATE construct become PTCs.

5.4.5 Scope of Variables

At the start of execution of the Test Case each test component is allocated its own fresh set of variables, timers, constraints, etc. These are limited in scope to the test component during the life of the Test Case: data is not shared between test components.

Only an MTC may use Test Suite Variables.

5.5 Verdicts

Concurrent TTCN includes a number of predefined variables for tracking the intermediate results and the final verdict. These are:

- each PTC maintains its own *local* result variable called **R**;
- the MTC also has a local result variable but this is called **MTC_R**;
- the MTC also maintains a *global* result variable called **R**.

A PTC may assign a preliminary result (for example (P)) in its verdict column. This has the effect of updating both the local R and the global R according to the priority table defined in ISO/IEC 9646-3 [1].

NOTE: These priority rules are unchanged from TTCN version 1. Also, MTC_R is now a TTCN reserved word.

A PTC may also assign a result without parentheses (for example P), in its verdict column. This updates the local and global result variables as usual and terminates execution of the PTC. However, this result is not to be considered as a final verdict.

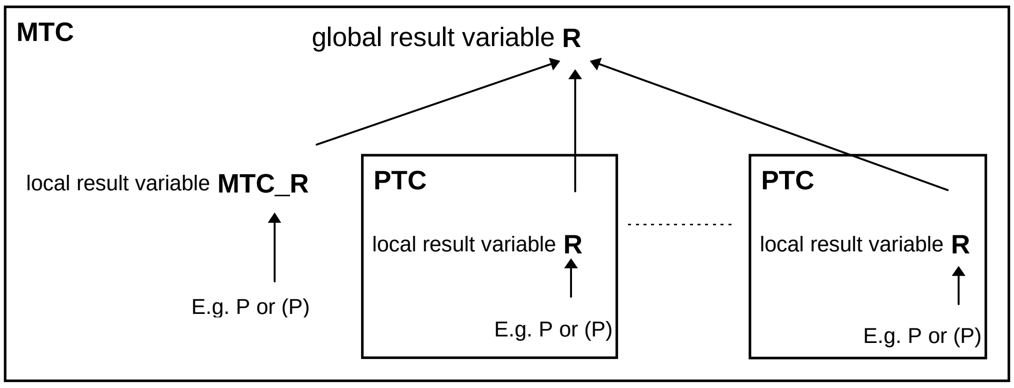

Figure 2: Relation between the different result variables

Similarly, the MTC will update its local result variable MTC_R and the global result variable R in accordance with the priority rules. A result without parentheses in the MTC is considered to be the final verdict and has the effect not only of terminating execution of the MTC but also of any PTCs that are still executing.

The TTCN standard does not say anything about the actual mechanisms that perform the updating of the local and global R variables the and MTC_R variable. This is a matter for the implementors of the test suite. Use these variables with care, for example always after DONE (see subclause 5.6).

5.5.1 R_Type

R_Type is a new predefined type associated with values of verdicts (pass, fail, inconc, none). It can be useful if verdicts need to be carried, say, in a Co-ordination message.

NOTE: This facility is less useful now because concurrent TTCN supports the implicit passing of verdicts.

5.6 The DONE Statement

The DONE statement is used to check whether or not a test component has terminated execution. It may be used by both MTCs and PTCs, but should generally be used by the MTC to ensure that all PTCs have ceased execution before assigning the final verdict.

It is strongly recommended to check that the PTCs have finished their execution, with the use of the DONE statement in MTC, before terminating the MTC.

Table 8: The DONE Statement

5		:			
6		?DONE (PTC1, PTC2)		PASS	
8		:			

The DONE statement has no effect on the test component being interrogated (it does not terminate execution of a test component). In a PTC termination will either occur:

- when the PTC behaviour tree ends in a leaf; or
- when an entry without parentheses is encountered in the verdicts column of the PTC or the MTC; or
- when the PTC is instructed to do so by an appropriate CM (which has to be defined in the test suite).

A DONE statement without an argument list has the same effect as interrogating all the PTCs that have been created prior to the DONE. Only an MTC may do this global interrogation.

Note that ISO/IEC 9646-3 [1] recommends the use of TIMEOUT as an alternative (subsequent) event to DONE. The present document discourages this practice since this would only handle test case errors and turn them into verdicts.

5.7 Summary of MTC and PTC characteristics

The table 9 summarizes the main characteristics of MTCs and PTCs.

Table 9: Characteristics of MTCs and PTCs

Capability	MTC	PTC
Is associated with a ...	Test Case	Test Step
Can use the CREATE construct	Yes	No
Can use the DONE statement	Yes	Yes
Results written to the global result variable R	Yes	Yes
Results written to the local result variable MTC_R	Yes	No
Results written to a local result variable R	No	Yes
Can assign a final verdict	Yes	No
Can use all TTCN types	Yes	Yes
Can use Test Suite Parameters and other Constants	Yes	Yes
Can use Test Suite Variables	Yes	No
Scope of Test Case Variables, Timers, Operations, etc.	Local to MTC	Local to PTC
Scope of attached Test Steps and Defaults	Local to MTC	Local to PTC

6 Encoding

6.1 Introduction to specifying encoding information in TTCN

This addition to TTCN allows test suites to specify the encoding of PDUs. This is useful where a base standard offers different choices of encoding rules and/or to test how an IUT handles invalid encoding. Mechanisms are provided for the definition of:

- general encoding;
- variations on the general encoding;
- invalid field encoding.

These mechanisms may be used both with tabular TTCN PDUs and ASN.1 PDUs. They shall not be used for ASPs or CMs, the encoding of which is an implementation matter.

These capabilities have introduced a number of additional proformas and added header entries and additional columns to some existing TTCN proformas.

- Additional proformas:
  - Encoding Definitions (e.g., table 10);
  - Encoding Variations (e.g., table 11);
  - Invalid Field Encoding Operation Definition(s) (see subclause 6.5).
- Changes to existing proformas:
  - Simple Type Definition(s) - one additional header entry, one additional column;
  - Structured Type Definition(s) - one additional header entry, one additional column;
  - ASN.1 Type Definition(s) - one additional header entry, some additional syntax;
  - ASN.1 Type Definitions by Reference - one additional column;
  - PDU Type Definition(s) - two additional header entries, one additional column;
  - ASN.1 PDU Type Definition(s) - two additional header entries, some additional syntax;
  - ASN.1 PDU Type Definitions by Reference - two additional columns;
  - Corresponding changes to the relevant Constraints tables have also been made.

NOTE: All these changes are optional and need not appear if encoding is not used.

6.2 Scope of application of encoding information

The mechanisms for TTCN Encoding Information shall only be used if the protocol or service specification defines or makes reference to a standardized set of encoding rules. New encoding rules could also be created when needed by an ATS (e.g. "direct encoding" rule when ASN.1 is used to defined tabular items where BER is not applicable). These new encodings shall be fully specified in a document provided with the ATS (e.g., ATS&PIXIT document).

Encoding Information can be of the following kinds:

- reference to the *general* encoding rule(s) applicable to the entire test suite;
- definitions of the encoding *variations* (if any) on the general encoding rule(s);
- definitions of *invalid* PDU field encoding (if any).

The encoding may be applied at five levels, given below in *increasing* order of priority:

- the top-level applies to the entire test suite, that is, the encoding rules apply to all PDUs sent or received in the test suite, unless overridden by one of the following cases;
- the second level applies to all PDUs of a particular type. All PDUs of this type will be encoded according to the *variant* given in the PDU type definition header;
- the third level allows specific fields of a particular PDU type to be given either a *variant* encoding or an *invalid* encoding;
- the fourth level allows an (entire) individual constraint to be given a *variant* encoding;
- finally, the fifth level allows specific fields of an individual PDU constraint to be given either a *variant* encoding or an *invalid* encoding.

NOTE: For simplicity, this subclause only discusses PDUs and PDU Constraints. The above equally applies to Structured Types and Structured Constraints (that are used in PDUs). Similarly, the same applies to Simple Types and Structure elements as to PDU fields. However, re-definition of Encoding Rules is not permitted in Structured Types.

6.3 Encoding Definitions

The Encoding Definitions figure states the encoding rules that are used in the ATS by referring to the appropriate standard where the rules are defined. In a Test Suite this table comes immediately before the Test Suite Operation Definition tables.

Table 10: Typical use of Encoding Definitions

Encoding Definitions			
Encoding Rule Name	**Reference**	**Default**	**Comments**
BER	ISO/IEC 8825-1: 1993	TSP1	
PER	ISO/IEC 8825-1: 1993	[TSP2 OR TSP3]	
DER	ISO/IEC 8825-1: 1993		
DirEnc	EN 300 130-4		Direct encoding defined in the ISDN standards.

The Boolean expression in the Default column is used to determine the default set of encoding rules. In our example, if the Test Suite Parameter TSP1 has the value TRUE then BER is used. If either TSP2 or TSP3 evaluates to TRUE then PER applies. No entry in the Default column has the same effect as writing FALSE. If this column is empty then this is a test case error (see also subclause 8.1.1).

The DirEnc encoding rule is not specified by a standardization body but is needed for this ATS. The encoding rule is than defined locally in the ATS&PIXIT document.

6.4 Encoding Variations

The Encoding Variations table states variations on the general encoding rules that are used in the ATS, if any. These variations are allowed according to the original encoding rules. In a Test Suite this table comes after the Encoding Definitions.

Table 11: Typical use of Encoding Variations

Encoding Variations			
Encoding Rule Name : BER Type List : INTEGER Comments :			
Encoding Variation	**Reference**	**Default**	**Comments**
SD LD (len:INTEGER)	6.3.3.1 6.3.3.1	TRUE	

Table 10 shows two variations of Basic Encoding Rules (BER, Short Definite and Long Definite encodings for INTEGER values). The default variant is Short Definite.

The Type List entry indicates to which types this encoding applies. If this entry is empty then the encoding applies to all types.

NOTE: The encoding variation may be followed by a formal parameter list, if required. In our example, *len* states the length of the LD encoding.

6.5 Invalid Field Encoding definitions

The Invalid Field Encoding Definitions table is used to define invalid encodings of PDU fields, if any. In a Test Suite these tables come after the Encoding Variation tables.

The invalid encodings are defined using the (new) TTCN syntax for procedures (not the free text format).

Table 12: Definition and use of invalid field encodings

Invalid Field Encoding Operation Definition
Group : ProtocolDiscriminator Operation Name : wrongPD (prDiscr : PD) Result Type : PD Comments : Generate a wrong encoded protocol discriminator.
Definition

VAR PD_OUT : PD
ENDVAR
BEGIN
PD_OUT := INT_TO_BIT ((BIT_TO_INT (prDiscr)) + 1 MOD 128)
RETURNVALUE PD_OUT
END

Table 12a

PDU Type Definition			
PDU Name	: ALERTING_PDU_wrong_PD		
PCO Type	: L		
Encoding Rule Name	:		
Encoding Variation	:		
Comments	: The alerting PDU with a wrong encoded protocol discriminator.		
Field Name	**Field Type**	**Field Encoding**	**Comments**
pd	PD	wrongPD	protocol discriminator is not correctly encoded.
cr	CR		
mt	BITSTRING[8]		
chi	CHI	DirEnc	Specified in tabular form in the protocol specification, but defined with an ASN.1 table in TTCN.
fie	FIES		
pi	PI		
dsp	DSP		
uui	UUI		

6.6 Using variant and invalid encodings in PDUs and Constraints

In the following example, let us assume that the default encoding rules BER apply (as stated in the Encoding Rules table of table 11).

Table 13: PDU Type definitions with encoding information

PDU Type Definition			
PDU Name	: A_PDU		
PCO Type	: L		
Encoding Rule Name	:		
Encoding Variation	: LD		
Comments	: All PDUs (Constraints) of this type will be encoded to BER. However, all fields of type INTEGER in this PDU will be encoded to the Long Definite (LD) variation.		
Field Name	**Field Type**	**Field Encoding**	**Comments**
F1	INTEGER		This field will be encoded Long Definite
F2	INTEGER		This field will be encoded Long Definite
F3	INTEGER		This field will be encoded Long Definite

Table 13a

PDU Type Definition			
PDU Name	: ALERTING_PDU		
PCO Type	: L		
Encoding Rule Name	:		
Encoding Variation	:		
Comments	: By Default, all the ASN.1 fields are using BER. However the field CHI, defined with ASN.1 tables uses a direct encoding.		
Field Name	**Field Type**	**Field Encoding**	**Comments**
pd	PD		
cr	CR		
mt	BITSTRING[8]		
chi	CHI	DirEnc	Specified in tabular form in the protocol specification, but defined with an ASN.1 table in TTCN.
fie	FIES		
pi	PI		
dsp	DSP		
uui	UUI		

Table 14: A constraint with encoding derived from the PDU type

PDU Constraint Declaration		
Constraint Name	:	C1
PDU Type	:	A_PDU
Derivation Path	:	
Encoding Rule Name	:	
Encoding Variation	:	
Comments	:	This constraint will be encoded according to the encoding information given in the definition of A_PDU (i.e. BER and Long Definite). However, the encoding of fields F1 and F2 is explicitly overridden.

Field Name	Field Value	Field Encoding	Comments
F1	123	SD	Switch back to Short Definite
F2	456	INVALID_LENGTH(2)	This field will given an invalid encoding
F3	789		This field will be encoded Long Definite

6.7 Using the ENC keyword

In ASN.1 PDU Type Definitions and ASN.1 PDU Constraint Declarations the field encodings are specified using the **ENC** keyword instead of the additional column. In ASN.1 the constraint of the example above would be:

Table 15: Use of the ENC keyword in ASN.1

ASN.1 PDU Constraint Declaration		
Constraint Name	:	C1
PDU Type	:	A_PDU
Derivation Path	:	
Encoding Rule Name	:	
Encoding Variation	:	
Comments	:	This constraint will be encoded according to the encoding information given in the definition of A_PDU. However, the encoding of fields F1 and F2 is explicitly overridden.

Constraint Value

```
SEQUENCE {
    f1 123 ENC SD;
    f2 456 ENC INVALID_ENCODING(2);
    f3 789 }
```

7 Modular TTCN

7.1 Introduction

The introduction of modularity in TTCN allows the separation of parts of an abstract test suite into modules. This is particular useful in view to maintenance and reusability issues. TTCN Edition 2 permits the modularization of test suites by allowing the specification of two separate entities:

- Test Suites (as in normal TTCN);
- Modules.

A Test Suite may now consist of five (rather than four) parts, that is:

- Test Suite Overview Part;
- Imports Part (new);
- Declarations Part;
- Constraints Part;
- Dynamic Part.

Whereby the Test Suite Overview Part contains a new section for the Exports Part:

- Test Suite Index;
- Test Suite Structure;
- Test Case Index;
- Test Step Index;
- Default Index;
- Test Suite Exports (new).

A Module consists of a similar five parts:

- Module Overview part;
- Module Imports part;
- Declarations part;
- Constraints part;
- Dynamic Part.

Whereby the Module Overview Part contains the section for the Module Exports:

- TTCN Module Exports;
- TTCN Module Structure;
- Test Case Index;
- Test Step Index;
- Default Index.

These capabilities have introduced a number of additional concepts and proformas to TTCN:

- Modules;
- Import and Export of objects;
- EXTERNAL objects;
- Imports proforma used in Test Suite and Modules.

7.2 Importing objects to a Test Suite or module

The types of objects that may be imported/exported are:

Table 16: List of object types which can be imported

SimpleType_Object	Timer_Object	StructTypeConstraint_Object
StructType_Object	Tcomp_Object	ASN1_TypeConstraint_Object
ASN1_Type_Object	TcompConfig_Object	TTCN_ASP_Constraint_Object
TS_Op_Object	TTCN_ASP_Type_Object	ASN1_ASP_ Constraint_Object
TS_Proc_Object	ASN1_ASP_Type_Object	TTCN_PDU_ Constraint_Object
TS_Par_Object	TTCN_PDU_Type_Object	ASN1_PDU_ Constraint_Object
SelectExpr_Object	ASN1_PDU_Type_Object	TTCN_CM_ Constraint_Object
TS_Const_Object	TTCN_CM_Type_Object	ASN1_CM_ Constraint_Object
TS_Var_Object	ASN1_CM_Type_Object	TestCase_Object
TC_Var_Object	EncodingRule_Object	TestStep_Object
PCO_Type_Object	EncodingVariation_Object	Default_Object
PCO_Object	InvalidFieldEncoding_Object	NamedNumber_Object
CP_Object	Alias_Object	Enumeration_Object

The imported objects are declared in the Imports proforma. In a Test Suite this table comes after the test suite overview.

Table 17: Use of the Import proforma

Imports			
Source Name : Module_1 Source Ref : Standards Ref : Comments :			
Object Name	**Object Type**	**Source Name**	**Comments**
SimpleType_A Timer_A PDU_A SETUP_S1(CREF: CALL_REF_TYPE)	SimpleType_Object Timer_Object TTCN_PDU_Type_Object TTCN_PDU_Object Constraint_Object	 Omit Module_2	 (*)
Detailed Comments: (*) the corresponding PDU type and the StructuredType definitions that are referenced by this constraint are implicitly imported but can not be used the test suite unless they are explicitly imported (and explicitly exported in the module).			

7.3 Exporting objects from a Test Suite or module

The exported objects are declared in the exports proforma which is located in the test suite overview part.

Table 18: Use of the Export proforma

Test Suite Exports				
Object Name	**Object Type**	**Source Name**	**Page Nr**	**Comments**
String5	SimpleTypeDef_Object		3	
Wait	TimerDcl_Object	Module_B		
INTC	TTCN_PDU_Type_Object		13	
DEF1	Default_Object	TestSuite_1		
TC_2	TestCase_Object	TestSuite_2		
TC_3	TestCase_Object		33	
Preamble	TestStep_Object	EXTERNAL		
Detailed Comments:				

7.4 TTCN modules

A module is very similar to a Test Suite in that it may contain declarations, constraints and dynamic behaviours but it is not complete in itself. In practice a module should concentrate on a particular aspect, for example, only constraints or only Test Steps. A Module also has an overview part, similar in function to the Test Suite Overview.

Modules are not foreseen to be executed and only consist of a library of TTCN items to be used in other test suites for execution.

It is strongly recommended to avoid complex structures made of modules and test suites with the use of the import/export function. The final tree consisting of all the module and test suites with their relationships should stay as flat as possible.

Objects defined in modules are intended to be imported by Test Suites or other Modules. The objects that are visible to (i.e., may be imported by) Test Suites and other Modules should be declared in the Module Exports proforma.

Table 19: Use of Module exports

Module Exports				
TTCN Module Name: Module_A Objective : TTCN ModuleRef : Standards Ref : PICS Ref : PIXIT Ref : Test Method(s) : Comments :				
Object Name	**Object Type**	**Source Name**	**Page Nr**	**Comments**
String5	SimpleTypeDef_Object		2	
Wait	TimerDcl_Object		3	
INTC	TTCN_PDU_Type_Object	Suite_1		
DEF1	Default_Object	Module_1		
TC_2	TestCase_Object		13	
TC_3	TestCase_Object		45	
Preamble	TestStep_Object		56	
			67	
		EXTERNAL		
Detailed Comments:				

NOTE: The Source Name column may either:

- be empty, i.e., the object is defined in this module; or
- contain a Module Identifier, i.e., the object is defined in another Module; or
- contain a Test Suite Identifier, i.e., the object is defined in another Test Suite; or
- contain the keyword EXTERNAL, i.e., The object is defined externally.

Objects that are defined in the Module, but not declared in the Module Exports table may not be imported directly although they may be used by the exported objects. For example, suppose the Module defines Test_Case_A which attaches Test_Step_B but only Test_Case_A is declared in the Module Exports table. This means that Test_Step_B is still used by Test_Case_A but cannot itself be imported from the Module.

7.5 External TTCN objects

This proforma list the objects being referenced by their identifier in a TTCN module. The object defined in the External Objects table need to be defined when importing the TTCN module.

Table 20: Use of External objects

External Objects		
Object Name	**Object Type**	**Comments**

CRC(P:A_Pdu)
CONSTRAINT_A(acstr:T_CONNECT)
TESTSTEP_A(I:INTEGER)
DEF3
	TS_Op_Object
	TTCN_PDU_Constraint_Object
	TestStop_Object
	Default_Object	

NOTE: For objects that have formal parameter lists then the list shall be provided too.

7.6 Renaming

It is possible that the identifier of an object that is imported from a module or test suite is the same as an identifier already existing in the importing instance. In this case a name clash occurs. The importing instance then needs to resolve the name clash by renaming the imported object to:

Source_Module_Or_Test_Suite_Identifier::Object_Identifier

That is, concatenation of the module or test suite identifier with the object identifier, separated by two colons. For example, with the module **Module1** and test step **Preamble1** we would get:

Module1::Preamble1.

The renaming of an object means that:

- the object definition; and/or
- the references to the object.

are renamed.

In most cases it can be expected that this re-naming will automatically be performed by the tool doing the import.

8 Other TTCN version 2 features

This clause describes various other minor, but important, features of TTCN version 2.

8.1 Grouping of constraints and other tables

It is now possible to group sets of similar tables, in the same way that Test Cases and Test Steps can be grouped in the old TTCN. This will probably be most useful for grouping PDU Type Definitions and PDU Constraint Declarations, for example.

Table 21: Example of a grouped constraint

PDU Constraint Declaration			
Constraint Name : C1 Group : B-ISDN-L3/NetworkSide/Invalid/ PDU Type : A_PDU Derivation Path : Encoding Rule Name : Encoding Variation : Comments :			
Field Name	**Field Value**	**Field Encoding**	**Comments**

The syntax for the group reference is the same as for test step groups etc. Note that the constraint name should still be unique across the entire test suite so that constraints can be referenced directly by name and not via the group path. This also means that the syntax for Derivation Path has not changed.

Note that grouping also applies to tables where in the old TTCN we would only have had one instance of a table, for example, it is now possible to have a whole set of Test Suite Parameter tables or Test Suite Constant tables. Again, bear in mind that the group mechanism is only a structuring mechanism. Names of objects that appear in the tables (e.g., a Test Suite Parameter) should still be unique across the test suite.

8.1.1 Global restrictions on groups of tables.

In several cases TTCN version 2 puts requirements on the contents of a single table that are not strictly correct when groups of tables are used. For example, the statement that a test component Declarations table should have at least one MTC and at least one PTC does not apply on a per table basis when these tables are grouped. For example, it may be desirable to put all the MTCs in one table and all the PTCs into another table. However, what is intended is that the *collection* of Test Component Declarations table has at least one MTC and at least one PTC.

Thus, in general, similar restrictions should be read to apply to the groups of a particular table as whole, that is, as if all the tables in a group were collected into a single table. Other examples of this are Encoding Definitions and Encoding Variations.

8.2 Passing Matching Symbols to constraints

Matching symbols may now be passed as actual parameters to constraints. For example:

Table 22: Use of matching symbols in a constraints reference

7		:			
8		L?A_PDU	C1(?, (1,2,3), *,(1..6), "ab?xy*z")		
9		:			

NOTE: Matching symbols may not be passed to constraints that are to be used in a send event.

8.3 Empty PDUs

PDU declarations need not have any field entries (i.e., the body of the table may be empty) if the corresponding PDU in the standard does not have any fields. Previously this applied only to ASPs. This also means that constraints for the 'empty' PDU need not be defined.

8.4 The RETURN statement

This statement may only appear in Default dynamic behaviour descriptions. It is intended to be used when incoming PDUs can occur at any time but which are not considered to be part of the test purpose and which should be ignored. A RETURN from a Default will cause processing to continue at the first alternative in the set of alternatives that caused the Default behaviour to be invoked. For example:

Table 23: Use of the RETURN statement

7		:			
8		L? PDU_1	C1		Ignore this PDU
9		RETURN			
10		?OTHERWISE		inconc	
11		:			

NOTE: Do not confuse RETURN with the RETURNVALUE keyword which is part of the procedural test suite operations syntax.

8.5 OTHERWISE and the fail verdict

It is no longer mandatory that an OTHERWISE in a Default should lead to a fail verdict.

8.6 Collective comments

Tables for multiple TTCN objects (e.g., Test Suite Parameters, Test Case Variables) may now contain additional lines called Collective Comments that can be used to group entries in the table. For example:

Table 24: Use of collective comments

Test Suite Parameter Declarations			
Parameter Name	**Type**	**PICS/PIXIT Ref**	**Comments**
The following parameters are used only in test cases for Valid behaviour			
VPAR1	INTEGER	ref1	
VPAR2	BOOLEAN	ref2	
The following parameters are used only in test cases for Invalid behaviour			
IPAR1	INTEGER	ref3	
IPAR2	BOOLEAN	ref4	

8.7 Test Suite Constants by Reference

Test Suite Constants that are defined externally in ASN.1 may now be referenced using the following proforma:

Table 25: Use of Test Suite Constant Declarations by Reference

Test Suite Constant Declarations by Reference			
Constant Name	**Type**	**Value Reference**	**Comments**
TC1	INTEGER	value_of_tc1	
TC2	BOOLEAN	value_of_tc2	

8.8 PCO Type Declarations

PCO Types should be declared in a (new) single PCO Types Declaration table. In a Test Suite this table comes before PCO Declarations.

Table 26: Declaration of Co-ordination Points

PCO Type Declarations		
Type Name	**Role**	**Comments**
TSAP	LT	
SSAP	UT	

Two different PCOs, having the same PCO Type, may now use the same ASPs and/or PDUs.

8.9 ACTIVATE statement

The ACTIVATE statement allows Defaults to be selectively activated/deactivated during test case execution. This feature is powerful and should be used with some care. See A.1.2.3 for an example of the use of ACTIVATE.

8.10 Test suite operations using the procedural definition

Table 27: Typical use and definition of a TTCN procedure

Test Suite Operation Procedural Definition	
Operation Name:	IncreaseNR (Value : INTEGER)
Result Type:	INTEGER
Comments:	Operation to increase a value by one with modulo 128
Description	

VAR INT : INTEGER
ENDVAR
BEGIN
INT := (Value + 1) MOD 128
RETURNVALUE INT
END	

Detailed Comments:	

Test Suite Operation Procedural Definition	
Operation Name:	Assign_CHI (basic_CHI, primary_CHI : CHI, basicAccess : BOOLEAN)
Result Type:	CHI
Comments:	This operation is used to assign a correct Channel identification information element to PDUs dependant on the type of access that is tested.
Description	

IF basicAccess THEN
RETURNVALUE basic_CHI
ELSE
RETURNVALUE primary_CHI
ENDIF	

Detailed Comments:	

8.11 Using ASN.1 '94

The TTCN version 2 endorses the use of ASN.1 '94 rather than earlier versions of ASN.1. This, however, is misleading as the X.680 series [3] which describes ASN.1 '94 includes several features (e.g., extensibility, information objects, parameterization etc.) which cannot easily be accommodated in the TTCN in its present form.

TR 101 295 [4] describes the transformation of these capabilities to forms which can be supported in TTCN's limited coverage of ASN.1 '94.

9 Some Do's and Don'ts when using TTCN

9.1 Use of matching symbols in receive constraints

Matching symbols are often used in the wrong way in constraints. When a matching symbol (? Or *) is used in a field of a PDU or Structured Type constraints, it means that the contents of the received buffer shall match the field type. But often the field type consists of BITSTRING, HEXSTRING or OCTETSTRING which are fitting with any kind of data.

The use of matching symbols should be restricted to particular fields of Structured Type tables.

For the fields of PDUs that are optional, it is recommended to define a Structured Type constraint containing at least values for the fixed part of the table (e.g. identifier, mandatory field) and matching symbols for the optional ones. This "dummy" Structured Type constraint can then be used instead of a "?" in the PDU constraint. To replace a "*"an IF_PRESENT can be added to the "dummy" constraint reference. For example:

Table 28: Use of wildcards and IF_PRESENT

PDU Constraint Declaration		
Constraint Name : RL_R1(FLAG: INTEGER; CALL_REF: CALL_REF_TYPE) PDU Type : RELEASE_PDU Derivation Path : Comments : Receive PDU with "don't care values"		
Field Name	**Field Value**	**Comments**
pd cr mt cau …	PROTOCOL_DISCRIMINATOR_Q931 CR1(FLAG,CALL_REF) MT_RELEASE **CAU_R_DUMMY IF_PRESENT**	

Table 28a

Structuired Type Constraint Declaration		
Constraint Name : CAU_R_DUMMY PDU Type : CAU Derivation Path : Comments : Receive constraint to match any Cause IE		
Field Name	**Field Value**	**Comments**
cau_i	'00001000'B	
cau_l	'0000????'B	length < 32
cau_e3_eb	?	
cau_e3_cs	'000'B	
cau_e3_loc	?	
cau_e4_rec	*	
cau_e5_eb	'1'B	
cau_e5_cv	?	
cau_di	*	

9.2 Restriction on using receive events

A RECEIVE, OTHERWISE or TIMEOUT event line shall only be followed by other RECEIVE, OTHERWISE and TIMEOUT event lines in a set of alternatives, even through an ATTACH construct.

1 L?PDUr;

2 +STEP;

3 ?TIMEOUT TAC.

STEP shall only contain RECEIVE, OTHERWISE and TIMEOUT event lines on its first set of alternatives.

9.3 First set of alternatives in default.

As a consequence of the restrictions on using receive events, the default trees shall contain only RECEIVE, OTHERWISE and TIMEOUT event lines on the first set of alternatives.

9.4 Declaration part and protocol standard

The declaration part shall reflect the table structure of the protocol specification. Thus, the Structured Type, ASP and PDU declaration shall use the same fields as in the protocol specification.

9.5 Declaration part and transfer syntax

Data is transmitted between communication entities as bit streams. Often the data stream should fit in an octet string format. This requirement is not only the matter of the test equipment but shall also be fully specified in the ATS.

The use of the INTEGER type could be convenient, e.g. for counter field and brings a better overview in the dynamic part. However it is strongly recommended to translate these INTEGER value into a BITSTRING value before to include in the corresponding constraints.

EXAMPLE:

Table 29: Use of TTCN predefined operation INT_TO_BIT

Structured Type Constraint		
Constraint Name:	CST1(CST_VAL: INTEGER)	
Structured Type:	CST	
Derivation Path:		
Comments:	Constraint with a parametrized Call state value used for sending and receiving.	
Element Name	**Element Value**	**Comments**
cst_i	ID_CST	
cst_l	'00000001'B	
cst_cs	'00'B	
cst_csv	**INT_TO_BIT(CST_VAL,6)**	
Detailed Comments:		

If ASN.1 is used without BER (direct encoding), the encoding format of an INTEGER should be specified, for example

SequenceNumberType ::= INTEGER(0..7) – to be encoded as bitstring of fixed length 3

9.6 Timers

ATSs can use operational timers (e.g. to fix a delay when a specific receive event is expected). These timers have no corresponding value in the protocol specification. Instead of fixing the value by a constant in the timer declaration, it is recommended to use a corresponding a test suite parameter (PIXIT).

9.7 Wild cards in send constraints

Send constraints should not include wildcards ("?" or "*") unless the corresponding fields are assigned a value in the dynamic behaviour. However it is strongly recommended not to use wildcards in send constraints but to pass the values over formal parameter in order to increase the overview.

9.8 Use of the metatype PDU

Parameters or fields of ASP and PDU type definitions can be specified as being of metatype PDU. Then in a corresponding constraint the value for that parameter or field should be specified as the name of a PDU constraint, or formal parameter.

According to the STATIC SEMANTIC, no reference shall be made (e.g. with assignment in the dynamic behaviour) to fields of the corresponding substructure.

EXAMPLE:

Table 30: typical use of the PDU metatype

ASP Type Definition		
ASP Name:	DL_DAT_IN (DL-DATA-INDICATION)	
PCO Type:	SAP	
Comments:	CEId: = (SAPI,CES) mapped onto DLCI: = (SAPI,TEI) This ASP is used to indicate the receipt of layer 3 PDUs using acknowledged operation (L2 ---> L3).	
Parameter Name	**Parameter Type**	**Comments**
mun	PDU	
Detailed Comments:		

Table 30a

ASP Constraint Declaration		
Constraint Name:	Mr(PARAM: PDU)	
ASP Type:	DL_DAT_IN	
Derivation Path:		
Comments:	ASP to indicate the receipt of layer 3 messages.	
Parameter Name	**Parameter Value**	**Comments**
mun	PARAM	
Detailed Comments:		

Table 30b

PDU Constraint Declaration		
Constraint Name:	SU_R1	
PDU Type:	SETUP_PDU	
Derivation Path:		
Comments:	Send PDU	
Field Name	**Field Value**	**Comments**
pd	PROTOCOL_DISCRIMINATOR_Q931	
cr	?	
mt	MT_SETUP	
…	…	…
Detailed Comments:	This PDU is used as base constraint for all SETUP messages to be sent.	

With the constraints above the following assignment is not allowed:

| 1 | L?DL_DAT_IN (CREF := **DL_DAT_IN.mun.cr**) | Ms(SU_R1) |

Actually the field **.cr** is not known in the data structure of DL_DAT_IN as a metatype PDU is only generic.

9.9 Page number in the test suite overview

The inclusion of the page numbers in the overview part is optional. However this information is very useful for the future ATS users and could be systematically included in the graphical form (paper version).

9.10 Test case end

The test case shall end leaving the IUT in a stable state from the protocol point of view. Not doing this could prevent the execution of further test cases unless the correct initial test conditions are reached.

9.11 ASN.1 definitions in TTCN

The ASN.1 definitions used in TTCN should be as simple as possible. Unnecessarily complicated structures should be avoided, for example a CHOICE with one item:

```
CHOICE  { unexpectedComponentSequence [3] IMPLICIT SEQUENCE { .. }
        }
```

is more simply written as:

```
      [3] IMPLICIT SEQUENCE { .. }
```

Explicit tagging should be avoided in ASN.1 test suite parameters, for example:

```
 $ASN1_TypeDefinition
    [1] IMPLICIT SEQUENCE {
      informationToSend [0] InformationToSend,
      ...
    }

    $ASN1_ConsValue
    {
      informationToSend TSPX_information_to_send,
      ...
    }
```

where a better rendition would be:

```
    $ASN1_ConsValue
    {
      informationToSend [0] TSPX_information_to_send,
      ...
    }
```

Often protocol specifications use macros (e.g. ROSE) for the ASN.1 coding requirements. The use of type definitions like ANY DEFINED BY type in these MACRO prevent the test tools from an automatic macro expansion. Thus it is strongly recommended to expanded the macro manually, according to the protocol specification, before including the resulting definitions in TTCN.

```
UserUserService_InvokeComponent ::= SEQUENCE {
  invokeID                     InvokeIDType,
  operation_value              Operation,
  argument                     Argument}

Argument ::= SEQUENCE { service  [1] IMPLICIT Service,
                        prefered [2] IMPLICIT Prefered}
```

is better than:

```
UserUserService_InvokeComponent ::= SEQUENCE {
  invokeID                     InvokeIDType,
  operation_value              Operation,
  argument                     ANY DEFINED BY operation_value}
```

9.12 Some common errors to avoid

The following is a list of frequently encountered errors found in test suites which should be avoided:

- RECEIVE, TIMEOUT and OTHERWISE events are used together with other event types (e.g. pseudo event, assignment, …);
- timers are not cancelled when the TIMEOUT branch is left before the timer expiration;
- timers are defined with constant values in the timer declaration, where a variable timer value is needed (value not defined in a protocol specification). Use a Test Suite Parameter instead;
- wildcards are used in send constraints without corresponding assignment in the dynamic behaviour;
- test Suite Parameters are defined without any PICS/PIXIT reference;
- reference are made to fields (e.g. assignment in the dynamic behaviour) of a substructure of a field defined as metatype PDU;
- sometimes ATS are distributed using a non-standardized MP format (e.g. editor specific format like ITEX);
- some postambles leave the IUT in an unstable state, provoking an error in the following test case execution;
- some possible events are not always expected in the default behaviour description (e.g., TIMEOUT, OTHERWISE are often missing);
- two successive receive events in the behaviour description with the second message embedded in the first one (an INAP message in a TCAP message) that actually correspond to only one event in the PCO.

EXAMPLE:

```
L?TCAP_message;
L?INAP_message,
```

with the INAP message being embedded in the TCAP message which consist of the actual PCO event.

Annex A (normative): Some examples of the use of TTCN

A.1 Concurrent TTCN

A.1.1 Example of EN 300 403-7: ATS&PIXIT for DSS1 layer 3 network

In the following example, the ATS uses a test configuration with a MTC connected to the IUT by one PCO. Two PTC are connected by remote access in order to initiate or to answer calls related to the behaviour of the MTC.

A.1.1.1 Description of ATM used

The requirement for testing the network IUT is to focus on the behaviour of the network IUT at the user-network interface where a T reference point or coincident S and T reference point applies. Thus the IUT is the network DSS1 protocol entity at a particular user-network interface and is not the whole network.

It is possible to specify an ATS based on a Single party (remote) test method for such an IUT. However, it is considered that an ATS based on such an approach is of limited use as the only way to specify IUT generated PDUs is to use the "implicit send" statement. Many users of such an ATS would replace the "implicit send" statements with descriptions of the behaviour at other interfaces.

An ATS based on a multi-party test method is considered to be more useful in that it is closer to how a real test suite would be constructed. Such a test method specifies behaviour at multiple network interfaces. One very important limitation here is that tests are focused on one particular interface. Thus the test system is made up one Main Test Component (MTC) and one or more Parallel Test Components (PTC), see figure A.1.

A.1.1.2 Conventions for test components and PCOs

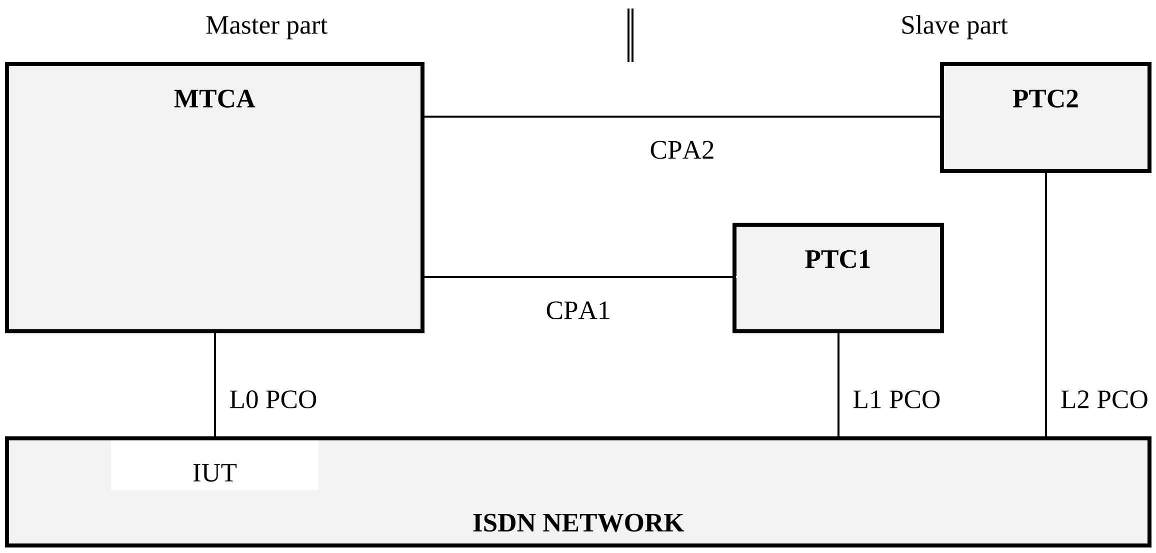

Figure A.1: test architecture

In a master/slave arrangement, the MTC is considered to be the master while the PTCs are the slaves. The "slave" testers are only an explicit description of how to deal with the "other" interfaces during the testing process, i.e. "how to make the IUT send the required message".

This means, in particular, that the verdict will only be assigned from the protocol aspects observed on *the* interface under test (i.e. by the "master" tester), as it would be observed by a terminal connected to this interface. A failure in the correlation between the protocol at the different interfaces to which the different testers are connected, i.e. in the mechanism of the functional service itself, will not cause a FAIL verdict. For instance, if the IUT fails to send a message on the tested interface after another interface has received the proper stimulus, the verdict will be INCONCLUSIVE.

The MTC MTCA has two functions in this configuration. Firstly, it has the MTC function of controlling the one or more PTCs. Thus it is responsible for starting the PTCs and afterwards co-ordinates activities by exchanging Co-ordination Messages (CM) with the PTCs. Secondly it is responsible for the behaviour of the Lower Tester (LT) at PCO L0.

A combination of the remote and multi-party test methods is applied. As can be seen from figure A.1, several PCOs are used. All PCOs reside at the service access points between layers 2 and 3.

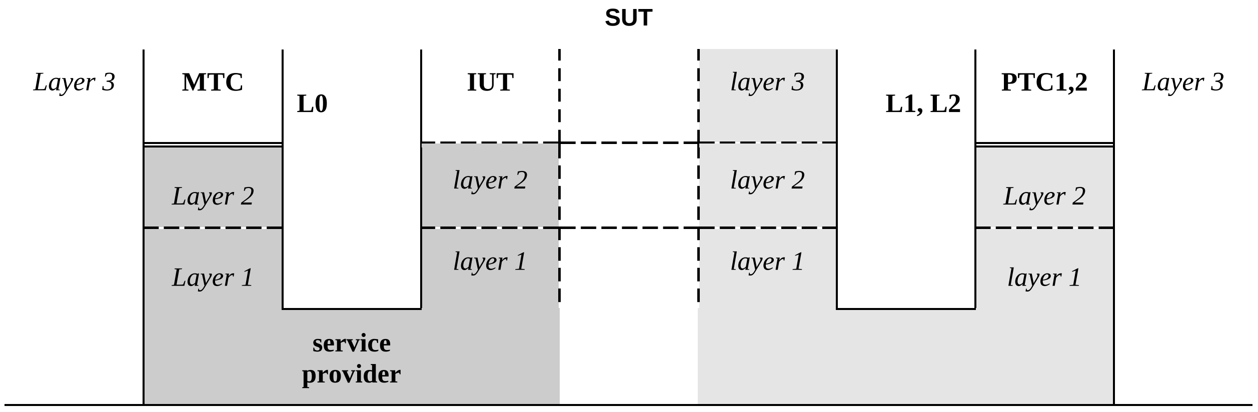

Figure A.2: Combination of the remote and multi-party test methods

The MTC PCO is named "L0" ("L" for Lower). The L0 PCO is used to control and observe the behaviour of the IUT and test case verdicts are assigned depending on the behaviour observed at this PCO. The PTCs PTC1, PTC2 etc. use PCOs L1, L2 etc. These PCOs are used to control and, in a limited way, observe the behaviour of the network equipment at interfaces other than the one under test. No verdicts are assigned at these PCOs.

As stated in a previous paragraph, the non-receipt of network generated messages at L0, which are stimulated by events at the L1, L2 etc., will result in INCONCLUSIVE rather than FAIL verdicts being assigned.

PTC2 is only activated in a small set of test cases that test the handling of two calls at one time. In test cases which verify that the IUT rejects invalid or unacceptable SETUP messages, no PTC is activated at all, as these rejection procedures are considered local to the access between IUT and MTC.

The capability of the IUT to send INFORMATION and PROGRESS messages is tested in different call states. Implicit send events have to be used in this small set of test cases, as the sending of those messages cannot be triggered via a PTC. Separate PIXIT questions are asked for each call state, if and how it is possible for the test operator to cause the sending of the messages.

A.1.1.3 The Test Component Configuration Declaration

Table A.1: Defining various test configurations

Test Component Configuration Declaration			
Configuration Name : CONFIG2 Comments : test configuration with 2 remote access.			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
MTCA PTC1 PTC2	L0 L1 L2	CPA1, CPA2 CPA1 CPA2	

A.1.1.4 Example of a test case

Table A.2: Using the CREATE statement

Test Case Dynamic Behaviour		
Test Case Name	:	L3N_N10O_V_006
Group	:	N10_Outgoing/Valid
Purpose	:	Ensure that the IUT in the Active call state N10, on receipt of a SUSPEND message indicating in the Call identity information element a call identity that is already in use, sends a SUSPEND REJECT message with a Cause information element indicating the cause value 84 "call identity in use" and remains in the Active call state N10.
Configuration	:	CONFIG2
Defaults	:	
Comments	:	

Nr	L	Behaviour Description	Constraint Ref	V	Comments
1		CREATE (PTC1 : PTC1_OUT)			1)
		CREATE (PTC2 : PTC2_OUT)			2)
		..			
		L0!PDUs	Ms(..)		3)
		L0?PDUr	Mr(..)		3)
		..			
		CPA1!CM	S_DISCONNECT		4)
		..			
Detailed comments :					

Comments:

1) This CREATE statement launch the PTC1 test component that executes the test step PTC1_OUT.
2) This CREATE statement launch the PTC1 test component that executes the test step PTC1_OUT.
3) PDU are sent and received by the MTC over the PCO L0.
4) The MTC expect the receipt of a DISCONNECT message. The sending of the corresponding message from the IUT is initiated by sending a DISCONNECT message to the IUT from the remote access that is connected to the PTC. A co-ordination message of type CM is sent to PTC1 over the co-ordination point CPA1 to initiate the sending of the DISCONNECT message.

NOTE: A co-ordination message could also be used to exchange values between test components.

A.1.1.5 Example of a co-ordination message

In the example above, the co-ordination message is used to trigger off the sending or the receipt of PDU between test components. Their content is not relevant as it only enables the recognition of the co-ordination message itself.

Table A.3: Definition of CMs

CM Type Definition		
CM Name	:	CP_M
Comments	:	coordination message

Parameter Name	Parameter Type	Comments
CM_content	IA5String [0 TO 26]	message content in clear text

Table A.3a

CM Constraint Declaration		
Constraint Name : S_DISCONNECT CM Type : CP_M Derivation Path : Comments :		
Parameter Name	**Parameter Value**	**Comments**
CM_content	"SEND_DISCONNECT"	

In this example, the parameter value is not significant from the formal point of view as it only serve to trigger off the receipt of the co-ordination message.

A.1.2 Example of ETS 300 806-2: ATS&PIXIT for the Generic Functional Protocol

This example is taken from for ATS for Generic Functional Protocol of the Inter-exchange signalling protocol for Private Integrated Services Network ETS 300 806-2 [5].

Contrary to the example above, in this ATS the MTC, when the multi-party configuration applies, do not use any PCO, but synchronize two PTC at which a PCO is connected.

A.1.2.1 Test Configurations and use of Concurrent TTCN

As this ATS covers both single-party testing using non-concurrent TTCN, and multi-party testing using concurrent TTCN, the notation chosen for the complete ATS is the concurrent TTCN syntax. Therefore, test components are defined to describe the two configurations: the "mono" configuration, and the "transit" configuration, as shown in figures A.5 and A.6.

The mono configuration is used in case of single-party testing, i.e. for the following:

- ROSE testing;
- Co-ordination Function testing;
- Protocol Control testing;

for End PINX and Transit PINX (single Transit interface active).

Only one test component, which is the Master Test Component MTC_MONO, connected to the IUT via the PCO LX, is needed in this case.

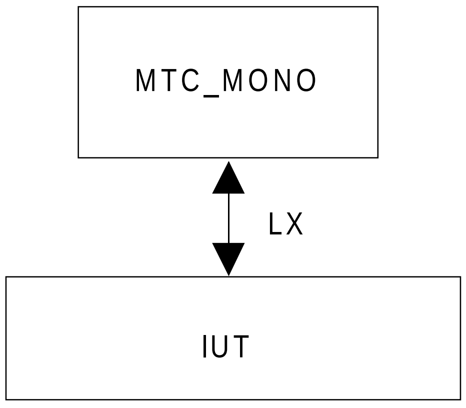

Figure A.3: Mono configuration

The transit configuration is used in case of multi-party testing, i.e. for the following:

- ROSE testing;
- Co-ordination Function testing;
- Protocol Control testing;

for Transit PINX (dual Transit interfaces active).

In this case, three test components are needed, these are the Master Test Component MTC_TRANSIT, and the two Parallel Test Components PTC_X and PTC_Y, which are connected to the Master Test Component via the two Co-ordination Points CPX and CPY. PTC_X and PTC_Y are further connected to the IUT via the two PCOs LX and LY.

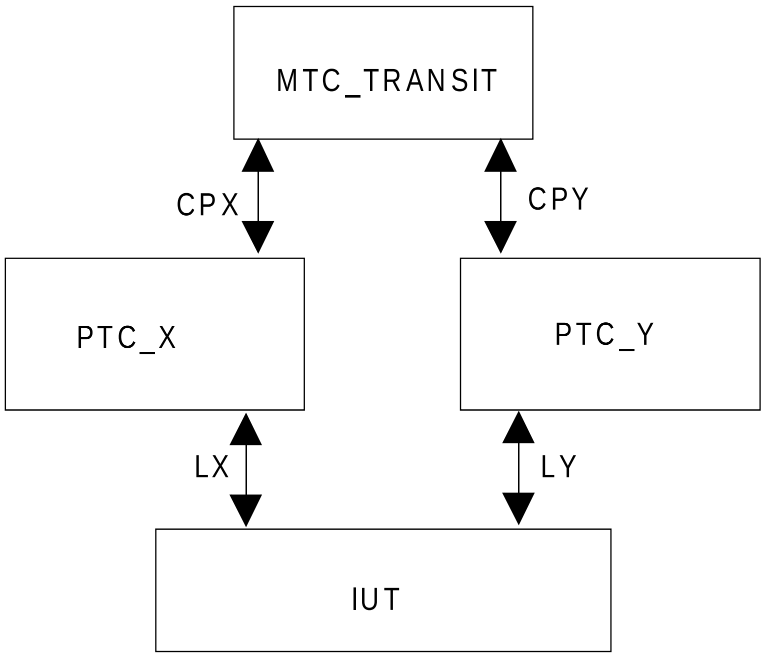

Figure A.4: Transit configuration

A.1.2.2 Test Component Configuration Declaration for the Transit configuration

Table A.4: Declaration of the Transit test configurations

Test Component Configuration Declaration			
Configuration Name : Config_Transit Comments : used for tarnsit test cases.			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
MTC_TRANSIT PTCX PTCY	 LX LY	CPX, CPY CPX CPY	

A.1.2.3 Example of a test case for Transit Configuration

Table A.5: Concurrent Test Case

Test Case Dynamic Behaviour					
Test Case Name : TC1068t Group : GFP/CR/COTA/BV/ Purpose : Ensure that the IUT as a Transit PINX, in state TCC_Idle, on receiving a SETUP message on interface X, containing a Facility IE which is to be relayed, sends a SETUP message on interface Y containing this Facility IE Configuration : Config_Transit Defaults : def_mtc					
Nr	**L**	**Behaviour Description**	**Constraint Ref**	**V**	**Comments**
		CREATE(PTCX:SUBTREE_X,PTCY:SUBTREE_Y)			1)
		+mtc_sync			2)
		..			
		?DONE(PTCX,PTCY)			3)
		..			
		SUBTREE_X			4)
		ACTIVATE(def_ptcx)			5)
		LX!SETUPrq	DLRQ(SET37(..))		6)
		..			
		SUBTREE_Y			4)
		ACTIVATE(def_ptcy)			5)
		LY?SETUPin	DLIN		6)
		..			

Comments:

1) This CREATE statement launch the PTCX and PTCY test components which executes respectively the local trees SUBTRE_X and SUBTREE_Y.

2) The mtc_sync test step contains events that consist of exchanging co-ordination messages in order to synchronize the actions of PTCX and PTCY.

3) This DONE event waits for the end of PTCX and PTCY before to execute the next event.

4) The local tree SUBTREE_X (SUBTREE_Y) is executed by PTCX (PTC_Y). The definition as local tree increases the readability of the of the test case from the point of view of the complete environment. It could be replaced by a test step if the behaviour is comment to several test cases

5) The role of the ACTIVATE operation is to apply the given tree as default tree from the point where the operation is called. In this example, the default tree for each PTC is different from the MTC default reference, so that each PTC is applied a specific default tree reference by the ACTIVATE operation.

6) A SETUP message is sent from PTCX to the IUT over the PCO LX, that should result in the receipt of a corresponding SETUP message over the PCO LY at PTCY.

A.2 Index table of TTCN topics

To illustrate the different TTCN topics, the following ATS from ETSI has been chosen because it contains good examples for the major TTCN parts:

- EN 300 286-6 - ATS for the UUS DSS1 [6] supplementary service, network side.

NOTE: Can be found in the free downloadable area on the ETSI server.

Examples on a specific TTCN topic can be found by referencing the page number (of the TTCN GR of EN 300 286-6 [6]) given in table A.6.

Table A.6: Index of the TTCN topics in EN 300 286-6 [6]

TTCN topics	[Item name -] Page Nr
Simple types definitions with: value list, value range, length, length range.	p 20
Structured Type Definition.	p 24 to 31
ASN.1 type definition for components defined in ASN.1 in the protocol specification.	p 32 to 37
ASN.1 type definition for an information element (Facility) defined in tabular form in the protocol specification.	p 33
Test suite operation definition with operation description in text.	p39
Test suite parameter related to a PICS item.	p 40
Test suite parameter related to a PIXIT item.	p 41
Test case selection expression.	p 43
Timer declaration using test suite parameter for the duration.	p 50
Test component declarations and configuration (1 PCO for the MTC, and 2 PTCs).	p 51, 52
Tabular ASP type definitions with and without parameter.	p 53 to 66
Tabular PDU type definitions, some containing a field defined in ASN.1	p 67 to 79
CM (co-ordination message) type definitions containing only a "text" (IA5String type) message to synchronize test components.	CP_M - p 80
CM (co-ordination message) type definitions containing also integer fields to pass values between test components.	CP_M_CAU_ERR - p 81
Alias definitions	p 82
Structured type constraints using direct values and Test Suite Parameters.	BCAP20 - p 84
Structured type constraints using matching symbols, an operation and a formal parameter.	CAU4 - p 87
Structured type constraints using IF_PRESENT.	CHI3p - p91
Tabular PDU constraint declaration using formal parameter, Structured Type Constraint reference, Test suite Operation, and containing a field defined in ASN.1	AL20 - p 128
Tabular CM constraint declaration using formal parameters which allow to pass integer values between test components.	R_DI_CAU_ERRs - p 203
A test case using the following TTCN features – a PTC of which behaviour description is defined in a test step, – a constraint reference is passed to the PTC in the CREATE statement, – co-ordination messages to synchronize the test components	UUS_N01_001 - p 219
Test step to be executed by a PTC, containing co-ordination messages	PTC1_wait_msg - p 489
Defaults dynamic behaviour table using the return statement	DF69901 - p 540

Annex B (normative): Application of TTCN to other types of testing

B.1 Introduction

This annex looks at the application of TTCN to other types of testing and identifies possible new requirements for future versions of TTCN. By the fulfilment of these requirements the scope of TTCN would be opened towards

- the use of TTCN for the specification of test cases for non-OSI compliant applications, e.g., CORBA based systems; and
- the application of TTCN for testing non functional properties.

This annex is restricted to discussing issues that have been raised by ETSI TC MTS and/or regular TTCN users within the ETSI membership and is not intended to be a complete study of the subject.

This annex is structured as follows: In clause B.2 application specific requirements are analysed. The requirements are driven by the fast dissemination of CORBA based applications. In clause B.3 the use of TTCN for performance testing is analysed. In clause B.4 TTCN based real-time testing is discussed. Architectural issues related to the requirements on TTCN are discussed in clause B.5. Finally, in clause B.6 a possible strategy for the introduction of new concepts is described.

B.2 Application specific requirements

B.2.1 CORBA

The use of CORBA (see bibliography 3) and IDL (see bibliography 3) is becoming of increasing relevance to ETSI. With respect to the relation of CORBA and IDL to TTCN two main areas of interest have been identified:

- dynamic test configurations; and
- mapping rules from IDL to TTCN.

NOTE: IDL has been defined by the Object Management Group in the context of CORBA.

B.2.1.1 Dynamic test configurations

The only thing that restricts TTCN test methods/architectures from being applied to CORBA is the test configuration, which is specified as a flat 'tree' of connected test components. The test configuration for each test case is static and fixed.

TTCN operational semantics require that only a Main Test Component be able to start a Parallel Test Component, which simply means that there can be only one main (MTC) process and as many as the number of PTCs of parallel sub-processes running at the same time. The PTC processes can communicate with each other and with the MTC process, but a PTC process can not activate any other PTC process.

This will restrict TTCN from being used to test situations such as when a major process activates one sub-process and this sub-process activates another sub-process and so on. It is suggested that the operational semantics of TTCN be extended so that any PTC process can activate another PTC process, but can not recursively activate itself either directly or indirectly. With this extension, TTCN will be able to apply to any protocol.

NOTE: The lack of dynamic test configurations in TTCN seems to be a more general problem for testing applications on architectures which cannot be directly mapped onto the OSI reference model. Dynamic test configurations are also needed for performance testing architectures (see clause B.3). The problem should be studied generally, not only in the context of CORBA.

B.2.1.2 IDL and TTCN mapping

A major problem in testing CORBA based applications with TTCN is that there is no IDL-TTCN mapping. The mapping proposed by Conformance Texting of TINA Service Components (see bibliography 4) as a result of TTCN based testing of TINA services implemented on CORBA platforms is summarized below:

- The IDL module provides a name scope and a mechanism to group interfaces. Due to the fact that name scoping is not supported by any tool. It has to be checked whether this can be represented (modelled) adequately by using modular TTCN.
- For each IDL interface type, a separate test group is introduced to structure the TTCN test suite in accordance with the interface structure of the tested TINA service component. For each TINA service component with multiple interfaces a test group is introduced which contains sub test groups, one for each provided interface.
- In contrast to IDL, interface inheritance is not supported by TTCN. Therefore, everything to be inherited will be duplicated for the derived type.
- TTCN uses asynchronous communication. Synchronous IDL operations may be modelled by using two ASP types.
  - an ASP type for operation requests; and
  - an ASP for replies of the operation.

  Name prefixes may improve the readability of the test suite, e.g., `sCALL_operationName` for an operation call and `sREPLY_operationName` for the answer of an operation.
- IDL exceptions may occur during the invocation or execution of an operation. Such an exception may also be mapped onto an ASP type. A special name prefix, e.g., `pRAISE_operationName`, denoting that the ASP represents an exception, may increase the readability of the test suite.
- An IDL attribute definition is logically equivalent to a pair of access functions: *get* to retrieve a value, *set* to change or set a value:
  - *set* is mapped to a one-way operation, i.e., modelled as one ASP type,
  - *get* is mapped to two ASP types, one (without parameters) for the request, one (with parameters) to retrieve the result.
- For *read-only* attributes only, the *get* function has to be defined.
- Most IDL basic types are mapped onto ASN.1 types of the TTCN test suite. The types *float* and *double* cannot be mapped, because these types are not supported by TTCN. IDL *typedef* are translated to TTCN ASN.1 type definitions.
- IDL constants are mapped to TTCN constant declarations.

NOTE: With the upcoming importance of CORBA, ETSI and ETSI members will be confronted with IDL specifications and with the test of interfaces described by means of IDL. The relation between TTCN and IDL has therefore to be studied in more detail. The first conclusion is that an IDL to TTCN mapping and vice versa is possible.

B.2.1.3 Dynamic parts of CORBA test cases

The IDL mapping rules only describe the mapping of the static parts of a TTCN test suite. Besides the dynamic test configurations, the TTCN test case dynamic behaviour tables seem to be applicable for the test specification of CORBA based applications.

B.2.2 Database applications

TTCN could also be used for functionality testing and for database application testing. There is no standard testing language/notation for database application at all. Database application test cases are written in an informal manner. While most test cases for database applications, such as Oracle forms, are nothing but the specification of database tables, columns, events, timing and verdicts of testing and they are part of OSI layer 7 'protocol'. All of these can easily be described by TTCN.

NOTE: While this is a completely new application area for TTCN it is probably outside the scope of the ETSI work.

B.3 TTCN based performance testing

The text in this clause mainly is based on the 'ATM Forum Performance Testing Specification - Baseline Text' (see bibliography 1) and on a non-standardized TTCN extension for performance testing, called PerfTTCN, which has been developed at the GMD in Berlin (see bibliography 5 and 6).

B.3.1 Goal of performance testing

The goal of performance testing is to measure the level of (performance-) quality of an implementation under test (IUT) under well-known conditions. The level of quality can be expressed in form of metrics such as latency, effective throughput or end-to-end delay. The well-known conditions are related to the reproduction of test results and to the fact that performance testing normally is carried out in normal and overload situations of the IUT.

B.3.2 Relations to Quality of Service metrics

Performance quality is not equivalent to Quality of Service. Most of the QoS metrics, such as cell transfer delay, cell delay variation, cell loss ratio (CLR), and so on, may or may not be reflected directly in the performance perceived by the user. For example, consider the comparison of two switches: If one gives a CLR of 0,1 % and a frame loss ratio of 0,1 % while the other gives a CLR of 1 % but a frame loss ratio of 0,05 %, the second switch will be considered superior by many users.

B.3.3 Relations to conformance testing

Conformance testing can be seen as a pre-requisite for performance testing because errors in the functionality should have no impact on the performance measurements. However, overload may degrade the functional behaviour and therefore test results of performance measurements in overload situations have to be analysed and interpreted carefully.

In contrast to conformance testing, the performance characteristics of the used equipment have big impact on the test results, e.g., testing the performance of the same application over TCP/IP and over an ATM may lead to totally different results. This means performance test suites will impose (performance-) requirements on the environment in which the IUT is tested, i.e., a performance test suite may be tailored to a specific environment of the IUT. Note that in conformance testing, an ATS requires the correct functional behaviour of the components in which the IUT may be embedded and the underlying service over which the IUT is tested.

B.3.4 Metrics to be measured

Performance testing is related to some metrics which should be measured. For ATM networks, these metrics are defined in (see bibliography 1). The content of the present document is restricted to the native ATM layer and the adaptation layer. All work related to performance measurements of the higher layers is being deferred for further study.

For performance measurement, we have to distinguish between foreground traffic and background traffic. Foreground traffic refers to the traffic whose performance is measured. Background traffic describes the traffic used to set the SUT to load situations. Performance tests can be conducted with and without background traffic.

NOTE: In ATM Forum Performance Testing Specification (see bibliography 1) the description of performance metrics, the abbreviation SUT (System Under Test) refers to an ATM switch.

The metrics defined in ATM Forum Performance Testing Specification (see bibliography 1) are throughput, frame latency, throughput fairness, frame loss ratio, maximum fame burst size, and call establishment latency. In the following, these performance metrics are briefly described. This summary provides the general information and abstracts from ATM characteristics. Note that a test specification language for performance testing should be able to describe such metrics.

B.3.4.1 Throughput

Throughput defines the number of information units (e.g. bits, bytes or ATM frames) transported in a certain amount of time. At frame level, there are three throughput metrics that are of interest to an ATM user:

1) **Loss-less throughput:** Loss-less throughput is the maximum rate at which none of the offered frames is dropped by the SUT.

2) **Peak throughput:** Peak throughput is the maximum rate at which the SUT operates regardless of frames dropped. The maximum rate can occur when the loss is not zero.

3) **Full-load throughput:** Full-load throughput is the rate at which the SUT operates when the input links are loaded at 100 % of their capacity.

Throughput measurements should be provided in effective bits/sec, counting only bits from frames excluding the overhead introduced by the ATM technology and transmission systems. This is preferred over the measurement of frames/sec or cells/sec. Frames/sec requires specifying the frame size; cells/sec is not a good unit for frame-level performance since cells aren't seen by the user.

Throughput tests require several test runs during which the relevant bits are counted. The preamble of a throughput test includes not only steps to bring the system into a situation where frames are exchanged, but also into the condition where the throughput rate to be checked can be measured. For example, for testing the loss-less throughput rate we have at first to set-up the required foreground traffic. A test run ends and the foreground traffic is stopped when the average cell transfer delay has not significantly changed (not more than 5 %) during a period of at least 5 minutes.

In ATM Forum Performance Testing Specification (see bibliography 1), only the measurement procedures for tests without background traffic are defined. Measurements of throughput with background traffic are under study.

For the foreground traffic, the throughput measurement may not only take place at one input and one output channel. There are several connection configurations defined. For *n* input/output channels we can distinguish the connection configurations:

- *n*-to-*n* straight;
- *n*-to-(n–1) full cross;
- *n*-to-*m* partial cross, ($1 \le m \le n-1$);
- *k*-to-1 ($1 < k < n$);
- 1-to-(n–1) multicast;
- *n*-to-(n–1) multicast.

These connection configurations may be comparable to the abstract test methods in ISO/IEC 9646.

B.3.4.2 Frame latency

Frame latency for an SUT is measured by using a *'Message-In Message-Out'* (MIMO) definition. One MIMO latency definition is:

$$\text{MIMO latency} = \begin{cases} \text{LILO latency} & \text{if Input Link Rate} \leq \text{Output Link Rate} \\ \text{FILO latency} - \text{NFOT} & \text{otherwise} \end{cases}$$

NOTE: ATM Forum Performance Testing Specification (see bibliography 1) also presents a second definition.

where:

- LILO latency = Time between the last-bit entry and the last-bit exit
- FILO latency = Time between the first-bit entry and the last-bit exit
- NFOT = Nominal Frame Output Time, defined as the time a frame needs to pass through the *zero-delay switch*

In case that the Input Link Rate is smaller than the Output Link Rate, the MIMO latency for a given frame can be calculated by recording the time when the last-bit of the frame enters into the SUT and when the last-bit of the frame exits from the SUT. A detailed description of the second case, a comprehensive explanation of MIMO latency and its justification can be found in ATM Forum Performance Testing Specification (see bibliography 1). Frame latency should be defined in µsec.

For the given foreground and background traffic, the required times and/or delays needed for MIMO latency calculation, are recorded for several frames. Based on these values, statistical values like mean *MIMO latency*, *standard error* and $100(1-\alpha)$-*percent confidence interval* should be calculated.

Frame latency is dependent on foreground and background traffic. Background traffic characteristics that affect frame latency are the type of background VCCs, connection configuration, service class, arrival pattern, frame length and input rate.

In the scope of the present document, the details are not interesting. It should only be noted that conformance testing and TTCN cannot cope with statistical values and background traffic. Even if the times for the occurrence of the bits of a frame can be measured by using TTCN, their interpretation and the determination of a test verdict (based on statistical values) is outside the scope of conformance testing.

B.3.4.3 Throughput fairness

The two throughput fairness metrics that are of interest to users are:

1) **Peak throughput fairness:** the fairness at a frame load for the peak throughput.

2) **Full-load throughput fairness:** the fairness at a frame load for the full-load throughput.

The procedures to measure the throughput fairness are performed similarly to the throughput measurements described in subclause B.3.4.1.

B.3.4.4 Frame Loss Ratio

Frame loss ratio is defined as the fraction of frames that are not forwarded by an SUT due to lack of resources. The two frame loss ratio metrics which are of interest to a user are:

1) **Peak throughput frame loss ratio:** the frame loss ratio at a frame load for the peak throughput.

2) **Full-load throughput frame loss ratio:** the frame loss ratio at a frame load for the full-load throughput.

The frame loss ratio metric is related to throughput:

Frame Loss Ratio = (Input Rate – Throughput)/Input Rate

For calculating the frame loss ratios no special measurements are necessary. The ratios can be calculated from throughput measurements of subclause B.3.4.1.

B.3.4.5 Maximum Frame Burst Size

Maximum Frame Burst Size (MFBS) is the maximum number of frames that each of the source end systems can send at the peak rate through the SUT without incurring any losses. MFBS measures the data buffering capability and its ability to handle back-to-back frames. MFBS should be expressed in octets of AAL payload fields.

MFBS is measured for a k-to-1 connection configuration as was briefly sketched in subclause B.3.4.1. The measurement procedure requires a number of tests. Each test includes simultaneous generations of fixed lengths of bursts through all k channels and counting of all cells transmitted by the SUT. If there is no loss of cells, the length of bursts is increased, but in case of loss, the length of bursts is decreased. The next test is performed with the new burst length. The procedure is finished when the MFBS is found. The tests are conducted without any background traffic.

B.3.4.6 Call Establishment Latency

Call establishment latency is an important part of the user perceived performance. Informally it can be defined by the time between submission of a *setup* message to a network and the reception of the *connect* message from the network. The time lost at the destination while the destination was deciding whether to accept the call is not under network control and is not included in the call establishment latency. This means:

Call Establishment Latency = MIMO Latency for *setup* message
+ MIMO Latency for the corresponding *connect* message

The procedures for measurement and calculation of latency metrics have been sketched in subclause B.3.4.2 and should not be repeated here.

B.3.5 Performance TTCN (PerfTTCN)

Performance TTCN (PerfTTCN) (see bibliography 5 and 6) is a performance extension of TTCN. The aim of PerfTTCN is to allow the specification of performance tests in an unambiguous and reusable way with the benefit of making performance test results comparable by using a TTCN oriented notation. In the following the performance specific TTCN extensions of PerfTTCN are briefly described.

B.3.5.1 Concepts of PerfTTCN

The concepts of the approach are related to test components, performance test configurations, measurements, and test behaviour.

B.3.5.1.1 PerfTTCN Test components

A performance test consists of distributed *foreground test components, background test components* and a *main test component* which co-ordinates the foreground and background test components. Foreground test components realize the communication with the IUT. Background test components generate continuous streams of data to cause load for the IUT. Background test components do not communicate directly with the IUT. They only influence the IUT implicitly by bringing the IUT into normal and overload situations.

Traffic models describe traffic patterns for continuous streams and data packets with varying inter-arrival times and varying packet length. Due to their generosity and efficiency, Markov Modulated Poison Processes are an often used model for the description of traffic patterns.

Points of control and observation (PCOs) and *co-ordination points* (CPs) are used in the same manner as in normal conformance testing with the one exception that PCOs are also used for connecting background test components to the system under test.

B.3.5.1.2 Performance test configurations

In analogy to the abstract test methods in conformance testing, different performance test configurations may have to be defined. In Testing of Communicating Systems (see bibliography 6), three performance test configurations are identified, they may be used for testing:

- an end-user telecommunication application;
- an end-to-end telecommunication service; or
- a communication protocol.

The last configuration corresponds to the distributed test method in conformance testing. However, additional configurations may have to be defined for testing further network components.

B.3.5.1.3 Measurements and Analysis

Measurements are one of the main new concepts to be introduced into TTCN in order to widen its scope to include performance testing. A *measurement* is started during a test run. It describes the period of time during which the performance is observed. A measurement may be performed by monitoring the test components that are sensitive to the test events to be measured, i.e., time stamps for the sensitive test events are collected. Constraints describe the format of the test events belonging to a measurement so that the monitor collects a time stamp whenever an event at a PCO matches that format. The constraints used follow the constraint definitions in TTCN. More elaborated *performance characteristics* such as means or standard deviations (see subclause B.3.4) can be calculated based on several measurements.

Performance characteristics may be evaluated *off-line*, i.e., after the performance test is finished, or *on-line*, i.e., during the performance test itself. For the on-line evaluation, *performance constraints* have to be specified. Performance constraints may be used to control the execution of a performance test. If the required performance characteristics is violated, the test run may be finished and a fail test verdict may be assigned.

B.3.5.1.4 Performance test behaviour

A performance test specification language has to offer features:

- to start and cancel background and foreground test components;
- to start and cancel measurements;
- to interact with the IUT;
- to generate a controlled load to the IUT; and
- to access recent measurements via performance constraints.

At the end of a performance test, a final test verdict has to be assigned. The result of a performance test should not only evaluate the observed performance, but also return the measured performance characteristics.

B.3.5.2 PerfTTCN specific language constructs

PerfTTCN is an extension of TTCN in order to use TTCN for performance test specification. In this clause, we sketch the additional language constructs of PerfTTCN as described in Testing of Communicating Systems (see bibliography 6) and An AAL5 Performance Test Suite in PerfTTCN (see bibliography 5). With the allowance of the authors, the examples are taken from Testing of Communicating Systems (see bibliography 6).

B.3.5.2.1 Foreground and background test components

Foreground and background test components are specified in an extended *Test Component Configuration Declaration Table* as shown in table B.1.

Table B.1: Extended Test Component Configuration Declaration Table

Test Component Configuration Declaration			
Configuration name: CONFIG_2			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
MTC	PCO_1	CP1	
PTC1	PCO_2	MCP2, CP1	
Background Test Components			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
traffic1	(PCO_B1)->(PCO_B2)	BCP1, BCP2	Point to Point
traffic2	(PCO_B1)->(PCO_B4)	BCP1, BCP2	Point to Point

B.3.5.2.2 Background traffic patterns

Background traffic is meant to be a continuous, uninterrupted, and predictable stream of data packets following a well defined traffic pattern. A traffic pattern defines the lengths and the inter-arrival times of the data packets. The format of the background data packets may be defined in the declarations and constraints part of the PerfTTCN test suite by using TTCN mechanisms.

Background traffic patterns may be defined in *Traffic Model Declaration Tables*. Two examples are shown in figure B.2.

Figure B.2 part a) specifies a Markov Modulated Poison Process.

Figure B.2 part b) describes a Constant Bit Rate. PerfTTCN provides different types of Traffic Model Declaration Tables for different kinds of background traffic patterns.

The relation between background traffic patterns and background test components is made by using a *Background Traffic Stream Declaration Table*. An example is shown in table B.3.

Traffic Model Declaration		
Name: on_off **Type:** MMPP **Comments:**		
Length	S1	10
Length	S2	1000
Rate	S1	2
Rate	S2	10
Transition	S1, S2	3
Transition	S2, S1	5

a) Markov Modulated Poisson Process

Traffic Model Declaration	
Name: const1 **Type:** CBR **Comments:**	
PCR	10 Mbit/s

b) Constant Bit Rate

Figure B.1: Examples of Traffic Model Declaration Tables

Table B.2: Background Traffic Stream Declaration

Background Traffic Stream Declaration			
Traffic Name	**Background Test Component**	**Model Name**	**Nr. of Instances**
load1	traffic1	on_off	6
load2	traffic1	const1	2
load3	traffic2	const1	8

B.3.5.2.3 Measurements and Analysis

PerfTTCN offers special tables to declare measurements and performance characteristics. An example for a measurement declaration is shown in table B.3. A measurement has a name, defines the metric to be measured and includes one or two test events that define the critical events of the measurement. The example in table B.3 measures the FILO delay between a Request message sent at PCO_1 and a Response message received at PCO_1.

NOTE: FILO (= first bit in, last bit out) is explained in clause B.3.4.2.

Table B.3: Measurement Declaration Table

Measurement Declaration						
Name	**Metric**	**Unit**	**event 1**	**constraint 1**	**event 1**	**constraint 2**
response_delay	DELAY_FILO	ms	PCO_1!Request	s_req_spc	PCO_1?Response	r_rep_spc

The evaluation of measurements is defined by performance characteristics. These are declared in Performance Characteristics Declaration Tables. An example is shown in table B.4 performance characteristic refers to a single measurement. In order to be statistically significant, a performance characteristics should be calculated only if the measurement has been repeated several times. Therefore it is possible to define a sample size or a time duration of the measurement for the calculation of a performance characteristic.

Table B.4: Performance Characteristics Declaration Table

Performance Characteristics Declaration				
Name	**Calculation**	**Measurement**	**Sample size**	**Duration**
res_dealay_mean	MEAN	response_delay	20	
res_delay_max	MAX	response_delay		1 min

B.3.5.2.4 Performance constraints

For the on-line analysis of performance characteristics, performance constraints have to be defined. In contrast to normal TTCN constraints, the evaluation of performance constraints is based on repeated measurement of test events rather than the matching of a single event. A performance constraint declaration consists of a name and a logical expression. The logical expression may refer to performance characteristics. For example, the performance constraint declarations shown in table B.5 refer to performance characteristics specified in table B.4.

Table B.5: Performance Constraints Declaration Table

Performance Constraints Declaration		
Name	**Constraint Value Expression**	**Comments**
p_resp	(res_delay_mean < 5) AND (res_delay_max < 10)	
n_p_resp	NOT (p_resp)	

B.3.5.2.5 Performance Test Behaviour

In PerfTTCN the behaviour of a performance test is specified in the dynamic part of the test suite. The main tester controlling the test run is specified in the test cases. All other test components are specified by test steps. Test components are created by a START command and their termination may be required explicitly via co-ordination messages. The control of measurements is similar to the control of timers, i.e. START and CANCEL commands are used to start and stop a measurement. Performance constraints may be used in the constraints reference column if an on-line evaluation of the performance characteristics is required.

An example of a PerfTTCN main test component is shown in table B.6. In line 1 the background test component `load2` (for the definition of `load2` see tables B.1 to B.2) is started and the timer `T_response_delay` is started. Afterwards (line 2) a `Request` message is sent via PCO1 and the measurement `response_delay` is started. If the IUT answers with a `Response` message according to the performance constraint `p_resp` a preliminary `pass` is assigned to the variable R (line 3) and the test case loops back to line 2. If the received Response message is not according to the performance constraint (line 5) a preliminary inconclusive is assigned to R and as specified in line 6 the test case continues with line 2. The test case ends with the timeout of `T_response_delay` (line 7). Then the measurement is cancelled (line 7) and the background test component is stopped (line 8). The final test verdict is defined by the value of R. The test execution of the test case may also end if something unexpected is received (lines 9, 10).

NOTE: It should be noted that after the assignment of (inconc) the value of R cannot be changed again to (pass), i.e., the verdict assignment in line 3 will have no effect on the final test verdict.

Table B.6: PerfTTCN dynamic behaviour description

Test Case Dynamic Behaviour					
Test Case Name:		www_get			
Group:					
Purpose:					
Configuration:		CONFIG_2			
Default:					
Comments:					
Nr	**Label**	**Behaviour Description**	**Constr. Ref.**	**Verdicts**	**Comments**
1		BCP1!START(load2) START T_response_delay			start backgr. Traffic load2
2	top	PCO_1!Request START response_delay	s_req		start measurements
3		PCO_1?Response	p_rep	(pass)	acceptable performance
4		GOTO top			
5		PCO_1?Response	n_p_rep	(inconc)	unacceptable performance
6		GOTO top			
7		? T_response_delay CANCEL response_delay			measurement terminates
8		BCP1!Stop(load2)		R	stop background traffic
9		PCO_1?OTHERWISE CANCEL response_delay CANCEL T_response_delay		fail	unexpected event, stop measurement
10		BCP1!Stop(load2)		R	stop background traffic
Detailed Comments:					

B.3.5.3 Application of PerfTTCN

Three applications have shown that PerfTTCN test cases can be implemented and executed. PerfTTCN was used to specify and implement performance tests for an HTTP server, an SMTP server Testing of Communicating Systems (see bibliography 6) and for AAL5 implementations (see bibliography 5).

B.3.5.4 Semantics of PerfTTCN

The operational semantics is under development.

B.3.6 Conclusions

From the work which has been done, we can conclude that there is a need for the specification of performance tests in an unambiguous and reusable manner with the aim of making performance test results comparable. The work on PerfTTCN shows that an extension of TTCN with performance concepts is feasible. The performance concepts which have to be integrated into TTCN are:

- *performance test components*, especially background test components with specific traffic models;
- *performance test configurations* which are comparable to the abstract test architectures in conformance testing;
- *measurements* defining the period of time within the test run during which the performance is measured; and
- *performance specific extensions of the test case behaviour*.

Furthermore, the concepts *performance metrics* and *performance constraints* have to be integrated to allow an on-line and off-line evaluation of performance characteristics. The integration of performance concepts into TTCN will be a major change of the language and should be studied carefully before integration. Especially the semantics should be clarified first.

B.4 TTCN based real-time testing

The ideas in this subclause are mainly based on a non-standardized real-time extension of TTCN, called RT-TTCN, which has been developed within a Cupertino of the Medical University of Luebeck (Germany) and the Swiss Federal Institute of Technology Zuerich (see bibliography 7 and 8).

B.4.1 Goal of real-time testing

The goal of real-time testing is to test *hard time conditions* of real-time systems, e.g., systems controlling physical devices and processes, flight control systems, control systems of nuclear power plants and also, multimedia applications and protocols. For these systems, real-time communication is essential for their correct behaviour. In contrast to that, performance testing as described in the previous sections deals with the measurement of *soft time conditions,* i.e., time conditions with statistical variations.

B.4.2 Relations to conformance testing

The relation of real-time and conformance testing is very close. The correct behaviour of a protocol implementation due to setting, timeout and cancellation of timers can be seen as part of the correct functional behaviour, i.e., the behaviour is tested by means of conformance testing. However, the timers in protocol standards are in most cases not critical. This means timers are used for driving the protocol into a stable state if a communication partner does not react or a resource is not available. In order to be in-line with the following statement in part 1 of ISO/IEC 9646-3 [1]: "The relative speed of the systems executing the test case should not have an impact on the test result", test equipment for conformance testing should be able to check these timers without any problems.

In real-time testing, timing constraints are essential for the correct behaviour of the system and they may be hard to test. Test equipment may have to be tuned in order to be able to check real-time requirements. In other words, the speed of test equipment running the test case may have an impact on the test result.

B.4.3 Problems of using TTCN for real-time testing

With the assumption that the test equipment running the TTCN test case is always much faster than the SUT, TTCN can be used to specify real-time tests. As a result of this assumption, during the test run the maximal number of ASPs or PDUs waiting for processing is one and their waiting time is negligible. Furthermore, timer setting, resetting, cancellation and timeout events can be seen as instantaneous, i.e., can be used to measure real-time.

From a telecommunication point of view, the performance of a protocol will involve two major things: one is the response time and the other is the throughput. While the response time can be easily described via TTCN START TIMER, READ TIMER, TIMEOUT as well as CANCEL TIMER events and the protocol throughput can be defined as the number of ASPs or PDUs or CMs sent/received during a time period, which can also be easily described via TTCN.

However, the assumption about the speed of the test equipment is an implicit requirement on the test equipment to be used. Implicit requirements are always dangerous, because if they are violated they influence the test results and the violation is difficult to detect.

This can be explained by using the example shown in table B.7. The purpose of the shown test case is to check if 1 000 `DATAind` messages arrive with an interval of at least 4 ms but not more than 6 ms. All violations to this requirement are interpreted as fail cases. The test case is specified as follows: After a execution of a preamble (line 1) the timer `T1` for the earliest arrival time of the next `DATAind` message is set (line 2). The counter for the `DATAind` messages counts (line 3) and the timeout for the `T1` is awaited. On arrival of the timeout message (line 4) the timer `T2` for the latest arrival time of the next `DATAind` message is started. Afterwards the DATAind message is awaited. If it arrives in time (line 5), a preliminary pass is assigned to R and depending on the number of already received `DATAind` messages, the test case may loop back to line 2 (lines 6, 7) or end by executing a postamble (lines 8, 9). If the `DATAind` message does not arrive in time T2 will produce a timeout message and the test case ends with a `FAIL` verdict. All alternatives are guarded by an `OTHERWISE` event (lines 11, 12) in order to cope with unexpected messages. On the reception of unexpected messages, the running timers are stopped and the test case ends with a `FAIL` verdict.

Table B.7: TTCN dynamic behaviour description for a real-time test

Test Case Dynamic Behaviour	
Test Case Name:	RT_Testing_Example
Group:	
Purpose:	1 000 DATAind messages should arrive with a distance of at least 4 ms but not bigger than 6 ms. All violations of this requirement have to be interpreted as fail.
Configuration:	
Default:	
Comments:	

Nr	Label	Behaviour Description	Constr. Ref.	Verdicts	Comments
1		+PREAMBLE			some preamble
2	top	START T1 (4 ms)			set timer for earliest arrival time
3		(counter := counter +1)			some calculation
4		?TIMEOUT T1 START T2 (2 ms)			timeout of T1 and set timer for latest arrival time
5		PCO1?DATAind STOP T2	C_DATA	(pass)	DATAind arrives in time ?
6		[counter<1 000]			
7		GOTO top			another DATAind required
8		[counter>=1 000]			
9		+POSTAMBLE			TC finishes with postamble
10		?TIMEOUT T2		FAIL	DATAind arrives not in time
11		PCO1?OTHERWISE STOP T2		FAIL	some other message
12		PCO1?OTHERWISE STOP T1		FAIL	some other message
Detailed Comments:					

This specification is incorrect with regard to statement "the relative speed of the systems executing the test case should not have an impact on the test result" in ISO/IEC IS 9646-3 [1]. The speed of the test equipment running the test case may very well have an influence on the test result. If the test equipment is very fast, i.e. the execution of a TTCN statement is negligible compared to the precision of the measurement (= ms), then the test case may run without problems. But, if the execution of a TTCN statement is slow, e.g., 3 ms, the following may happen due to the TTCN snapshot semantics:

A first `DATAind` message may arrive after 2 ms and a second one arrives after 10 ms. Both should lead to a `FAIL` verdict. However, the start of the timer `T1` (line 1) takes 3 ms and the first `DATAind` message has already arrived. The execution of the statement in line 3 costs another 3 ms. This means after 6 ms the first snapshot is taken. The timer `T1` is already expired and its timeout message is part of the actual snapshot. The timeout is executed which takes again 3 ms (line 4) and `T2` is started then after 9 ms the next snapshot is taken. The first `DATAind` is available, causing the cancellation of `T2` (which already is expired) and via line 6 and line 7 the test case continues after 18 ms with a new setting of `T1` in line 2. It is obvious that the violation of the time distance requirement between the two `DATAind` messages will also not be detected. Exchanging the order of timeout and `DATAind` receive events in the TTCN description will not help. It is possible to construct another erroneous behaviour which will also pass the test.

From this example it can be seen that for testing real-time requirements by using TTCN test specifications, the speed of the test equipment may very well influence the test result. The problematic part of the TTCN test description for real-time testing is the timer construct. In our example the effect of the timer on the test case behaviour is delayed. Even worse, due to the possibility of delaying the execution of timeout events inadequate or slow test equipment will be able to interpret test cases in the way explained by the example. A new or modified timer concept may help to overcome these problems.

B.4.4 Real-time TTCN

The only known proposal for a real-time extension of TTCN is Real-time TTCN or for short RT-TTCN which has been developed within a Cupertino of the Medical University of Luebeck (Germany) and the Swiss Federal Institute of Technology Zuerich (Switzerland). The following summary is based on Testing of Communicating Systems (see bibliography 7) and A Proposal for a Real-Time Extension for TTCN (see bibliography 8).

B.4.4.1 The RT-TTCN Approach

The approach of RT-TTCN is to allow the annotation of TTCN statements with time labels. The time labels define the *earliest execution time* (EET) and the *latest execution time* (LET) of the statement. This means the time labels form a time interval during which the statement has to be executed. The time labels may be specified relative to the enabling time of the statement or the execution time of previous statements.

NOTE: The abbreviations `LET` and `EET` are keywords for default time labels in the RT-TTCN syntax. Different fonts are used to distinguish between syntax, `EET` and `LET`, and semantics, EET and LET.

B.4.4.2 RT-TTCN specific language constructs

For being somehow compatible with the existing TTCN definition, RT-TTCN proposes only small language extensions. With the allowance of the authors, some examples in this subclause are taken from Testing of Communicating Systems (see bibliography 7).

B.4.4.2.1 Extensions of the TTCN declaration part

In order to allow the comfortable specification of time labels by using meaningful names, an Execution Time Declarations Table has to be introduced. An example is shown in table B.8. Execution Time Declarations Tables are used for the specification of time names, time values and units. Apart from the headings, the table looks much like the TTCN Timer Declarations Table. Time names are declared in the Time Names column. Their values and the corresponding time units are specified on the same line in the Value and Units column. The declaration of the time values is optional. `EET` and `LET` are predefined time names (keywords) with default values zero and infinity. As shown in table B.8 these default time values can be overwritten.

Table B.8: Execution Time Declarations Table

Execution Time Declarations			
Time Name	**Value**	**Unit**	**Comments**
EET	1	s	Redefined EET value
LET	1	min	Redefined LET value
WFN	5	ms	Wait For Nothing
NoDur		min	No value specified

Besides of the static declarations of time values in an Execution Time Declarations Table, changing these values within a behaviour description table can be done by means of assignments. Examples are shown in line 2 and line 4 of table B.9. However, evaluation of time labels attached to a test event should always result in EET and LET values for which $0 \leq \text{EET} \leq \text{LET}$ holds.

B.4.4.2.2 Extensions of the TTCN dynamic part

RT-TTCN introduces two new columns in dynamic behaviour description tables. These are a Time and a Time Options column. In table B.9 a RT-TTCN Test Case Dynamic Behaviour table is shown. The same additional columns are also introduced in Test Step Dynamic Behaviour and Default Dynamic Behaviour tables.

Table B.9: RT-TTCN Test Case Dynamic Behaviour Table

Test Case Dynamic Behaviour							
Test Case Name:	RT_TTCN_Example						
Group:							
Purpose:	just an example						
Configuration:							
Default:							
Comments:							
Nr	**Label**	**Time**	**Time Options**	**Behaviour Description**	**Constr. Ref.**	**Verdicts**	**Comments**
1	L1	2, 4	M	A?DATAind			Time label Mandatory EET
2				(NoDur := 3)			Time assignment
3		2, NoDur		A!DATAack			
4				(LET := 50)			LET update (ms)
5				A?DATAind			
6		L1+WFN, L1+LET	M, N	B?ALARM			Mandatory EET not pre-emptive
Detailed Comments							

An entry in the Time column specifies EET and LET for the corresponding TTCN statement. Entries may be constants (line 1), time names (use of NoDur in line 3), or expressions (line 6).

In general, EET and LET values are interpreted relative to the enabling time of alternatives at a level of indentation, i.e., the time when the level of indentation becomes the current level. However, some applications may require to define EET and LET values relative to the execution of an earlier test event, i.e., not restricted just to the previous one. In support of this requirement, a label in the Label column may not only be used in a GOTO but can also be used in the Time column, so that EET and LET values are computed relative to the execution time of the alternative identified by the label: In table B.9 on line 6, the time `labels (L1+WFN, L1+LET)` are referring to the execution time of the alternative in line 1 (for which label L1 is defined). Statements which are not annotated with time labels are by default annotated with `EET` and `LET`.

Entries in the Time Options column are combinations of symbols `M` and `N`. Similar to using labels in expressions, time option `N` allows to express time values relative to the alternative's own enabling time, even though some TTCN statements may be executed in between two successive visits of the same level of indentation.

Thus, the amount of time needed to execute the sequence of TTCN statements in between two successive visits is compensated. If time option `N` is defined, then execution of this alternative is not pre-emptive with respect to the timing of all alternatives at the same level of indentation.

In some executions of a test case, a receive or otherwise event may be evaluated successfully before it has been enabled for EET units. If the intention to define EET as a mandatory lower bound when an alternative may be evaluated successfully, then time option `M` has to be specified. Informally, if time option `M` is specified and the corresponding alternative can be successfully evaluated before it has been enabled for EET units, then this results in a `FAIL` verdict.

B.4.4.2.3 Assigning test verdicts

RT-TTCN introduces one additional rule for the assignment of test verdicts to test runs. This is the following: A Fail verdict is assigned and the test execution ends if a specified real-time requirement is violated during a test run. All other TTCN rules for assigning test verdicts remain the same.

B.4.4.3 Semantics of RT-TTCN

RT-TTCN has an elaborated semantics which is based on timed transition systems (see bibliography 2). Additionally, a refined TTCN snapshot semantics that takes time annotations into account is provided in Testing of Communicating Systems (see bibliography 7).

B.4.4.4 Application of RT-TTCN

RT-TTCN was developed in an academic environment. Therefore the RT-TTCN semantics is well elaborated and its applicability has been shown by means of some examples. However, until now no big case study has been done. Table B.10 provides a RT-TTCN specification of the example discussed in subclause B.4.3 and presented in table B.7. The example and its correct specification in table B.10 may give some indication about the elegance and power of RT-TTCN.

B.4.5 Conclusion

As long as the test equipment is faster than the applications, the need of a TTCN extension for real-time testing may not be urgent. However, as seen by the example of RT-TTCN a small extension may help to avoid future problems. It should also be noted that the RT-TTCN annotation with time labels might also be a more obvious or more understandable way to express the already existing timing requirements in pure conformance testing.

Table B.10: RT-TTCN description of the TTCN example shown in table B.7

Test Case Dynamic Behaviour							
Test Case Name:	RT_Testing_Example2						
Group:							
Purpose:	1 000 DATAind messages should arrive with a distance of at least 4 ms but not bigger than 6 ms. All violations of this requirement have to be interpreted as fail.						
Configuration:							
Default:							
Comments:							
Nr	**Label**	**Time**	**Time Options**	**Behaviour Description**	**Constr. Ref.**	**Verdicts**	**Comments**
1				+PREAMBLE			
2				(counter := counter +1)			
3		4, 6	M	PCO1?DATAind			
4				[counter<1 000]			
5				GOTO top			
6				[counter>=1 000]			
7				+POSTAMBLE			
8				PCO1?OTHERWISE			
Detailed Comments							

B.5 Advanced test architectures

The TTCN extensions and requirements described in the previous sections mainly address the problem of describing the dynamic test case behaviour properly. But, testing has also architectural aspects. According to ISO/IEC 9646-3 [1], an abstract test suite is written for a specific abstract test method. Opening the scope of TTCN towards the test of new applications, real-time testing and performance testing also pose new requirements on test architectures. This means that a future test architecture model will include new test components beyond lower and upper tester functions (load generators or monitor components) and should also represent the communication points adequately (the PCO concept may be extended towards synchronous communications, pure monitor points, or dynamic PCO creation).

A generic test architecture model has been proposed in Testing of Communicating Systems (see bibliography 9). The basic idea of the approach is to provide a tool box of elements, which can be combined generically into a test architecture which is suitable for a specific application or system to be tested. A test architecture comprises possibly several instances of different types of components:

- An *Implementation under test* (IUT) represents the implementation or parts of the distributed system to be tested. In principle, an IUT may be distributed over physically separate real systems, i.e., it is allowed to have several IUT components in a concrete test architecture.

- An *Interface Component* (IC) is a component which is needed for interfacing IUTs, e.g., an underlying service or a system in which an IUT is embedded.

- A *Test Component* (TC) describes a component which contributes to the test verdict by co-ordinating other TCs or controlling and observing IUTs. A test configuration identifies all TCs necessary for the execution of a specific test case. A TC exists from the start of a test case or is created dynamically by other TCs. In each test architecture, there should be one special *Main Test Component* (MTC) which starts, ends and co-ordinates the test run.

- A *Controlled Component* (CC) is a component which does not contribute to the test verdict but provides SUT specific data to TCs or the SUT, e.g., a load generator, an emulator or a simulator.

- A *Communication Point* (CoP) represents a point at which communication takes place and at which communication can be observed, controlled or monitored. CoPs may be placed somewhere in the IUT, thus CoPs may be used for controlling and observing state information internal to the IUT or to monitor communication between IUTs.

- A *Communication Link* (CL) is a means for describing possible communication flows between TCs, IUTs, ICs and CCs and the kind of communication which may take place. We distinguish between active and passive CLs. An active CL can be characterized by its kind (synchronous or asynchronous) and its direction (unidirectional or bidirectional). A passive CL allows to monitor communication, i.e., to listen at a CoP. For the dynamic creation of TCs it is assumed that CLs are also created dynamically and that the CoPs linked to the CLs are all known before test execution, i.e., CoPs cannot to be created dynamically.

- The term *System Under Test* (SUT) denotes a combination of ICs and IUTs.

An example for the use of the proposed model is shown in figure B.2. The different hard- and software components of the architecture are shown as boxes and ellipses. The communication flow, the kind of communication and the creation of TCs is indicated by different types of arrows.

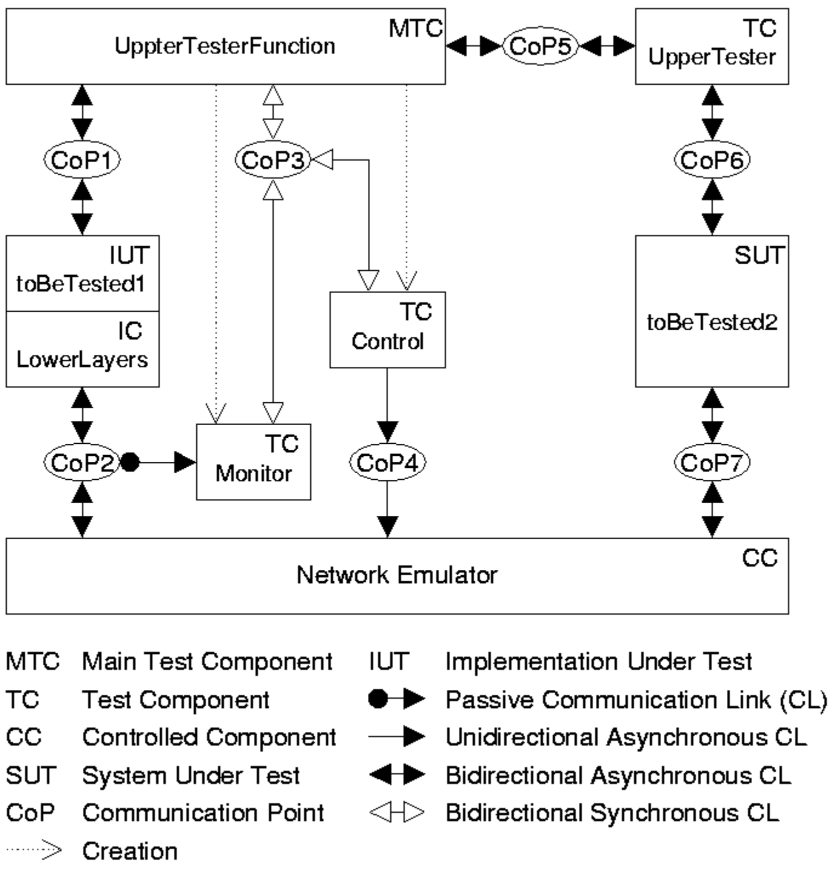

Figure B.2: Example for the use of the generic test architecture

Figure B.2 describes an architecture proposed for interoperability testing in Testing of Communicating Systems (see bibliography 9). There are two IUTs to be tested. One is called *toBeTested1*; the other is embedded in the SUT *toBeTested2*. The IUTs communicate by using an underlying network which in our case is emulated by the CC *Network Emulator*. The IUT *toBeTested1* needs the IC *Lower Layers* for having the interface *CoP2* with *CC Network Emulator*. The communication at *CoP2* is monitored by TC *Monitor*. This is described by the passive CL between *CoP2* and *Monitor*. If necessary, the CC *Network Emulator* can be controlled by the TC *Control*.

The MTC is called *UpperTesterFunction*. It communicates asynchronously via *CoP5* with the peer TC *UpperTester*. As indicated by the dotted arrows, the TCs *Monitor* and *Control* are created by the MTC. It is assumed that they are running on the same computer and perform a synchronous communication with the MTC.

NOTE: When extending TTCN, new test methods should also be investigated. The generic test architecture model of Testing of Communicating Systems (see bibliography 9) is very general and may be used for this purpose.

B.6 On the introduction of new TTCN concepts

The introduction of new TTCN concepts should be done carefully. The following proposes a three-step procedure where each step widens the scope of TTCN in one specific direction. In parallel to all steps, corresponding abstract test architectures should be developed.

B.6.1 Step 1: Generalisation of TTCN concepts

The first step should widen the scope of TTCN towards the test of applications which are not OSI compliant, e.g., CORBA applications. As a consequence, some TTCN restrictions should be dropped and some new concepts should be introduced. Major topics of this step are:

- It should be allowed to distribute the IUT instead of having only one black box representing the IUT.
- New communication mechanisms should be supported, e.g., synchronous communication, or broadcast messages.
- Dynamic creation and destruction of test components and communication links should be supported.

The work for this first step should be co-ordinated with the ongoing discussions about a simplification of TTCN and additional constructs proposed for facilitating the use of TTCN.

B.6.2 Step 2: TTCN based real-time testing

In the second step TTCN should be extended to support real-time testing. This may be done by introducing a new timer concept or by modifying the existing timer concept. In both cases, the semantics of TTCN has to be adapted.

B.6.3 Step 3: TTCN based performance testing

The TTCN extensions to be introduced in a third step should support performance testing. From the point of view of the semantics, this third step might be the most critical one. The steps one and two are straight forward and can build on the existing semantics. In the third step, statistical models have to be introduced and interpreted.

B.6.4 Related issues

Some applications use their own description techniques for purposes which affect the TTCN use. For example, IDL is used for the description of interfaces. The relation of IDL and TTCN is not defined yet and therefore not clear. In cases where the relationship between TTCN and the other description techniques is important, it should be studied in detail. As a result of these investigations, mapping rules between the two notations should be defined. In a first step this should be done for clarifying the relation between TTCN and IDL.

Bibliography

The following material, though not specifically referenced in the body of the present document (or not publicly available), gives supporting information.

1) R. Jain, G. Babic, A. Durresi. *ATM Forum Performance Testing Specification - Baseline Text.* ATM Forum Document Number: BTD-TEST-TM-PERF.00.05 (96-0810R8), February 1998.

2) T. Henzinger, Z. Manna, A. Pnueli. Timed Transition Systems. In Real-Time: Theory and Practice. LNCS 600, 1991.

3) OMG. *The Common Object Request Broker Architecture and Specification*. Version 2.2, Feb. 1998.

4) I. Schieferdecker, M. Li, A. Hoffmann. *Conformance Testing of TINA Service Components - the TTCN/CORBA Gateway*. To appear in ISN.98.

5) I. Schieferdecker, M. Li, A. Rennoch. An AAL5 Performance Test Suite in PerfTTCN. Proceedings of the 1997 GI/ITG technical workshop on formal description techniques in Berlin, 1997.

6) I. Schieferdecker, B. Stepien, A. Rennoch. PerfTTCN, a TTCN language extension for Performance Testing. In: Testing of Communicating Systems, Volume 10 (M. Kim, S. Kang, K. Hong editors), Chapman & Hall, September 1997.

7) T. Walter, J. Grabowski. Realtime TTCN for Testing Real-time and Multimedia Systems. In: Testing of Communicating Systems, Volume 10 (M. Kim, S. Kang, K. Hong editors), Chapman & Hall, September 1997.

8) T. Walter, J. Grabowski. A Proposal for a Real-Time Extension for TTCN. In Kommunikation in Verteilten Systemen'97 (KiVS'97), Informatik aktuell, Springer Verlag 1997.

9) T. Walter, I. Schieferdecker, K. Grabowski. Test Architectures for Distributed Systems - State of the Art and Beyond. In: Testing of Communication Systems, Volume 11 (A. Petrenko, N. Yevtushenko editors), Chapman & Hall, September 1998.

History

Document history		
V1.1.1	February 1999	Membership Approval Procedure MV 9917: 1999-02-23 to 1999-04-23
V1.1.1	May 1999	Publication

ISBN 2-7437-3093-5
Dépôt légal : Mai 1999