ETSI ES 203 119-6 V1.3.1 (2022-05)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Part 6: Mapping to TTCN-3

Reference
RES/MTS-1196v1.3.1

Keywords
methodology, model, testing, TTCN-3

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

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

The present document is part 6 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document specifies how the elements of the Test Description Language (TDL) should be mapped to Testing and Test Control Notation version 3 (TTCN-3) [2]. The intended use of the present document is to serve as the basis for the development of TDL tools. The meta-model of TDL and the meanings of the meta-classes are described in ETSI ES 203 119-1 [1].

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI ES 203 119-1 (V1.6.1): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: Abstract Syntax and Associated Semantics".

[2] ETSI ES 201 873-1 (V4.13.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[3] ETSI ES 203 119-3 (V1.5.1): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 3: Exchange Format".

[4] ETSI ES 203 119-2 (V1.5.1): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 2: Graphical Syntax".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 136 523-1 (V10.2.0): "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 1: Protocol conformance specification (3GPP TS 36.523-1 version 10.2.0 Release 10)".

[i.2] ETSI TS 186 011-2: "Core Network and Interoperability Testing (INT); IMS NNI Interoperability Test Specifications (3GPP Release 10); Part 2: Test descriptions for IMS NNI Interoperability".

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

behaviour function: function used in TTCN-3 code that describes the behaviour of a TDL component instance

TTCNname: name of a TDL meta-model element that is used in the TTCN-3 code

NOTE: A TTCNname of a TDL element follows the syntactical rules of identifiers specified in ETSI ES 201 873-1 [2]. A TTCNname of a TDL element may contain a part that is derived from the TDL name with some prefixes and/or postfixes determined by a naming convention used in the TTCN-3 code.

3.2 Symbols

Void.

3.3 Abbreviations

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

IMS	IP Multimedia Subsystem
MTC	Main Test Component
PTC	Parallel Test Component
SUT	System Under Test
TDL	Test Description Language
TTCN-3	Testing and Test Control Notation version 3

4 Basic Principles

4.1 Introduction

While both TDL and TTCN-3 are standardized languages, there are various ways how TTCN-3 code can be derived from a TDL test description. This may result in different or even incompatible code intended to implement the same test description. Without a standardized mapping of TDL to TTCN-3, there could be a proliferation of different and possibly incompatible tool- and user-specific mappings of TDL test descriptions to executable test cases which can present new challenges to users and tool vendors.

A standardized mapping between the two languages provides a consistent approach for producing executable tests from high level test descriptions specified in TDL. This enables the generation of executable tests from TDL test descriptions in a (semi-) automatic way, and by extension of the re-use of existing test tools and frameworks for test execution. This way, test engineers can concentrate on the specification of test descriptions at a higher level of abstraction, while having a clear expectation of what the resulting test implementation will look like.

4.2 Document Structure

The meta-model of the Test Description Language is specified in ETSI ES 203 119-1 [1]. The present document specifies how the elements of the meta-model of TDL in locally ordered 'TestDescriptions' should be mapped to TTCN-3 code. The mapping of the globally ordered 'TestDescription's is outside the scope of the present document.

The structure of the present document follows the structure of the meta-model specification in ETSI ES 203 119-1 [1]. The clauses 5 to 10 describe the standardized mappings of the meta-model elements with identical clause numbers in ETSI ES 203 119-1 [1]. In each clause, first the description of the mapping of the corresponding meta-model element is described. It may be followed by a Constraints section, if the mapping is provided only with limitations. At the end of a clause an Example clause may exist to illustrate the mapping of the corresponding meta-model element. In the Examples the textual (specified in ETSI ES 203 119-1 [1]) or the graphical (specified in ETSI ES 203 119-3 [3]) notations of TDL can be used.

In some cases the structure of the TTCN-3 code may differ from the structure of the TDL specification or it requires some additional specification in TTCN-3. These special cases are described in clauses 8.2.11, 8.2.12 and 8.3.

At the end of the present document in Annex A several examples illustrate how the TTCN-3 code will look like after the rules of mapping specified in the present document are applied.

4.3 Notational Conventions

4.3.0 General

Elements (e.g. meta-classes, properties, etc.) from the TDL meta-model [1] are typed in between 'single quotes', e.g. 'StructuredDataType' or 'returnType'.

The TTCN-3 code elements (keywords, symbols, etc.) are typed in **`bold Courier New`** font, e.g. **`type port`** or **`{`** .

The TTCN-3 code to be generated is described by production rules, where applicable. The production rules are specified in between << and >> symbols. Inside a production rule, the concatenation between elements of that production rule is specified by a plus (+) symbol.

Iterations over collections of attributes of a metaclass make use of a function collect( ) with the following syntax: *propertyName*.collect(*VariableName* ':' *expression*), where *VariableName* is an alphanumeric word signifying the variable used in the subsequent *expression*, . *propertyName* is a string that shall be the same as the name of a property of a TDL metaclass. The type of this property determines the type of the variable denoted by *VariableName*.

The separator between the elements of an iteration is specified by the concat( ) function.

EXAMPLE 1:

The production rule:

`type record` <<self.name>> **`{`**

<< member.collect(m | m.dataType.name + " " + m.name()).concat("**`,`**")>>

}

for this TDL description

Type MSG (sessionID of type integer, content of type charstring);

will provide the following TTCN-3 code snippet:

```
type record MSG {
    integer sessionID,
    charstring content
}
```

The function select() selects a TDL element with a given value of a property.

EXAMPLE 2:

componentInstance.select(c | c.role = Tester) selects a 'componentInstance' whose 'role' property has a value of 'Tester'.

Other helper functions used in the production rules are collected in clause 4.3.1, while the predefined 'AnnotationType's that can be used to control the TTCN-3 code generation are listed in clause 4.3.2.

4.3.1 Functions used in production rules

- behaviourFunctionInReferencedTD(): returns the name of the behaviour function used in a referenced 'TestDescription' of the same tester component.
- equivalent(): returns the equivalent of the corresponding TDL element. If none of the structural modifications - described in clause 8.3 - on a TDL configuration is to be applied then the element.equivalent() is the element itself, otherwise what is specified in the corresponding sub-clause of clause 8.3.
- getKind(): returns the kind of an 'importedElement' (e.g. **`type`**, **`template`**, **`const`**, **`function`**, etc.) that can be used in a TTCN-3 **`import`** statement.
- toLower(): returns the value of a literal converted to all lowercase characters.
- TTCNname(): returns the name of the corresponding TDL element that will be used in the TTCN-3 code.

4.3.2 Predefined Annotations

A Predefined Annotation is an 'Annotation', whose 'key' is one of the following predefined 'AnnotationTypes'. The Predefined Annotations are used to help the TTCN-3 code generation in cases where the TTCN-3 code to be generated cannot be determined just from the TDL description:

- TTCN3Code: this 'AnnotationType' indicates that the 'body' of the 'Annotation' or of an 'InlineAction' contains a valid TTCN-3 code.
- Value: this 'AnnotationType' indicates that the annotated element shall not be treated as a template or a template type.

4.4 Conformance

For an implementation claiming to conform to this version of the mapping from TDL to TTCN-3, all features specified in the present document and in ETSI ES 203 119-1 [1] shall be implemented consistently with the requirements given in the present document and ETSI ES 203 119-1 [1].

5 Foundation

5.1 Overview

'Package's are mapped to TTCN-3 modules, 'ElementImport's to import statements, while 'Comment's, 'Annotation's, 'AnnotationType's and 'TestObjective's to TTCN-3 comments.

5.2 Mapping of Foundation Elements

5.2.1 Element

This is an abstract metaclass, therefore no mapping is defined.

Naming is different in TDL and in TTCN-3, therefore the names of the 'Element's used in TDL may not be used in TTCN-3. On one hand the set of characters allowed to be used in a TDL name is larger than the set allowed in TTCN-3 and on the other hand a TDL name may be a reserved keyword in TTCN-3. That is why the term TTCNname is introduced. A TTCNname of an 'Element' is the name of the 'Element' that is used in the TTCN-3 code.

A TTCNname may contain a part that is derived from the TDL name with some prefixes and/or postfixes determined by a naming convention used in the TTCN-3 code.

The present document does not specify how a TTCNname is generated from a TDL name. Neither the method how the TDL names are converted to valid TTCN-3 names nor the naming convention to be used in the TTCN-3 code, however the present document recommends a naming convention. The basic assumption of the recommended TTCNname is that it contains a part which is generated from the TDL name and it may be extended by some prefix(es) and/or postfix(es).

NOTE 1: The naming convention used in the present document is only a recommendation, in a concrete tool or implementation a different one may be used.

NOTE 2: In the following clauses the function TTCNname() will be used to get the TTCNname of the corresponding 'Element'.

5.2.2 NamedElement

This is an abstract metaclass, therefore no mapping is defined.

5.2.3 PackageableElement

This is an abstract metaclass, therefore no mapping is defined.

5.2.4 Package

A 'Package' shall be mapped to a module.

module <<self.TTCNname()>> **{**

}

For all import: as defined in clause 5.2.5.

NOTE: In TTCN-3 a module cannot contain another module, therefore a contained 'Package' will also be mapped to a "standalone" module. If information about the 'Package' structure needs to be kept in TTCN-3, then use a suitable naming convention.

5.2.5 ElementImport

The 'ElementImport' shall be mapped to **import** statement(s).

If the 'importedElement' is empty then an **import** … **all** statement shall be used:

import from <<self.importedPackage.TTCNname()>> **all;**

otherwise for all the 'importedElement' a selected **import** statement shall be used:

<< importedElement.collect(i | "**import from** " +" " + self.importedPackage.TTCNname() +" " +i.getKind() + " " + i.TTCNname()).concat("**;**")>>

NOTE: How the kind of the 'importedElement' (e.g. **type**, **template**, **const**, **function**, etc.) is determined is outside the scope of the present document. For this purpose e.g. an annotation or a naming convention can be used.

5.2.6 Comment

A 'Comment' shall be mapped to a comment:

/* <<self.body>> ***/**

5.2.7 Annotation

If the 'key' of the 'Annotation' is the predefined 'AnnotationType' TTCN3Code, then the 'Annotation' shall be mapped to its 'value' (that is to the TTCN-3 code itself), otherwise it shall be mapped to a comment:

```
/*
ANNOTATION <<self.key.TTCNname()>>
<<self.value>>
*/
```

5.2.8 AnnotationType

'AnnotationType' shall be mapped to a comment:

```
/*
ANNOTATION TYPE <<self.TTCNname()>>
*/
```

If the 'AnnotationType' has an extension:

```
/*
ANNOTATION TYPE <<self.TTCNname()>> EXTENDS <<self.extension.extending.TTCNname()>>
*/
```

5.2.9 TestObjective

The 'TestObjective' shall be mapped to a comment:

```
/*
Test Objective  <<self.name>>
Description: <<self.description>>
Objective URI: <<self.objectiveURI>>
*/
```

5.2.10 Extension

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

6 Data

6.1 Overview

Mapping of data definitions can either be done by the explicit 'DataElementMapping's provided by the user or if no 'DataElementMapping' is provided, then the TDL data definitions shall be mapped as they are specified in the following clauses. If there is a 'DataElementMapping' provided for a 'StructuredDataType' then mappings shall be provided for all of its 'Member's.

TDL does not make a distinction if a data instance is a value or a template, while TTCN-3 does. By default, all TDL 'DataInstance's, 'FormalParameter's, 'Variable's and return values of the 'Function's shall be mapped to a TTCN-3 template unless an 'Annotation' with the predefined 'AnnotationType' Value instructs otherwise. The 'PredefinedFunctionCall's and the predefined instances of the 'SimpleDataType' and 'SimpleDataInstance' elements shall be mapped to their TTCN-3 counterparts, while predefined instance 'Second' of 'Time' element shall be mapped to TTCN-3 data type **`float`**.

The 'DataUse's are mapped to their TTCN-3 counterparts, the 'DataInstanceUse' is a usage of a template or a constant; 'SpecialValueUse's are mapped to the TTCN-3 AnyValue (?), AnyValueOr None (*) and the special value omit, respectively; 'FunctionCall's and 'PredefinedFunctionCall's to function calls or operator invocation, 'FormalParameterUse' and 'VariableUse' to usage of a formal parameter or a variable. The inline modification of the 'DataUse's are mapped to a (sequence of) template modification(s).

6.2 Mapping of Data Definition Elements

6.2.1 DataResourceMapping

If 'DataResourceMapping' is provided, then its 'resourceURI' shall be used to locate the resource, where the 'DataElementMapping'(s) can be found.

6.2.2 MappableDataElement

This is an abstract metaclass, therefore no mapping is defined.

6.2.3 DataElementMapping

If a 'DataElementMapping' is provided by the user, then it shall be used for mapping the corresponding data definitions.

6.2.4 ParameterMapping

If there is a 'ParameterMapping' provided by the user for a 'Member' of a 'StructuredDataType' or a 'FormalParameter' of an 'Action' or a 'Function' then it shall be used for mapping the corresponding data definitions.

6.2.5 DataType

This is an abstract metaclass, therefore no mapping is defined.

6.2.6 DataInstance

This is an abstract metaclass, therefore no mapping is defined.

6.2.7 SimpleDataType

If there is a 'DataElementMapping' provided for a 'SimpleDataType' then it shall be used for the mapping, otherwise a 'SimpleDataType' shall be mapped to the TTCN-3 data type **`charstring`**, except for the Predefined Instances of the 'SimpleDataType' Element that shall be mapped according to the rules specified in clause 10.2.

The following TTCN-3 type definition of SimpleType shall be made in the declaration part of the module:

```
type charstring SimpleDataType;
```

For each SimpleDataType:

```
type SimpleDataType <<self.TTCNname()>> ;
```

6.2.8 SimpleDataInstance

A 'SimpleDataInstance' shall be mapped to a **const** if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **template**.

If there is a 'DataElementMapping' provided for a 'SimpleDataInstance' then it shall be used for the mapping, otherwise a 'SimpleDataInstance' shall be mapped to:

```
template <<self.datatype.TTCNname()>> <<self.TTCNname()>> := " <<self.TTCNname()>> ";
```

or if an 'Annotation' with the predefined 'AnnotationType' Value is used, to:

```
const <<self.datatype.TTCNname()>> <<self.TTCNname()>> := " <<self.TTCNname()>> ";
```

6.2.9 StructuredDataType

If there is a 'DataElementMapping' provided for a 'StructuredDataType' then it shall be used for the mapping, otherwise if no 'Constraint' is associated to the 'StructuredDataType', it shall be mapped to a record:

```
type record <<self.TTCNname()>> {
    << allMembers().collect(m |  m.dataType.TTCNname() + " " + m.TTCNname() ).concat(",")>>
}
```

If a 'member' is optional (self.member.isOptional = true) then the TTCN-3 keyword **optional** shall be inserted after the TTCNname of that 'member'.

If a 'Constraint' with predefined 'ConstraintType' 'union' is associated to a 'StructuredDataType' then it shall be mapped to a union type:

```
type union <<self.TTCNname()>> {
    << allMembers().collect(m |  m.dataType.TTCNname() + " " + m.TTCNname() ).concat(",")>>
}
```

Constraints

If there is a 'DataElementMapping' provided for a 'StructuredDataType' then a mapping shall be provided for all of its 'Member's.

6.2.10 Member

This metaclass has no dedicated mapping, it is used solely in the mapping of other metaclasses.

6.2.11 StructuredDataInstance

A 'StructuredDataInstance' shall be mapped to a **const** if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **template**.

If there is a 'DataElementMapping' provided for a 'StructuredDataInstance' then it shall be used for the mapping, otherwise a 'StructuredDataInstance' shall be mapped to:

```
template <<self.datatype.TTCNname()>> := {
    << memberAssignment.collect(m | m.member.TTCNname() + ":= " + m.memberSpec).concat(",")>>
    }
```

or if an 'Annotation' with the predefined 'AnnotationType' Value is used, to:

```
const <<self.datatype.TTCNname()>> := {
    << memberAssignment.collect(m | m.member.TTCNname() + ":= " m.memberSpec).concat(",")>>
    }
```

A 'memberSpec' shall be mapped to a corresponding 'DataUse' according to clause 6.3.1 of the present document.

If 'Annotation' with the predefined 'AnnotationType' Value is not used and a 'StructuredDataInstance' has no 'MemberAssignment' for a given 'Member' of its 'StructuredDataType', and:

- 'unassignedMember' is set to 'AnyValue' then the 'memberSpec' of that member shall be mapped to **?**;
- 'unassignedMember' is set to 'AnyValueOrOmit' then the 'memberSpec' of that member shall be mapped to ***** if it is optional or **?** if it is non-optional.

Constraints

For a constant, 'memberSpec's shall be specified for all of its 'member's.

6.2.12 MemberAssignment

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

6.2.13 CollectionDataType

If there is a 'DataElementMapping' provided for a 'CollectionDataType' then it shall be used for the mapping, otherwise if no 'Constraint' is associated to the 'CollectionDataType', it shall be mapped to an unrestricted **record of** itemType:

```
type record of <<self. itemType.TTCNname()>> <<self.TTCNname()>> ;
```

If a 'Constraint' with the predefined 'ConstraintType' 'length' is associated to a 'CollectionDataType' and a single 'quantifier' element, it shall be mapped to a restricted **record of** itemType with the exact length specified by the 'quantifier':

```
type record length( <<self.constraint.quantifier>> ) of
    <<self. itemType.TTCNname()>> <<self.TTCNname()>> ;
```

If a 'Constraint' with the predefined 'ConstraintType' 'length' is associated to a 'CollectionDataType' and two 'quantifier' elements, it shall be mapped to a restricted **record of** itemType with a minimum and maximum length defined by the two 'quantifier' elements:

```
type record length( << self.contstraint.quantifier.collect(q | q.TTCNname() ).concat("..")>> ) of
   <<self. itemType.TTCNname()>> <<self.TTCNname()>> ;
```

6.2.14 CollectionDataInstance

A 'CollectionDataInstance' shall be mapped to a **const** if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **template**.

If there is a 'DataElementMapping' provided for a 'CollectionDataInstance' then it shall be used for the mapping, otherwise a 'CollectionDataInstance' shall be mapped to:

```
template <<self.datatype.TTCNname()>> := { << item.collect(i | i).concat(",")>> }
```

or if an 'Annotation' with the predefined 'AnnotationType' Value is used, to:

```
const <<self.datatype.TTCNname()>> := { << item.collect(i | i).concat(",")>> }
```

The 'item' shall be mapped to a corresponding 'DataUse' according to clause 6.3.1 of the present document.

6.2.15 ProcedureSignature

If there is a 'DataElementMapping' provided for a 'ProcedureSignature' then it shall be used for the mapping, otherwise a 'ProcedureSignature' shall be mapped to a **signature** definition:

```
signature <<self.TTCNname()>> ( << parameter.select(p | p.kind = In or p.kind = Out).collect(p |
    p.kind.toLower() + " " + p.dataType.TTCNname()+ " " +p.TTCNname() ) ).concat(",")>> ) exception (
    << parameter.select(p | p.kind = ExceptionX).collect(p | p.dataType.TTCNname() ).concat(",")>> )
```

The exception type list shall not contain the same type more than once.

6.2.16 ProcedureParameter

This metaclass has no dedicated mapping, it is used solely in the mapping of other metaclasses.

6.2.17 ParameterKind

This metaclass has no dedicated mapping, it is used solely in the mapping of other metaclasses.

6.2.18 Parameter

This is an abstract metaclass, therefore no mapping is defined.

6.2.19 FormalParameter

This metaclass has no dedicated mapping, it is used solely in the mapping of other metaclasses.

6.2.20 Variable

A 'Variable' shall be mapped to a **var** if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **var template**:

```
var template <<self.dataType.TTCNname()>> <<self.TTCNname()>>;
```

or if an 'Annotation' with the predefined 'AnnotationType' Value is used,

```
var <<self.dataType.TTCNname()>> <<self.TTCNname()>>;
```

6.2.21 Action

An 'Action' shall be mapped to a TTCN-3 function with no return type.

If there is a 'DataElementMapping' provided for an 'Action' then it shall be used for the mapping, otherwise an 'Action' shall be mapped to:

```
function <<self.TTCNname()>> ( << formalParameter.collect(f | f. dataType.TTCNname() + "  " +
    f.TTCNname()).concat(",")>> ) {

        /*
```

```
        <<self.body>>
        */
    }
```

A 'formalParameter' shall be mapped to a non-template parameter if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **`template`** parameter.

For a **`template`** parameter, the TTCN-3 **`template`** keyword shall be inserted in front of its type name.

Constraints

If there is a 'DataElementMapping' provided for an 'Action' then 'DataElementMapping' shall be provided for all its 'formalParameter's.

6.2.22 Function

A 'Function' shall be mapped to a TTCN-3 function with a return type.

If there is a 'DataElementMapping' provided for a 'Function' then it shall be used for the mapping, otherwise a 'Function' shall be mapped to:

```
function <<self.TTCNname()>> ( << formalParameter.collect(f | f. dataType.TTCNname() + "  " +
    f.TTCNname()).concat(",")>> ) return  <<self.returnType.TTCNname()>> {
        /*
        <<self.body>>
        */
    }
```

A 'formalParameter' shall be mapped to a non-template parameter if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **`template`** parameter. For a **`template`** parameter, the TTCN-3 **`template`** keyword shall be inserted in front of its type name.

The 'returnType' shall be mapped to a data type if an 'Annotation' with the predefined 'AnnotationType' Value instructs this, otherwise it shall be mapped to a **`template`** data type. If the 'returnType' is a **`template`**, then the TTCN-3 **`template`** keyword shall be inserted after the **`return`** keyword.

Constraints

If there is a 'DataElementMapping' provided for a 'Function' then 'DataElementMapping' shall be provided for all its 'formalParameter's and for its returnType.

6.2.23 UnassignedMemberTreatment

This metaclass has no dedicated mapping, it is used solely in the mapping of other metaclasses.

6.2.24 PredefinedFunction

This metaclass has no dedicated mapping. See clause 10.5.

6.2.25 EnumDataType

If there is a 'DataElementMapping' provided for an 'EnumDataType' then it shall be used for the mapping, otherwise an 'EnumDataType' shall be mapped to an enumerated type:

```
type enumerated <<self.TTCNname()>> {
```

<< value.collect(v | v.TTCNname()).concat("**,**")>>

}

6.3 Mapping of Data Use Elements

6.3.1 DataUse

This is an abstract metaclass, therefore no mapping is defined. Mapping of 'DataUse' depends on it sub-class.

In case a 'reduction' is provided, it shall be mapped to dot-notation or array access expression.

If in a 'MemberReference' of a 'reduction' the 'member' is defined then it shall be mapped to:

<<self.member.TTCNname()>>

If in a 'MemberReference' of a 'reduction' the 'collectionIndex' is defined then it shall be mapped to:

[<<self.collectionIndex>> **]**

6.3.2 ParameterBinding

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

6.3.3 StaticDataUse

This is an abstract metaclass, therefore no mapping is defined. Mapping of 'StaticDataUse' depends on it sub-class.

6.3.4 DataInstanceUse

If self.dataInstance is provided, a 'DataInstanceUse' shall be mapped to a usage of a template or a constant if an 'Annotation' with the predefined 'AnnotationType' Value instructs this.

If a 'DataInstanceUse' has argument(s) then it shall be mapped to an inline template modification. If at least one argument has also argument(s), then before the actual usage of the data, a sequence of template modifications shall be generated.

If no argument specified:

<<self.dataInstance.TTCNname()>>

If argument(s) are specified, but the argument(s) themselves have no argument(s):

modifies <<self.dataInstance.TTCNname()>> **:= {** << argument.collect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat("**,**")>> **}**

If self.dataInstance is not provided, then:

<<self.resolveDataType().TTCNname()>> **: {** << argument.collect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat("**,**")>> **}**

or if the 'item' is specified, to:

<<self.resolveDataType().TTCNname()>> **: {** << item.collect(i | i).concat("**,**")>> **}**

where the 'item' shall be mapped to a corresponding 'DataUse' according to clause 6.3.1 of the present document.

In case of nested arguments (there is at least one argument that has argument(s)) the argument(s) shall be processed in an iterative way:

- Take the "innermost" argument (the argument that has no further argument).

- Define a temporary template to which the proper template modification (as described above) is assigned.
- Use this temporary template in the template modification at the next, "outer" level.
- Repeat until the "outermost" argument is processed.

EXAMPLE:

TDL:

```
Type SESSIONS (id1 of type Integer, id2 of type Integer);
Type MSG (ses of type SESSIONS, content of type String);
Type ENCAPSULATED_MSG (header of type String, msg of type MSG);

SESSIONS s1(id1 = 1, id2 = 2);
SESSIONS s2(id1 = 11, id2 = 22);
MSG m1(ses = s1, content = "m1");

SESSIONS c_s1(id1 = 1, id2 = 2) with {VALUE} ;  -- predefined annotation
SESSIONS c_s2(id1 = 11, id2 = 22) with {VALUE} ;  -- predefined annotation
MSG c1(ses = c_s1, content = "c1") with {VALUE} ;  -- predefined annotation

ENCAPSULATED_MSG e_m(header = "h", msg = m1);
ENCAPSULATED_MSG e_m2(header = "hh", msg = (ses = s2, content = "e_m2");
ENCAPSULATED_MSG e_m3(header = "hhh", msg = m1(ses := s2, content := "e_m3");

Action ACT(MSG m);

…

variable v1 of type MSG with {VALUE} ;  -- predefined annotation
variable v2 of type MSG;
variable v3 of type MSG;

perform action ACT(m1(content = "a1"));
perform action ACT(m1(ses = s2, content = "a2"));
perform action ACT(c1(ses = c_s2, content = "c2"));
perform action ACT(m1(ses = s1(id1 = 111), content = "a3"));

v1 = m1(content = "v1");
v2 = m1(content = "v2");
v3 = m1(ses = s1(id1 = 111), content = "v3");
```

TTCN-3:

```
type record SESSIONS {
  integer id1,
  integer id2
 }

type record MSG {
  SESSIONS ses,
  charstring content
}

type record ENCAPSULATED_MSG {
  charstring header,
  MSG
}

function ACT(template MSG m){};

template SESSIONS s1 := {id1 := 1, id2 := 2}
template SESSIONS s2 := {id1 := 1, id2 := 2}
template MSG m1 := {ses := s1, content := "m1"}

const SESSIONS c_s1 := {id1 := 1, id2 := 2}
const SESSIONS c_s2 := {id1 := 1, id2 := 2}
const MSG c1 := {ses := c_s1, content := "c1"}

template ENCAPSULATED_MSG e_m := {header := "h", msg := m1}
template ENCAPSULATED_MSG e_m2 := {header := "hh", msg := {ses := s2, content := "e_m2"}}
```

If the argument has an argument (nested arguments), an iterative template modification shall be used:

```
template MSG t_m1_1 modifies m1 :={ses := s2, content := "e_m3"}
```

```
template ENCAPSULATED_MSG e_m4 := {header := "hh", msg := t_m1_1}

// …

var MSG v1;
var template MSG v2;
var template MSG v3;

ACT(modifies m1 := {content := "a1"});
ACT(modifies m1 := {ses := s2, content := "a2"});
```

If the dataInstance used in 'DataInstanceUse' is not a template, but a value, then it shall be assigned to a temporary template which can later be modified:

```
template MSG t_tc1 := c1;
ACT(modifies t_tc1 := {ses := c_s2, content := "c2"});
```

If the argument has an argument (nested arguments), an iterative template modification shall be used:

```
template SESSIONS t_s1_1 modifies s1 := {id1 := 111};
ACT( modifies m1 := {ses := t_s1_1, content := "a3"});
```

If the 'DataInstanceUse' is used as a value, valueof( ) function shall be used:

```
v1 := valueof(modifies m1 := {content := "v1"});
```

If the 'DataInstanceUse' with argument(s) is used at the right hand side of an 'Assignment', then a temporary template (t_t2 and t_t3 in the example) shall be defined and this temporary template shall be used in the 'Assignment'.

```
template MSG t_t2 modifies m1 := {content := "v2"}
v2 := t_t2;

template SESSIONS t_s1_2 modifies s1 := {id1 := 111}
template MSG t_t3 modifies m1 := {ses := t_s1_2, content := "v3"}
v3 := t_t3;
```

The following applies in case 'DataInstanceUse' is used at the right hand side of:

- 'memberSpec' of 'MemberAssignment';
- 'item' of 'CollectionDataInstance;
- 'dataUse' of 'ParameterBinding'; or
- 'expression' of 'Assignment'.

If the 'StructuredDataType' of the 'DataInstanceUse' extends the 'StructuredDataType' (`targetType`) at the left hand side then field assignment shall be used:

> **{** <<`targetType`.allMembers().collect(m | <<m.TTCNname()>> + " := " + <<self.dataInstance.TTCNname()>> + "." + <<m.TTCNname()>>).concat("**,**")>> **}**

6.3.5 SpecialValueUse

This is an abstract metaclass, therefore no mapping is defined. Mapping of 'SpecialValueUse' depends on its sub-class.

6.3.6 AnyValue

'AnyValue' shall be mapped to a TTCN-3 AnyValue symbol: **?**

6.3.7 AnyValueOrOmit

'AnyValueOrOmit' shall be mapped to a TTCN-3:

- AnyValue symbol: **?** when it is assigned to a non-optional 'Member'.
- AnyValueOrNone symbol: ***** when it is assigned to an optional 'Member'.

6.3.8 OmitValue

'OmitValue' shall be mapped to a TTCN-3 keyword **omit**.

6.3.9 DynamicDataUse

This is an abstract metaclass, therefore no mapping is defined. Mapping of ' DynamicDataUse' depends on its sub-class.

6.3.10 FunctionCall

A 'FunctionCall' shall be mapped to a TTCN-3 function call.

If no argument is specified (for the return value):

<<self.function.TTCNname()>> **(** << argument.collect(a | a.dataUse).concat(",")>> **)**

If argument(s) are specified (for the return value), but the argument(s) themselves have no argument(s), a temporary template shall be defined. The name of this template (`temp`) shall be unique:

template <<self.parameter.dataType.TTCNname()>> `temp` **:=**
<<self.function.TTCNname()>> **(** << argument.collect(a | a.dataUse).concat(",")>> **)**

modifies `temp` **:= {** << argument.collect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat(",")>> **}**

In case of nested arguments (there is at least one argument that has argument(s)) the argument(s) shall be processed in an iterative way:

- Take the "innermost" argument (the argument that has no further argument).
- Define a temporary template to which the proper template modification (as described above) is assigned.
- Use this temporary template in the template modification at the next, "outer" level.
- Repeat until the "outermost" argument is processed.

EXAMPLE:

TDL:

```
Type SESSIONS (id1 of type Integer, id2 of type Integer);
Type MSG (ses of type SESSIONS, content of type String);

SESSIONS s1(id1 = 1, id2 = 2)
SESSIONS s2(id1 = 11, id2 = 22);
MSG m1(ses = s1, content = "m1");
Action ACT(MSG m);

…
Function f ( ) returns MSG;
variable vv of type MSG;

vv := instance returned from f();

perform action ACT(instance returned from f());
perform action ACT(instance returned from f()(content = "f"));
perform action ACT(instance returned from f()(ses = s1(id1 = 111), content = "f2"));

vv = instance returned from f()(content = "ff"));
```

TTCN-3:

```
type record SESSIONS {
  integer id1,
  integer id2
 }

type record MSG {
  SESSIONS ses,
  charstring content
```

```
}

function ACT(template MSG m){};

template SESSIONS s1 := {id1 := 1, id2 := 2}
template SESSIONS s2 := {id1 := 1, id2 := 2}
template MSG m1 := {ses := s1, content := "m1"}

function f () return MSG {…};

var template MSG vv;
```

'FunctionCall' with no argument:

```
vv := f();
ACT(f());
```

If the 'FunctionCall' has an argument, a temporary template (t_f_1 in the example) shall be defined that can be modified:

```
template MSG t_f_1 := f();
ACT(modifies t_f_1 := {content := "f"});
```

If the argument has an argument (nested arguments), an iterative template modification shall be used:

```
template SESSIONS t_s1_1 modifies s1 := {id1 := 111};
template MSG t_f_2 := f();
ACT( modifies t_f_2 := {ses := t_s1_1, content := "f2"});
```

If the 'FunctionCall' with argument(s) is used at the right hand side of an 'Assignment', then a temporary template (t_f_3_ in the example) shall be defined and this temporary template shall be used in the 'Assignment'.

```
template MSG t_f_3 := f();
template MSG t_f_3_ modifies t_f_3 := {content := "ff"};
vv := t_f_3_;
```

6.3.11 FormalParameterUse

If no argument specified, 'FormalParameterUse' shall be mapped to:

<<self.parameter.TTCNname()>>

If argument(s) are specified, but the argument(s) themselves have no argument(s), a temporary template shall be defined. The name of this template (`temp`) shall be unique:

`template` <<self.parameter.dataType.TTCNname()>> `temp` **`:=`** <<self.parameter.TTCNname()>>

`modifies` `temp` **`:= {`** << argument.collect(a | a.parameter.TTCNname() + " **`:=`** " + a.dataUse).concat("**`,`**")>> **`}`**

In case of nested arguments (there is at least one argument that has argument(s)) the argument(s) shall be processed in an iterative way:

- Take the "innermost" argument (the argument that has no further argument).
- Define a temporary template to which the proper template modification (as described above) is assigned.
- Use this temporary template in the template modification at the next, "outer" level.
- Repeat until the "outermost" argument is processed.

EXAMPLE:

TDL:

```
Test Description TD ( p of type MSG) ….

    Type SESSIONS (id1 of type Integer, id2 of type Integer);
    Type MSG (ses of type SESSIONS, content of type String);

    SESSIONS s1(id1 = 1, id2 = 2)
    SESSIONS s2(id1 = 11, id2 = 22);
```

```
MSG m1(ses = s1, content = "m1");

Action ACT(MSG m);

…
variable vv of type MSG;

vv := p;

//suppose action ACT is performed on 'ComponentInstance' CI
perform action ACT(p);
perform action ACT(p(content = "p"));
perform action ACT(p(ses = s1(id1 = 111), content = "p2"));

vv = p(content = "pp"));
```

TTCN-3:

```
type record SESSIONS {
  integer id1,
  integer id2
 }

type record MSG {
  SESSIONS ses,
  charstring content
}

function ACT(template MSG m){};

template SESSIONS s1 := {id1 := 1, id2 := 2}
template SESSIONS s2 := {id1 := 1, id2 := 2}
template MSG m1 := {ses := s1, content := "m1"}

//suppose action ACT is performed on 'ComponentInstance' CI
f_behaviourOfCIInTD(MSG p){

   var template MSG vv;
```

'FormalParameterUse' with no argument:

```
   vv := p;
   ACT(p);
```

If the 'FormalParameterUse' has an argument, a temporary template (t_p_1 in the example) shall be defined that can be modified:

```
   template MSG t_p_1 := p;
   ACT(modifies t_p_1 := {content := "p"});
```

If the argument has an argument (nested arguments), an iterative template modification shall be used:

```
   template SESSIONS t_s1_1 modifies s1 := {id1 := 111};
   template MSG t_p_2 := p;
   ACT( modifies t_p_2 := {ses := t_s1_1, content := "p2"});
```

If the 'FormalParameterUse' with argument(s) is used at the right hand side of an 'Assignment', then a temporary template (t_p_3_ in the example) shall be defined and this temporary template shall be used in the 'Assignment':

```
   template MSG t_p_3 := p;
   template MSG t_p_3_ modifies t_p_3 := {content := "pp"};
   vv := t_p_3_;
}
```

6.3.12 VariableUse

A 'VariableUse' shall be mapped to a usage of a variable. If the 'VariableUse' is used in a function that runs on a 'ComponentInstance' and the 'ComponentInstance' is not the same as the 'componentInstance' of the 'VariableUse' then the 'VariableUse' is ignored.

If no argument is specified:

<<self.variable.TTCNname()>>

If argument(s) are specified, but the argument(s) themselves have no argument(s), a temporary template shall be defined. The name of this template (`temp`) shall be unique:

`template` <<self.variable.dataType.TTCNname()>> `temp` **`:=`** <<self.variable.TTCNname()>>

`modifies` `temp` **`:= {`** << argument.collect(a | a.parameter.TTCNname() + " **`:=`** " + a.dataUse).concat("**`,`**")>> **`}`**

In case of nested arguments (there is at least one argument that has argument(s)) the argument(s) shall be processed in an iterative way:

- Take the "innermost" argument (the argument that has no further argument).
- Define a temporary template to which the proper template modification (as described above) is assigned.
- Use this temporary template in the template modification at the next, "outer" level.
- Repeat until the "outermost" argument is processed.

EXAMPLE:

TDL:

```
Type SESSIONS (id1 of type Integer, id2 of type Integer);
Type MSG (ses of type SESSIONS, content of type String);

SESSIONS s1(id1 = 1, id2 = 2)
SESSIONS s2(id1 = 11, id2 = 22);
MSG m1(ses = s1, content = "m1");

Action ACT(MSG m);

…
variable v of type MSG;
variable vv of type MSG;

v = m1;
v.ses = s2;

vv := v;
perform action ACT(v);

perform action ACT(v(content = "v"));
perform action ACT(v(ses = s1(id1 = 111), content = "v2"));

vv = v(content = "vv"));
```

TTCN-3:

```
type record SESSIONS {
  integer id1,
  integer id2
 }

type record MSG {
  SESSIONS ses,
  charstring content
}

function ACT(template MSG m){};

template SESSIONS s1 := {id1 := 1, id2 := 2}
template SESSIONS s2 := {id1 := 1, id2 := 2}
template MSG m1 := {ses := s1, content := "m1"}

var template MSG v;
var template MSG vv;

v := m1;
v.ses := s2;
```

'VariableUse' with no argument:

```
vv := v;
ACT(v);
```

If the 'VariableUse' has an argument, a temporary template (t_v_1 in the example) shall be defined that can be modified:

```
template MSG t_v_1 := v;
ACT(modifies t_v_1 := {content := "v"});
```

If the argument has an argument (nested arguments), an iterative template modification shall be used:

```
template SESSIONS t_s1_1 modifies s1 := {id1 := 111};
template MSG t_v_2 := v;
ACT( modifies t_v_2 := {ses := t_s1_1, content := "v2"});
```

If the 'VariableUse' with argument(s) is used at the right hand side of an 'Assignment', then a temporary template (t_v_3_ in the example) shall be defined and this temporary template shall be used in the 'Assignment':

```
template MSG t_v_3 := v;
template MSG t_v_3_ modifies t_v_3 := {content := "vv"};
vv := t_v_3_;
```

6.3.13 PredefinedFunctionCall

The 'PredefinedFunctionCall' shall be mapped to the TTCN-3 equivalent of the predefined function as is specified in clause 10.5. The actual parameters shall be mapped as is specified in clause 6.3.1.

6.3.14 LiteralValueUse

The 'LiteralValueUse' shall be mapped to TTCN-3 literal depending on its type. Unless otherwise specified, the value is mapped as is:

<<self.value>>

If 'intValue' is specified then the 'LiteralValueUse' shall be mapped to:

<<self.intValue>>

If 'boolValue' is specified then the 'LiteralValueUse' shall be mapped to:

<<self.boolValue>>

If the type is the predefined 'SimpleDataType' named String then the value is mapped to:

" <<self.value>> "

If the type has an annotation with annotation type whose name is "TTCN_DT_<TYPE_NAME>" (where <TYPE_NAME> is the name of a TTCN-3 built in type) then the value is mapped according to syntax specified in ETSI ES 201 873-1 [2].

6.3.15 DataElementUse

If the 'dataElement' is not specified or it is a 'DataInstance' then the 'DataElementUse' is mapped according to the rules specified in clause 6.3.4 with the property 'dataElement' of 'DataElementUse' used instead of the 'dataInstance' property of 'DataInstanceUse'.

If the 'dataElement' a 'Function' then the 'DataElementUse' is mapped according to the rules specified in clause 6.3.10.

If the 'dataElement' a 'FormalParameter' then the 'DataElementUse' is mapped according to the rules specified in clause 6.3.11.

7 Time

7.1 Overview

In TDL, the 'Time' is a monotonically increasing function. In TTCN-3, in the function that describes the behaviour of a 'ComponentInstance' a timer called **T_elapsedTimeOfComponent** is started.

The 'TimeLabel's are mapped to float arrays with 3 elements to store the first, previous and last timestamps of execution of an 'AtomicBehaviour'.

The mapping of 'TimeConstraint's is provided for some well-defined cases.

The 'TimeOperations' are mapped to timer start and timeout operations, while the 'TimerOperation's are mapped to their TTCN-3 counterparts.

7.2 Mapping of Time Elements

7.2.1 Time

For handling the 'Time', in each component type a timer **T_elapsedTimeOfComponent** shall be defined, that shall be started after a component instance is created by the 'f_startOf' function. This timer shall be started for a 'long enough' duration. This duration can be specified e.g. by a module parameter **mp_componentElapsedTimerMaxDuration.**

7.2.2 TimeLabel

Each 'TimeLabel' shall be mapped to a variable of TimeLabel type, which is an array of 3 **float** elements. These variables shall be defined at the beginning of the behaviour function of that 'ComponentInstance' in which the 'TimeLabel' is defined.

In the definitions part of the module a TimeLabel data type and shall be defined:

```
type float TimeLabel[3];
```

In addition to that, three constants - to handle the indexes first, previous, last - shall be defined. The recommended naming convention is **cg_tlk_** (timelabel kind):

```
const integer cg_tlk_first := 0;
const integer cg_tlk_previous := 1;
const integer cg_tlk_last := 2;
```

At the beginning of the behaviour function of that 'ComponentInstance' in which the 'TimeLabel' is specified, a variable shall be defined for that 'TimeLabel'. The recommended naming convention is **v_tl_**.

```
var TimeLabel <<self.TTCNname()>>;
```

If front of the code of that 'AtomicBehaviour' for which the 'TimeLabel' is defined, the following code shall be inserted:

```
if (not isbound(<<self.TTCNname()>>[cg_tlk_first])) {
    <<self.TTCNname()>>[cg_tlk_last] := T_elapsedTimeOfComponent.read;
    <<self.TTCNname()>>[cg_tlk_first] := <<self.TTCNname()>>[cg_tlk_last];
    <<self.TTCNname()>>[cg_tlk_ previous] := <<self.TTCNname()>>[cg_tlk_last]
}
```

```
else {
    <<self.TTCNname()>>[cg_tlk_ previous] := <<self.TTCNname()>>[cg_tlk_last];
    <<self.TTCNname()>>[cg_tlk_last] := T_elapsedTimeOfComponent.read;
}
```

7.2.3 TimeLabelUse

The 'TimeLabelUse' shall be mapped to the following:

<<self.timeLabel.TTCNname()>>**.[**<<self.kind>>**]**

7.2.4 TimeLabelUseKind

The literals of the 'TimeLabelUseKind' shall be mapped to the names of those constants that are defined for this purpose (see clause 7.2.2). The recommended names are **`cg_tlk_first`**, **`cg_tlk_previous`**, and **`cg_tlk_last`**, respectively.

7.2.5 TimeConstraint

A timeConstraintExpression can only be mapped if it satisfies the following requirements.

For tester-input events:

- The timeConstraintExpression shall be the return value of one of the following predefined functions:
  - _<_: **instanceOf**(Time), **instanceOf**(Time) → Boolean
    Denotes the standard mathematical less-than operation.
  - _>_: **instanceOf**(Time), **instanceOf**(Time) → Boolean
    Denotes the standard mathematical greater-than operation.
  - _<=_: **instanceOf**(Time), **instanceOf**(Time) → Boolean
    Denotes the standard mathematical less-or-equal operation.
  - _>=_: **instanceOf**(Time), **instanceOf**(Time) → Boolean
    Denotes the standard mathematical greater-or-equal operation.
  - _==_: **instanceOf**(DataUse), **instanceOf**(DataUse) → Boolean
    Denotes equality of the results from the evaluation of the 'DataUse's supplied as arguments. The 'DataUse's, shall refer to the same 'DataType'. Equality shall be determined based on content and not on identity.
- The arguments of these functions shall be one of the following two options:
  - First argument is the return value of the predefined function _+_: **instanceOf**(Time), **instanceOf**(Time) → **instanceOf**(Time) where its first argument is a 'TimeLabelUse' that is interpreted as a point in time and its second argument is an InstanceOf(Time) that is interpreted as a duration; and the second argument is a TimeLabelUse that is interpreted as a point in time
  - First argument is the return value of the predefined function _-_: **instanceOf**(Time), **instanceOf**(Time) → **instanceOf**(Time) where its arguments are 'TimeLabelUse's that are interpreted as points in time; and the second argument is an InstanceOf(Time) that is interpreted as a duration
- None of the arguments of the functions above shall be 'SpecialValueUse' elements.
- The timeConstraintExpression shall be mapped to the following right after the code of the constrained 'AtomicBehaviour' in the behaviour function(s) of the related component(s):

```
if( <<timeConstraintExpression>> ) { setverdict (pass)} else
{setverdict(fail)};
```

For other 'AtomicBehaviour's:

- The timeConstraintExpression shall be the return value of the following predefined function:
  - _==_: **instanceOf**(DataUse), **instanceOf**(DataUse) → Boolean
    Denotes equality of the results from the evaluation of the 'DataUse's supplied as arguments. The 'DataUse's, shall refer to the same 'DataType'. Equality shall be determined based on content and not on identity).
- The arguments of this function shall be:
  - The first argument shall be a 'TimeLabelUse' that is interpreted as a point in time (point2). This shall use the timeLabel of the constrained 'AtomicBehaviour'.
  - The second argument shall be the return value of the predefined function _+_: **instanceOf**(Time), **instanceOf**(Time) → **instanceOf**(Time) where its first argument is a 'TimeLabelUse' that is interpreted as a point in time (point1) and its second argument is an InstanceOf(Time) that is interpreted as a duration.
  - The 'TimeLabelUse' used in the first argument shall denote a point in time that happens later than the point in time denoted by the 'TimeLabelUse' used in the second argument.
- None of the arguments of the functions above shall be 'SpecialValueUse' elements.
- The timeConstraintExpression shall be mapped to the following:
  - a timer shall be declared in the behaviour function of that component(s) that contains the constrained 'AtomicBehaviour'. The name of the timer (t1 in the example below) shall be unique. The recommended naming convention is **`Tl_constraint_:`**

    ```
    timer Tl_constraint_t1;
    ```

  - in the behaviour function(s) of the related component right after the code of that 'AtomicBehaviour' whose timeLabel is used as point1 a timer start instruction shall be generated, whose argument shall be the InstanceOf(Time) that is interpreted as a duration (duration in the example below):

    ```
    Tl_constraint_t1.start(duration);
    ```

  - in front of the code of the of the constrained 'AtomicBehaviour' the following shall be added:

    ```
    if(not Tl_constraint_t1.running) {
      setverdict(false)
    }
    else {
      alt{
        [] Tl_constraint_t1.timeout {};
        [else] {repeat;}
      }
    }
    ```

NOTE: The [else] branch is used to suppress the potential active defaults.

7.2.6 TimeOperation

This is an abstract metaclass, therefore no mapping is defined. Mapping of 'TimeOperation' depends on its sub-class.

The TTCN-3 code of the sub-classes of 'TimeOperation' shall be placed into the behaviour function of the 'componentInstance'.

7.2.7 Wait

The wait instruction is mapped to a timer start and a timeout. To suppress the potential defaults to be activated before the timer expires, the timeout shall be placed into an alt instruction, whose second alternative shall be [else].

A timer shall be defined in the behaviour function of that component instance in which 'Wait' occurs:

```
timer Tl_WAIT;
```

The 'Wait' itself shall be mapped to (<<self. period>> shall be a **float** expression):

```
Tl_WAIT.start(<<self. period>>);
alt {
    [] Tl_WAIT.timeout {}
    [else] {}
}
```

7.2.8 Quiescence

The 'Quiescence' is mapped to a timer start operation and an alt instruction. The '_id' part of the timer name shall be a unique identifier (that is each 'Quiescence' shall have its own timer). If a 'Quiescence occurs as the first tester-input event of a 'Block' of an 'AlternativeBehaviour', then follow the steps specified in clause 9.3.11:

```
timer Tl_quiescence_id.start(<<self.period>>);
    alt {
        [] Tl_quiescence_id.timeout {setverdict(pass)}
```

If the 'componentInstance' or gateReference.component contains a message gate:

```
        [] any port.receive;{setverdict(fail)}
```

If the 'componentInstance' or gateReference.component contains a procedure gate:

```
        [] any port.getcall;{setverdict(fail)}
    }
```

7.2.9 Timer

A 'Timer' shall be mapped to a TTCN-3 timer definition:

```
timer <<self.TTCNname()>>;
```

The recommended naming convention for timers defined in a component type is **T_** while for other timers **Tl_**.

7.2.10 TimerOperation

This is an abstract metaclass, therefore no mapping is defined.

The TTCN-3 code of the sub-classes of 'TimerOperation' shall be placed into the behaviour function of the 'componentInstance'.

7.2.11 TimerStart

The 'TimerStart' shall be mapped to the following (<<self. period>> shall be a float expression):

```
<<self.timer.TTCNname()>>.start(<<self. period>>);
```

7.2.12 TimerStop

The 'TimerStop' shall be mapped to:

<<self.timer.TTCNname()>>**.stop;**

7.2.13 TimeOut

The 'TimeOut' shall be mapped to:

<<self.timer.TTCNname()>>**.timeout;**

8 Test Configuration

8.1 Overview

A TDL 'GateType' definition is mapped to a TTCN-3 port type definition, a 'ComponentType' definition to a component type definition, a 'GateInstance' definition to a port instance specification within a component type definition, a 'ComponentInstance' specification to a parallel test component and the connections to `connect` or `map` instructions.

Apart from mapping the elements of the test configuration defined in TDL, some additional definitions are needed:

- definition of the component type of `mtc`
- definition of the component type of `system`

In TDL, the types of the allowed connections are wider than those in TTCN-3. If a TDL 'TestConfiguration' contains connections that are not allowed in TTCN-3, then the configuration in TTCN-3 will differ from the configuration in TDL, e.g. the number of port instances defined in a component type, name or type of ports, etc. may be different. The function 'equivalent()' will be used to denote the TTCN-3 equivalent of a TDL construct.

8.2 Mapping of TestConfiguration Elements in Non-special Cases

8.2.1 Introduction

This clause provides the mapping of elements of a TDL 'TestConfiguration', if it does not contain any of the following:

a) a 'GateType' whose instances take part both in a 'Connection' between a 'ComponentInstance' in the role of 'Tester' and a 'ComponentInstance' in the role of 'SUT' (Tester-SUT connection) and in a 'Connection' between 'ComponentInstance's in the role of 'Tester' (Tester-Tester connection);

b) a 'GateReference' that is connected both to a 'ComponentInstance' in the role of 'Tester' and a 'ComponentInstance' in the role of 'SUT';

c) more than one 'ComponentInstance' in the role of 'SUT';

d) a 'GateReference' of a 'ComponentInstance' in the role of 'Tester' that is connected to two different 'ComponentInstance's in the role of 'SUT';

e) a 'GateReference' of a 'ComponentInstance' in the role of 'Tester' that is connected to two different 'GateInstance's of the same 'ComponentInstance'.

8.2.2 GateType

A 'GateType' is mapped to a port type definition, where for each 'DataType' referenced in the 'dataType' property, the corresponding message data type (in the case if property 'kind' is set to 'Message') or signature (in the case if property 'kind' is set to 'Procedure') is listed as 'inout' in the port type definition. The recommended naming convention for the GateType is <<self.name>>**_PT**.

If property 'kind' is set to 'Message':

type port <<self.equivalent().TTCNname()>> **message {**

inout << allDataTypes().collect(d | d.equivalent().name).concat(",") **;**>>

}

If property 'kind' is set to 'Procedure'

type port <<self.equivalent().name>> **procedure {**

inout << allDataTypes().collect(d | d.equivalent().name).concat(",") **;**>>

}

EXAMPLE:

TDL:


GT1
Data Type: DT1, DT2

TTCN-3:

```
type port GT1_PT message {
     inout DT1,DT2;
}
```

8.2.3 GateTypeKind

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

8.2.4 GateInstance

A 'GateInstance' definition is mapped to a port instance definition within a component type definition. The recommended TTCNname is <<self. name>>:

port <<self.type.TTCNname()>> << self.TTCNname() >> **;**

A 'GateInstance' definition will only be used in a 'ComponentType' definition.

8.2.5 ComponentType

In TTCN-3, only the behaviour of the testers can be defined. Therefore only those 'ComponentType's shall be mapped that have at least one 'ComponentInstance' in the role of Tester.

A 'ComponentType' is mapped to a component type definition. The contained 'Variable's, 'Timer's and 'GateInstance's are mapped to the corresponding TTCN-3 constructs according to the mapping specifications in the respective clauses.

The recommended naming convention for a component type is <<self.name>>**_CT**, for a timer is **T**_<<self.name>>, while the recommended naming convention for a variable is **v**_<<self.name>>:

type component <<self.equivalent().TTCNname()>> **{**

timer T_elapsedTimeOfComponent := mp_componentElapsedTimerMaxDuration;

<< timer.collect(t | "**timer** " +t.TTCNname()).concat("**;**")>>

<< variable.collect(v | "**var** " +v.dataType.TTCNname()+" "+v.TTCNname()).concat("**;**")>>

<< gateInstance.collect(g | "**port** " +g.equivalent().type.TTCNname() + " " +
g.equivalent().TTCNname()).concat("**;**")>>

}

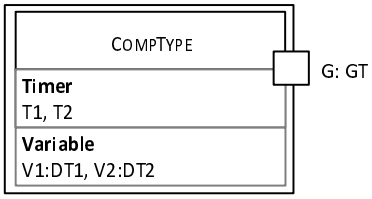
NOTE: **mp_componentElapsedTimerMaxDuration** is a **float** module parameter that determines the maximal duration of the lifetime of componentInstances.

If the 'ComponentType' has an extension then:

type component <<self.equivalent().TTCNname()>> **extends**
<<self.extension.extending.equivalent().TTCNname())>> **{}**

EXAMPLE:

TDL:



TTCN-3:

```
type component CompType_CT {
     timer T_elapsedTimeOfComponent := mp_componentElapsedTimerMaxDuration;
     timer T_T1;
     timer T_T2;
     var DT1 v_V1;
     var DT2 v_V2;
     port GT_PT G;
}
```

8.2.6 ComponentInstance

A 'ComponentInstance' in the role of 'Tester' is mapped to a parallel test component (PTC). The PTC is created by the 'TestConfiguration'. The variable that holds the address of the PTC is defined in the component type of MTC (see clause 8.2.11).

8.2.7 ComponentInstanceRole

This metaclass is not to be mapped.

8.2.8 GateReference

A 'GateReference' shall be mapped to:

<<self.gate.TTCNname()>>

8.2.9 Connection

In TTCN-3, only the behaviour of the testers can be defined. Therefore only the Tester-SUT and inter-Tester 'Connection's shall be mapped, while the inter-SUT 'Connection's shall not be mapped. The Tester-SUT connections shall be mapped to **map** instructions, while the inter-Tester connections to **connect** instructions.

Tester-SUT 'Connection's (If one <<self.endpoint.component.role>> is SUT):

map(<<self.endPoint->at(0).component.TTCNname()>> **:** <<self.endPoint->
at(0).equivalent().gate.TTCNname()>> **,** <<self.endPoint->at(1).component.TTCNname()>> **:**
<<self.endPoint->at(1).equivalent().gate. TTCNname()>>**);**

where at the "SUT side" of the 'Connection' **system** shall be used instead of
self.endpoint.component.TTCNname().

Inter-Tester connections (If both <<self.endpoint.component.role>> are Tester):

connect(<<self.endPoint->at(0).component.TTCNname()>> **:** <<self.endPoint->
at(0).gate.equivalent().TTCNname()>> **,** <<self.endPoint->at(1).component.TTCNname()>> **:**
<<self.endPoint->at(1).gate.equivalent().TTCNname()>>**);**

Inter-SUT connections (If both <<self.endpoint.component.role>> are SUT): no code shall be generated.

8.2.10 TestConfiguration

A 'TestConfiguration' is mapped to a function that creates and connects the corresponding Parallel Test Components (PTCs). The variables that store the addresses of the created PTCs are defined in the component type of MTC.

Alternatively, if in an implementation usage of a function for this purpose is inconvenient, then the body of the below defined function may be generated into the testcase at the same place where this function is called (see clause 9.2.1):

function f_setupTestConfiguration<<self.TTCNname()>>**()runs on MTC_CT {**

<< componentInstance.select(c | c.role = Tester).collect(c | c.TTCNname() + " **:=** " +
c.type.equivalent().TTCNname()+"**.create**").concat("**;**")>>

<< connection.collect(c | as in 8.2.9).concat("**;**")>>

}

8.2.11 Definition of the component type of MTC

In TTCN-3, the test cases are executed on a component instance called Main Test Component (MTC). The TTCN-3 code shall also contain the definition of the component type of the MTC. This component type shall contain no gate instance definitions, but shall contain variable definitions that will be used to store the addresses of the PTCs used in the test configuration. The recommended naming convention for these variables are: **vc_** <<self.name>>:

type component MTC_CT {

<< componentInstance.select(c | c.role = Tester).collect(c | "**var** "+c.type.equivalent().TTCNname() + " **vc_**"
+ c.TTCNname()).concat("**;**")>>

}

8.2.12 Definition of the component type of system

If a 'TestConfiguration' contains only one 'ComponentInstance' in the role SUT, then its type will be the type of the **system** component in TTCN-3. If it contains 'Variable's, 'Timer's, they shall not to be mapped, only the 'GateInstance's:

type component System_CT {

<< gateInstance.collect(g | "**port** " +g.equivalent().type.equivalent().TTCNname() + "
"+g.equivalent().TTCNname()).concat("**;**")>>

}

8.3 Mapping of TestConfiguration Elements in Special Cases

8.3.1 Introduction

This clause collects the cases when the TTCN-3 test configuration will differ from the TDL test configuration due to connectivity restrictions in TTCN-3. These situations can be eliminated from the TDL 'TestDescriptions' by applying a proper model transformation or can be handled as special cases during the mapping.

In a TDL description the combination of some or all the following cases may exist.

NOTE: Because mapping of these special cases will require to define additional/different ports, components in TTCN-3, it may mean that two component or gate instances that had the same type in TDL may have different types in TTCN-3. It may cause duplications in the TTCN-3 code and may cause that referencing 'TestDescription's on compatible 'TestConfiguration's will not be possible without regenerating the code of the referenced 'TestDescription' with the 'TestConfiguration' of the referencing 'TestDescription'.

8.3.2 Connectable and mappable GateType

In several TTCN-3 frameworks, it shall be declared at the port type definition if the instances of that port type are either connectable or mappable, but not both. In these frameworks a 'GateType' whose instances take part both in 'Connection's between a 'ComponentInstance' in the role of 'Tester' and a 'ComponentInstance' in the role of 'SUT' (Tester-SUT connection) and in 'Connection's between 'ComponentInstance's in the role of 'Tester' (inter-Tester connection) shall be mapped to two different port types.

The recommended naming convention: <<self.type.name>>**_to_map_GT** and <<self.type.name>>**_to_connect_GT**.

EXAMPLE:

TDL:

TTCN-3:

```
type port GT1_to_map_PT message {
     inout DT1, DT2;
}
type port GT1_to_connect_PT message {
     inout DT1, DT2;
}
```

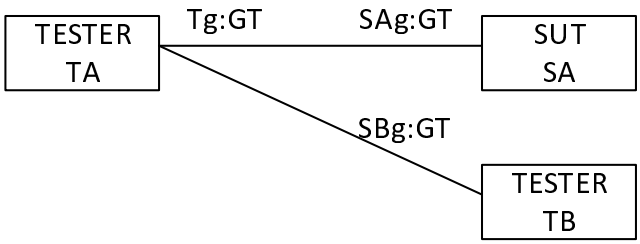
8.3.3 A gate connected to a Tester and an SUT

In TDL, a 'GateReference' can be connected both to a 'ComponentInstance' in the role of 'Tester' and a 'ComponentInstance' in the role of 'SUT', which is not allowed in TTCN-3. In this case each related 'GateInstance' shall be mapped to two port instances, one for connections, one for mappings.

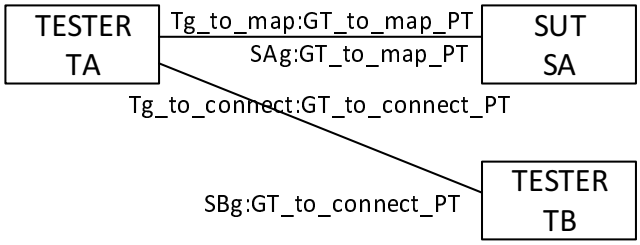
The recommended naming convention for 'GateInstance's: <<self.name>>**_to_map** and <<self.name>>**_to_connect**.

EXAMPLE:

TDL:



TTCN-3:



8.3.4 More than one SUT

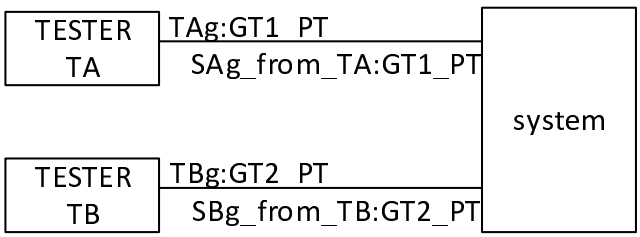
In TDL, more SUTs can be specified, while in TTCN-3 only one **`system`** can exist. In this case the 'SUT' shall be "united" to one **`system`**. This means that in the type definition of the system component as may ports shall be specified, as many 'GateInstance's all the 'ComponentInstance's in the role of 'SUT' have altogether.

The recommended naming convention for the port names of the component type of the system: use the name of that Tester 'ComponentInstance' from which the gate is connected.

EXAMPLE:

TDL:

TTCN-3 equivalent:




```
type component System_CT {
     port GT1_PT   SAg_from_TA;
     port GT2_PT   SBg_from_TB;
}
```

8.3.5 A gate of a Tester is connected to more SUTs

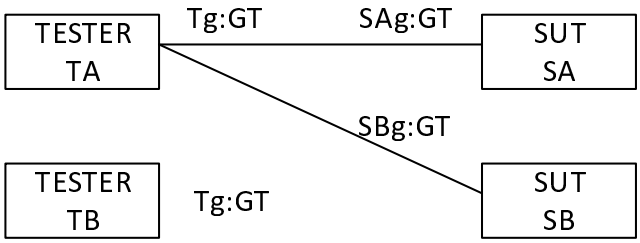
In TDL, a 'GateReference' of a 'ComponentInstance' in the role of 'Tester' may be connected to more than one different 'ComponentInstance' in the role of 'SUT', but in TTCN-3 a port can be mapped only to one system port. In these cases the 'GateInstance' of a 'Tester' shall be mapped to multiple ports. Consequence: a new component type with multiple port instances of the same type is required for that 'ComponentInstance'.

NOTE: The TTCN-3 component type may be different of 'ComponentInstance's of the same TDL 'ComponentType'.

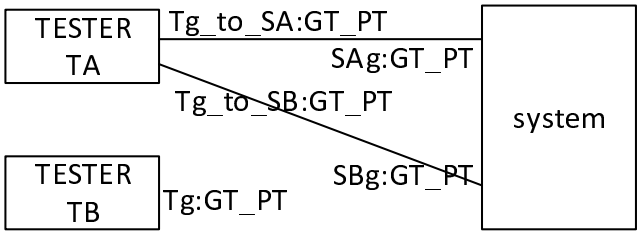
Naming convention for the 'multiplied' Tester ports: use the name of that SUT to which it is connected.

EXAMPLE:

In TDL (type of TA, TB is the same: TType):



In TTCN-3 the type of TA and TB become different:




```
type component TType_CT {  //'regular' component type; type of TB
     port GT_PT_Tg;
}
type component TType_for_TA_CT {  // component type of TA
     port GT_PT    Tg_to_SA;
     port GT_PT    Tg_to_SB;
}
```

8.3.6 A gate is connected to more gates of the same component

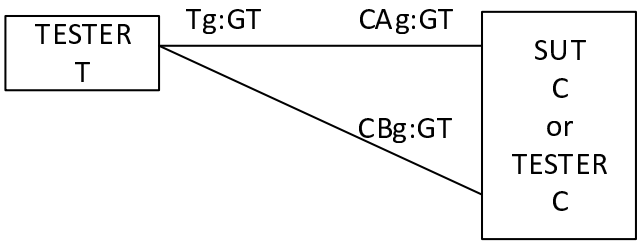
In TDL, a 'GateReference' of a 'ComponentInstance' in the role of 'Tester' may be connected to more than one different GateReference's of another 'ComponentInstance', while in TTCN-3 it is not allowed to connect or map a port instance to multiple port instances of the same component instance or to the system. In these cases the 'GateInstance' in question of a 'Tester' shall be mapped to multiple ports. Consequence: a new component type with multiple port instances of the same type is required for that 'ComponentInstance'.

NOTE: The TTCN-3 component type may be different of 'ComponentInstance's of the same TDL 'ComponentType'.

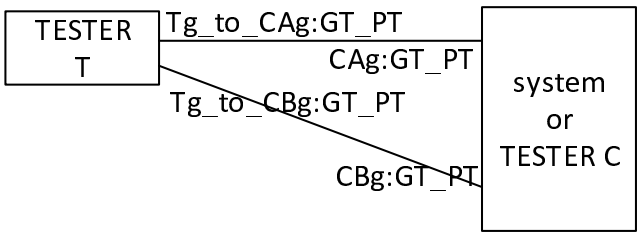
The recommended naming convention for the "multiplied" Tester ports: use the name of the gate to which it is connected.

EXAMPLE:

In TDL (Type of T is TType, Type of C is CType):



TTCN-3 equivalent:




```
type component TType_for_T_CT {  // component type of T
     port GT_PT    Tg_to_CAg;
     port GT_PT    Tg_to_CBg;
}
```

9 Test Behaviour

9.1 Overview

In TDL, a 'TestDescription' describes the behaviour of all the 'ComponentInstances' as a whole, while in TTCN-3, every component executes its own behaviour, independently from the others. The other main semantical difference between TDL and TTCN-3 is that in TDL a 'TestDescription' can call another 'TestDescription' by 'TestDescriptionReference', while in TTCN-3 a testcase cannot be called by another testcase.

The overview of the mapping of the behavioural concepts of TDL is the following.

A 'TestDescription' is mapped to:

- A testcase that runs on the MTC_CT and System_CT.
- Each 'ComponentInstance' will have its own function called f_startOf' that consists of two parts. It performs the necessary initializations (activate defaults, starting the timer that is used to measure the time in the component, etc.) and calls the behaviour function (called f_behaviourOf') of that component instance that describes the behaviour of that component. The function f_startOf will be started by the testcase.
- If a 'TestDescription' is called by another 'TestDescription' then the behaviour functions related to the called 'TestDescription's will be called from the behaviour functions related to the calling 'TestDescription'.

The 'CombinedBehaviour's will be mapped to their TTCN-3 counterparts in each participating component's behaviour function.

The 'Interaction's will be mapped to send and/or receive instructions in the behaviour functions of the related 'Tester' 'ComponentInstance's.

9.2 Mapping of Test Description Elements

9.2.1 TestDescription

A 'TestDescription' shall be mapped to a testcase and to two functions running on each 'ComponentInstance'. The first function is called f_startOf*ComponentInstanceName* and the second function is called f_behaviourOf*ComponentInstanceName*In*TestDescriptionName*.

The testcase:

testcase <<self.TTCNname()>> **(** << self.formalParameter.collect(fp | fp.dataType.TTCNname() + " "+ fp.TTCNname()).concat(", ")>> **) runs on MTC_CT system System_CT {**

If there is a 'testObjective':

<< self.testObjective.collect(t | t).concat("**\n**")>>

Call the function establishes the configuration:

f_setupTestConfiguration<<self.testConfiguration.TTCNname()>>**( );**

If there is a 'behaviourDescription' defined:

<<self.testConfiguration.componentInstance.select(c | c.role = Tester).collect(c | " **vc_**" + c.TTCNname() + "**.start(f_startOf**" + c.TTCNname() +" **(**" + self.formalParameter.collect(fp |fp.TTCNname()).concat(", ") + " **))**").concat("**;\n**")>>

If there is a port(s) in the Tester component that is connected to more than one other Tester component ("multiple connection"), then the f_startOf function - after the "testcase parameters" specified above - shall have as many additional actual parameters, as the number of Tester components which are used in 'multiple connection's. These additional actual parameters shall be the names of the corresponding component reference variables that are defined in the component type of the MTC (see clause 8.2.11).

```
    all component.done;
}
```

Apart from the testcase definition above, the following specifications shall also be made:

If there is 'behaviourDescription' defined, an altstep to handle the deviations from specification:

```
altstep to_handle_deviations_from_TDL_description_AS ( ){
    [ ] any port.receive {
        setverdict(fail);
        }
    [ ] any port.getcall {
        setverdict(fail);
        }
}
```

NOTE 1: Component type for MTC (MTC_CT) and system (System_CT) has to be created as well, see clauses 8.2.11 and 8.2.12, respectively.

If there is a 'behaviourDescription' defined, then for each 'Tester' 'componentInstance' of the 'testConfiguration' a function to be started on that component shall be generated. This function will perform the initializations (default, timer) and call the behaviour function that describes the behaviour of that 'ComponentInstance':

```
<<self.testConfiguration.componentInstance.select(c | c.role = Tester).collect(c | "f_startOf" +
c.TTCNname() +"(" + self.formalParameter.collect(fp | fp.dataType.TTCNname() + " " +
fp.TTCNname()).concat(", ") + " ) runs on " + c.type.equivalent().TTCNname() + "{

        activate (to_handle_deviations_from_TDL_description_AS ());

        T_elapsedTimeOfComponent.start;

        f_behaviourOf" + c.TTCNname() + "In" +self.TTCNname() + "(" +
    self.formalParameter.collect(fp | fp.TTCNname()).concat(", ") + ");"

}" ).concat("\n")>>
```

NOTE 2: The name of the behaviour function contains the name of the component instance and the name of the test description.

If there is a port(s) in the 'Tester' component that is connected to more than one other 'Tester' component ("multiple connection"), then:

- The f_startOf function above - after the "testcase parameters" - shall have as many additional formal parameters, as the number of Tester components which are used in 'multiple connection's. The type name of these additional formal parameters shall be the TTCN-3 type name of the corresponding Tester component, while the name of that formal parameter shall be the same as the name of the corresponding component reference variable (defined in the component type of the MTC (see clause 8.2.11)) but the recommended naming convention is **pl**_ instead of **vc**_.

- The f_behaviourOf function above - after the "testcase parameters" - shall have as many additional actual parameters, as the number of Tester components which are used in 'multiple connection's. These additional actual parameters shall be the names of the corresponding formal parameters of the startOf function.

If there is a 'behaviourDescription' defined, then for each 'Tester' 'componentInstance' of the 'testConfiguration' a behaviour function shall be generated. The body of this function will contain the 'Behaviour's to be executed by that 'ComponentInstance' as described in clauses 9.3 and 9.4 and activates 'ExceptionalBehaviour's for that 'ComponentInstance' as described in clause 9.3.14:

```
<<self.testConfiguration.componentInstance.select(c | c.role = Tester).collect(c | "f_behaviourOf" +
c.TTCNname() + "In" +self.TTCNname() + "(" + self.formalParameter.collect(fp | fp.dataType.TTCNname() +
" " + fp.TTCNname()).concat(", ") + " ) runs on " + c.type.equivalent().TTCNname() + " {

    Here comes the behaviour of that 'componentInstance'

}" ).concat("\n")>>
```

If there are port(s) in the Tester component that are connected to more than one other Tester component ("multiple connection"), then the f_behaviourOf function above - after the "testcase parameters" - shall have as many additional formal parameters, as the number of Tester components which are used in 'multiple connection's. The type name of these additional formal parameters shall be the TTCN-3 type name of the corresponding Tester component, while the name of the formal parameter shall be the same as the name of the corresponding component reference variable (defined in the component type of the MTC (see clause 8.2.11)), but the recommended naming convention is **pl**_ instead of **vc**_.

If 'ExceptionalBehaviour'(s) whose target-of-exceptional is a Tester component with "multiple connections" , then these additional parameters shall be passed to every altstep and default activation that are related to this behaviour function (see clause 9.3.14).

Constraints

Only 'TestDescription's with the property 'isLocallyOrdered' set to 'true' can be mapped.

9.2.2 BehaviourDescription

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses (see clause 9.2.1).

9.3 Mapping of Combined Behaviour elements

9.3.1 Behaviour

This is an abstract metaclass, therefore no mapping is defined. Mapping of 'Behaviour' depends on its sub-class.

If self.testObjective is not empty:

```
/*
Test Objective Satisfied: << testObjective.collect(t | t.name).concat(",")>>
*/
```

9.3.2 Block

'Block' cannot be mapped in general, mapping depends on in which 'combinedBehaviour' it is used.

9.3.3 LocalExpression

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

9.3.4 CombinedBehaviour

This is an abstract metaclass, therefore no mapping is defined. A 'CombinedBehaviour' shall be mapped to its TTCN-3 counterparts depending on its sub-class. The corresponding TTCN-3 counterpart shall be generated into the behaviour function(s) of the participating component(s).

9.3.5 SingleCombinedBehaviour

This is an abstract metaclass, therefore no mapping is defined.

9.3.6 CompoundBehaviour

A 'CompoundBehaviour' shall be mapped to a TTCN-3 instruction group (if it has no guard) or to an **if** instruction (if it has a guard):

If self.block has no guard:

```
{
    self.block.behaviour
}
```

If self.block has a 'guard' then the 'guard' specified for the corresponding 'ComponentInstance' shall be used:

```
if (<<self.block.guard.expression>>) {
    self.block.behaviour
}
```

9.3.7 BoundedLoopBehaviour

A 'BoundedLoopBehaviour' shall be mapped to a TTCN-3 **for** cycle. In the cycle the 'numIteration' specified for the corresponding 'ComponentInstance' shall be used:

The cycle variable (cv in the example below) shall be unique.

for(var integer cv :=0; cv < <<self.numIteration.expression>>**; cv:=cv+1) {**

self.block.behaviour

}

Constraints

Only 'BoundedLoopBehaviour's with an integer numIteration.expression can be mapped.

9.3.8 UnboundedLoopBehaviour

An 'UnboundedLoopBehaviour' shall be mapped to a TTCN-3 **while** cycle.

If self.block has a guard then the 'guard' specified for the corresponding 'ComponentInstance' shall be used:

while (<<self.block.guard.expression>>**) {**

self.block.behaviour

}

If self.block has no guard:

while (true) {

self.block.behaviour

}

9.3.9 OptionalBehaviour

The 'OptionalBehaviour' shall be mapped as the followings:

- At the source side:
  - In the behaviour function of the source component an **if** instruction, in which the condition is determined by the <<self.block.guard.expression>>.

    if (<<self.block.guard.expression>> **) {***self.block.behaviour***}**

- At the target(s) side:
  - An **altstep** shall be defined. This **altstep** shall have a unique name, it shall have a **runs on** clause with the (equivalent()) component type of the target component and it shall contain one alternative branch with the behaviour of the 'OptionalBehaviour' at the target side. As the last instruction of that alternative a TTCN-3 **repeat** instruction shall be generated.
  - In the behaviour function of the target 'ComponentInstance'(s), the corresponding **altstep** has to be activated as the mapped code of the given 'OptionalBehaviour' and shall be deactivated when the next 'Instruction' whose source is the same tester as the source of the 'OptionalBehaviour' (will be called as 'next input from the same Tester'). For the default activation a unique **default** type variable (**vd_dv** in the example below) shall be used.

Since the TTCN-3 standard does not specify in details how the deactivation of a default shall be implemented and the possible solutions for mapping the 'OptionalBehaviour' may have performance and/or memory consumption side-effects in certain cases, there is no standardized mapping provided how the 'OptionalBehaviour' shall be mapped at the target side, though some possible solutions are listed below.

NOTE: This may cause that the TTCN-3 code generated by different tools using different approaches may be incompatible and may require the re-generation of the TTCN-3 code from the TDL descriptions by the same tool.

The possible solutions:

a) The altstep (**as_for_optional_id_AS**, where the id part is different for each 'OptionalBehaviour') containing the code of the 'Optional'Behaviour' at the target side is not deactivated when it is activated and executed, only when the next input from the same tester arrives:

```
altstep as_for_optional_id_AS () runs on
    <<target.type.equivalent().TTCNname()>>{

  [] receiving the first tester-to-tester interaction{

       rest of the behaviour of the 'OptionalBehaviour'

       repeat;

       }

  }
```

Into the behaviour function of the target component:

```
var default vd_dv := activate (as_for_optional_id_AS());

code following 'OptionalBehaviour' up to the input from the same Tester

deactivate(vd_dv);
```

b) In each 'ComponentType' a Boolean array is defined, that controls if the altstep containing the code of the 'Optional'Behaviour' at the target side is executed or not. When the corresponding altstep is activated a new element is added to the end of the array with a 'true' value indicating that the altstep can be executed. If the altstep is activated, the corresponding element of the array is set to false that will cause to disable further execution of that altstep. The disadvantage of this solution is the potentially infinite size of the controlling array, and the fact that the defaults will not be deactivated, but the size of the elements of the controlling array is small:

```
type record of boolean BoolArray;

type component CompType_CT {
    var BoolArray optionalEnabled := {};
    var integer optionalCount := 0;
    //other definitions
}

altstep as_for_optional_id_AS(integer index) runs on CompType_CT {
    [optionalEnabled[index]] receiving the first tester-to-tester interaction {
       rest of the behaviour of the 'OptionalBehaviour'
       optionalEnabled[index] := false;
       repeat;
       }
}
function behaviourOfCompType_CTInTestDescriptionTD () runs on
    CompType_CT {
       //...
       //when 'OptionalBehaviour' occurs:
       optionalCount := lengthof(optionalEnabled);
       optionalEnabled[optionalCount] := true;
```

```
            activate(as_for_optional_id_AS (optionalCount));
            //...
            }
```

c) It is similar to the previous solution. The main difference is that instead of a Boolean array the elements of the controlling array are records with two fields: the first field is Boolean with the same purpose as in the previous case, while the second element can store a default reference that can be used for deactivation of the corresponding default from the altstep. But since according to the TTCN-3 standard, if a deactivated default is deactivated again, it causes a test case error, to prevent the situation when a default is deactivated from the altstep and at the place where the next input from the same tester arrives a local variable shall be defined for each 'OptionalBehaviour' to store the index of the last element of the controlling array at the activation. The disadvantage of this solution is the potentially infinite size of the controlling array, and the size of the elements is larger than in the previous solution, but the defaults will be deactivated:

```
type record DefEn {
    boolean optEn,
    default def
}

type record of DefEn DefEnArray;

type component CompType_CT {
    var DefEnArray optDefaultEnabler := {};
    // other definitions
}
altstep as_for_optional_id(integer index) runs on C {
    [optDefaultEnabler[index].optEn] receiving the first tester-to-tester interaction {
        rest of the behaviour of the 'OptionalBehaviour'
        optDefaultEnabler[index].optEn := false;
        deactivate (optDefaultEnabler[index].def);
        repeat;
        }
}

function behaviourOfCompType_CTInTestDescriptionTD() runs on CompType_CT {
var integer optionalCountAtActivationOfas_for_optional_id_AS :=
   lengthof(optDefaultEnabler);

   //...
   //when 'OptionalBehaviour' occurs:
   optDefaultEnabler[optionalCountAtActivationOfas_for_optional_id_AS].optEn :=
true;
   optDefaultEnabler[optionalCountAtActivationOfas_for_optional_id_AS].def :=
   activate(Optional_1_AS(optionalCountAtActivationOfas_for_optional_id_AS));

   //...

   //when the next input from same Tester arrives

   if (optDefaultEnabler[optionalCountAtActivationOfas_for_optional_id_AS].optEn ==
   true) {

      // The default was not deactivated in the body of optional

      deactivate
   (optDefaultEnabler[optionalCountAtActivationOfas_for_optional_id_AS].def);

   }

}
```

9.3.10 MultipleCombinedBehaviour

This is an abstract metaclass, therefore no mapping is defined.

9.3.11 AlternativeBehaviour

An 'AlternativeBehaviour' shall be mapped to a TTCN-3 **`alt`** instruction into the behaviour function of the target-of-alt component. In the **`alt`** instruction each 'Block' shall represent one alternative in the order of their definition:

`alt{`

If self.block has a guard

`[`<<self.block.guard.expression>>**`]`** *first tester-input event of self.block* **`{`**

rest of self.block.behaviour

`}`

If self.block has no guard:

`[ ]` first tester-input event of self.block **`{`**

rest of self.block.behaviour

`}`

`}`

If a 'Quiescence' occurs as the first tester-input event of a 'Block' of an 'AlternativeBehaviour' then:

- the timer (**`Tl_quiescence_id`** - see clause 7.2.8) associated to the corresponding 'Quiescence' shall be started before the **`alt`** instruction;
- the *first tester-input event* in the corresponding alt branch shall be **`Tl_quiescence_id.timeout;`**
- the body of the corresponding alt branch shall start with **`setverdict(pass);`** followed by the *rest of self.block.behaviour*.

9.3.12 ConditionalBehaviour

A 'ConditionalBehaviour' shall be mapped to a TTCN-3 **`if`** instruction. For the condition of the **`if`** instruction the 'guard' specified for the corresponding 'ComponentInstance' shall be used.

If the 'ConditionalBehaviour' contains more than one 'Block' then the following block(s) shall be added as **`else if`** instruction(s) or, if the last 'Block' has no 'guard' then by an **`else`** instruction:

`if (`<<self.block.guard.expression>> **`) {`***self.block.behaviour***`}`**

For the additional 'Block'(s):

`else if …`

If the last 'Block' has no guard:

`else …`

9.3.13 ParallelBehaviour

This metaclass is not to be mapped.

9.3.14 ExceptionalBehaviour

For each 'CombinedBehaviour' to which an 'ExceptionalBehaviour' is attached, for each target-of-exceptional tester component an **altstep** shall be defined. (A tester component is called as target-of-exceptional if the target of the first tester-input event of an 'ExceptionalBehaviour is that component.)

This **altstep:**

- shall have a unique name;
- shall have a **runs on** clause with the (equivalent()) component type of that target-of-exceptional component;
- shall contain - as alternatives - all the 'ExceptionalBehaviour's that are targeted to that 'ComponentInstance' in their definition order.

In the behaviour function of the target-of-exceptional 'ComponentInstance'(s), the corresponding **altstep** has to be activated in front of the mapped code of the given 'CombinedBehaviour' and shall be deactivated right after the mapped code of the given 'CombinedBehaviour.

If the 'ExceptionalBehaviour' is attached to a 'TestDescription', then the corresponding **altstep**(s) shall be activated at the beginning of the behaviour functions of the targeted 'ComponentInstances' of that 'TestDescription' and shall be deactivated at the end of them.

For each default activation a unique **default** type variable (**vd_dv** in the example below) shall be used. The recommended naming convention for these default variables is **vd_**.

In case of an 'InterruptBehaviour', a TTCN-3 **repeat** instruction shall be generated as the last instruction of its block.

NOTE 1: If the behaviour function of the target-of-exceptional component has additional parameters related to "multiple connections" (as is specified in clause 9.2.1) then these parameters have to be added to the parameter list of the **altstep**.

altstep as_AS() runs on <<*target-of-exceptional*.type.equivalent().TTCNname()>>**{**

[] ….

[] ….

}

Into the behaviour function of the target-of-exceptional component:

var default vd_dv := activate (as_AS());

code of combinedBehaviour

deactivate(vd_dv);

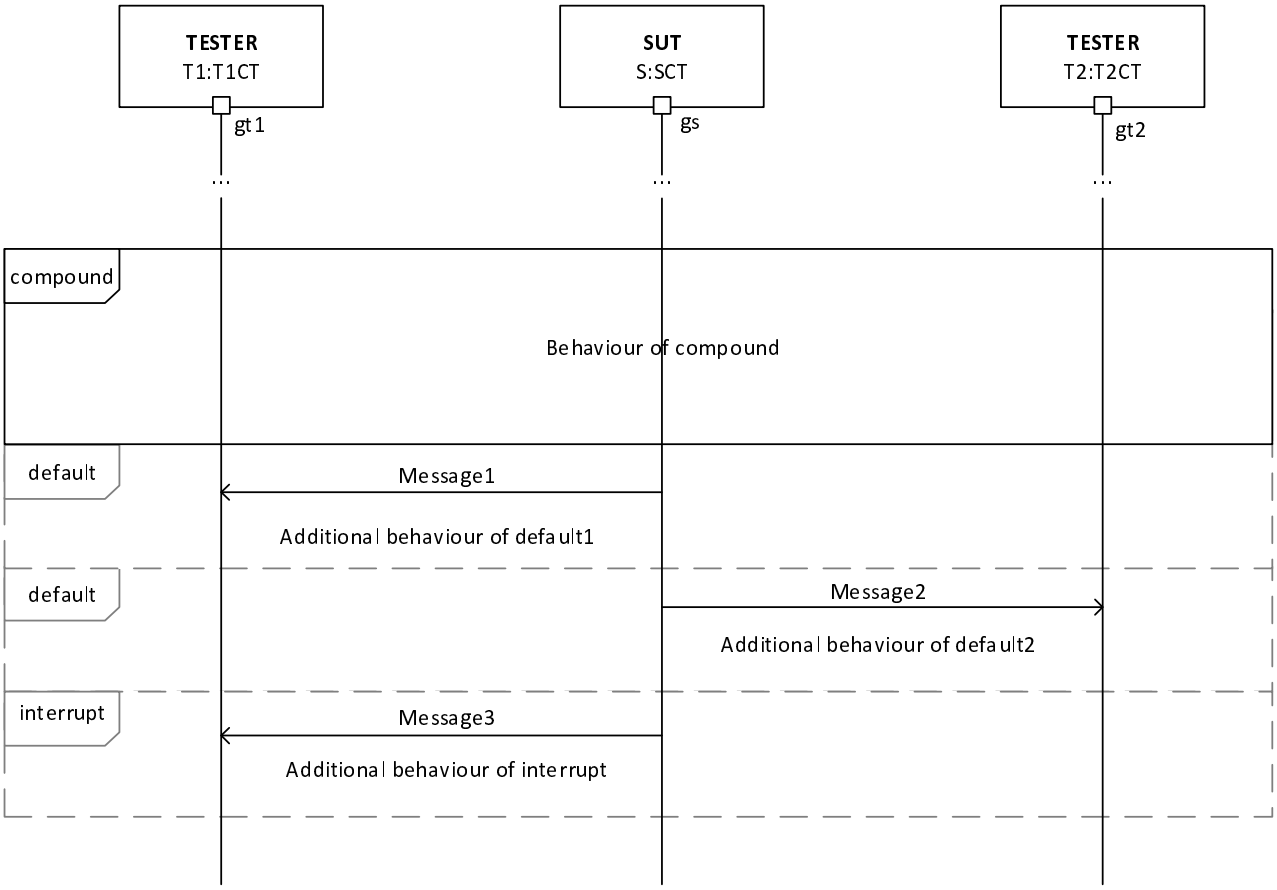
NOTE 2: If the behaviour function of the target-of-exceptional component has additional parameters related to "multiple connections" (as is specified in clause 9.2.1) then these parameters have to be added to the parameter list of the **altstep** at the activation.

Constraints

There is no standardized mapping defined for 'ExceptionalBehaviour's with 'Quiescence' as the first tester-input event.

EXAMPLE:

TDL:



TTCN-3:

```
altstep t1_AS() runs on T1CT_CT {
     [] gt1.receive(Message1){
     //Additional behaviour of default1
     }
     [] gt1.receive(Message3){
       //Additional behaviour of interrupt
       repeat;
     }
}
altstep t2_AS() runs on T2CT_CT {
     [] gt2.receive(Message2){
     //Additional behaviour of default1
     }
}
```

Into the behaviour function of tester T1:

```
f_behaviourOfT1InTestDescription1 ( ) runs on T1CT_CT{
     var default vd_1;
     //. . .
     vd_1 := activate( t1_AS( ) );
     //Behaviour of compound
     deactivate(vd_1);
     //. . .
}
```

Into the behaviour function of tester T2:

```
f_behaviourOfT2InTestDescription1 ( ) runs on T2CT_CT{
     var default vd_2;
     //. . .
     vd_2 := activate( t2_AS( ) );
     //Behaviour of compound
     deactivate(vd_2);
     //. . .
}
```

9.3.15 DefaultBehaviour

Mapping of this metaclass is specified in clause 9.3.14.

9.3.16 InterruptBehaviour

Mapping of this metaclass is specified in clause 9.3.14.

9.3.17 PeriodicBehaviour

This metaclass is not to be mapped.

9.4 Mapping of Atomic Behaviour Elements

9.4.1 AtomicBehaviour

This is an abstract metaclass, therefore no mapping is defined.

9.4.2 Break

The 'CombinedBehaviour' containing the 'ConditionalBehaviour' in which the 'Break' occurs is called as enclosing 'CombinedBehaviour'. If the enclosing 'CombinedBehaviour' is a 'BoundedLoopBehaviour', an 'UnboundedLoopBehaviour' or an 'AlternativeBehaviour' then the 'Break' shall be mapped to a **`break`** instruction, otherwise to a label at the end of the enclosing 'CombinedBehaviour' and a goto jumping to that label. If 'ExceptionalBehaviour's are defined for the containing 'ConditionalBehaviour' and/or for the enclosing 'CombinedBehaviour', then the corresponding defaults shall be deactivated before the **`break`** or after the label, respectively in reverse order of their activation. The TTCN-3 code shall be inserted to the behaviour function of all the participating components.

If the enclosing 'CombinedBehaviour' of a 'Break' is a 'BoundedLoopBehaviour', an 'UnboundedLoopBehaviour' or an 'AlternativeBehaviour':

> *here comes the deactivation of all defaults defined in the containing 'ConditionalBehaviour' and/or the enclosing 'CombinedBehaviour' and/or- -if there were any - in reverse order of their activation.*

```
break;
```

If the enclosing 'CombinedBehaviour' of a 'Break' is other than 'BoundedLoopBehaviour', 'UnboundedLoopBehaviour' or 'AlternativeBehaviour'.

At the place where 'Break' occurs:

```
goto Label_id;
```

At the end of the enclosing 'CombinedBehaviour':

```
label Label_id;
```

> *here comes the deactivation of all defaults defined in that 'CombinedBehaviour' - -if there were any - in reverse order of their activation.*

The '**_id**' part of the label name shall be a unique identifier (each label shall be different).

9.4.3 Stop

A 'Stop' shall be mapped to:

```
mtc.stop;
```

The TTCN-3 code shall be inserted to the behaviour function of all components.

9.4.4 VerdictAssignment

The 'VerdictAssignment' shall be mapped to a setverdict operation in the behaviour function of all the participating components.

setverdict (<<self.verdict>> **);**

Constraints

Only predefined 'SimpleDataInstance's ('pass', 'fail' and 'inconclusive') of the predefined 'SimpleDataType' 'Verdict' shall be mapped.

9.4.5 Assertion

Into the behaviour function determined by self.componentInstance:

If 'otherwise' is specified:

if (<<self.condition>> **) {setverdict(pass)}**

else {setverdict(<<self.otherwise>> **)};**

If 'otherwise' is not specified:

if (<<self.condition>> **) {setverdict(pass)}**

else {setverdict(fail)};

Constraints

Only predefined 'SimpleDataInstance's ('pass', 'fail' and 'inconclusive') of the predefined 'SimpleDataType' 'Verdict' shall be mapped.

9.4.6 Interaction

This is an abstract metaclass, therefore no mapping is defined.

9.4.7 Message

A 'Message' shall be mapped to TTCN-3 **send** and/or **receive**/**trigger** instructions in the behaviour function(s) of the tester component(s) involved in the 'Message'.

If the sender of the 'Message' is a 'Tester' (self.sourceGate.componentInstance.role = ComponentInstanceRole::Tester), then into the behaviour function of that 'ComponentInstance' the following TTCN-3 code shall be inserted:

<<self.sourceGate.gate.equivalent().TTCNname()>>**.send(**<<self.argument>>**);**

If the sending gate is connected to more than one gate, then the TTCN-3 **to** clause shall be used to determine the target(s) of the interaction.

If only one target is specified:

<<self.sourceGate.gate.equivalent().TTCNname()>>**.send(**<<self.argument>>**) to**
<<self.target.targetGate.component.TTCNname()>>**;**

If more than one targets are specified (multi-cast message):

<<self.sourceGate.gate.equivalent().TTCNname()>>**.send(**<<self.argument>>**) to (**
<<self.target.collect(t | t.targetGate.component.TTCNname()).concat(",")>> **);**

NOTE: In situations described in clause 8.3, if the original gate is 'split' to multiple ports, then in case of a multi-cast message sending instructions may be generated on several 'splitted' ports.

If a target of the 'Message' is a 'Tester' (any of self.target.componentInstance.role = ComponentInstanceRole::Tester), then into the behaviour function of that/those 'ComponentInstance'(s) the following TTCN-3 code shall be inserted:

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.receive(**<<self.argument>>**);**

If a target gate is connected to more than one gate, then the TTCN-3 **from** clause shall be used to determine the sender of the interaction:

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.receive(**<<self.argument>>**) from**
<<self.sourceGate.component.equivalent().TTCNname()>>**;**

If a variable is specified at a target, then the TTCN-3 value redirection shall be used:

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.receive(**<<self.argument>>**) -> value**
<<self.target.valueAssignment.variable.TTCNname()>> **;**

If the type of the argument extends the type of the variable then an additional variable (`argVariable`) of argument's type is used and the value shall be assigned to the actual variable using field assignment:

<<self.target.valueAssignment.variable.TTCNname()>> **:= {**
<<self.target.valueAssignment.variable.allMembers().collect(m | <<m.TTCNname()>> + " := " +
`argVariable` + "." + <<m.TTCNname()>>).concat(",")>> **}**

If the 'isTrigger' property is set, then instead of a **receive**, a **trigger** instruction shall be used.

9.4.8 ProcedureCall

A ProcedureCall' shall be mapped to TTCN-3 Procedure-based communication instructions.

A procedure call consists of one calling and one or more reply 'ProcedureCall's. The lifeline of the called component instance of a procedure call if notation (b) defined in clause 6.5.1 in ETSI ES 203 119-2 [4] is used or the lifeline of the corresponding gate instance of that component instance if notation (a) defined in clause 6.5.1 in ETSI ES 203 119-2 [4] is used shall be modified between the calling and the last reply *ProcedureCall*s: instead of a line a narrow rectangle, a so called 'ExecutionSymbol' shall be used.

If 'the caller of the procedure call is a 'Tester', and there is only one reply, which is not in a 'Block' of an 'AlternativeBehaviour', then into the behaviour function of the caller 'ComponentInstance' the following TTCN-3 code shall be inserted:

<<self.sourceGate.gate.equivalent().TTCNname()>>**.call(** <<self.signature.TTCNname()>> : **{**
<<self.argument.collect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat(",")>> **}) {**

If the reply contains OUT parameters:

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.getreply(**
<<self.signature.TTCNname()>> : **{** <<self.argument.collect(a | a.parameter.TTCNname() + " **:=** "
a.dataUse).concat(",")>> **}) { }**

}

or if the reply contains an EXCEPTION parameter:

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.catch(** <<self.signature.TTCNname()>> **,** <<self.argument.dataUse>> **) { }**

}

If 'valueAssignment'(s) are specified in the reply, then:

if the reply contains OUT parameters:

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.getreply(** <<self.signature.TTCNname()>> : **{** <<self.argumentcollect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat("**,**")>> **}) -> param (** <<self.target.valueAssignment.collect(v | v.variable.TTCNname() + " **:=** " + v.parameter.TTCNname()>> **){ };**

}

or if the reply contains an EXCEPTION parameter:

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.catch(** <<self.signature.TTCNname()>> **,** <<self.argument.dataUse>> **) -> value** <<self.target.valueAssignment.variable.TTCNname()>> { }

}

If the reply (replies) is (are) in 'Block's of an 'AlternativeBehaviour' then for the calling 'ComponentInstance':

<<self.sourceGate.gate.equivalent().TTCNname()>>**.call(** <<self.signature.TTCNname()>> : **{** <<self.argument.collect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat("**,**")>> **}, nowait);**

Each reply shall be in a 'Block' of an 'AlternativeBehaviour':

If the reply contains OUT parameters without 'valueAssignment':

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.getreply(** <<self.signature.TTCNname()>> : **{** <<self.argumentcollect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat("**,**")>> **}){***additional behaviour of that alternative* **}**

If the reply contains OUT parameters with 'valueAssignment':

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.getreply(** <<self.signature.TTCNname()>> : **{** <<self.argumentcollect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat("**,**")>> **}) -> param (** <<self.target.valueAssignment.collect(v | v.variable.TTCNname() + " **:=** " + v.parameter.TTCNname()>>**){***additional behaviour of that alternative* **}**

If the reply contains an EXCEPTION parameter without 'valueAssignment:

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.catch(** <<self.signature.TTCNname()>> **,** <<self.argument.dataUse>> **) {** *additional behaviour of that alternative* **}**

}

or if the reply contains an EXCEPTION parameter with 'valueAssignment:

[] <<self.target.targetGate.gate.equivalent().TTCNname()>>**.catch(** <<self.signature.TTCNname()>> **,** <<self.argument.dataUse>> **) -> value** <<self.target.valueAssignment.variable.TTCNname()>> **{** *additional behaviour of that alternative* **}**

}

If 'the called party of the procedure call is a 'Tester' then into the behaviour function of the called 'ComponentInstance' the following TTCN-3 code shall be inserted:

For receiving the call without 'valueAssignment':

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.getcall(** <<self.signature.TTCNname()>> : **{** <<self.argumentcollect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat(",")>> **});**

For receiving the call with 'valueAssignment':

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.getcall(** <<self.signature.TTCNname()>> : **{** <<self.argumentcollect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat(",")>> **}) -> param (** <<self.target.valueAssignment.collect(v | v.variable.TTCNname() + " **:=** " + v.parameter.TTCNname()>> **);**

For sending a reply with OUT parameters:

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.reply(** <<self.signature.TTCNname()>> : **{** <<self.argumentcollect(a | a.parameter.TTCNname() + " **:=** " + a.dataUse).concat(",")>> **});**

For sending a reply with an EXCEPTION parameter:

<<self.target.targetGate.gate.equivalent().TTCNname()>>**.raise(** <<self.signature.TTCNname()>> **,** <<self.argument.dataUse>> **);**

Constraints

- Procedure calls for which the reply is not in a 'Block' of an 'AlternativeBehaviour' can only be mapped, if either its 'signature' has no EXCEPTION parameters, and the reply contains arguments only with OUT parameters, or if its 'signature' has no OUT parameters but it has only one EXCEPTION parameter, and the reply contains one argument with that EXCEPTION parameter.
- If a reply in a 'Block' of an 'AlternativeBehaviour' carries an exception, then it shall have only one argument.

9.4.9 Target

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

9.4.10 ValueAssignment

This metaclass has no dedicated mapping, it is used solely in mapping of other metaclasses.

9.4.11 TestDescriptionReference

A 'TestDescriptionReference' shall be mapped to calling the behaviour functions of the 'ComponentInstance's of the called 'TestDescription' from the behaviour functions of the corresponding 'ComponentInstance's of the called 'TestDescription'.

Into the behaviour function of each tester component the following code shall be generated:

<<behaviourFunctionInReferencedTD() +"**(**" + self.argument.collect(a | a.dataUse).concat("**,** ") + "**);**">>

If there is a port(s) in the Tester component that is connected to more than one other Tester component ("multiple connection"), then the called behaviour function above - after the "testcase parameters" - shall have as many additional actual parameters, as the number of Tester components which are used in 'multiple connection's. The values of these additional actual parameters shall be the corresponding formal parameters of the calling behaviour function.

NOTE: The behaviourFunctionInReferencedTD() returns by the name of the behaviour function of the corresponding tester component in the referenced 'TestDescription'.

Constraints

A 'TestDescriptionReference' can only be mapped if no 'componentInstanceBinding' is defined (that is when the configurations of the calling and called 'TestDescription's are the same).

9.4.12 ComponentInstanceBinding

This metaclass is not to be mapped.

9.4.13 ActionBehaviour

This is an abstract metaclass, therefore no mapping is defined. Mapping of 'ActionBehaviour' depends on its sub-class.

9.4.14 ActionReference

An 'ActionReference' shall be mapped to a TTCN-3 function call. The TTCN-3 code shall be inserted to the behaviour function of that 'ComponentInstance' which is referred to by the 'componentInstance' property:

<<self.action.TTCNname()>> **(** << argument.collect(a | a.dataUse).concat(",")>> **);**

9.4.15 InlineAction

If the predefined annotation TTCN3Code is used then an 'InlineAction' shall be mapped to its body (that is a TTCN-3 code itself), otherwise it shall be mapped to a comment.

The TTCN-3 code or the comment, respectively shall be inserted to the behaviour function of that 'ComponentInstance' which is referred to by the 'componentInstance' property if it is set, otherwise to the behaviour function of all the participating components.

If the predefined annotation TTCN3Code is used:

```
//INLINE ACTION
    <<self.body>>
```

If the predefined annotation TTCN3Code is not used:

```
/* INLINE ACTION
    <<self.body>>
*/
```

9.4.16 Assignment

An 'Assignment' shall be mapped to a TTCN-3 assignment. The TTCN-3 code shall be inserted to the behaviour function of that 'ComponentInstance' which is referred to by the self.variable.componentInstance:

<<self.variable.variable.TTCNname()>> **:=** <<self.expression>> **;**

10 Predefined TDL Model Instances

10.1 Overview

10.2 Mapping of Predefined Instances of the 'SimpleDataType' Element

10.2.1 Boolean

The predefined 'SimpleDataType' 'Boolean' shall be mapped to TTCN-3 data type **`boolean`**.

10.2.2 Integer

The predefined 'SimpleDataType' 'Integer' shall be mapped to TTCN-3 data type **`integer`**.

10.2.3 String

The predefined 'SimpleDataType' 'String' shall be mapped to TTCN-3 data type **`charstring`** or to **`universal charstring`**.

10.2.4 Verdict

The predefined 'SimpleDataType' 'Verdict' shall be mapped to TTCN-3 data type **`verdicttype`**.

10.3 Mapping of Predefined Instances of 'SimpleDataInstance' Element

10.3.1 True

The predefined 'SimpleDataInstance' 'True' shall be mapped to **`true`** value of the TTCN-3 data type **`boolean`**.

10.3.2 False

The predefined 'SimpleDataInstance' 'False' shall be mapped to **`false`** value of the TTCN-3 data type **`boolean`**.

10.3.3 pass

The predefined 'SimpleDataInstance' 'pass' shall be mapped to **`pass`** value of the TTCN-3 data type **`verdicttype`**.

10.3.4 fail

The predefined 'SimpleDataInstance' 'fail' shall be mapped to **`fail`** value of the TTCN-3 data type **`verdicttype`**.

10.3.5 inconclusive

The predefined 'SimpleDataInstance' 'inconclusive' shall be mapped to **`inconc`** value of the TTCN-3 data type **`verdicttype`**.

10.4 Mapping of Predefined Instances of 'Time' Element

10.4.1 Second

The predefined instance 'Second' of the 'Time' element shall be mapped to TTCN-3 data type **`float`**.

10.5 Mapping of Predefined Instances of the 'Function' Element

10.5.1 Overview

10.5.2 Functions of Return Type 'Boolean'

- _==_: **instanceOf**(DataUse), **instanceOf**(DataUse) → Boolean
  Denotes equality of the results from the evaluation of the 'DataUse's supplied as arguments. The 'DataUse's, shall refer to the same 'DataType'. Equality shall be determined based on content and not on identity.

  This predefined function shall be mapped to the TTCN-3 **==** operator.

- _!=_: **instanceOf**(DataUse), **instanceOf**(DataUse) → Boolean
  Denotes inequality of the results from the evaluation of the 'DataUse's supplied as arguments. The 'DataUse's, shall refer to the same 'DataType'. Inequality shall be determined based on content and not on identity.

  This predefined function shall be mapped to the TTCN-3 **!=** operator.

- _and_: Boolean, Boolean → Boolean
  Denotes the standard logical AND operation.

  This predefined function shall be mapped to the TTCN-3 **`and`** operator.

- _or_: Boolean, Boolean → Boolean
  Denotes the standard logical OR operation.

  This predefined function shall be mapped to the TTCN-3 **`or`** operator.

- _xor_: Boolean, Boolean →
  Denotes the standard logical exclusive OR operation.

  This predefined function shall be mapped to the TTCN-3 **`xor`** operator.

- not: Boolean → Boolean
  Denotes the standard logical NOT operation.

  This predefined function shall be mapped to the TTCN-3 **`not`** operator.

- _<_: Integer, Integer → Boolean
  Denotes the standard mathematical less-than operation.

  This predefined function shall be mapped to the TTCN-3 **<** operator, where the arguments shall be **`integer`**.

- _>_: Integer, Integer → Boolean
  Denotes the standard mathematical greater-than operation.

  This predefined function shall be mapped to the TTCN-3 **>** operator, where the arguments shall be **`integer`**.

- _<=_: Integer, Integer → Boolean
  Denotes the standard mathematical less-or-equal operation.

  This predefined function shall be mapped to the TTCN-3 **<=** operator, where the arguments shall be **`integer`**.

- _>=_: Integer, Integer → Boolean
  Denotes the standard mathematical greater-or-equal operation.

  This predefined function shall be mapped to the TTCN-3 **>=** operator, where the arguments shall be **integer**.

10.5.3 Functions of Return Type 'Integer'

The following functions of return type 'Integer' shall be predefined:

- _+_: Integer, Integer → Integer
  Denotes the standard arithmetic addition operation.

  This predefined function shall be mapped to the TTCN-3 **+** operator, where the arguments shall be **integer**.

- _-_: Integer, Integer → Integer
  Denotes the standard arithmetic subtraction operation.

  This predefined function shall be mapped to the TTCN-3 **-** operator, where the arguments shall be **integer**.

- _*_: Integer, Integer → Integer
  Denotes the standard arithmetic multiplication operation.

  This predefined function shall be mapped to the TTCN-3 ***** operator, where the arguments shall be **integer**.

- _/_: Integer, Integer → Integer
  Denotes the standard arithmetic integer division operation.

  This predefined function shall be mapped to the TTCN-3 **/** operator, where the arguments shall be **integer**.

- _mod_: Integer, Integer → Integer
  Denotes the standard arithmetic modulo operation.

  This predefined function shall be mapped to the TTCN-3 **mod** operator, where the arguments shall be **integer**.

- size: **instanceOf**(CollectionDataInstance) → Integer
  Returns the number of members in the 'CollectionDataInstance'.

- This predefined function shall be mapped to the TTCN-3 **lengthof** predefined function.

10.5.4 Functions of Return Type of Instance of 'Time'

The following functions of return type of instance of the 'Time' meta-model element shall be predefined:

- _+_: **instanceOf**(Time), **instanceOf**(Time) → **instanceOf**(Time)
  Returns the sum of two time values of the same time data type, i.e. all parameters of the function definition shall refer to the same instance of the 'Time' element as data type.

  This predefined function shall be mapped to the TTCN-3 **+** operator, where the arguments shall be **float**.

- _-_: **instanceOf**(Time), **instanceOf**(Time) → **instanceOf**(Time)
  Returns the difference of two time values of the same time data type, i.e. all parameters of the function definition shall refer to the same instance of the 'Time' element as data type.

  This predefined function shall be mapped to the TTCN-3 **-** operator, where the arguments shall be **float**.

- _value_: **instanceOf**(Time) → Integer
  Returns the value of the 'Time' instance as an instance of 'Integer' for use with other functions.

Annex A (informative): Examples of mapping TDL to TTCN-3

A.1 Introduction

This annex provides the mappings of the same examples that were used in Annex B in ETSI ES 203 119-1 [1] and in Annex A of ETSI ES 203 119-3 [3].

A.2 A 3GPP Conformance Example in Textual Syntax

This example describes one possible way to translate clause 7.1.3.1 from ETSI TS 136 523-1 [i.1] into the proposed TDL textual syntax, by mapping the concepts from the representation in the source document to the corresponding concepts in the TDL meta-model by means of the proposed textual syntax. The example has been enriched with additional information, such as explicit data definitions and test configuration details for completeness where applicable.

TDL:

```
Package Layer_2_DL_SCH_Data_Transfer {
  //Procedures carried out by a component of a test configuration
  //or an actor during test execution
  Action preCondition : "Pre-test Conditions:
            RRC Connection Reconfiguration" ;
  Action preamble : "Preamble:
          The generic procedure to get UE in test state Loopback
          Activated (State 4) according to ETSI TS 136 508 clause 4.5
          is executed, with all the parameters as specified in the
          procedure except that the RLC SDU size is set to return no
          data in uplink.
          (reference corresponding behavior once implemented" ;

  //User-defined verdicts
  //Alternatively the predefined verdicts may be used as well
  Type Verdict ;
  Verdict PASS;
  Verdict FAIL;

  //User-defined annotation types
  Annotation TITLE ;          //Test description title
  Annotation STEP ;           //Step identifiers in source documents
  Annotation PROCEDURE ;      //Informal textual description of a test step
  Annotation PRECONDITION ;   //Identify pre-condition behaviour
  Annotation PREAMBLE ;       //Identify preamble behaviour.

  //Test objectives (copied verbatim from source document)
  Test Objective TP1 {
    from : "36523-1-a20_s07_01.doc::7.1.3.1.1 (1)" ;
    description : "with { UE in E-UTRA RRC_CONNECTED state }
             ensure that {
               when { UE receives downlink assignment on the PDCCH
                      for the UE's C-RNTI and receives data in the
                  associated subframe and UE performs HARQ
                  operation }
                then { UE sends a HARQ feedback on the HARQ
                  process }
                 }" ;
  }
  Test Objective TP2 {
    from : "36523-1-a20_s07_01.doc::7.1.3.1.1 (2)" ;
    description : "with { UE in E-UTRA RRC_CONNECTED state }
             ensure that {
               when { UE receives downlink assignment on the PDCCH
                      with a C-RNTI unknown by the UE and data is
                  available in the associated subframe }
               then { UE does not send any HARQ feedback on the
                  HARQ process }
             }" ;
```

```
}

//Relevant data definitions
Type PDU;
PDU mac_pdu ;

Type ACK ;
ACK harq_ack ;

Type C_RNTI;
C_RNTI ue;
C_RNTI unknown;

Type PDCCH (optional c_rnti of type C_RNTI);
PDCCH;

Type CONFIGURATION;
CONFIGURATION RRCConnectionReconfiguration ;

//User-defined time units
Time Second;
Second five;

//Gate type definitions
Gate Type defaultGT accepts ACK, PDU, PDCCH, C_RNTI, CONFIGURATION ;

//Component type definitions
Component Type defaultCT having {
  gate g of type defaultGT;
}

//Test configuration definition
Test Configuration defaultTC {
  create Tester SS of type defaultCT;
  create SUT UE of type defaultCT ;
  connect UE.g to SS.g ;
}

//Test description definition
Test Description TD_7_1_3_1 uses configuration defaultTC {
  //Pre-conditions and preamble from the source document
  perform action preCondition with { PRECONDITION ; } ;
  perform action preamble  with { PREAMBLE ; } ;

  //Test sequence
  SS.g sends pdcch (c_rnti=ue) to UE.g with {
    STEP : "1" ;
    PROCEDURE : "SS transmits a downlink assignment
           including the C-RNTI assigned to
           the UE" ;
  } ;
  SS.g sends mac_pdu to UE.g with {
    STEP : "2" ;
    PROCEDURE : "SS transmits in the indicated
           downlink assignment a RLC PDU in
           a MAC PDU" ;
  } ;
  UE.g sends harq_ack to SS.g with {
    STEP : "3" ;
    PROCEDURE : "Check: Does the UE transmit an
           HARQ ACK on PUCCH?" ;
    test objectives : TP1 ;
  } ;
  set verdict to PASS ;
  SS.g sends pdcch (c_rnti=unknown) to UE.g with {
    STEP : "4" ;
    PROCEDURE : "SS transmits a downlink assignment
             to including a C-RNTI different from
           the assigned to the UE" ;
  } ;
  SS.g sends mac_pdu to UE.g with {
    STEP : "5" ;
    PROCEDURE : "SS transmits in the indicated
           downlink assignment a RLC PDU in
           a MAC PDU" ;
  } ;

  //Interpolated original step 6 into an alternative behaviour,
```

```
    //covering both the incorrect and the correct behaviours of the UE
    alternatively {
      UE.g sends harq_ack to SS.g ;
        set verdict to FAIL ;
    } or {
        gate SS.g is quiet for five ;
      set verdict to PASS ;
    } with {
            STEP : "6" ;
            PROCEDURE : "Check: Does the UE send any HARQ ACK
                       on PUCCH?" ;
            test objectives : TP2 ;
    }
  } with {
    Note  : "Note 1: For TDD, the timing of ACK/NACK is not
         constant as FDD, see Table 10.1-1 of TS 36.213." ;
  }
} with {
  Note : "Taken from 3GPP TS 36.523-1 V10.2.0 (2012-09)" ;
  TITLE : "Correct handling of DL assignment / Dynamic case" ;
}
```

TTCN-3 equivalent:

```
module Layer_2_DL_SCH_Data_Transfer { //Example in MM Annex B.2

    /*
    ANNOTATION TYPE TITLE
    */
    /*
    ANNOTATION TYPE STEP
    */
    /*
    ANNOTATION TYPE PROCEDURE
    */
    /*
    ANNOTATION TYPE PRECONDITION
    */
    /*
    ANNOTATION TYPE PREAMBLE
    */
    /*
    Test Objective TP1
        Description:
            with { UE in E-UTRA RRC_CONNECTED state }
               ensure that {
                 when { UE receives downlink assignment on the PDCCH
                        for the UE's C-RNTI and receives data in the
                    associated subframe and UE performs HARQ
                      operation }
                     then { UE sends a HARQ feedback on the HARQ
                      process }
        Objective URI: 36523-1-a20_s07_01.doc::7.1.3.1.1 (1)
    */

    /*
    Test Objective TP2
        Description:
            with { UE in E-UTRA RRC_CONNECTED state }
              ensure that {
                when { UE receives downlink assignment on the PDCCH
                       with a C-RNTI unknown by the UE and data is
                   available in the associated subframe }
                   then { UE does not send any HARQ feedback on the
                      HARQ process }
        Objective URI: 36523-1-a20_s07_01.doc::7.1.3.1.1 (2)
    */

//=========================================================================
// Module Parameters
//=========================================================================

    modulepar Second mp_componentElapsedTimerMaxDuration;

//=========================================================================
// Data Types
//=========================================================================

    type charstring SimpleDataType;
```

```
    type float Second;

    type SimpleDataType PDU;
    type SimpleDataType ACK;
    type SimpleDataType C_RNTI;

    type record PDCCH {
        C_RNTI optional
    }

    type SimpleDataType CONFIGURATION;

//=========================================================================
//Port Types
//=========================================================================

    type port defaultGT_PT message {
        inout charstring, PDCCH, ACK, PDU,  C_RNTI, CONFIGURATION ;
    }

//=========================================================================
//Component Types
//=========================================================================

    type component MTC_CT { // component type for MTC
        var defaultCT_CT vc_SS;
    }

    type component System_CT {
        port defaultGT_PT   g;
    }

    type component defaultCT_CT {
        timer T_elapsedTimeOfComponent:=mp_componentElapsedTimerMaxDuration;
        port defaultGT_PT   g;
    }

//=========================================================================
// Constants
//=========================================================================

    const Second five := 5.0;

//=========================================================================
// Templates
//=========================================================================

    template PDU mac_pdu := "mac_pdu";
    template ACK harq_ack := "harq_ack";
    template C_RNTI ue := "ue";
    template C_RNTI unknown := "unknown";
    template PDCCH := {};
    template CONFIGURATION RRCConnectionReconfiguration := "RRCConnectionReconfiguration";

//=========================================================================
// Altsteps
//=========================================================================

    altstep to_handle_deviations_from_TDL_description_AS() {
        [ ] any port.receive {
                setverdict(fail);
            }
        [ ] any port.getcall {
                setverdict(fail);
            }
    }

//=========================================================================
// Functions
//=========================================================================

    function preCondition (){
        /*
        Pre-test Conditions:
                RRC Connection Reconfiguration
        */
    }
```

```
    function preamble (){
      /*
        Preamble:
              The generic procedure to get UE in test state Loopback
              Activated (State 4) according to ETSI TS 136 508 clause 4.5
              is executed, with all the parameters as specified in the
              procedure except that the RLC SDU size is set to return no
              data in uplink.
              (reference corresponding behavior once implemented)
       */
    }

    function f_setupTestConfigurationdefaultTC () runs on MTC_CT {
        vc_SS := defaultCT_CT.create;
        map (vc_SS:g,system:g);
    }

    function f_startOfSS() runs on defaultCT_CT{
        activate (to_handle_deviations_from_TDL_description_AS ());
        T_elapsedTimeOfComponent.start;
        f_behaviourOfSSInTD_7_1_3_1();
    }

    function f_behaviourOfSSInTD_7_1_3_1() runs on defaultCT_CT {
        preCondition(); /*Annotation: PRECONDITION */
        preamble()      /*Annotation: PREAMBLE */
        g.send(modifies pdcch := {c_rnti := ue})
          /*Annotation STEP "1" */
          /*Annotation PROCEDURE
                        "SS transmits a downlink assignment including the C-RNTI assigned to the UE"
             */

        g.send(mac_pdu);
          /*Annotation STEP "2" */
          /*Annotation PROCEDURE
                        "SS transmits in the indicated downlink assignment a RLC PDU in a MAC PDU"
             */

        g.receive(harq_ack);
          /*Annotation STEP "3" */
          /*Annotation PROCEDURE
                        "Check: Does the UE transmit an HARQ ACK on PUCCH?"
             */
          /*
                Test Objective Satisfied: TP1
             */
        setverdict(pass);

        g.send(modifies pdcch := {c_rnti := unknown});
          /*Annotation STEP "4" */
          /*Annotation PROCEDURE
                        "SS transmits a downlink assignment to including a C-RNTI different from the
assigned to the UE"
             */

        g.send(mac_pdu);
          /*Annotation STEP "5" */
          /*Annotation PROCEDURE
                        "SS transmits in the indicated downlink assignment a RLC PDU in a MAC PDU"
             */

        timer Tl_quiescence_1;
        Tl_quiescence_1.start(five);
        alt{
           [] g.receive(harq_ack){
              setverdict(fail);
              }
           [] Tl_quiescence_1.timeout{
                setverdict(pass);
                /*Annotation STEP "6" */
                /*Annotation PROCEDURE
                             "Check: Does the UE send any HARQ ACK on PUCCH?"
                    */
                /*
                       Test Objective Satisfied:  TP2
                    */
               }
           [] any port.receive{
```

```
                setverdict(fail);
                }
        }
        /*
            Note  : "Note 1: For TDD, the timing of ACK/NACK is not constant as FDD, see
Table 10.1-1 of TS 36.213."
        */
    }

//=====================================================================
// Testcases
//=====================================================================

   testcase TD_7_1_3_1() runs on MTC_CT system System_CT {
       activate (to_handle_deviations_from_TDL_description_AS ());
       f_setupTestConfigurationdefaultTC ();
       vc_SS.start(f_startOfSS());
        all component.done;
   }

    /*
        Note : "Taken from 3GPP TS 36.523-1 V10.2.0 (2012-09)" ;
   */
   /*
        ANNOTATION TITLE
               "Correct handling of DL assignment / Dynamic case"
    */
}
```

A.3 An IMS Interoperability Example in Textual Syntax

This example describes one possible way to translate clause 4.5.1 from ETSI TS 186 011-2 [i.2] into the proposed TDL textual syntax, by mapping the concepts from the representation in the source document to the corresponding concepts in the TDL meta-model by means of the proposed textual syntax. The example has been enriched with additional information, such as explicit data definitions and test configuration details for completeness where applicable.

TDL:

```
Package IMS_NNI_General_Capabilities {
    //Procedures carried out by a component of a test configuration
    //or an actor during test execution
    Action preConditions : "Pre-test conditions:
                - HSS of IMS_A and of IMS B is configured according to table 1
                - UE_A and UE_B have IP bearers established to their respective
                  IMS networks as per clause 4.2.1
                - UE_A and IMS_A configured to use TCP for transport
                - UE_A is registered in IMS_A using any user identity
                - UE_B is registered user of IMS_B using any user identity
                - MESSAGE request and response has to be supported at II-NNI
                  (ETSI TS 129 165 [16] see tables 6.1 and 6.3)" ;

  //User-defined verdicts
  //Alternatively the predefined verdicts may be used as well
  Type Verdict ;
    Verdict PASS ;
    Verdict FAIL ;

    //User-defined annotation types
    Annotation TITLE ;          //Test description title
    Annotation STEP ;           //Step identifiers in source documents
    Annotation PROCEDURE ;      //Informal textual description of a test step
    Annotation PRECONDITION ;   //Identify pre-condition behaviour
    Annotation PREAMBLE ;       //Identify preamble behaviour.
    Annotation SUMMARY ;        //Informal textual description of test sequence

    //Test objectives (copied verbatim from source document)
    Test Objective TP_IMS_4002_1 {
        //Location in source document
        from : "ts_18601102v030101p.pdf::4.5.1.1 (CC 1)" ;
        //Further reference to another document
        from : "ETSI TS 124 229 [1], clause 4.2A, paragraph 1" ;
        description : "ensure that {
                            when { UE_A sends a MESSAGE to UE_B
```

```
                                containing a Message_Body greater than 1 300
                                bytes }
                        then { IMS_B receives the MESSAGE containing the
                                Message_Body greater than 1 300 bytes }
                    }" ;
}
Test Objective UC_05_I {
    //Only a reference to corresponding section in the source document
    from : "ts_18601102v030101p.pdf::4.4.4.2" ;
}

//Relevant data definitions
Type MSG (optional TCP of type CONTENT);
MSG MESSAGE ;
MSG DING ;
MSG DELIVERY_REPORT ;
MSG M_200_OK

Type CONTENT ;
CONTENT tcp;

Time Second;
Second default_timeout;

//Gate type definitions.
Gate Type defaultGT accepts MSG, CONTENT ;

//Component type definitions
//In this case they may also be reduced to a single component type
Component Type USER having {
    gate g of type defaultGT ;
}
Component Type UE having {
    gate g of type defaultGT ;
}
Component Type IMS having {
    gate g of type defaultGT ;
}
Component Type IBCF having {
    gate g of type defaultGT ;
}

//Test configuration definition
Test Configuration CF_INT_CALL {
  create Tester USER_A of type USER;
  create Tester UE_A of type UE;
  create Tester IMS_A of type IMS;
  create Tester IBCF_A of type IBCF;
  create Tester IBCF_B of type IBCF;
  create SUT IMS_B of type IMS;
  create Tester UE_B of type UE;
  create Tester USER_B of type USER;
    connect USER_A.g to UE_A.g ;
    connect UE_A.g to IMS_A.g ;
    connect IMS_A.g to IBCF_A.g ;
    connect IBCF_A.g to IBCF_B.g ;
    connect IBCF_B.g to IMS_B.g ;
    connect IMS_B.g to UE_B.g ;
    connect UE_B.g to USER_B.g ;
}

//Test description definition
Test Description TD_IMS_MESS_0001 uses configuration CF_INT_CALL {
        //Pre-conditions from the source document
        perform action preConditions with { PRECONDITION ; };

        //Test sequence
        USER_A.g sends MESSAGE to UE_A.g with { STEP : "1" ; } ;
        UE_A.g sends MESSAGE to IMS_A.g with { STEP : "2" ; } ;
        IMS_A.g sends MESSAGE to IBCF_A.g with { STEP : "3" ; } ;
        IBCF_A.g sends MESSAGE to IBCF_B.g with { STEP : "4" ; } ;
        IBCF_B.g sends MESSAGE (TCP = tcp) to IMS_B.g with { STEP : "5" ; } ;
        IMS_B.g sends MESSAGE to UE_B.g with { STEP : "6" ; } ;
        UE_B.g sends DING to USER_B.g with { STEP : "7" ; } ;
        UE_B.g sends M_200_OK to IMS_B.g with { STEP : "8" ; } ;
        IMS_B.g sends M_200_OK to IBCF_B.g with { STEP : "9" ; } ;
        IBCF_B.g sends M_200_OK to IBCF_A.g with { STEP : "10" ; } ;
        IBCF_A.g sends M_200_OK to IMS_A.g with { STEP : "11" ; } ;
```

```
            IMS_A.g sends M_200_OK to UE_A.g with { STEP : "12" ; } ;
            alternatively {
                UE_A.g sends DELIVERY_REPORT to USER_A.g with { STEP : "13" ; } ;
            } or {
                gate USER_A.g is quiet for default_timeout;
            }
    } with {
        SUMMARY : "IMS network shall support SIP messages greater than
                        1 500 bytes" ;
    }
} with {
    Note : "Taken from ETSI TS 186 011-2 [i.2] V3.1.1 (2011-06)" ;
    TITLE : "SIP messages longer than 1 500 bytes" ;
}
```

TTCN-3 equivalent:

```
module IMS_NNI_General_Capabilities //Example in MM Annex B.3
{
    /*
    ANNOTATION TYPE TITLE
    */
    /*
    ANNOTATION TYPE STEP
    */
    /*
    ANNOTATION TYPE PROCEDURE
    */
    /*
    ANNOTATION TYPE PRECONDITION
    */
    /*
    ANNOTATION TYPE PREAMBLE
    */
    /*
    ANNOTATION TYPE SUMMARY
    */
    /*
    Test Objective TP_IMS_4002_1
       Description:
             "ensure that {
                           when { UE_A sends a MESSAGE to UE_B
                                  containing a Message_Body greater than 1 300
                                  bytes }
                           then { IMS_B receives the MESSAGE containing the
                                  Message_Body greater than 1 300 bytes }
                         }" ;
        Objective URI: ts_18601102v030101p.pdf::4.5.1.1 (CC 1)
                       ETSI TS 124 229 [1], clause 4.2A, paragraph 1
    */

    /*
    Test Objective UC_05_I
        Description:
          //Only a reference to corresponding section in the source document
        Objective URI: from : "ts_18601102v030101p.pdf::4.4.4.2" ;
    */

  //=========================================================================
  // Module Parameters
  //=========================================================================

  modulepar Second mp_componentElapsedTimerMaxDuration;

  //=========================================================================
  // Data Types
  //=========================================================================

  type charstring SimpleDataType;
  type float Second;

  type record MSG {
    CONTENT TCP optional
  }

  type SimpleDataType CONTENT;

  //=========================================================================
```

```
//Port Types
//=========================================================================

// Insert port type definitions here if applicable!
// You can use the port_type skeleton!

type port defaultGT_to_connect_PT message {
  inout MSG, CONTENT;
}

type port defaultGT_to_map_PT message {
  inout MSG, CONTENT;
}

//=========================================================================
//Component Types
//=========================================================================

type component MTC_CT {
  var USER    vc_USER_A;
  var UE      vc_UE_A;
  var IMS     vc_IMS_A;
  var IBCF    vc_IBCF_A;
  var IBCF    vc_IBCF_B;
  var UE      vc_UE_B;
  var USER    vc_USER_B;
  }

type component USER {
  timer T_elapsedTimeOfComponent:=mp_componentElapsedTimerMaxDuration;
  port defaultGT_to_connect_PT g_to_connect;
  port defaultGT_to_map_PT    g_to_map;
}

type component UE {
  timer T_elapsedTimeOfComponent:=mp_componentElapsedTimerMaxDuration;
  port defaultGT_to_connect_PT g_to_connect;
  port defaultGT_to_map_PT    g_to_map;
}

type component IMS {
  timer T_elapsedTimeOfComponent:=mp_componentElapsedTimerMaxDuration;
  port defaultGT_to_connect_PT g_to_connect;
  port defaultGT_to_map_PT    g_to_map;
}

type component IBCF {
  timer T_elapsedTimeOfComponent:=mp_componentElapsedTimerMaxDuration;
  port defaultGT_to_connect_PT g_to_connect;
  port defaultGT_to_map_PT    g_to_map;
}

type component System_CT {
  port defaultGT_to_map_PT    g_to_map;
}

//=========================================================================
// Constants
//=========================================================================

const Second c_default_timeout := 5.0;

//=========================================================================
// Templates
//=========================================================================

template MSG MESSAGE := {};
template MSG DING := {};
template MSG DELIVERY_REPORT := {};
template MSG M_200_OK := {};

template CONTENT tcp := "tcp";

//=========================================================================
// Altsteps
//=========================================================================

altstep to_handle_deviations_from_TDL_description_AS() {
```

```
      [ ] any port.receive {
              setverdict(fail);
          }
      [ ] any port.getcall {
              setverdict(fail);
          }
  }

//=====================================================================
// Functions
//=====================================================================

function preConditions (){
  /*
  Pre-test conditions:
               - HSS of IMS_A and of IMS B is configured according to table 1
               - UE_A and UE_B have IP bearers established to their respective
                 IMS networks as per clause 4.2.1
               - UE_A and IMS_A configured to use TCP for transport
               - UE_A is registered in IMS_A using any user identity
               - UE_B is registered user of IMS_B using any user identity
               - MESSAGE request and response has to be supported at II-NNI
                 (ETSI TS 129 165 [16] see tables 6.1 and 6.3)
  */
}

function f_setupTestConfigurationCF_INT_CALL () runs on MTC_CT {

  vc_USER_A := USER.create;
  vc_UE_A := UE.create;
  vc_IMS_A := IMS.create;
  vc_IBCF_A := IBCF.create;
  vc_IBCF_B := IBCF.create;
  vc_UE_B := UE.create;
  vc_USER_B := USER.create;

  connect(vc_USER_A:g_to_connect, vc_UE_A:g_to_connect);
  connect(vc_UE_A:g_to_connect, vc_IMS_A:g_to_connect);
  connect(vc_IMS_A:g_to_connect, vc_IBCF_A:g_to_connect);
  connect(vc_IBCF_A:g_to_connect, vc_IBCF_B:g_to_connect);
  map(vc_IBCF_B:g_to_map,system:g_to_map);
  map(system:g_to_map, vc_UE_B:g_to_map);
  connect(vc_UE_B:g_to_connect, vc_USER_B:g_to_connect);
}

function f_startOfUSER_A () runs on USER {

  activate (to_handle_deviations_from_TDL_description_AS ());
  T_elapsedTimeOfComponent.start;
  f_behaviourOfUSER_AInTD_IMS_MESS_001();
}

function f_behaviourOfUSER_AInTD_IMS_MESS_001 () runs on USER {

  g_to_connect.send(MESSAGE);
  /*Annotation STEP "1" */

  timer T_quiescence_1;
  T_quiescence_1.start(c_default_timeout);
  alt{
   [] g_to_connect.receive(DELIVERY_REPORT){
   }
   [] T_quiescence_1.timeout{
     setverdict(pass);
   }
  }
}

function f_startOfUE_A (USER pl_USER_A, IMS pl_IMS_A) runs on UE {

  activate (to_handle_deviations_from_TDL_description_AS ());
  T_elapsedTimeOfComponent.start;
  f_behaviourOfUE_AInTD_IMS_MESS_001(pl_USER_A, pl_IMS_A);
}

function f_behaviourOfUE_AInTD_IMS_MESS_001 (USER pl_USER_A, IMS pl_IMS_A) runs on UE {
                              /*if a port is connected to multiple components
                              these components shall be passed as parameters
```

```
                              to  be able refer to them in send/receive to/from*/
  g_to_connect.receive(MESSAGE) from pl_USER_A;
  g_to_connect.send(MESSAGE) to pl_IMS_A;
  /*Annotation STEP "2" */

  g_to_connect.receive(M_200_OK) from pl_IMS_A;
  g_to_connect.send(DELIVERY_REPORT) to pl_USER_A;
  /*Annotation STEP "13" */
}

function f_startOfIMS_A (UE pl_UE_A, IBCF pl_IBCF_A) runs on IMS {

  activate (to_handle_deviations_from_TDL_description_AS ());
  T_elapsedTimeOfComponent.start;
  f_behaviourOfIMS_AInTD_IMS_MESS_001(pl_UE_A, pl_IBCF_A);
}

function f_behaviourOfIMS_AInTD_IMS_MESS_001 (UE pl_UE_A, IBCF pl_IBCF_A) runs on IMS {

  g_to_connect.receive(MESSAGE) from pl_UE_A;
  g_to_connect.send(MESSAGE) to pl_IBCF_A;
  /*Annotation STEP "3" */

  g_to_connect.receive(M_200_OK) from pl_IBCF_A;
  g_to_connect.send(M_200_OK) to pl_UE_A;
  /*Annotation STEP "12" */
}

function f_startOfIBCF_A (IMS pl_IMS_A, IBCF pl_IBCF_B) runs on IBCF {

  activate (to_handle_deviations_from_TDL_description_AS ());
  T_elapsedTimeOfComponent.start;
  f_behaviourOfIBCF_AInTD_IMS_MESS_001(pl_IMS_A, pl_IBCF_B);
}

function f_behaviourOfIBCF_AInTD_IMS_MESS_001 (IMS pl_IMS_A, IBCF pl_IBCF_B) runs on IBCF {

  g_to_connect.receive(MESSAGE) from pl_IMS_A;
  g_to_connect.send(MESSAGE) to pl_IBCF_B;
  /*Annotation STEP "4" */

  g_to_connect.receive(M_200_OK) from pl_IBCF_B;
  g_to_connect.send(M_200_OK) to pl_IMS_A;
  /*Annotation STEP "11" */
}

function f_startOfIBCF_B () runs on IBCF {

  activate (to_handle_deviations_from_TDL_description_AS ());
  T_elapsedTimeOfComponent.start;
  f_behaviourOfIBCF_BInTD_IMS_MESS_001();
}

function f_behaviourOfIBCF_BInTD_IMS_MESS_001 () runs on IBCF {

  g_to_connect.receive(MESSAGE);
  g_to_map.send(modifies MESSAGE := {TCP := tcp});
  /*Annotation STEP "5" */

  g_to_map.receive(M_200_OK);
  /*Annotation STEP "9" */

  g_to_connect.send(M_200_OK);
  /*Annotation STEP "10" */

}

function f_startOfUE_B () runs on UE {

  activate (to_handle_deviations_from_TDL_description_AS ());
  T_elapsedTimeOfComponent.start;
  f_behaviourOfUE_BInTD_IMS_MESS_001();
}

function f_behaviourOfUE_BInTD_IMS_MESS_001 () runs on UE {

  g_to_map.receive(MESSAGE);
  /*Annotation STEP "6" */
```

```
    g_to_connect.send(DING);
    /*Annotation STEP "7" */
    g_to_map.send(M_200_OK);
    /*Annotation STEP "8" */
  }

  function f_startOfUSER_B () runs on USER {

    activate (to_handle_deviations_from_TDL_description_AS ());
    T_elapsedTimeOfComponent.start;
    f_behaviourOfUSER_BInTD_IMS_MESS_001();
  }

  function f_behaviourOfUSER_BInTD_IMS_MESS_001 () runs on USER {

    g_to_connect.receive(DING);
  }

  //=========================================================================
  // Testcases
  //=========================================================================

  testcase TD_IMS_MESS_001 () runs on MTC_CT system System_CT {

    activate (to_handle_deviations_from_TDL_description_AS ());

    f_setupTestConfigurationCF_INT_CALL();
    vc_USER_A.start(f_startOfUSER_A ());
    vc_UE_A.start(f_startOfUE_A (vc_USER_A, vc_IMS_A));
    vc_IMS_A.start(f_startOfIMS_A (vc_UE_A, vc_IBCF_A));
    vc_IBCF_A.start(f_startOfIBCF_A (vc_IMS_A, vc_IBCF_B));
    vc_IBCF_B.start(f_startOfIBCF_B ());
    vc_UE_B.start(f_startOfUE_B ());
    vc_USER_B.start(f_startOfUSER_B ());

    preConditions();
    /*Annotation PRECONDITION*/
    all component.done;

    /*Annotation SUMMARY
                "IMS network shall support SIP messages greater than 1 500 bytes"
    */
   }
}
```

History

Document history		
V1.1.1	June 2018	Publication
V1.2.1	August 2020	Publication
V1.3.1	March 2022	Membership Approval Procedure MV 20220527: 2022-03-28 to 2022-05-27
V1.3.1	May 2022	Publication