Final draft ETSI ES 203 119-2 V1.1.0 (2015-04)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Part 2: Graphical Syntax

Reference
DES/MTS-203 119-2

Keywords
graphical notation, language, MBT, methodology, testing

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 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is 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 (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, 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.

Foreword

This final draft ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 2 of a multi-part deliverable covering the Test Description Language as identified below:

Part 1: "Abstract Syntax and Associated Semantics";

Part 2: "Graphical Syntax";

Part 3: "Exchange Format";

Part 4: "Structured Test Objective Specification (Extension)".

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 the concrete graphical syntax of the Test Description Language (TDL). The intended use of the present document is to serve as the basis for the development of graphical TDL tools and TDL specifications. The meta-model of TDL and the meanings of the meta-classes are described in [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 reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://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 (V1.2.0): "Methods for Testing and Specification (MTS); Test Description Language; Part 1: Abstract Syntax and Associated Semantics".

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 reference 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) (2012-10): "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 (V3.1.1) (2011-06): "IMS Network Testing (INT); IMS NNI Interoperability Test Specifications; Part 2: Test Description for IMS NNI Interoperability".

[i.3] ETSI ES 203 119-3 (V1.1.0): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 3: Exchange Format".

[i.4] ETSI ES 203 119-4 (V1.1.0): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: Structured Test Objective Specification (Extension)".

3 Definitions and abbreviations

3.1 Definitions

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

diagram: placeholder of TDL shapes

lifeline: a vertical line originates from a gate instance, to which behavioural elements can be attached

NOTE: A lifeline from top to down represents how time passes.

shape: layout of the graphical representation of a TDL meta-class

3.2 Abbreviations

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

BNF	Backus-Naur Form
EBNF	Extended Backus-Naur Form
IMS	IP Multimedia Subsystem
IP	Internet Protocol
OCL	Object Constraint Language
TDL	Test Description Language
URI	Unified Resource Identifier

4 Basic principles

4.1 Introduction

The meta-model of the Test Description Language is specified in ETSI ES 203 119-1 [1]. The presentation format of the meta-model can be different according to the needs of the users or the requests of the domain, where the TDL is applied. These presentation formats can either be text-oriented or graphic-oriented and may cover all the functionalities of the TDL meta-model or just a part of it, which is relevant to satisfy the needs of a specific application domain.

The present document specifies a concrete graphical syntax that provides a graphical representation for the whole functionality of the TDL meta-model.

The document specifies the TDL diagram, where the graphical representations of the instances of the TDL meta-classes can be placed. A graphical representation can contain a shape with textual labels placed into it. The rules, how these labels shall be interpreted are described in OCL-like expressions.

4.2 Document Structure

The present document specifies the concrete graphical syntax of the Test Description Language (TDL).

Clause 5 specifies the TDL Diagram.

Clause 6 specifies the concrete shapes defined for the TDL meta-classes. (The meta-model of TDL and the meanings of the meta-classes are described in ETSI ES 203 119-1 [1].)

- Foundation (clause 6.1)
- Data (clause 6.2)
- Time (clause 6.3)
- Test Configuration (clause 6.4)
- Test Behaviour (clause 6.5)

At the end of the document several examples illustrating the features of the TDL Graphical Syntax can be found.

4.3 Notational Conventions

4.3.0 General

Elements from the TDL meta-model 1 are typed in italic, e.g. *StructuredDataType*.

The definition of the TDL Concrete Graphical Syntax consists of both shapes and textual labels placed into these shapes. Textual labels are differentiated into non-terminal textual labels and terminal textual labels. The production rule of a non-terminal textual label is specified by a combination of EBNF symbols and OCL-like expressions to navigate over the abstract syntax meta-model of TDL.

4.3.1 Symbols and meanings for shapes

Shapes consist of outermost borders, compartments, and textual labels (i.e. non-terminal textual labels and terminal-textual labels). The following conventions apply:

- Non-terminal textual labels are typed in small capitals (e.g. PRODUCTIONRULELABEL). The name of the label refers to a production rule with the same name that specifies how the result of the production rule is determined.
- If a non-terminal symbol name is typed in special, e.g. UNDERLINED or **BOLD** small capitals, underlined or bold font shall be used in the shape for the result of the production rule of that non-terminal symbol, e.g. SIMPLEDATAINSTANCENAMELABEL (non-terminal) and MyValue:MyType (a result of the production rule of that non-terminal) or **COMPONENTROLELABEL** (non-terminal) and **TESTER** (a result of the production rule of that non-terminal)), etc.
- Terminal textual labels are typed in non-small capital characters. They shall be typeset in the same font, as they appear on the figure, e.g. if a terminal textual label is typed in **bold**, bold font shall be used in the shape for that terminal textual symbol, e.g. **timer**, etc.
- The outermost border of a shape shall not be hidden, unless it is stated explicitly.
- Compartments and non-terminal textual labels may be hidden to simplify the internal structure of the shape.
- In the figures, optional compartments are shaded in a light grey colour, while optional non-terminal textual labels are typed in grey colour. However, the colour and the shading indicates only the optionality of a compartment or a non-terminal label. That is, if they are actually present in a test description, they shall not be shaded and shall be typed in black.
- If a non-terminal textual label is defined to be optional, that non-terminal textual label can only be shown if the surrounding compartment is shown and the corresponding non-terminal textual production rule results in a non-empty string or a non-empty collection of strings.
- If an optional compartment contains a mandatory terminal or non-terminal textual label, the text shall only be shown if the surrounding compartment is shown.
- References to non-terminal textual production rules external to the given shape are represented by the name of the referenced production rule enclosed in angle brackets (e.g. <REFERENCEDPRODUCTIONRULE>).
- A non-terminal textual label in between hashmarks (e.g. #ELEMENT#) denotes a placeholder for a shape identified by that non-terminal textual label.

4.3.2 Symbols for non-terminal textual labels

Non-terminal textual labels are specified by production rules (so called non-terminal textual label production rule). The formal specification of a non-terminal textual label production rule is expressed by OCL. The context meta-model element for the OCL expression is specified prior to the non-terminal textual label specification. In some cases, the definition of OCL expression would be too complex for understanding. In that case, pseudo-code like helper notations are used.

The OCL expressions are combined with a variant of the Backus-Naur Form (BNF). The conventions within the present document for the production rules are:

- OCL keywords and helper functions are typed in **bold.**
- The keyword **context** followed by the name of TDL metaclass determines the context element for the following production rule (e.g. **context** Package).
- Non-terminal textual labels production rule identifiers are always represented in small capitals (e.g. LABELPRODUCTIONRULE).
- Non-terminal textual label production rule definitions are signified with the '**::=** ' operator.
- OCL expressions are written in lower case characters (e.g. self.name).

- Non-terminal textual labels may contain terminal symbols. A terminal symbol is enclosed in single quotes (e.g. 'keyword' or '[').
- Alternative choices between symbols in a production are separated by the '|' symbol (e.g. symbol1 | symbol2).
- Symbols that are optional are enclosed in square brackets ' [ ] ' (e.g. [symbol]).
- In case the context of an OCL expression need to be changed for non-terminal textual label production rule, the predefined function *variable* **as context in <LABELPRODUCTIONRULE>** shall be used to invoke a production rule of a different metaclass, where v*ariable* refers to an instance of a metaclass that complies with the context of the invoked **<LABELPRODUCTIONRULE>.**
- If the OCL expression of a production rule results in a collection of strings, a collection helper function **separator(String)** can be used to specify the delimiter between any two strings in the collection, e.g. self.collectionProperty->**separator(**','**)**. The collection helper function **newline( )** inserts a line break between any two strings in the collection.
- Iterations over collections of attributes of a metaclass use a verbatim (non-OCL) helper function *foreach* with the following syntax: **foreach** *VariableName* ':' *VariableType* [**separator(String)|newline( )**] **in** *OCLexpression* **end**. *VariableName* is an alphanumeric word signifying the variable used for subsequent statement. *VariableType* is a string that shall be the same as a TDL metaclass name. *OCLexpression* is an OCL statement that resolves in a collection of metaclass elements compliant to the metaclass given in *VariableType*. For example, the statement LABEL **::= foreach** *e:Element* ***in*** *self.attribute* **end**, iterates of the elements in the collection *self.attribute* and stores resulting element of each iteration in variable *e*. The variable *e* can be used in the body of the loop for further calculations. In every iteration, the non-terminal textual production rule LABEL is invoked, and the respective instance of metaclass *Element* that is stored in *e* will be used in the invoked production rule. The collection helper functions **separator(String)** and **newline( )** can also be applied directly to the **foreach** construct.

4.3.3 Example

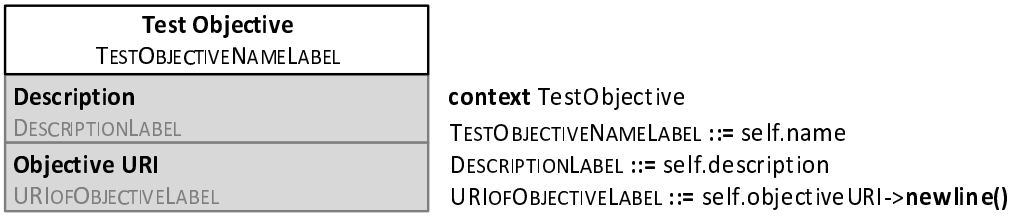
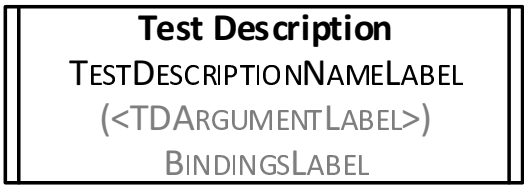


Figure 4.1: Notational convention example 1

In figure 4.1, the following notational concepts of the TDL Concrete Graphical Syntax are shown:

- The uppermost compartment contains a terminal textual label (a keyword) 'Test Objective' typed in bold.
- The context meta-model element of this shape is *TestObjective*.
- The non-terminal textual label production rule TESTOBJECTIVENAMELABEL results in the name of the context element (i.e. self.name).
- There are two optional compartments (i.e. shaded grey) shown ordered from top to down.
- Both compartments contain a mandatory terminal textual label (i.e. the label shall be shown if the surrounding compartment is shown). The terminal textual labels shall be typed in bold (**Description** and **Objective URI**, respectively).
- Both compartments contain an optional non-terminal textual label (i.e. the label shall be shown if the surrounding compartment is shown and the production rules results in a non-empty string or a non-empty collection of strings).
- The separator between the elements of the self.objectiveURI in production rule URIOFOBJECTIVELABEL is a new line.




```
context TestDescriptionReference
TESTDESCRIPTIONNAMELABEL ::= self.testDescription.name

TDARGUMENTLABEL ::= foreach d:DataUse in self.actualParameter separator(',')
                                d as context in <DATAUSELABEL>
                    end

BINDINGSLABEL ::= foreach c : ComponentInstanceBinding in self.componentInstanceBinding separator(',')
                            c.componentInstanceBinding.actualComponent.name '->'
                            c.componentInstanceBinding.formalComponent.name
                  end
```

Figure 4.2: Notational convention example showing the foreach helper function

In figure 4.2, the use of a non-OCL *foreach* helper function is illustrated. The context element when entering the foreach loop is *TestDescriptionReference*. The first foreach loop assigns iteratively each element in the collection *self.actualParameter* to the variable *d* of type *DataUse*. The variable *d* then used as it is described in the referenced production rule DATAUSELABEL. The separator between the results of the iterations is ',' (a comma character). The second foreach loop assigns iteratively each element in the collection *self.componentInstanceBinding* to the variable *c* of type *ComponentInstanceBinding*. The variable *c* is then used in a subsequent non-terminal textual label production rule to build the label for the production rule. The separator between the results of the iterations is ',' (a comma character).

4.4 Conformance

For an implementation claiming to conform to this version of the TDL Concrete Graphical Syntax, 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 Diagram

There is only one diagram kind provided by TDL Concrete Graphical Syntax. This diagram is called TDL Diagram. There can be multiple instances of a TDL Diagram at the same time.

The shapes that can be placed onto a TDL Diagram are specified in clause 6.

6 Shapes

6.1 Foundation

6.1.1 NamedElement

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

Formal Description

```
context NamedElement
NAMEDELEMENTLABEL ::= self.name
```

Comments

No comments.

6.1.2 ElementImport

Concrete Graphical Notation

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

Formal Description

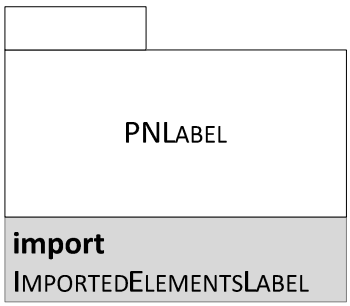
```
context ElementImport
IMPORTLABEL ::= 'from' self.importedPackage.qualifiedName
        if self.importedElement->isEmpty() then
            'all'
        else
            self.importedElement.name->separator(',')
        endif
```

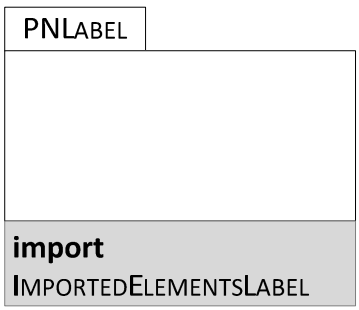
Comments

No comments.

6.1.3 Package

Concrete Graphical Notation





Formal Description

```
context Package
PNLABEL ::= self.name
IMPORTEDELEMENTSLABEL ::= foreach i:ElementImport in self.import
                              i as context in <IMPORTLABEL> separator(',')
                          end
```

Comments

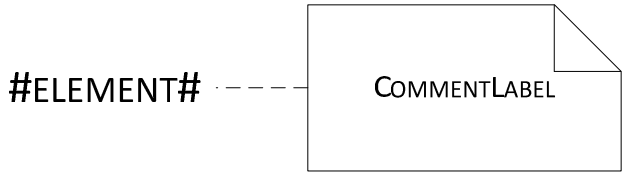
The figures above indicate the two possible representations of the *Package* shape: the PNLABEL can be written either in the top, small compartment or in the middle one.

The elements the package contains (packagedElements) may be shown within the large rectangle in the middle. In this case the PNLABEL shall be in the upper small compartment.

The lower **import** compartment is optional, it shall only be represented if the package imports other package(s) or elements from other package(s). If this compartment is present, its content shall also be present.

6.1.4 Comment

Concrete Graphical Notation



Formal Description

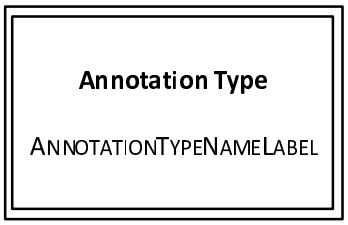
context Comment
COMMENTLABEL ::= self.body

Comments

A *Comment* shape shall be attached to the commented element by a thin dashed line.

6.1.5 AnnotationType

Concrete Graphical Notation



Formal Description

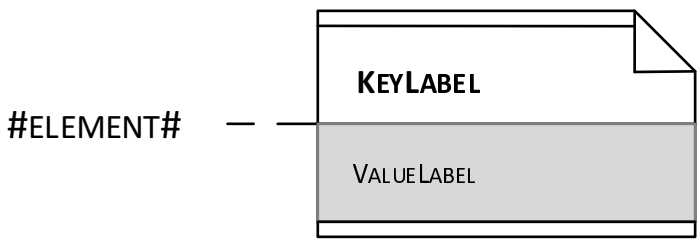
context AnnotationType
ANNOTATIONTYPENAMELABEL ::= self.name

Comments

No comments.

6.1.6 Annotation

Concrete Graphical Notation



Formal Description

context Annotation
KEYLABEL ::= self.key.name
VALUELABEL ::= self.value

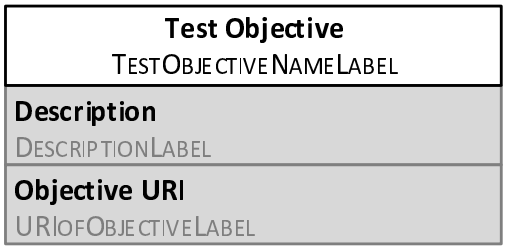
Comments

The lower compartment is optional, it shall be shown if the value of the *Annotation* is given.

An *Annotation* shape shall be attached to the annotated element by a thin dashed line.

6.1.7 TestObjective

Concrete Graphical Notation



Formal Description

context TestObjective
TESTOBJECTIVENAMELABEL ::= self.name
DESCRIPTIONLABEL ::= self.description
URIOFOBJECTIVELABEL ::= self.objectiveURI->**newline()**

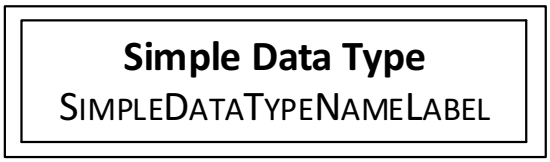
Comments

The compartments containing **Description** and **ObjectiveURI** are optional (that is any of them or both can be omitted). If an optional compartment is present, the contained terminal symbol (**Description** or **ObjectiveURI**, respectively) is mandatory, but the result of the production rule of the non-terminals (DESCRIPTIONLABEL or URIOFOBJECTIVELABEL), respectively) is optional.

6.2 Data

6.2.1 SimpleDataType

Concrete Graphical Notation



Formal Description

context SimpleDataType

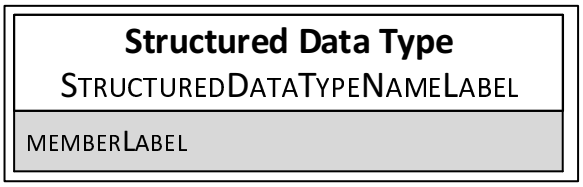
SIMPLEDATATYPENAMELABEL ::= self.name

Comments

No comments.

6.2.2 StructuredDataType

Concrete Graphical Notation



Formal Description

context StructuredDataType

STRUCTUREDDATATYPENAMELABEL ::= self.name

MEMBERLABEL ::= **foreach** m: Member **in** self.member **newline()**

if m.isOptional **then** '['m **as context in** <PARAMETERLABEL>']'

else

m **as context in** <PARAMETERLABEL>

endif

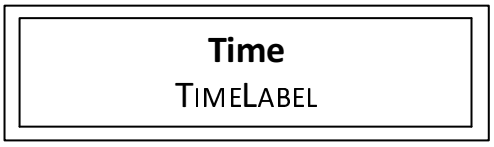
end

Comments

The compartment containing MEMBERLABEL is optional, it shall be shown if the *StructuredDataType* has at least one member.

6.2.3 Time

Concrete Graphical Notation



Formal Description

context Time

TIMELABEL ::= self.name

Comments

No comments.

6.2.4 DataInstance

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

Formal Description

context DataInstance

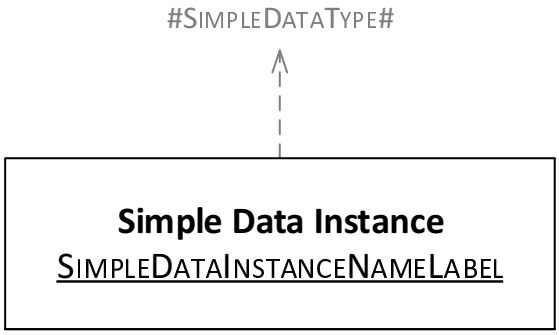
DATAINSTANCELABEL ::= self.name ':' self.dataType.name

Comments

No comments.

6.2.5 SimpleDataInstance

Concrete Graphical Notation



Formal Description

context SimpleDataInstance
SIMPLEDATAINSTANCENAMELABEL **::=** self **as context in** <DATAINSTANCELABEL>

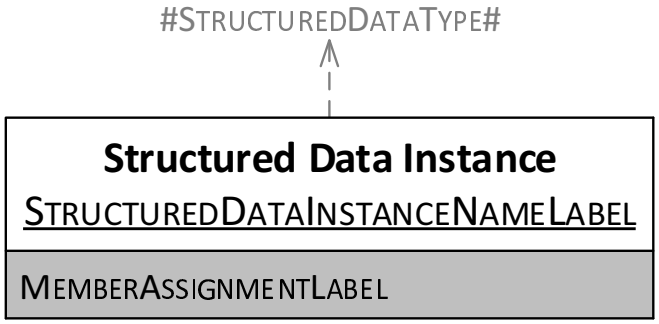
Comments

The result of the production rule of SIMPLEDATAINSTANCENAMELABEL shall be typed by underline font.

A *SimpleDataInstance* shape can optionally be connected to a *SimpleDataType* shape by dashed arrow. If this connection is present, then the ':' and the self.dataType.name can be omitted in the SIMPLEDATAINSTANCENAMELABEL.

6.2.6 StructuredDataInstance

Concrete Graphical Notation



Formal Description

context StructuredDataInstance
STRUCTUREDDATAINSTANCENAMELABEL **::=** self **as context in** <DATAINSTANCELABEL>
MEMBERASSIGNMENTLABEL ::= **foreach** m : MemberAssignment **in** self.memberAssignment **newline()**
if not self.member.name.**oclIsUndefined() then**
[self.member.name ':=']
else
''
endif
self.memberSpec **as context in** <STATICDATAUSELABEL>
end

Comments

The result of the production rule of STRUCTUREDDATAINSTANCENAMELABEL shall be typed by underline font.

The lower compartment containing MEMBERASSIGNMENTLABEL is optional. For each member the name is optional.

The *StructuredDataInstance* shape can optionally be connected to a *StructuredType* shape by dashed arrow. If this connection is present, then the ':' and the self.dataType.name can be omitted in the STRUCTUREDDATAINSTANCENAMELABEL.

6.2.7 Parameter

Concrete Graphical Notation

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

Formal Description

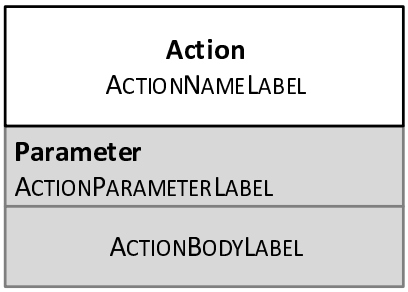
context Parameter
PARAMETERLABEL ::= self.name ':' dataType.name

Comments

No comments.

6.2.8 Action

Concrete Graphical Notation



Formal Description

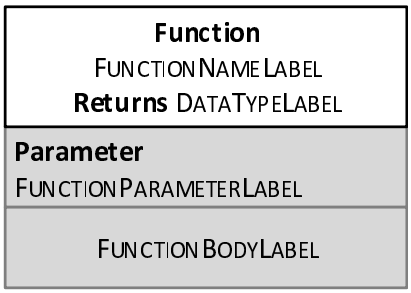
context Action
ACTIONNAMELABEL ::= self.name
ACTIONPARAMETERLABEL ::= **foreach** p:Parameter **in** self.formalParameter **separator**(',')
p **as context in** <PARAMETERLABEL>
end
ACTIONBODYLABEL ::= self.body

Comments

The compartments containing **Parameter** and ACTIONBODYLABEL are optional (that is any of them or both can be omitted). If an optional compartment is present, its content shall also be present.

6.2.9 Function

Concrete Graphical Notation



Formal Description

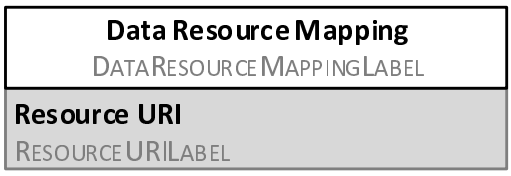
```
context Function
FUNCTIONNAMELABEL ::= self.name
DATATYPELABEL ::= self.returnType.name
FUNCTIONPARAMETERLABEL ::= foreach p:Parameter in self.formalParameter separator(',')
                                p as contex in <PARAMETERLABEL>
                           end
FUNCTIONBODYLABEL ::= self.body
```

Comments

The compartments containing **Parameter** and FUNCTIONBODYLABEL are optional (that is any of them or both can be omitted). If an optional compartment is present, its content shall also be present.

6.2.10 DataResourceMapping

Concrete Graphical Notation



Formal Description

```
context DataResourceMapping
DATARESOURCEMAPPINGLABEL ::= self.name
RESOURCEURILABEL ::= self.resourceURI
```

Comments

The DATARESOURCEMAPPINGLABEL is optional.

The compartment containing the **Resource URI** is optional. This compartment shall be shown when the optional RESOURCEURILABEL is present.

6.2.11 ParameterMapping

Concrete Graphical Notation

This metaclass has no dedicated shape, it is used solely in the *DataElementMapping* shape.

Formal Description

```
context ParameterMapping
```

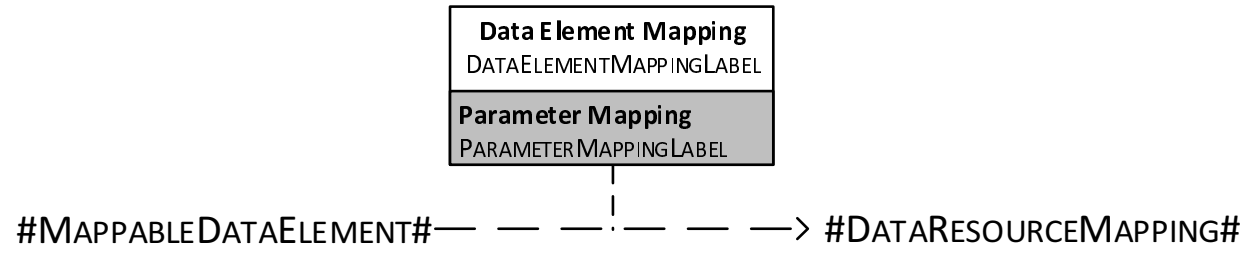
PARAMETERURLLABEL ::= self.parameter.name [':=' self.memberURI]

Comments

No comments.

6.2.12 DataElementMapping

Concrete Graphical Notation



Formal Description

```
context DataElementMapping
DATAELEMENTMAPPINGLABEL ::= self.name [':=' self.elementURI]
PARAMETERMAPPINGLABEL ::= foreach p:ParameterMapping in self.parameterMapping newline()
                              p as context in <PATAMETERURLLABEL>
                          end
```

Comments

In the DATAELEMENTMAPPINGLABEL the elementURI is optional.

The lower compartment containing Parameter Mapping is optional.

6.2.13 DataUse

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

Formal Description

```
context DataUse
DATAUSELABEL ::= if self.oclIsKindOf(StaticDataUse) then
                    self as context in <STATICDATAUSELABEL>
                 else if self.oclIsKindOf(DynamicDataUse) then
                    self as context in <DYNAMICDATAUSELABEL>
                 endif

DATAUSEARGUMENTLABEL ::= if not self.argument->isEmpty() then
                             self as context in <ARGUMENTLABEL>
                         else
                             ' '
                         endif

ARGUMENTLABEL ::= '('foreach p:ParameterUse in self.argument separator(',')
                      p.parameter.name' := ' p.dataUse as context in <DATAUSELABEL>
                  end')'
```

Comments

No comments.

6.2.14 StaticDataUse

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

Formal Description

```
context StaticDataUse
STATICDATAUSELABEL ::= if self.oclIsKindOf(DataInstanceUse) then
        self as context in <DATAINSTANCEUSELABEL>
    else if self.oclIsKindOf(AnyValue) then
        self as context in <ANYVALUELABEL>
    else if self.oclIsKindOf(AnyValueOrOmit) then
        self as context in <ANYVALUEORMITLABEL>
    else if self.oclIsKindOf(OmitValue) then
        self as context in <OMITVALUELABEL>
    endif
```

Comments

No comments.

6.2.15 DataInstanceUse

Concrete Graphical Notation

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

Formal Description

```
context DataInstanceUse
DATAINSTANCEUSELABEL ::= self.name self as context in <ARGUMENTLABEL>
                         if not self.reduction->isEmpty() then
                             'locate' self as context in <REDUCTIONLABEL>
                         else
                             ''
                         endif
```

Comments

No comments.

6.2.16 AnyValue

Concrete Graphical Notation

?

Formal Description

```
context AnyValue

ANYVALUELABEL ::= '?'
```

Comments

No comments.

6.2.17 AnyValueOrOmit

Concrete Graphical Notation

*

Formal Description

context AnyValueOrOmit
ANYVALUEOROMITLABEL ::= '*'

Comments

No comments.

6.2.18 OmitValue

Concrete Graphical Notation

omit

Formal Description

context OmitValue

OMITVALUELABEL ::= 'omit'

Comments

No comments.

6.2.19 DynamicDataUse

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

Formal Description

context DynamicDataUse
DYNAMICDATAUSELABEL ::= **if** self.**oclIsTypeOf**(VariableUse) **then**
self **as context in** <VARIABLEUSELABEL>
else if self.**oclIsTypeOf**(FormalParameterUse) **then**
self **as context in** <FORMALPARAMETERUSELABEL>
else if self.**oclIsTypeOf**(FunctionCall) **then**
self **as context in** <FUNCTIONCALLLABEL>
else if self.**oclIsTypeOf**(TimeLabel) **then**
self **as context in** <TIMELABEL>
endif

Comments

No comments.

6.2.20 FunctionCall

Concrete Graphical Notation

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

Formal Description

context FunctionCall

FUNCTIONCALLLABEL ::= self **as context in** <DataUseARGUMENTLABEL>
if not self.reduction->**isEmpty() then**
'returns' self **as context in** <REDUCTIONLABEL>
else
''
endif

Comments

No comments.

6.2.21 FormalParameterUse

Concrete Graphical Notation

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

Formal Description

context FormalParameterUse

FORMALPARAMETERUSELABEL ::= self.name self **as context in** <DATAUSEARGUMENTLABEL> self **as context in** <REDUCTIONLABEL>

Comments

No comments.

6.2.22 VariableUse

Concrete Graphical Notation

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

Formal Description

context VariableUse

VARIABLEUSELABEL ::= self.componentInstance.name'.'variable.name self **as context in** <DATAUSEARGUMENTLABEL>
if not self.reduction->**isEmpty() then**
'locate' self **as context in** <REDUCTIONLABEL>
else
''
endif

Comments

No comments.

6.3 Time

6.3.1 TimeLabel

Concrete Graphical Notation

#ATOMICBEHAVIOUR# — — @TIMELABELLABEL

Formal Description

context TimeLabel
TIMELABELLABEL ::= self.name

Comments

A *TimeLabel* shape shall be attached to the labelled *AtomicBehaviour* by a thin dashed line.

6.3.2 TimeLabelUse

Concrete Graphical Notation

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

Formal Description

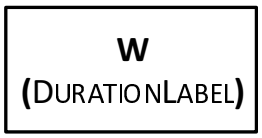
context TimeLabelUse
TIMELABELUSELABEL ::= self.timeLabel.name

Comments

No comments.

6.3.3 Wait

Concrete Graphical Notation



Formal Description

context Wait

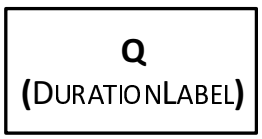
DURATIONLABEL ::= self.period **as context in** <DATAUSELABEL>

Comments

The *Wait* shape shall cover all the lifelines of that component instance, which is referred to by self.componentInstance.

6.3.4 Quiescence

Concrete Graphical Notation



Formal Description

context Quiescence

DURATIONLABEL ::= self.period **as context in** <DATAUSELABEL>

Comments

If the *Quiescence* refers to a component instance (self.componentInstance is set), then the *Quiescence* shape shall cover all the lifelines of that component instance, otherwise the *Quiescence* shape shall cover only the lifeline of that gate, which is referred to by self.gateReference.

6.3.5 TimeConstraint

Concrete Graphical Notation

#ATOMICBEHAVIOUR# — —{ TIMECONSTRAINTLABEL }

Formal Description

context TimeConstraint

TIMECONSTRAINTLABEL **::=** self.timeConstraintExpression **as context in** <DATAUSELABEL>

Comments

A *TimeConstraint* shape shall be attached to an *AtomicBehaviour* shape by a thin dashed line.

6.3.6 TimerStart

Concrete Graphical Notation

Formal Description

context TimerStart
TIMERSTARTLABEL **::=** self.timer.name
DURATIONLABEL **::=** self.period **as context in** <DATAUSELABEL>

Comments

The *TimerStart* shape shall cover all the lifelines of that component instance, which is referred to by self.componentInstance.

6.3.7 TimeOut

Concrete Graphical Notation

Formal Description

context TimeOut
TIMEOUTLABEL **::=** self.timer.name

Comments

The *TimeOut* shape shall cover all the lifelines of that component instance, which is referred to by self.componentInstance.

6.3.8 TimerStop

Meta-Model Reference

Concrete Graphical Notation

Formal Description

context TimerStop
TIMERSTOPLABEL **::=** self.timer.name

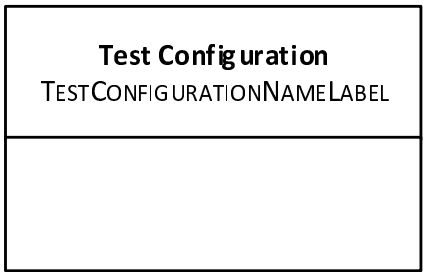
Comments

The *TimerStop* shape shall cover all the lifelines of that component instance, which is referred to by self.componentInstance.

6.4 Test Configuration

6.4.1 TestConfiguration

Concrete Graphical Notation



Formal Description

context TestConfiguration
TESTCONFIGURATIONNAMELABEL **::=** self.name

Comments

Into the lower empty compartment the elements of the *TestConfiguration* shall be placed.

6.4.2 GateType

Concrete Graphical Notation

Formal Description

context GateType
GATETYPENAMELABEL **::=** self.name
INTERACTIONLISTLABEL **::=** self.dataType.name->**separator**(',')

Comments

No comments.

6.4.3 GateInstance

Concrete Graphical Notation

Formal Description

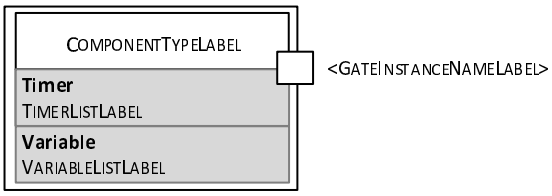
```
context GateInstance
GATEINSTANCENAMELABEL ::= self.name [':' self.type.name]
```

Comments

In GATEINSTANCENAMELABEL the ':' self.type.name is optional.

6.4.4 ComponentType

Concrete Graphical Notation



Formal Description

```
context ComponentType
COMPONENTTYPELABEL ::= self.name
TIMERLISTLABEL ::= self.timer.name->separator(',')
VARIABLELISTLABEL ::= foreach v:Variable in self.variable separator(',')
                          self.variable.name ':' self.variable.dataType.name
                      end
```

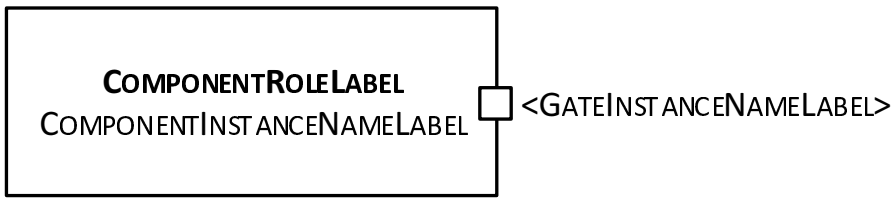
Comments

A *ComponentType* shape may contain one or more *GateInstance* shapes at any side or corner.

The compartments containing **Timer** and **Variable** are optional (that is any of them or both can be omitted). If an optional compartment is present, its content shall also be present.

6.4.5 ComponentInstance

Concrete Graphical Notation



Formal Description

```
context ComponentInstance
COMPONENTROLELABEL ::= if self.role = ComponentInstanceRole::SUT then 'SUT' else 'TESTER' endif
COMPONENTINSTANCENAMELABEL ::= self.name':'self.type.name
```

Comments

A *ComponentInstance* shape may contain one or more *GateInstance* shapes at any side or corner.

The terminal symbol '**SUT**' or '**TESTER**' shall be typed in bold.

6.4.6 Connection

Concrete Graphical Notation

#GATEREFERENCE# —NAMEOFCONNECTIONLABEL— #GATEREFERENCE#

Formal Description

context Connection
NAMEOFCONNECTIONLABEL **::=** self.name

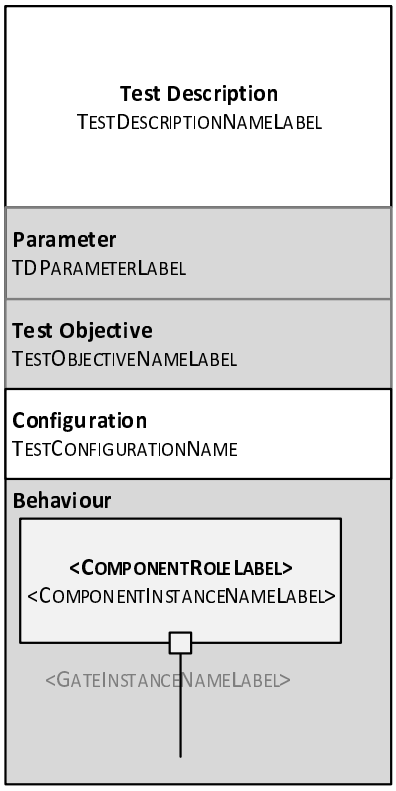
Comments

NAMEOFCONNECTIONLABEL is optional.

6.5 Test Behaviour

6.5.1 TestDescription

Concrete Graphical Notation



Formal Description

context TestDescription
TESTDESCRIPTIONNAMELABEL **::=** self **as context in** <NAMEDELEMENTLABEL>

TDPARAMETERLABEL **::= foreach** p:Parameter **in** self.formalParameter **separator(**',')
p **as context in** <ParameterLabel>
end

TESTOBJECTIVENAMELABEL **::= foreach** t:TestObjective **in** self.testObjectivet **newline()**
t **as context in** <NAMEDELEMENTLABEL>
end

TESTCONFIGURATIONNAME **::=** self.testConfiguration **as context in** <NAMEDELEMENTLABEL>

Comments

The compartments containing **Parameter**, **TestObjective** and **Behaviour** are optional (that is any or all of them can be omitted). If an optional compartment is present, its content shall also be present.

In the lowest compartment describing the behaviour of the test description:

- There shall be as many *ComponentInstance* shapes as there are component instances defined in the *TestConfiguration* referenced in a **Configuration** compartment.
- If a component instance has only one gate, the rectangle representing the *GateInstance* and the *GateInstanceNameLabel* are optional.
- From each gate instance a vertical line ("lifeline") originates, to which each *Behaviour* element defined in that test description and associated with that gate is attached.

6.5.2 Behaviour

Concrete Graphical Notation

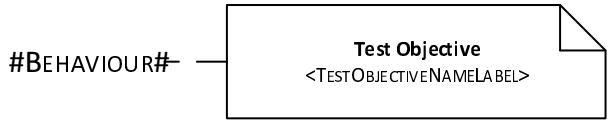
This is an abstract metaclass, therefore no graphical representation is defined.

Formal Description

n.a.

Comments

To a shape of any subclass of *Behaviour*, the following test objective reference shape can be attached by a thin dashed line.



6.5.3 CombinedBehaviour

Concrete Graphical Notation

Formal Description

n.a.

Comments

CombinedBehaviour is an abstract metaclass that can be refined to several subclasses. The figure above gives a general overview, how the combined behaviour elements shall be organized. Further constraints are explained in the respective subclauses describing the symbols of subclasses of *CombinedBehaviour*. Depending on the concrete type of the *CombinedBehaviour*, it may or may not contain more than one blocks. The outermost border of the contained *Block*(s) are not visible. If more than one blocks are defined, they shall be separated by thin dashed lines. Any number of periodic and/or exceptional behaviour can be attached in any order to a *CombinedBehaviour*.

A *CombinedBehaviour* shape shall cover all the lifelines.

6.2.4 Block

Concrete Graphical Notation

Formal Description

context Block

GUARDLABEL ::= self.block.guard **as context in** <DATAUSELABEL>

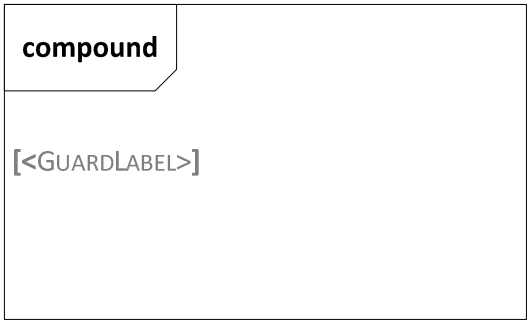
Comments

A *Block* cannot stand on its own, only as a part of a *CombinedBehaviour*. Therefore the border of the *Block* is not visible (the border on the figure above is indicated only for visualization purposes). If a *CombinedBehaviour* contains more than one *Block*, they are separated by dashed lines.

The [GUARDLABEL] is optional. If a *Block* of a *CombinedBehaviour* contains a GUARDLABEL, it shall be placed in between square brackets ('[' and ']').

6.5.5 CompoundBehaviour

Concrete Graphical Notation



Formal Description

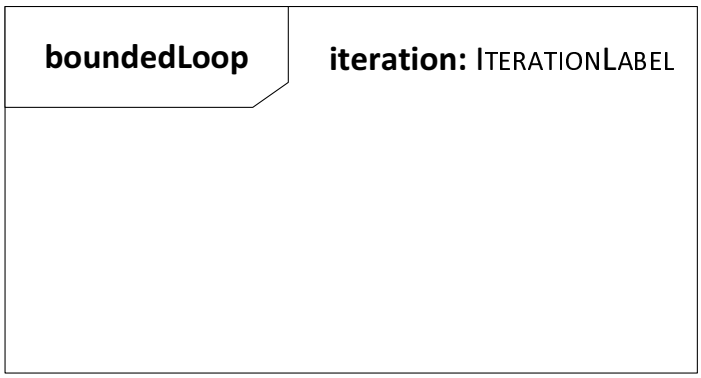
n.a.

Comments

[GUARDLABEL] is optional.

6.5.6 BoundedLoopBehaviour

Concrete Graphical Notation



Formal Description

context BoundedLoopBehaviour
ITERATIONLABEL **::=** self.numIteration **as context in** <DATAUSELABEL>

Comments

No comments.

6.5.7 UnboundedLoopBehaviour

Concrete Graphical Notation

Formal Description

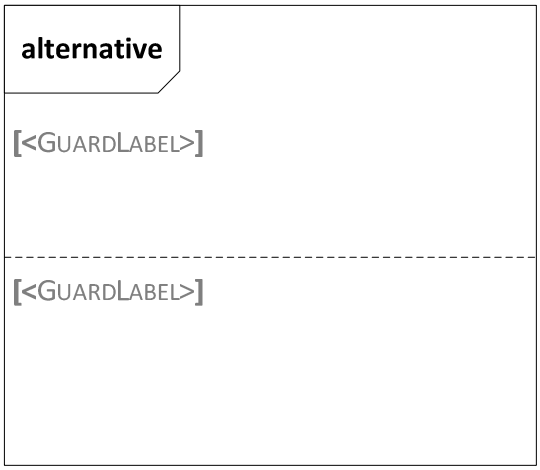
n.a.

Comments

[GUARDLABEL] is optional.

6.5.8 AlternativeBehaviour

Concrete Graphical Notation



Formal Description

n.a.

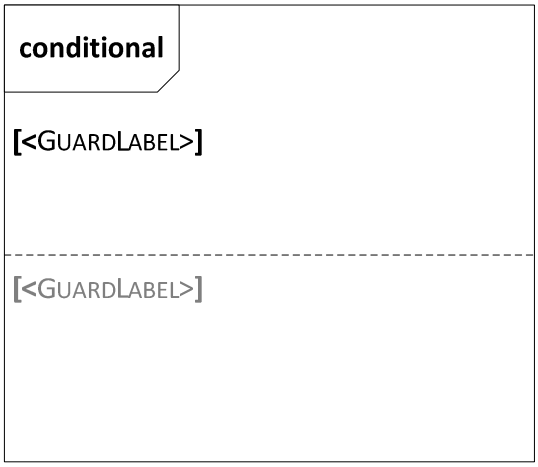
Comments

Any number of blocks can be contained, they are separated by dashed lines.

[GUARDLABEL] of any block is optional.

6.5.9 ConditionalBehaviour

Concrete Graphical Notation



Formal Description

n.a.

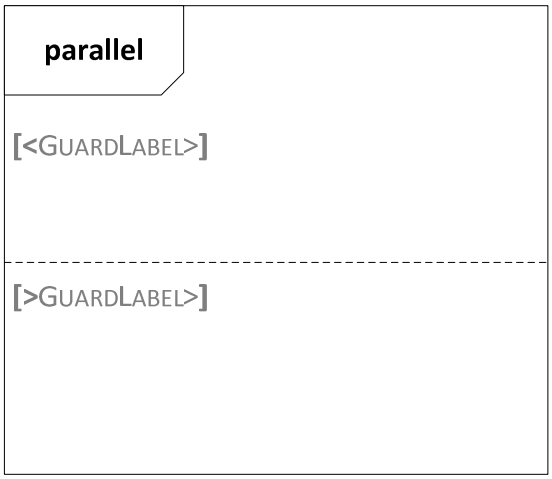
Comments

Any number of blocks can be contained, they are separated by dashed lines.

[GUARDLABEL] of the last block is optional.

6.5.10 ParallelBehaviour

Concrete Graphical Notation



Formal Description

n.a.

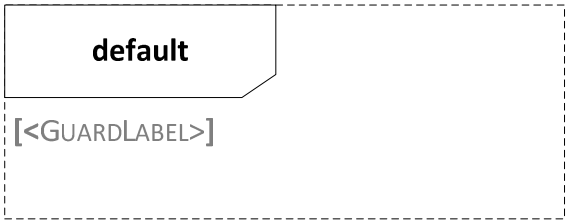
Comments

Any number of blocks can be contained, they are separated by dashed lines.

[GUARDLABEL] of any block is optional.

6.5.11 DefaultBehaviour

Concrete Graphical Notation



Formal Description

n.a.

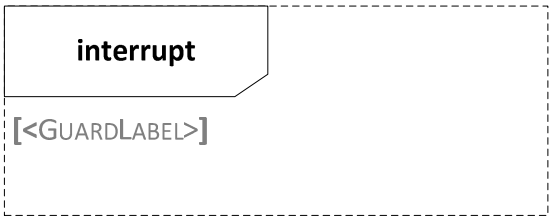
Comments

A *DefaultBehaviour* shape can be attached to any *CombinedBehaviour*.

[GUARDLABEL] is optional.

6.5.12 InterruptBehaviour

Concrete Graphical Notation



Formal Description

n.a.

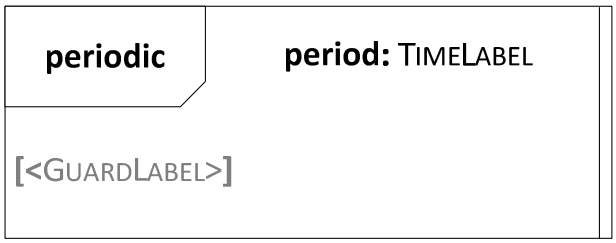
Comments

An *InterruptBehaviour* shape can be attached to any *CombinedBehaviour*.

[GUARDLABEL] is optional.

6.5.13 PeriodicBehaviour

Concrete Graphical Notation



Formal Description

```
context PeriodicBehaviour
TIMELABEL ::= self.period as context in <DATAUSELABEL>
```

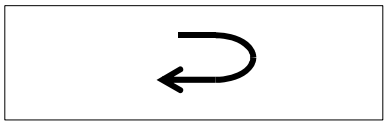
Comments

A *PeriodicBehaviour* shape can be attached to any *CombinedBehaviour*.

[GUARDLABEL] is optional.

6.5.14 Break

Concrete Graphical Notation



Formal Description

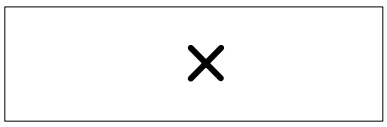
n.a.

Comments

The *Break* shape shall cover all the lifelines.

6.5.15 Stop

Concrete Graphical Notation



Formal Description

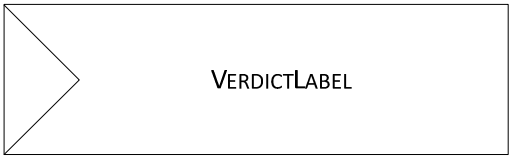
n.a.

Comments

The *Stop* shape shall cover all the lifelines.

6.5.16 VerdictAssignment

Concrete Graphical Notation



Formal Description

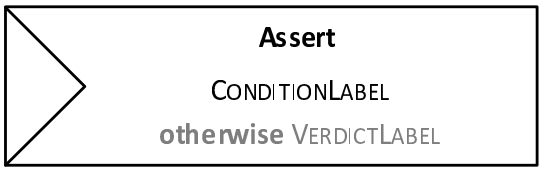
```
context Verdict
VERDICTLABEL ::= self.verdict as context in <DATAUSELABEL>
```

Comments

The *VerdictAssignment* shape shall cover all the lifelines.

6.5.17 Assertion

Concrete Graphical Notation



Formal Description

context Assertion
CONDITIONLABEL **::=** self.condition **as context in** <DATAUSELABEL>
VERDICTLABEL **::=** self.otherwise **as context in** <DATAUSELABEL>

Comments

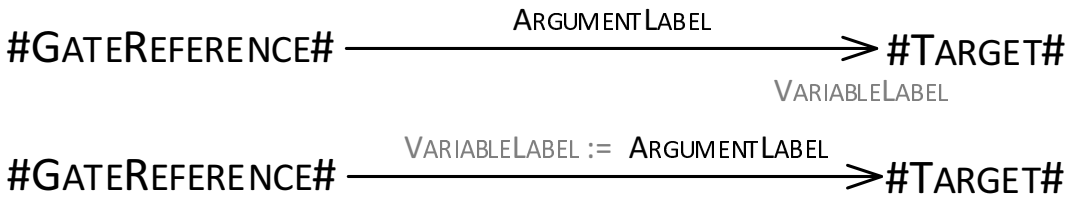
'**otherwise**' and VERDICTLABEL are optional.

The *Assertion* shape shall cover all the lifelines.

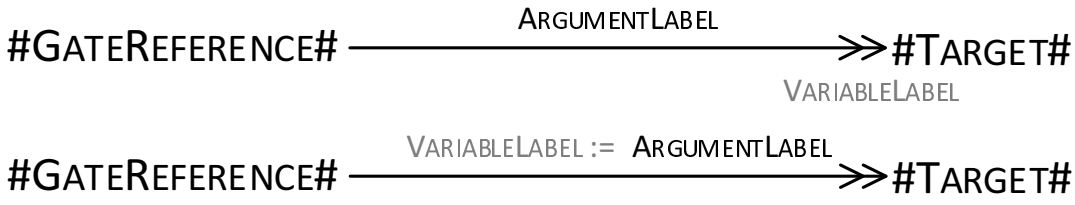
6.5.18 Interaction

Concrete Graphical Notation

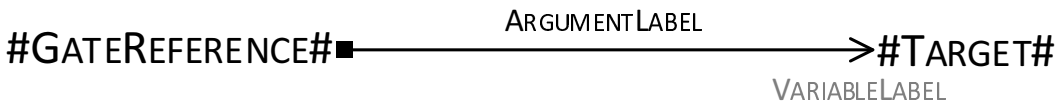
Unicast Interaction



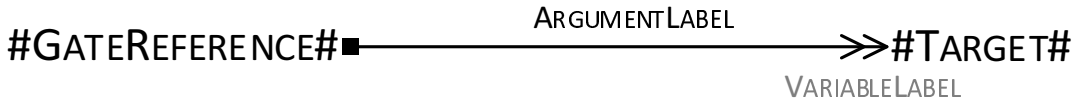
Unicast Trigger Interaction



Multicast Interaction



Multicast Trigger Interaction



Formal Description

```
context Interaction
ARGUMENTLABEL ::= self.argument as context in <DATAUSELABEL>
VARIABLELABEL ::= self.target.variable.name
```

Comments

The two ends of an interaction (GATEREFERENCE and TARGET) shall be placed onto the lifeline of the corresponding gate instances.

VARIABLELABEL and ':= ' are optional.

In case of a unicast or unicast trigger interaction, the VARIABLELABEL - if present - can be placed either above the arrow as an assignment or under the arrowhead.

In case of a multicast or multicast trigger interaction, the originating GATEREFERENCE shall be indicated by a small black square, and there shall be as many arrows present as there are destinations in the multicast interaction. In this case, optionally there may be a VARIABLELABEL presented under each arrowhead.

6.5.19 ActionReference

Concrete Graphical Notation

Formal Description

```
context ActionReference
ACTIONREFNAMELABEL ::= self.action as context in <ACTIONNAMELABEL>
ACTIONARGUMENTLABEL ::= foreach d:DataUse in self.actualParameter separator(',')
                            d as context in <DATAUSELABEL>
                        end
```

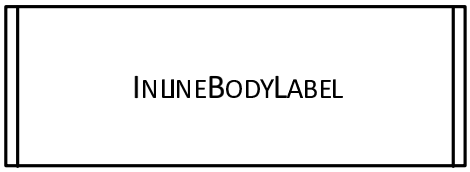
Comments

(ACTIONARGUMENTLABEL) is optional.

In case the *ActionReference* is not related to a *ComponentInstance* (the componentInstance property is not set), the *ActionReference* shape shall cover all the lifelines, otherwise only all the lifelines of the referenced *ComponentInstance*.

6.5.20 InlineAction

Concrete Graphical Notation



Formal Description

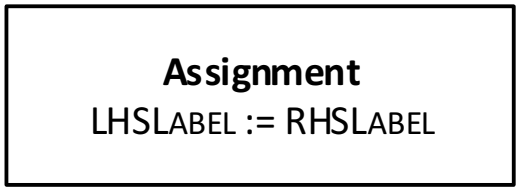
```
context InlineAction
INLINEBODYLABEL ::= self.body
```

Comments

In case the *InlineAction* is not related to a *ComponentInstance* (the componentInstance property is not set), the *InlineAction* shape shall cover all the lifelines, otherwise only all the lifelines of the referenced *ComponentInstance*.

6.5.21 Assignment

Concrete Graphical Notation



Formal Description

context Assignment

LHSLABEL **::=** self.variable **as context in** <VARIABLELABEL>

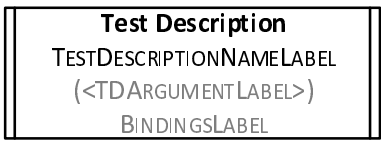
RHSLABEL **::=** self.expression **as context in** <DATAUSELABEL>

Comments

The *Assignment* shape shall cover all the lifelines of that *ComponentInstance* which is referred to by the componentInstance property of the *Assignment*.

6.5.22 TestDescriptionReference

Concrete Graphical Notation



Formal Description

```
context TestDescriptionReference
TESTDESCRIPTIONNAMELABEL ::= self.testDescription.name

TDARGUMENTLABEL ::= foreach d:DataUse in self.actualParameter separator(',')
                                d as context in <DATAUSELABEL>
                    end

BINDINGSLABEL ::= foreach c : ComponentInstanceBinding in self.componentInstanceBinding separator(',')
                            c.componentInstanceBinding.actualComponent.name '->'
                            c.componentInstanceBinding.formalComponent.name
                  end
```

Comments

(ARGUMENTLABEL) and BINDINGSLABEL are optional (that is any of them or both can be omitted).

The *TestDescriptionReference* shape shall cover all the lifelines.

Annex A (informative): Examples

A.0 Overview

This annex provides several examples to illustrate how the different elements of the TDL Graphical Syntax can be used and demonstrates the applicability of TDL in several different areas.

The first example in clause A.1 demonstrates the usage of data-related concepts.

The second example in clause A.2 shows a scenario when a 'Tester' performs a test scenario on one interface of the 'SUT'. The example is taken from ETSI TS 136 523-1 [i.1].

The third example in clause A.3 provides an example for interoperability testing in IMS. The example is taken from ETSI TS 186 011-2 [i.2].

A.1 Illustration of Data use in TDL Graphical Syntax

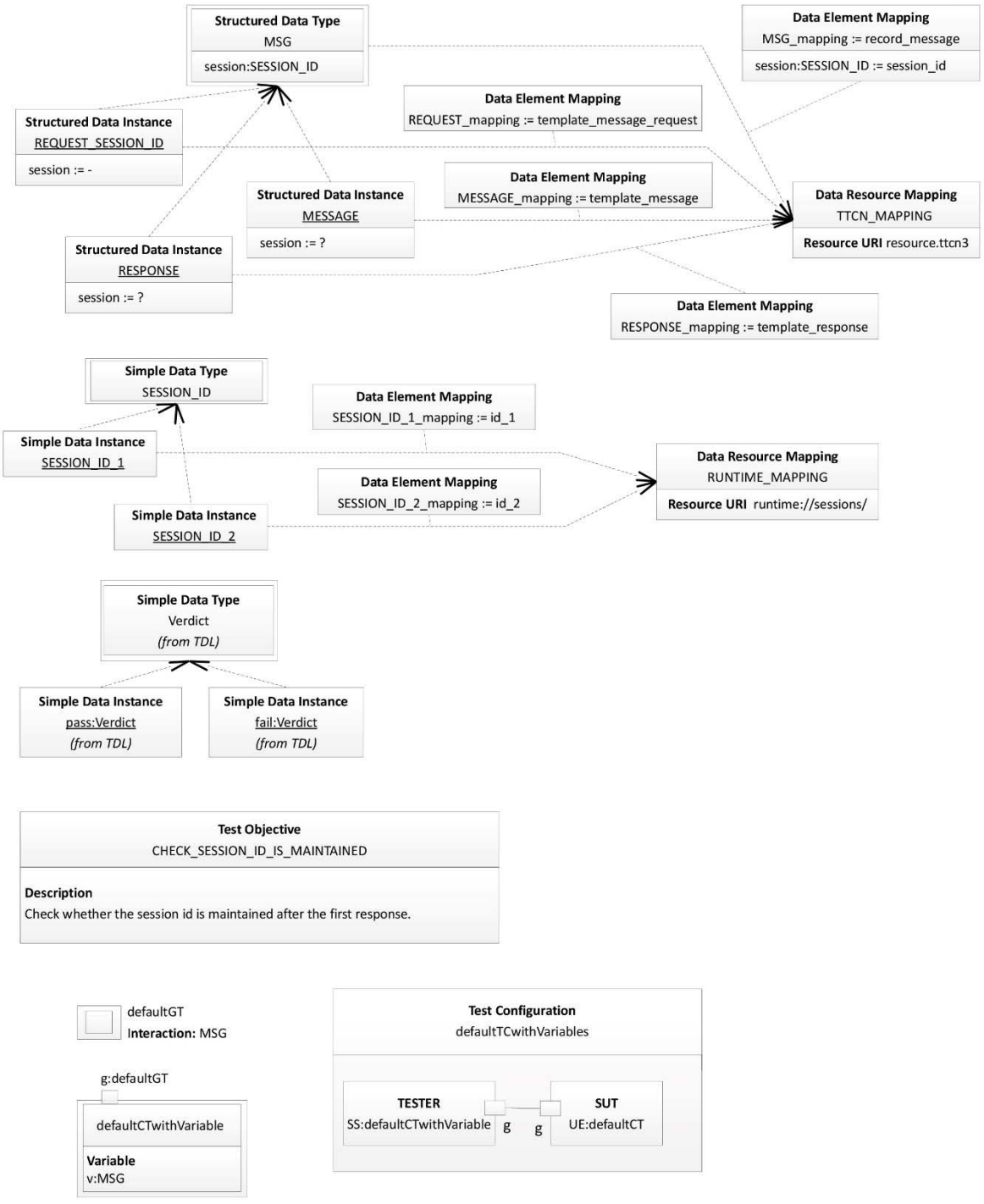


Figure A.1.1: Illustration of Data use in TDL Graphical Syntax Part 1

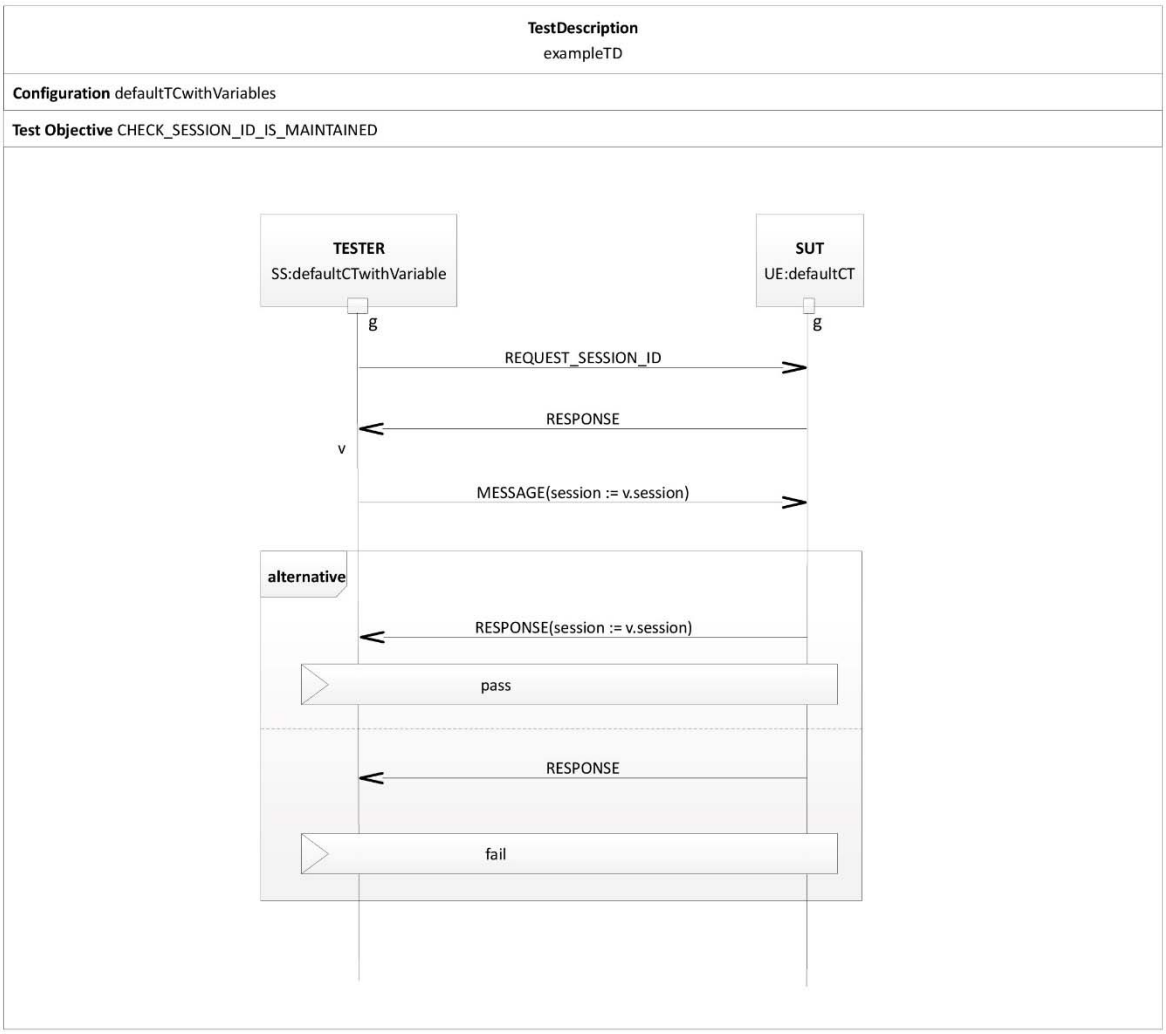


Figure A.1.2: Illustration of Data use in TDL Graphical Syntax Part 2

A.2 Interface Testing

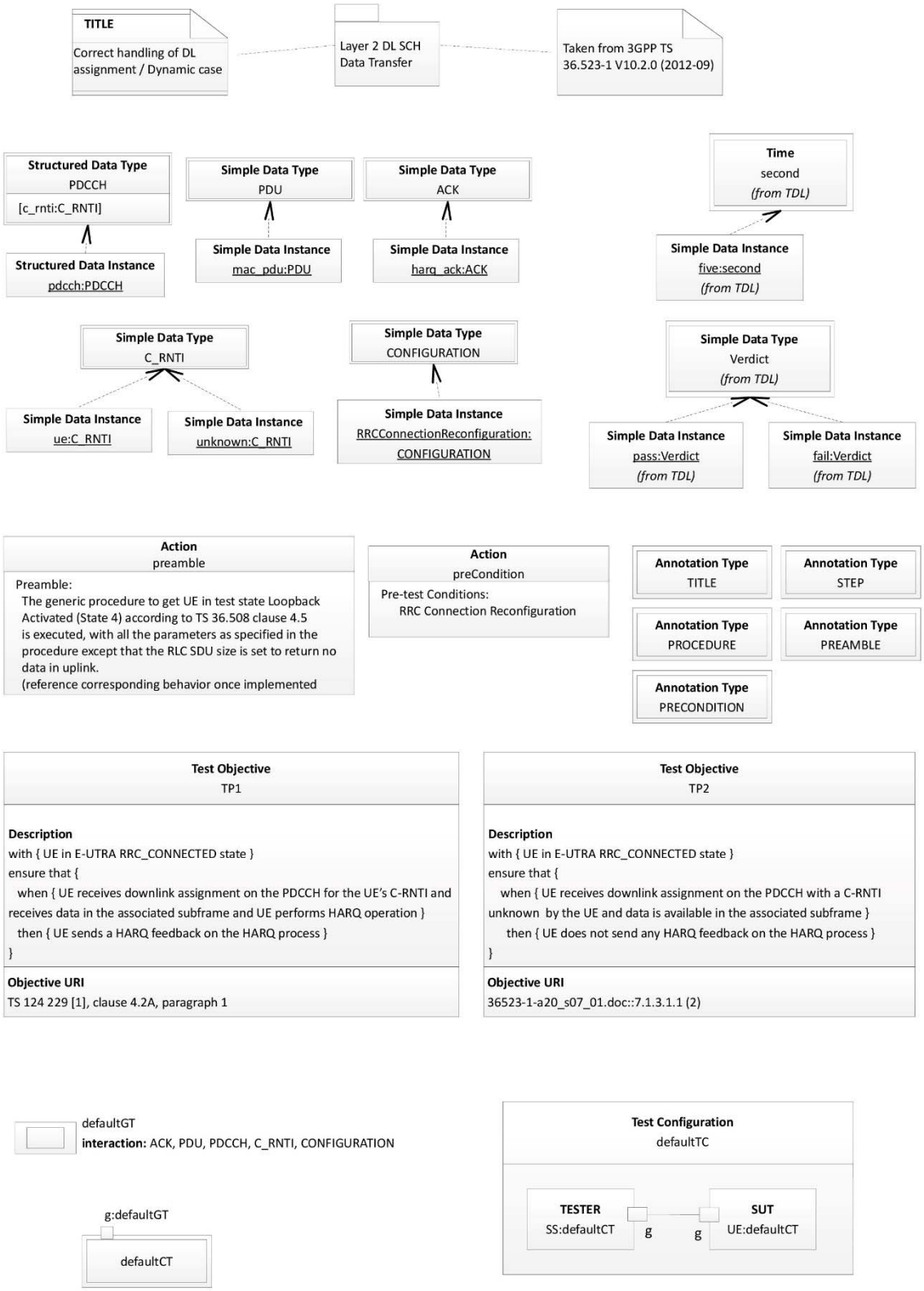


Figure A.2.1: Illustration of an interface testing in TDL Graphical Syntax Part 1

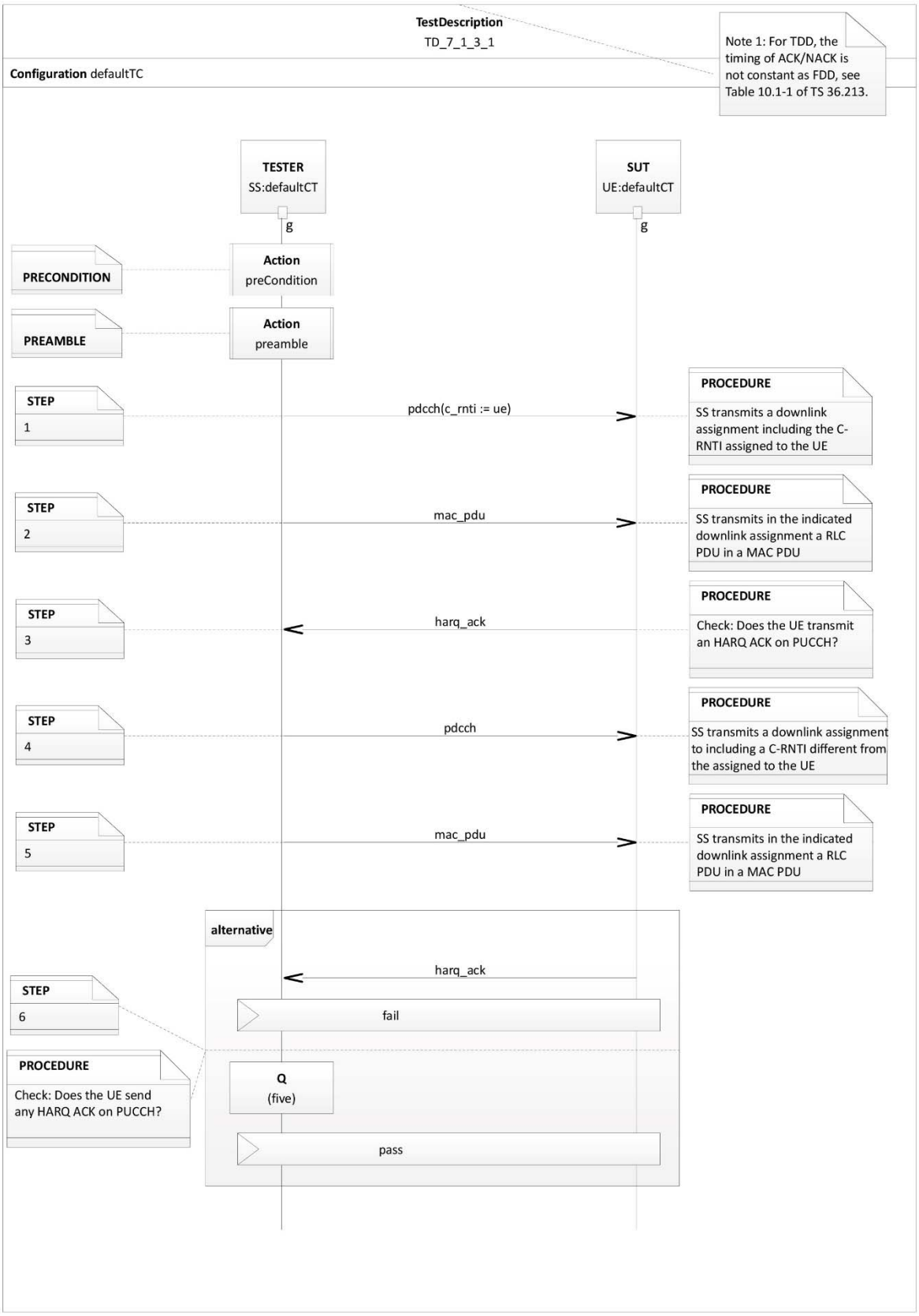


Figure A.2.2: Illustration of an interface testing in TDL Graphical Syntax Part 2

A.3 Interoperability Testing

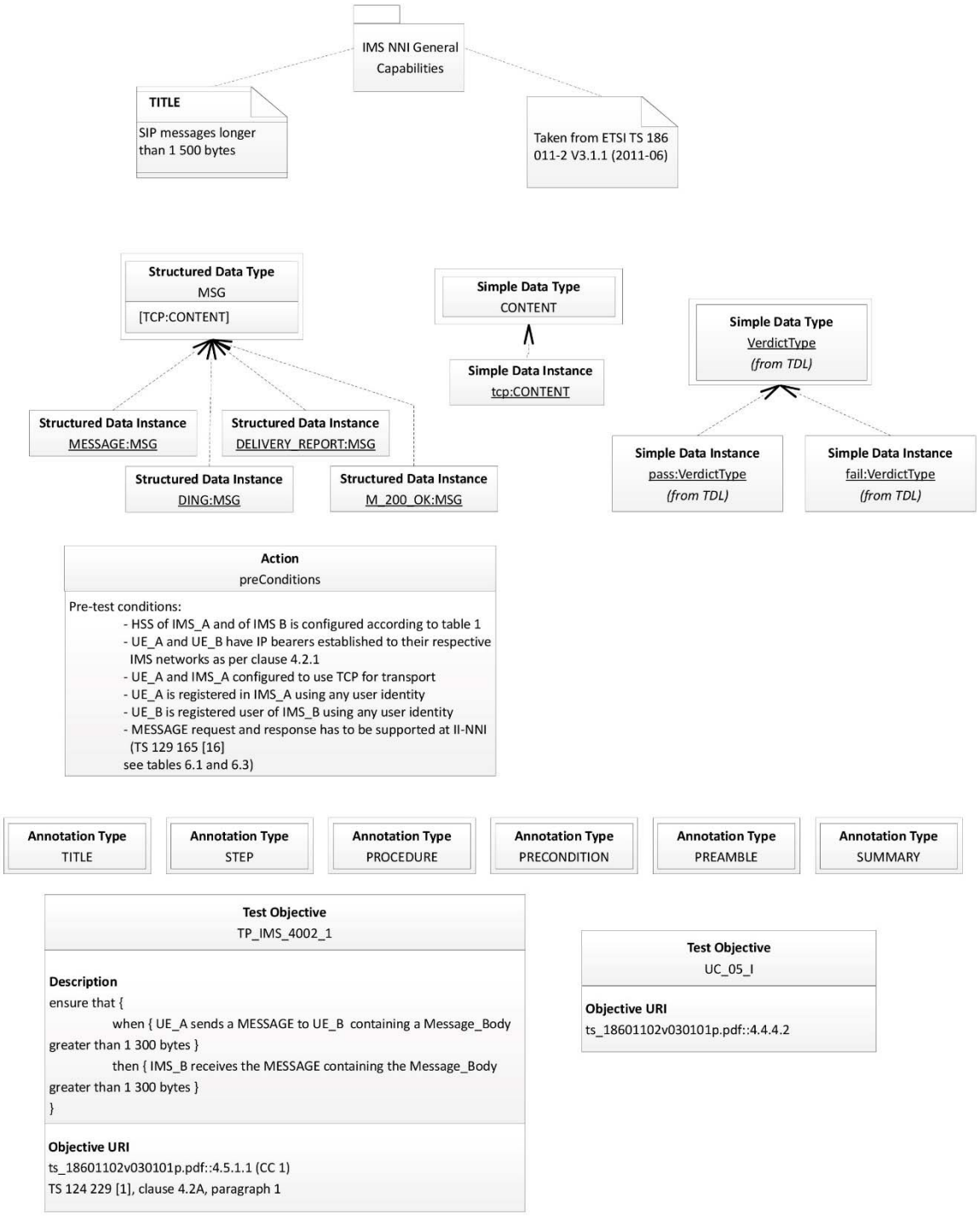


Figure A.3.1: Illustration of an interoperability testing in TDL Graphical Syntax Part 1

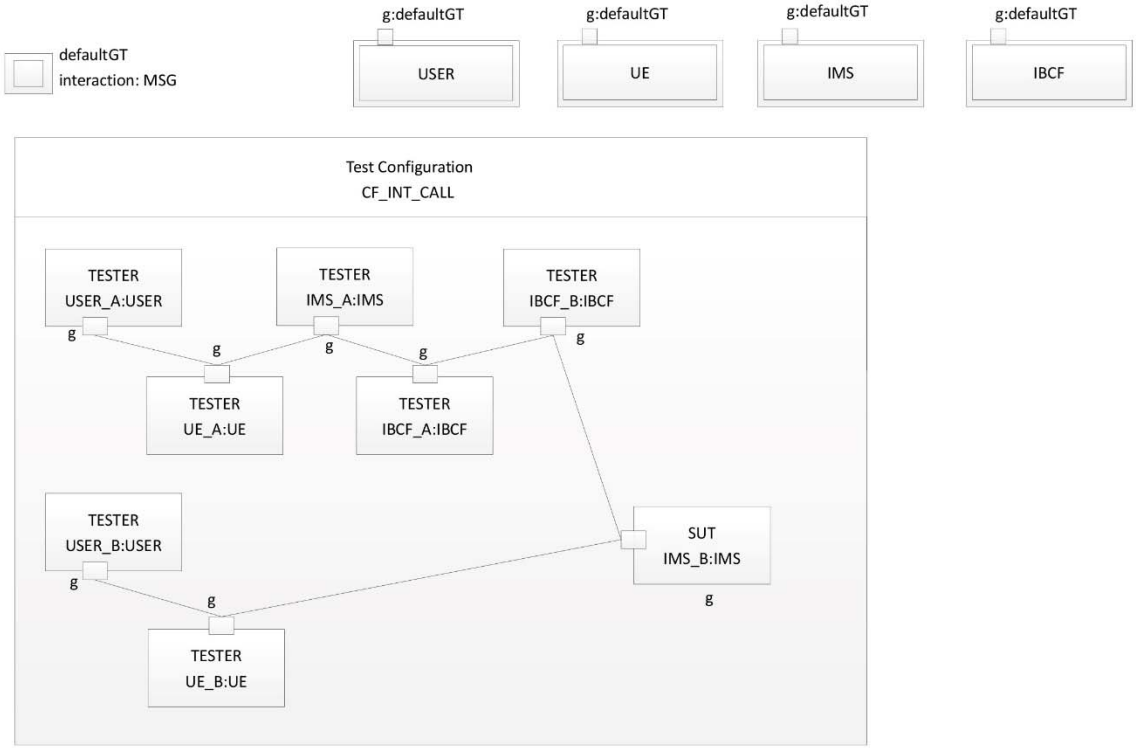


Figure A.3.2: Illustration of an interoperability testing in TDL Graphical Syntax Part 2

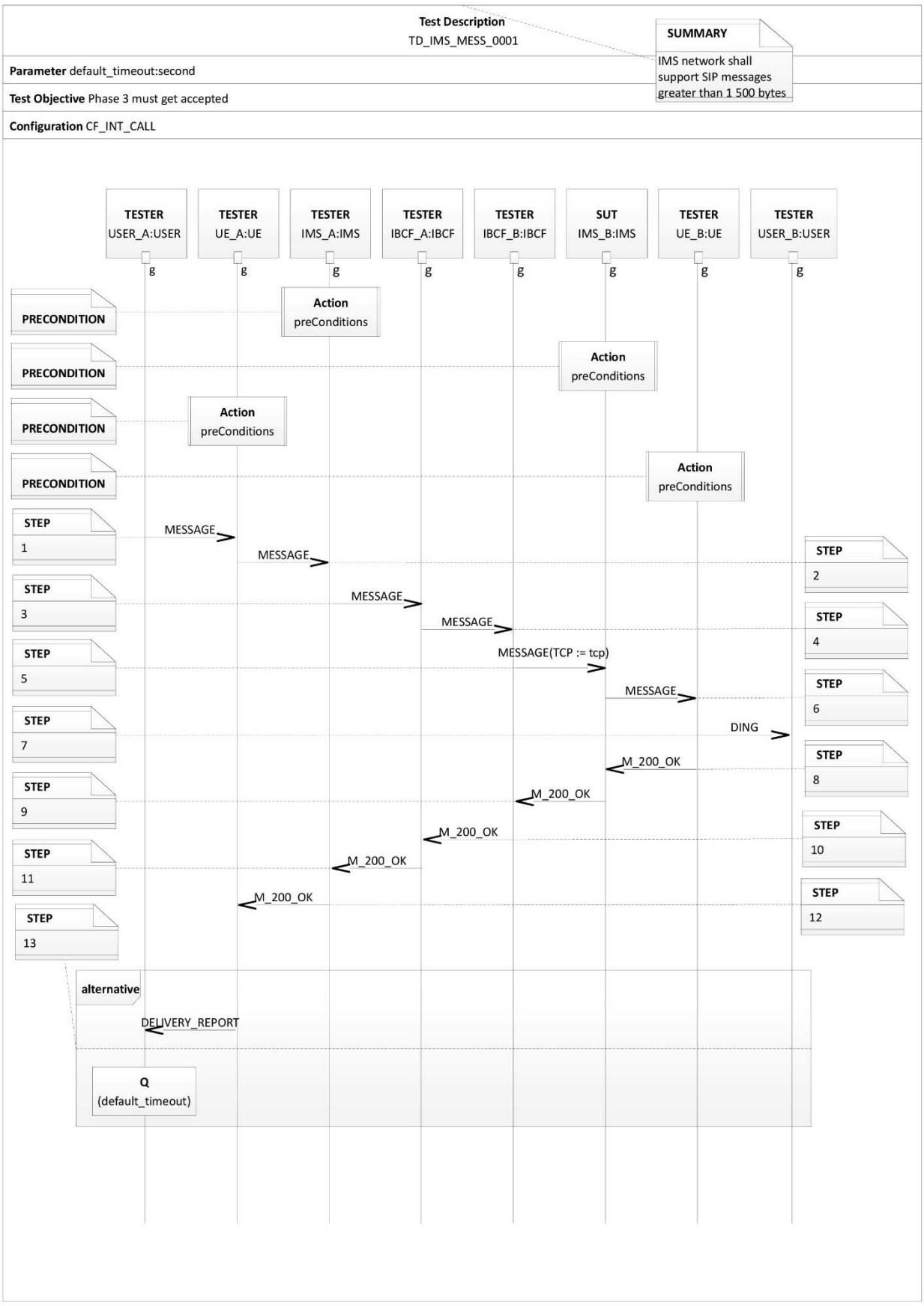


Figure A.3.3: Illustration of an interoperability testing in TDL Graphical Syntax Part 3

History

Document history		
V1.1.0	April 2015	Membership Approval Procedure MV 20150619: 2015-04-20 to 2015-06-19