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

Intellectual Property Rights ........................................................................................................................................... 5
Foreword ............................................................................................................................................................................. 5
Modal verbs terminology .................................................................................................................................................... 5
1 Scope ............................................................................................................................................................................. 6
2 References ..................................................................................................................................................................... 6
2.1 Normative references ................................................................................................................................................ 6
2.2 Informative references ............................................................................................................................................. 6
3 Definitions and abbreviations ....................................................................................................................................... 6
3.1 Definitions ............................................................................................................................................................... 6
3.2 Abbreviations .......................................................................................................................................................... 7
4 Basic principles ............................................................................................................................................................. 7
4.1 Introduction .............................................................................................................................................................. 7
4.2 Document Structure ................................................................................................................................................ 7
4.3 Notational Conventions ........................................................................................................................................ 8
4.3.0 General .............................................................................................................................................................. 8
4.3.1 Symbols and meanings for shapes ....................................................................................................................... 8
4.3.2 Symbols for non-terminal textual labels ........................................................................................................... 8
4.3.3 Examples .......................................................................................................................................................... 9
4.4 Conformance ........................................................................................................................................................... 10
5 Diagram ......................................................................................................................................................................... 11
6 Shapes ........................................................................................................................................................................... 11
6.1 Foundation .............................................................................................................................................................. 11
6.1.1 Element ............................................................................................................................................................. 11
6.1.2 NamedElement ................................................................................................................................................ 11
6.1.3 ElementImport ................................................................................................................................................. 12
6.1.4 Package ............................................................................................................................................................ 12
6.1.5 Comment .......................................................................................................................................................... 13
6.1.6 AnnotationType ............................................................................................................................................... 13
6.1.7 Annotation ....................................................................................................................................................... 14
6.1.8 TestObjective ................................................................................................................................................ 14
6.2 Data .......................................................................................................................................................................... 15
6.2.1 SimpleDataType ............................................................................................................................................... 15
6.2.2 StructuredDataType ....................................................................................................................................... 15
6.2.3 CollectionDataType ....................................................................................................................................... 16
6.2.4 ProcedureSignature ....................................................................................................................................... 16
6.2.5 Time ................................................................................................................................................................. 17
6.2.6 DataInstance .................................................................................................................................................... 17
6.2.7 SimpleDataInstance ..................................................................................................................................... 17
6.2.8 StructuredDataInstance ................................................................................................................................. 18
6.2.9 CollectionDataInstance ..................................................................................................................................... 19
6.2.10 Parameter ....................................................................................................................................................... 19
6.2.11 Action ............................................................................................................................................................. 20
6.2.12 Function ........................................................................................................................................................ 20
6.2.13 DataResourceMapping ................................................................................................................................. 21
6.2.14 ParameterMapping ....................................................................................................................................... 21
6.2.15 DataElementMapping .................................................................................................................................. 22
6.2.16 DataUse ........................................................................................................................................................ 22
6.2.17 StaticDataUse ............................................................................................................................................... 23
6.2.18 DataInstanceUse ........................................................................................................................................... 23
6.2.19 AnyValue ....................................................................................................................................................... 24
6.2.20 AnyValueOrOmit ........................................................................................................................................... 24
6.2.21 OmitValue ...................................................................................................................................................... 25
6.2.22 DynamicDataUse ........................................................................................................................................... 25
Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables 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 (https://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.

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.

Foreword

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

The present document is part 2 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 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 ETSI ES 203 119-1 [1].

NOTE: OMG®, UML®, OCL™ and UTP™ are the trademarks of OMG (Object Management Group). This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the products named.

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.


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

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**: vertical line originates from a gate instance or a component instance, to which behavioural elements may 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:

- EBNF: Extended Backus-Naur Form
- IMS: IP Multimedia Subsystem
- 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 may be placed. A graphical representation may 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. \textit{StructuredDataType}.

The definition of the TDL concrete graphical syntax consists of both shapes and textual labels placed into these shapes. Textual labels are differentiation 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. \textit{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. \texttt{UNDERLINED} or \texttt{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. \texttt{SIMPLEDATATYPENAMELABEL} (non-terminal) and \texttt{MyValue:MyType} (a result of the production rule of that non-terminal) or \texttt{COMPONENTROLELABEL} (non-terminal) and \texttt{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 \texttt{bold}, bold font shall be used in the shape for that terminal textual symbol, e.g. \texttt{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 shall 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. \texttt{<REFERENCEDPRODUCTIONRULE>}).

- A non-terminal textual label in between hashmarks (e.g. \texttt{#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 (Extended Backus-Naur Form - EBNF).

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 rule 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 needs 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 `variable` 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)` is 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()` may also be applied directly to the `foreach` construct.
- For the `PredefinedFunction` instances whose name starts and ends by a character '_' (actually they are infix operators) the (non-OCL) helper function `getOperatorSymbol()` is used to retrieve the operator symbol from the name. `getOperatorSymbol()` returns by the name of the `PredefinedFunction` instance without the character '_' at the beginning and at the end.

### 4.3.3 Examples

#### Test Objective

<table>
<thead>
<tr>
<th>Description</th>
<th>Objective URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>URI</td>
</tr>
</tbody>
</table>

**context TestObjective**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>OBJECTIVE</th>
<th>LABEL</th>
<th>LABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.name</td>
<td>self.description</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 `self.objectiveURI` in production rule `URIOFOBJECTIVELABEL` is a new line.

```
context TestDescriptionReference

TDARGUMENTLABEL := self.testDescription.name

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

END

BOUNDINGSLABEL ::= 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 may be multiple instances of a TDL Diagram at the same time.

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

6 Shapes

6.1 Foundation

6.1.1 Element

Concrete Graphical Notation

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

Formal Description

context Element

ELEMENT_NAMELABEL ::= self.name

Comments

To a shape of any subclass of Element, the name of that Element may be attached by a thin dashed line unless it is stated otherwise in the shape definition of a given subclass of Element.

6.1.2 NamedElement

Concrete Graphical Notation

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

Formal Description

context NamedElement

QUALIFIED_ELEMENTLABEL ::= self.qualifiedName

Comments

To a shape of any subclass of NamedElement, the qualified name of that NamedElement may be attached by a thin dashed line, except for those subclasses where it is specified otherwise.
6.1.3 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.

6.1.4 Package

Concrete Graphical Notation
### Formal Description

**context** Package

\[
\text{PNLABEL} ::= \text{self.name}
\]

**IMPORTED ELEMENTS**

\[
\text{IMPORTED ELEMENTS} ::= \text{foreach :} \text{ElementImport in self.import as context in} \langle \text{IMPORT LABEL} \rangle \text{separator(,)}
\]

**end**

**Comments**

The figures above indicate the two possible representations of the Package shape: the PNLABEL may 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.

#### 6.1.5 Comment

**Concrete Graphical Notation**

![Comment Notation](image)

**Formal Description**

**context** Comment

\[
\text{COMMENTLABEL} ::= \text{self.body}
\]

**Comments**

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

#### 6.1.6 AnnotationType

**Concrete Graphical Notation**

![Annotation Type](image)

**Formal Description**

**context** AnnotationType

\[
\text{ANNOTATION TYPE NAME LABEL} ::= \text{self.name}
\]
Comments
No comments.

6.1.7 Annotation
Concrete Graphical Notation

Formal Description
context Annotation

class KeyLabel := self.key.name

class 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.

6.1.8 TestObjective
Concrete Graphical Notation

Formal Description
context TestObjective

class TestObjectiveNameLabel := self.name

class Description := self.description

URIOfObjectiveLabel := self.objectiveURI->newline()

Comments
The compartments containing Description and ObjectiveURI are optional (that is any of them or both may 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

```
Simple Data Type
SIMPLEDATATYPENAMELABEL
```

Formal Description

```
context SimpleDataType
SIMPLEDATATYPENAMELABEL ::= self.name
```

Comments

No comments.

6.2.2 StructuredDataType

Concrete Graphical Notation

```
Structured Data Type
STRUCTUREDDATATYPENAMELABEL
MEMBERLABEL
```

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 CollectionDataType

Concrete Graphical Notation

<table>
<thead>
<tr>
<th>Collection Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTIONDATATYPENAME LABEL</td>
</tr>
<tr>
<td>ITEMTYPELABEL</td>
</tr>
</tbody>
</table>

Formal Description

context CollectionDataType

COLLECTIONDATATYPENAMELABEL ::= self.name
ITEMTYPELABEL ::= 'of' self.itemType.name

Comments

No comments.

6.2.4 ProcedureSignature

Concrete Graphical Notation

<table>
<thead>
<tr>
<th>Procedure Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCEDURESIGNATURENAME LABEL</td>
</tr>
<tr>
<td>PROCEDUREPARAMETERLABEL</td>
</tr>
</tbody>
</table>

Formal Description

context ProcedureSignature

PROCEDURESIGNATURENAMELABEL ::= self.name
PROCEDUREPARAMETERLABEL ::= foreach p: ProcedureParameter in self.parameter newline() if self.kind = ParameterKind::IN then 'IN'
else if self.kind = ParameterKind::OUT then 'OUT'
else if self.kind = ParameterKind::EXCEPTION then 'EXCEPTION'
endif
self as context in <PARAMETERLABEL>
end

Comments

No comments.
6.2.5 Time

Concrete Graphical Notation

![Diagram of Time]

Formal Description

c\text{\textbf{context Time}}

\text{TIMELABEL} ::= \text{self.name}

Comments
No comments.

6.2.6 DataInstance

Concrete Graphical Notation

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

Formal Description

c\text{\textbf{context DataInstance}}

\text{DATAINSTancelABEL} ::= \text{self.name} ': ' \text{self.dataType.name}

Comments
No comments.

6.2.7 SimpleDataInstance

Concrete Graphical Notation

![Diagram of Simple Data Instance]

Formal Description

c\text{\textbf{context SimpleDataInstance}}

\text{SIMPLEDATAINSTANCELABEL} ::= \text{self ascontext in } \text{<DATAINSTancelABEL>}

\text{#SIMPLEDATATYPE#}
Comments

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

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

6.2.8 StructuredDataInstance

Concrete Graphical Notation

Formal Description

context StructuredDataInstance

StructuredDataInstanceNameLabel ::= self as context in <DataInstanceLabel>

UnassignedMemberLabel ::= if self.unassignedMember = UnassignedMemberTreatment::AnyValue then
"Unassigned Members as ?"
else if self.unassignedMember = UnassignedMemberTreatment::AnyValueOrOmit then
"Unassigned Members as *
endif

MemberAssignmentLabel ::= foreach m : MemberAssignment in self.memberAssignment newline()
if not self.member.nameoclIsUndefined() then
[self.member.name ':=']
else
end
self.memberSpec as context in <StaticDataUseLabel>
end

Comments

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

The UnassignedMemberLabel is optional.

The lower compartment containing MemberAssignmentLabel is optional.

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

Concrete Graphical Notation

```
#COLLECTIONDATATYPE#

Collection Data Instance

COLLECTIONDATAINSTANCE NAME LABEL

ITEM LABEL
```

Formal Description

```
context CollectionDataInstance

COLLECTIONDATAINSTANCE NAME LABEL ::= self as context in <DATAINSTANCELABEL>

ITEM LABEL ::= foreach i : StaticDataUse in self.item newline()

   i as context in <STATICDATASELABEL>
```

Comments

The result of the production rule of COLLECTIONDATAINSTANCE NAME LABEL shall be typed by underline font.

The lower compartment containing ITEM LABEL is optional.

The CollectionDataInstance shape may optionally be connected to a CollectionDataType shape by a dashed arrow. If this connection is present, then the `:` and the self.dataType.name may be omitted in the COLLECTIONDATAINSTANCE NAME LABEL.

6.2.10 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 ':': self.dataType.name
```

Comments

No comments.
6.2.11  Action

Concrete Graphical Notation

<table>
<thead>
<tr>
<th>Action</th>
<th>ACTIONNAMELABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>ACTIONPARAMETERLABEL</td>
</tr>
<tr>
<td>Body</td>
<td>ACTIONBODYLABEL</td>
</tr>
</tbody>
</table>

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 Body are optional (that is any of them or both may be omitted). If an optional compartment is present, its content shall also be present.

6.2.12  Function

Concrete Graphical Notation

<table>
<thead>
<tr>
<th>Function</th>
<th>FUNCTIONNAMELABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns</td>
<td>DATATYPELABEL</td>
</tr>
<tr>
<td>Parameter</td>
<td>FUNCTIONPARAMETERLABEL</td>
</tr>
<tr>
<td>Body</td>
<td>FUNCTIONBODYLABEL</td>
</tr>
</tbody>
</table>

Formal Description

context Function

FUNCTIONNAMELABEL ::= self.name

DATATYPELABEL ::= self.returnType.name

FUNCTIONPARAMETERLABEL ::= foreach p:Parameter in self.formalParameter separator(',')
               p as context in <PARAMETERLABEL>
               end

FUNCTIONBODYLABEL ::= self.body
Comments
The compartments containing Parameter and Body are optional (that is any of them or both may be omitted). If an optional compartment is present, its content shall also be present.

6.2.13 DataResourceMapping

Concrete Graphical Notation

```
Data Resource Mapping
  DATARESOURCEMAPPINGLABEL
  RESOURCEURILABEL

Resource URI
  RESOURCEURILABEL
```

Formal Description

```
context DataResourceMapping

DATARESOURCEMAPPINGLABEL ::= self.name

RESOURCEURILABEL ::= self.resourceURI
```

Comments
The DATARESOURCEMAPPINGLABEL is optional.
The compartment containing the Resource URI is optional. When the optional RESOURCEURILABEL is present, this compartment shall be shown.

6.2.14 ParameterMapping

Concrete Graphical Notation

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

Formal Description

```
context ParameterMapping

PARAMETERURILABEL ::= self.parameter.name ['=' self.memberURI]
```

Comments
No comments.
6.2.15 DataElementMapping

Concrete Graphical Notation

\[
\text{DataElementMapping}\ |
\text{Parameter Mapping}
\]

#MAPPABLE\text{DataElement}\# --- --- --- \rightarrow \#\text{DataResourceMapping}\#

Formal Description

context DataElementMapping

\text{DataElementMappingLabel} ::= \text{self.name} ['::=' \text{self.elementURI}]

\text{ParameterMappingLabel} ::= \text{foreach} p:ParameterMapping in self.parameterMapping newline()

\text{ ParameterLabel} ::= p \text{as context in } <\text{ParameterURILabel}>

Comments

In the \text{DataElementMappingLabel} the elementURI is optional.

The lower compartment containing Parameter Mapping is optional.

6.2.16 DataUse

Concrete Graphical Notation

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

Formal Description

context DataUse

\text{DataUseLabel} ::= \text{if self.oclIsKindOf(StaticDataUse) then}

\text{self as context in } <\text{StaticDataUseLabel}>

\text{else if self.oclIsKindOf(DynamicDataUse) then}

\text{self as context in } <\text{DynamicDataLabel}>

\text{endif}

\text{DataUseArgumentLabel} ::= \text{if not self.argument->isEmpty() then}

\text{self as context in } <\text{ArgumentLabel}>

\text{else}

\text{''}

\text{endif}

\text{ArgumentLabel} ::= '\text{foreach} p:ParameterBinding in self.argument separator(',')

p.parameter.name' ::= \text{''p.dataUse as context in } <\text{DataUseLabel}>

\text{end}''
REDUCTIONLABEL ::= \textbf{foreach} mRef : MemberReference in self.reduction
    if not mRef.member.oclIsUndefined() then
      "." mRef.member.name
    else
      ""
    endif
    if not mRef.collectionIndex.oclIsUndefined() then
      "[" mRef.collectionIndex as context in <DATAUSELABEL> "]"
    else
      ""
    endif

Comments

In ARGUMENTLABEL p.parameter.name' := ' is optional.

6.2.17 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 <DATAINSTANCEDATAUSELABEL>
else if self.oclIsKindOf(AnyValue) then
  self as context in <ANYVALUELABEL>
else if self.oclIsKindOf(AnyValueOrOmit) then
  self as context in <ANYVALUEOROMITLABEL>
else if self.oclIsKindOf(OmitValue) then
  self as context in <OMITVALUELABEL>
endif

Comments

No comments.

6.2.18 DataInstanceUse

Concrete Graphical Notation

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

**context** DataInstanceUse

```
DATA_INSTANCE_USE_LABEL ::= if not self.dataInstance ->isEmpty() then
  self.dataInstance.name
else
  ' '
endif
if not self.dataType ->isEmpty() then
  'new ' self.dataType.name ':'
else
  ' 'endif
if not self.unassignedMember ->isEmpty() then
  '( self as context in <UNASSIGNED_MEMBERLABEL> ')'endif
else
  ' 'endif
self as context in <ARGUMENT_LABEL>
if not self.reduction ->isEmpty() then
  self as context in <REDUCTION_LABEL>
else
  ' 'endif
```

Comments
No comments.

6.2.19 AnyValue

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

Formal Description

```
context AnyValue
ANYVALUE_LABEL ::= '?'
```

Comments
No comments.

6.2.20 AnyValueOrOmit

Concrete Graphical Notation
This metaclass has no dedicated shape, it is used solely in the shapes of other metaclasses.
Formal Description
context AnyValueOrOmit
ANYVALUEOROMITLABEL ::= '*'

Comments
No comments.

6.2.21 OmitValue

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

Formal Description
context OmitValue
OMITVALUELABEL ::= 'omit'

Comments
No comments.

6.2.22 DynamicDataUse

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

Formal Description
context DynamicDataUse
DYNAMICDATASELABEL := 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 <FUNCTIONCALBLEABEL>
else if self.oclIsTypeOf(TimeLabelUse) then
  self as context in <TIMELABELUSE>
else if self.oclIsTypeOf(PredefinedFunctionCall) then
  self as context in <PREDEFINEDFUNCTIONCALBLEABEL>
endif

Comments
No comments.

6.2.23 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.function.name self as context in <DATAUSEARGUMENTLABEL>
    if not self.reduction->isEmpty() then
        self as context in <REDUCTIONLABEL>
    else
        ..
    endif

Comments
No comments.

6.2.24 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.

6.2.25 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> self as context in
    <REDUCTIONLABEL>

Comments
No comments.

6.2.26 PredefinedFunctionCall

Concrete Graphical Notation
This metaclass has no dedicated shape, it is used solely in the shapes of other metaclasses.
Formal Description

context PredefinedFunctionCall

PREDEFINED_FUNCTION_CALLLABEL ::= if (self.name.startsWith('_') and self.name.endsWith('_')) then

    self.argument -> at(0).dataUse as context in <DATAUSELABEL>
    self.name ->
    getOperatorSymbol() self.argument -> at(1).dataUse as context in <DATAUSELABEL>

else if (self.name = 'not') then 'not' self.argument.dataUse as context in <DATAUSELABEL>
else if (self.name = 'size') then 'size(' self.argument.dataUse as context in <DATAUSELABEL> ')
endif

Comments

The description above shall be applied for the predefined instances of the PredefinedFunction element. For the user-defined PredefinedFunction instances other, user-defined syntax can be used.

6.3 Time

6.3.1 TimeLabel

Concrete Graphical Notation

#ATOMIC_BEHAVIOUR#  ---  @TIME_LABELLABEL

Formal Description

context TimeLabel

TIME_LABELLABEL ::= 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

TIME_LABELUSELABEL ::= self.timeLabel.name self as context in <KIND_LABEL>

KIND_LABEL ::= [''  if self.kind = TimeLabelUseKind::first then 'first'
  else if self.kind = TimeLabelUseKind::previous then 'previous'
  else if self.kind = TimeLabelUseKind::last then 'last'
  endif ']

Comments

If self.kind = TimeLabelUseKind::last then <KIND_LABEL> is optional.
6.3.3 Wait

Concrete Graphical Notation

![Wait Graphical Notation]

Formal Description

context Wait

\[ \text{DURATIONLABEL} := \text{self.period as context in } <\text{DATAUSELABEL}> \]

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

![Quiescence Graphical Notation]

Formal Description

context Quiescence

\[ \text{DURATIONLABEL} := \text{self.period as context in } <\text{DATAUSELABEL}> \]
\[ \text{GATELABEL} := \text{self.gateReference.gate as context in } <\text{GATEINSTANCENAMELABEL}> \]

Comments

GATELABEL is optional.

If the Quiescence refers to a component instance (property self.componentInstance is set), then the Quiescence shape:

- shall cover all the lifelines of that component instance; and
- GATELABEL shall not be present,

otherwise the Quiescence shape shall:

- either cover only the lifeline of that gate, which is referred to by self.gateReference if notation (a) defined in clause 6.5.1. is used; or
- the GATELABEL shall be present if notation (b) defined in clause 6.5.1 is used.
6.3.5 TimeConstraint

Concrete Graphical Notation

\#{ATOMICBEHAVIOUR\#} \rightarrow \{ 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

\[\text{TIMERSTARTLABEL (DURATIONLABEL)}\]

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

\[\text{TIMEOUTLABEL}\]

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

Concrete Graphical Notation

```
  TIMERSTOPLABEL
```

Formal Description

**context** TimerStop

\[
\text{TIMERSTOPLABEL} ::= \text{self.timer.name}
\]

Comments

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

6.4 Test Configuration

6.4.1 TestConfiguration

Concrete Graphical Notation

```
  Test Configuration
  TESTCONFIGURATIONLABEL
```

Formal Description

**context** TestConfiguration

\[
\text{TESTCONFIGURATIONLABEL} ::= \text{self.name}
\]

Comments

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

Concrete Graphical Notation

If self.kind = GateTypeKind::Message, then

\[
\text{GATE\_TYPE\_NAME\_LABEL} \quad \text{Data Type: DATA\_TYPE\_LIST\_LABEL}
\]

If self.kind = GateTypeKind::Procedure, then

\[
\text{GATE\_TYPE\_NAME\_LABEL} \quad \text{Signature: DATA\_TYPE\_LIST\_LABEL}
\]

Formal Description

c\text{ontext} GateType

\[
\text{GATE\_TYPE\_NAME\_LABEL} :: = \text{self.name}
\]

\[
\text{DATA\_TYPE\_LIST\_LABEL} :: = \text{self.dataType.name->separator(',')}\]

Comments

No comments.

6.4.3 GateInstance

Concrete Graphical Notation

\[
\text{GATE\_INSTANCE\_NAME\_LABEL}
\]

Formal Description

c\text{ontext} GateInstance

\[
\text{GATE\_INSTANCE\_NAME\_LABEL} :: = \text{self.name [' self.type.name]}
\]

Comments

In GATE\_INSTANCE\_NAME\_LABEL the ‘:’ self.type.name is optional.
6.4.4 ComponentType

Concrete Graphical Notation

```
COMPONETYPETLABEL

Timer
TIMELISTLABEL

Variable
VARIELISTLABEL

<INSTANCELABEL>
```

Formal Description

```
context ComponentType

COMPONENTLABEL ::= self.name

TIMELISTLABEL ::= self.timer.name->separator(,'

VARIABLELISTLABEL ::= foreach v:Variable in self.variable separator(,'

    self.variable.name ':' self.variable.dataType.name

end
```

Comments

A ComponentType shape shall contain all GateInstance shapes defined for the corresponding ComponentType, at any side or corner.

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

6.4.5 ComponentInstance

Concrete Graphical Notation

```
COMPONENTROLELABEL

COMPONENTINSTANCENAMELABEL

<INSTANCELABEL>
```

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 shall contain all GateInstance shapes defined for the corresponding ComponentType, at any side or corner.

The terminal symbols 'SUT' and 'TESTER' shall be typed in bold.

NOTE: If the ComponentInstance shape is used inside the Behaviour compartment of a TestSpecification shape, all the rectangles representing the GateInstance(s) of a ComponentInstance may be left out, see notation (b) in clause 6.5.1.
6.4.6 Connection

Concrete Graphical Notation

NAMEOFCONNECTIONLABEL := self.name

Comments

NAMEOFCONNECTIONLABEL is optional.

6.5 Test Behaviour

6.5.1 TestDescription

Concrete Graphical Notation

```
ORDERINGLABEL
Test Description
TESTDESCRIPTIONNAME LABEL

Parameter
TD PARAMETERLABEL

Test Objective
TEST OBJECTIVELABEL

Configuration
TEST CONFIGURATIONNAME

Behaviour
```

#GATEREFERENCE# NAMEOFCONNECTIONLABEL #GATEREFERENCE#
Formal Description

**context** TestDescription

**ORDERING**::= if self.isLocallyOrdered = false then 'Globally Ordered'

else 'Locally Ordered'
endif

**TestDescription**::= self as context in <NamedElementLabel>

**TDParameter**::= foreach p:Parameter in self.formalParameter separator(‚,‘)

p as context in <ParameterLabel>
end

**TestObjective**::= foreach t:TestObjective in self.testObjective separator(‚,‘)

t as context in <NamedElementLabel>
end

**TestConfiguration**::= self.testConfiguration as context in <NamedElementLabel>

Comments

In case of a globally ordered **TestDescription** (self. isLocallyOrdered = false) then the **ORDERING** is optional. The result of the production rule of **ORDERING** shall be typed by bold font.

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

In the lowest compartment the behaviour of the test description may be described. In this compartment, there shall be as many **ComponentInstance** shapes as many component instances are defined in the **TestConfiguration** referenced in a **Configuration** compartment.

For each **ComponentInstance** shape either the rectangles representing the **GateInstance**(s) shall be (a) shown or (b) not shown.

- In notation (a) from each gate instance a vertical line ("lifeline") shall originate, to which each **Behaviour** element defined in that test description and associated with that gate shall be attached.
  
  - If a component instance has only one gate then the **GateInstanceNameLabel** is optional.
  
  - If a **GateInstance** of a **ComponentInstance** is not connected in the **TestConfiguration** referenced in a **Configuration** compartment, it is optional if that **GateInstance** and its lifeline are shown or not.

- In notation (b) from the **ComponentInstance** shape only one vertical line ("lifeline") shall originate, to which each **Behaviour** element defined in that test description and associated with any of the **GateInstance**(s) of that **ComponentInstance** shall be attached.

The time of a lifeline passes from top to down.

Implementation only of one of the two notations (a) and (b) is required, the implementation of the other is optional.

If both notations are implemented, for a given **ComponentInstance**, the two notations, (a) and (b) shall not be mixed.

NOTE: In a **TestDescription** the two notations, (a) and (b) may be mixed for different **ComponentInstances**, that is for some **ComponentInstance**(s) the notation (a) while for other **ComponentInstance**(s) the notation (b) may be used.

6.5.2 Behaviour

Concrete Graphical Notation

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

**Concrete Graphical Notation**

A CombinedBehaviour shape shall cover all the lifelines.

**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 clauses describing the symbols of subclasses of CombinedBehaviour. Depending on the concrete type of the CombinedBehaviour, it may or may not contain more than one block. The outermost border of the contained Block(s) shall not be visible. If more than one block is defined, they shall be separated by thin dashed lines. Any number of periodic and/or exceptional behaviour may be attached in any order to a CombinedBehaviour.

A CombinedBehaviour shape shall cover all the lifelines.
6.5.4 Block

Concrete Graphical Notation

Formal Description

context Block

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

Comments

A Block shall not 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 shall be separated by dashed lines.

The GUARDLABEL is optional if it is not stated otherwise in the containing CombinedBehaviour. If GUARDLABEL is present, it shall be placed in between square brackets ('[' and ']'). and in a globally ordered TestDescription the GUARDLABEL shall be placed at the top left part of the symbol of the Block, while in a locally ordered TestDescription the GUARDLABEL(s) shall be placed close to the top border of the symbol of the Block and close to the lifeline of the related ComponentInstance.

6.5.5 CompoundBehaviour

Concrete Graphical Notation

Formal Description

n.a.

Comments

[GUARDLABEL] in its contained Block is optional.
6.5.6 BoundedLoopBehaviour

Concrete Graphical Notation

![boundedLoop]

```
context BoundedLoopBehaviour

ITERATIONLABEL ::= self.numIteration as context in <DATASETLABEL>
```

Comments

In a globally ordered TestDescription, the iteration: ITERATIONLABEL shall be placed at the top right part of the symbol of the Block, while in a locally ordered TestDescription the iteration: ITERATIONLABEL(s) shall be placed at the top left part of the symbol of the Block, while in a locally ordered TestDescription the iteration: ITERATIONLABEL(s) shall be placed close to the top border of the symbol of the Block, and close to the lifeline of the related ComponentInstance.

6.5.7 UnboundedLoopBehaviour

Concrete Graphical Notation

![unboundedLoop]

```
unboundedLoop

[<GUARDLABEL>]
```

Formal Description

n.a.

Comments

[GUARDLABEL] in its contained Block is optional.
6.5.8 OptionalBehaviour

Concrete Graphical Notation

```
optional

[<GUARDLABEL>]
```

Formal Description
n.a.

Comments
[GUARDLABEL] in its contained Block is optional.

6.5.9 AlternativeBehaviour

Concrete Graphical Notation

```
alternative

[<GUARDLABEL>]

[<GUARDLABEL>]
```

Formal Description
n.a.

Comments
Any number of Blocks may be contained, they shall be separated by dashed lines.

[GUARDLABEL] in any Block is optional.
6.5.10 ConditionalBehaviour

Concrete Graphical Notation

```
conditional

[<GUARDLABEL>]

[<GUARDLABEL>]
```

Formal Description
n.a.

Comments
Any number of Blocks may be contained, they shall be separated by dashed lines.
If there are more than one Block, then the [GUARDLABEL] in the last Block is optional.

6.5.11 ParallelBehaviour

Concrete Graphical Notation

```
parallel

[<GUARDLABEL>]

[<GUARDLABEL>]
```

Formal Description
n.a.

Comments
Any number of Blocks may be contained, they shall be separated by dashed lines.
[GUARDLABEL] in any Block is optional.
6.5.12 DefaultBehaviour

Concrete Graphical Notation

Formal Description

context DefaultBehaviour

DEFAULTCOMPONENTLABEL ::= if not self.guardedComponent-&gt;isEmpty() then
  'for Component ' self.guardedComponent.name
  else
  ..
  endif

Comments

A DefaultBehaviour shape may be attached to any CombinedBehaviour.

[GUARDLABEL] in its contained Block is optional.

DEFAULTCOMPONENTLABEL shall only present if guardedComponent is set.

6.5.13 InterruptBehaviour

Concrete Graphical Notation

Formal Description

context InterruptBehaviour

INTERRUPTCOMPONENTLABEL ::= if not self.guardedComponent-&gt;isEmpty() then
  'for Component ' self.guardedComponent.name
  else
  ..
  endif
Comments

An InterruptBehaviour shape may be attached to any CombinedBehaviour.
[GUARDLABEL] in its contained Block is optional.
INTERRUPTCOMPONENTLABEL shall only present if guardedComponent is set.

6.5.14 PeriodicBehaviour

Concrete Graphical Notation

```
periodic     period: TIMELABEL

[GUARDLABEL]
```

Formal Description

```
context PeriodicBehaviour

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

Comments

A PeriodicBehaviour shape may be attached to any CombinedBehaviour.
[GUARDLABEL] in its contained Block is optional.

6.5.15 Break

Concrete Graphical Notation

```
<shape>
```

Formal Description

n.a.

Comments

The Break shape shall cover all the lifelines.
6.5.16 Stop

Concrete Graphical Notation

Formal Description
n.a.

Comments
The Stop shape shall cover all the lifelines.

6.5.17 VerdictAssignment

Concrete Graphical Notation

Formal Description

`context Verdict

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

Comments
The VerdictAssignment shape shall cover all the lifelines.

6.5.18 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. Either none of them or both of them shall be shown.
The **Assertion** shape shall cover all the lifelines, if componentInstance is not specified, otherwise it shall cover all the lifelines of that componentInstance.

### 6.5.19 Message

**Concrete Graphical Notation**

**Point-to-point Message**

```
SOURCEGATELABEL: #GATEREFERENCE#  
ARGUMENTLABEL                  #TARGET#   
VARIABLELABEL                  TARGETGATELABEL
```

**Point-to-point Trigger Message**

```
SOURCEGATELABEL: #GATEREFERENCE#  
ARGUMENTLABEL                  #TARGET#   
VARIABLELABEL                  TARGETGATELABEL
```

**Point-to-multipoint Message**

```
SOURCEGATELABEL: #GATEREFERENCE#  
ARGUMENTLABEL                  #TARGET#   
VARIABLELABEL                  TARGETGATELABEL
```

**Point-to-multipoint Trigger Message**

```
SOURCEGATELABEL: #GATEREFERENCE#  
ARGUMENTLABEL                  #TARGET#   
VARIABLELABEL                  TARGETGATELABEL
```

**Formal Description**

**context Message**

\[
\text{ARGUMENTLABEL} ::= \text{self.argumem as context in } <DataUseLabel>
\]

\[
\text{VARIABLELABEL} ::= \text{self.target.valueAssignment.variable.name}
\]

\[
\text{SOURCEGATELABEL} ::= \text{self.sourceGate.gate as context in } <GateInstanceNameLabel>
\]

\[
\text{TARGETGATELABEL} ::= \text{self.target.targetGate.gate as context in } <GateInstanceNameLabel>
\]

**Comments**

\[
\text{SOURCEGATELABEL, TARGETGATELABEL, VARIABLELABEL ' ::= ' are optional.}
\]

The ends of a message (GATEREFERENCE and TARGET) shall be placed onto the lifeline of the corresponding gate instances, if notation (a) defined in clause 6.5.1 is used. If notation (b) defined in clause 6.5.1 is used, then the corresponding end of a message shall be placed on the lifeline of the corresponding component instance and SOURCEGATELABEL, and/or TARGETGATELABEL shall be present, respectively.

In case of a point-to-point or a point-to-point trigger message, the VARIABLELABEL - if present - may be placed either above the arrow as an assignment or under the arrowhead.

In case of a point-to-multipoint or a point-to-multipoint trigger message, the source GATEREFERENCE shall be indicated by a small black square, and there shall be as many arrows present as many targets are in the point-to-multipoint message. In this case, optionally there may be a VARIABLELABEL presented under each arrowhead.
6.5.20 ProcedureCall

Concrete Graphical Notation

Formal Description

```
context ProcedureCall

PROC_CALL_ARGUMENT_LABEL ::= self.signature.name '(' 
foreach a: ParameterBinding in self.argument separator(',') 
  a.parameter.name ' := ' a.dataUse as context in <DATA_USE_LABEL>
end ')'  

VALUE_ASSIGNMENT_LABEL ::= foreach v: ValueAssignment in self.Target.valueAssignment separator(',') 
v.variable.name ' := ' v.parameter.name
end
```

Comments

SOURCE_GATE_LABEL, TARGET_GATE_LABEL, VALUE_ASSIGNMENT_LABEL' are optional.

The ends of a message (GATE_REFERENCE and TARGET) shall be placed onto the lifeline of the corresponding gate instances, if notation (a) defined in clause 6.5.1 is used. If notation (b) defined in 6.5.1 is used, then the corresponding end of a message shall be placed on the lifeline of the corresponding component instance and SOURCE_GATE_LABEL, and/or TARGET_GATE_LABEL shall be present, respectively.

A procedure call consists of one calling and one or several reply ProcedureCalls. The lifeline of the called component instance of a procedure call if notation (b) defined in clause 6.5.1 is used or the lifeline of the corresponding gate instance of that component instance if notation (a) defined in clause 6.5.1 is used shall be modified between the calling and the last reply ProcedureCalls: instead of a line a narrow rectangle, a so called 'ExecutionSymbol' shall be used.

NOTE: The reply/replies may be in block(s) of an AlternativeBehaviour.
6.5.21 ActionReference

Concrete Graphical Notation

Formal Description

context ActionReference

ACTIONREFERENCELABEL ::= self.action as context in <ACTIONREFERENCELABEL>

ACTIONARGUMENTLABEL ::= foreach p:ParameterBinding in self.argument separator(‘,’)
                         p.dataUse as context in <DATASURELABEL>
                         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.
6.5.22 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.

6.5.23 Assignment

Concrete Graphical Notation

Formal Description

```
context Assignment
LHSLABEL ::= self.variable as context in <VARIABLEUSELABEL>
RHSLABEL ::= self.expression as context in <DATASENTURELABEL>
```

Comments

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

Concrete Graphical Notation

```
<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST DESCRIPTION NAME LABEL</td>
</tr>
<tr>
<td>(TAARGUMENT LABEL)</td>
</tr>
<tr>
<td>BINDINGS LABEL</td>
</tr>
</tbody>
</table>
```

Formal Description

```
context TestDescriptionReference

TEST DESCRIPTION NAME LABEL ::= self.testDescription.name

TAARGUMENT LABEL ::= foreach p:ParameterBinding in self.argument separator(’,’) p.dataUse as context in <DATA USE LABEL>
end

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

Comments

(TAARGUMENT LABEL) and BINDINGS LABEL are optional (that is any of them or both may 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

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.1.1</td>
<td>June 2015</td>
<td>Publication</td>
</tr>
<tr>
<td>V1.2.1</td>
<td>September 2016</td>
<td>Publication</td>
</tr>
<tr>
<td>V1.3.1</td>
<td>March 2018</td>
<td>Membership Approval Procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MV 20180520; 2018-03-21 to 2018-05-21</td>
</tr>
<tr>
<td>V1.3.1</td>
<td>May 2018</td>
<td>Publication</td>
</tr>
</tbody>
</table>