Early draft ES 203 119 V00.00.01 (2013-05)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL)

Reference
DES/MTS-140_TDL

Keywords
METHODOLOGY, Language, MBT, TESTING,
TSS&TP, TTCN-3, UML

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

If you need to update the table of content you would need to first unlock it.
To unlock the Table of Contents: select the Table of Contents, click simultaneously: Ctrl + Shift + F11.
Then lock it: reselect the Table of Contents and then click simultaneously: Ctrl + F11.

Intellectual Property Rights 5
Foreword 5
Introduction 5
1 Scope 6
2 References 6
2.1 Normative references 6
2.2 Informative references 7
3 Definitions, symbols and abbreviations 7
3.1 Definitions 7
3.2 Symbols 9
3.3 Abbreviations 9
4 Introduction to TDL 10
4.1 Motivation 10
4.2 General Design Principles 10
5 TDL Specification Structure 11
5.1 Foundation 11
5.2 Packaging 11
6 Data Values 12
6.1 Data Instance 12
6.2 Simple Data Instance 13
6.3 Structured Data Instance 13
7 Test Architecture 14
7.1 Component Type 14
7.2 Gate Type 14
7.3 Test Configuration 15
7.3.1 Component instances 15
7.3.2 Gate instances 16
7.3.3 Connections 16
8 Test Behaviour 17
8.1 Test Description 17
8.2 Interaction Flow 17
8.3 Atomic Behaviour 18
8.4 Combined Behaviour 19
8.5 Exceptional and Periodic Behaviour 20
9 Time 21
9.1 Time Observations 21
9.2 Time Constraints 21
10 Miscellaneous 22
10.1 Comments 22
10.2 Annotations 23
10.3 Test Objectives 24
Annex A (normative): TDL Meta-Model 25
Annex B (normative): TDL Concrete Syntaxes 26
B.1 TDL Textual Syntax 26

B.2 TDL Graphical Syntax 26
Annex C (informative): Examples of TDL Specifications....27
C.1 Example: 3GPP 27
C.2 Example: IMS 27
History 28

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

Introduction

This clause is optional. If it exists, it is always the third unnumbered clause.

Clause numbering starts hereafter.
Check http://portal.etsi.org/edithelp/Files/other/EDRs_navigator.chm clauses 5.2.3 and A.4 for help.

<PAGE BREAK>

1 Scope

The ES (ETSI Standard) shall be chosen when the document contains normative provisions and it is considered preferable or necessary that the document be submitted to the whole ETSI membership for its approval.

The scope shall always be clause 1 of each ETSI deliverable and shall start on a new page (more details can be found in clause 11 of the EDRs).

No text block identified. Forms of expression such as the following should be used:

*The Scope **shall not** contain requirements.*

This document provides the Early Draft of the TDL ETSI Standard. It contains the general structure an table of contents and an early discussion of the features of the TDL abstract syntax. The intended TDL use case is the application of TDL in the design of test descriptions as they already exist at ETSI for interoperability testing of, e.g., IMS and conformance testing in the scope of 3GPP. Follow-up versions will focus on making a consistent language design for TDL and will avoid the introduction of additional features if they are not absolutely necessary to cover the intended first TDL use case.

2 References

The following text block applies. More details can be found in clause 12 of the EDRs.

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 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.

2.1 Normative references

Clause 2.1 only shall contain normative (essential) references which are cited in the document itself. These references have to be publicly available and in English.

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

- Use the **EX** style, enclose the number in square brackets and separate it from the title with a tab (you may use sequence fields for automatically numbering references, see clause A.4: "Sequence numbering") (see example).

EXAMPLE:

[1] ETSI EN 301 025-3: "<Title>".

[2] ETSI EN 300 163: "<Title>".

2.2 Informative references

Clause 2.2 shall only contain informative references which are cited in the document itself.

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.

- Use the **EX** style, add the letter "i" (for informative) before the number (which shall be in square brackets) and separate this from the title with a tab (you may use sequence fields for automatically numbering references, see clause A.4: "Sequence numbering") (see example).

EXAMPLE:

[i.1] ETSI TR 102 473: "<Title>".

[i.2] ETSI TR 102 469: "<Title>".

3 Definitions, symbols and abbreviations

Delete from the above heading the word(s) which is/are not applicable, (see clauses 13 and 14 of EDRs).

*Definitions and abbreviations extracted from ETSI deliverables can be useful when drafting documents and can be consulted via the **Terms and Definitions Interactive Database (TEDDI) (http://webapp.etsi.org/Teddi/).***

3.1 Definitions

Clause numbering depends on applicability.

- ***A definition shall not take the form of, or contain, a requirement.***
- ***The form of a definition shall be such that it can replace the term in context. Additional information shall be given only in the form of examples or notes (see below).***
- ***The terms and definitions shall be presented in alphabetical order.***

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

[**Editor note**: The following list serves as a glossary derived from the meta-model to define a common vocabulary for TDL. It might be moved to a different location in the document in future.]

Alternative: Combined behaviour that contains 1 to many branches. If two or more branches are executable to branch that appears first in the list is executed.

Annotation: An element that provides a key attribute and an optionally empty value attribute. Any element except an annotation itself can contain any number of annotations.

Behaviour: A generic, abstract element that is refined into atomic and combined behaviour elements. Every behaviour operates on 1 to many gate instances.

Block: A container of behaviours that executed in a strictly sequential manner (c.f. strict sequencing in UML) that is optionally guarded.

Bounded loop: Combined behaviour that contains 1 block. The block is executed a number of times defined by its min/max attributes.

Branch: A special block with the following restriction: First Interaction in the branch should refer to a sending event of a GateInstance that belongs to a component instance that assumes the role 'SUT' and to a receiving event needs to occur on a GateInstance that belongs to a component instance that assumes the role of 'Tester'.

Combined behaviour: Behaviour that operates on 2 to many gate instances and must contain 1 to many blocks required to carry out the refined combined behaviour types.

Comment: An element that provides a value attribute. Any element except a comment itself can contain any number of comments.

Component instance: An element that is in a test configuration. A component instance contains one or more gate instances. It can only own gate instances of gates that are referred to in the corresponding component (type). It has a role that is either of component kind 'Tester' or 'SUT'.

Component: An element that refers to one or many gates.

Conditional: A combined behaviour, whose blocks must define guards that are mutually exclusive.

Connection: An element contained in the test configuration. A connection refers to two gate instances as its end points. Each connection refers uniquely to two different gate instances, which shall belong to different component instances.

Element import: Defines means to reference packageable elements defined outside of the containing package.

Element instance [to be renamed]: A packageable element that represents either a simple instance or a structured instance of data values.

Element: The most basic element of TDL. It contains an optionally empty name attribute and any number of comments and annotations. Every element has the inheritance capability of other elements.

Exceptional behaviour: Behaviour that consists of one to many branches. It is optionally contained within a combined behaviour. Exceptional behaviour can be either of the kind 'default' or 'interrupt'. A default exceptional behaviour defines behaviour that is executed once if one of the associated branches becomes executable and terminates the combined behaviour it is contained in. An interrupt exceptional behaviour defines behaviour that is executed if one of the associated branches becomes executable and returns execution to the combined behaviour it is contained in.

Exit event: A multiple gate event that terminates the execution of a test description.

Gate event: A behaviour that operates exactly on one gate instance.

Gate instance: An element that is contained in component instance and serves as the end point of a connection. A gate instance is an instance of a gate (type).

Gate interaction event: A gate event that represents either the sending event or receiving event of an interaction. A receive event shall be blocking; a send event shall be non-blocking.

Gate: An element that belongs to zero or many components and that specifies the element instances (data values) that can be sent or received via its gate instances.

Interaction flow: Element of the test description that contains a block which must not declare guard expression. It is the root element of the behaviour expression of the test description.

Interaction: Describes the exchange of values defined as element instances. It is described by gate interaction events. An interaction covers all different interaction kinds such as message-passing, procedure-calls, shared variable access.

Local action event: A gate event that is used to describe an activity at a gate instance of a component instance as informal description such as a local computation.

Multiple gate event: Behaviour that operates on all gate instances defined in the test configuration referenced by the containing test description.

Package: A packageable element used as a container for any number of further packageable elements. The package also contains any number of element imports of package elements. The 'viewpoint' attribute of a package defines a restriction on that package used to limit the scope of a package.

Packageable element: A generic, abstract element that can be contained or imported in a package. Packagable elements are packages, components, gates, test configurations, test descriptions, and test objectives.

Parallel: A combined behaviour of 2 to many blocks, which are executed concurrently during runtime. All blocks shall not declare guard expressions.

Periodic: Behaviour that consists of one to many blocks. It is optionally contained within a combined behaviour. A periodic behaviour defines behaviour that is executed periodically in parallel to the enclosing combined behaviour. The attribute 'period' specifies the frequency of the execution.

Simple instance [to be renamed]: An element instance that represents a simple data value specified as a literal.

Structured instance [to be renamed]: An element instance that refers to one or more element instances that in turn are refined as simple or structured instances. A structured instance is used to define tuple of data values.

TDL specification: The root element of a TDL model that contains all packageable elements needed to describe this TDL model.

Test configuration: A packageable element that contains two or more component instances and one or many connections. It is referenced in a test description and shall contain at least one 'Tester' component instance and one 'SUT' component instance.

Test description reference: A multiple gate event that refers to an existing test description that shall be executed and contains a possibly empty list of actual parameters. This list of actual parameters shall match the list of parameters in the referenced test description. After execution of the referenced test description the calling behaviour is continued.

Test description: A packageable element that contains an interaction flow and refers to zero or many test objectives. It contains the attribute 'isTestCase' that indicates whether a test description can serve as the starting point for execution. In addition it contains a possibly empty list of parameters that is currently not further detailed.

Test objective: A packageable element that contains the specification of that test objective. Test objectives can be referenced by a test description. Test objectives are typically system requirements, test purposes or system use cases.

Time constraint: [to be discussed further]

Time observation: [to be discussed further]

Time specification: [to be discussed further]

Timeout event: [to be discussed further] A gate event that is used as the first event in a branch. It is used to describe the end of a time duration.

Unbounded loop: Combined behaviour that contains 1 block. The block is executed potentially an infinite number of times, unless the guard expression evaluates to 'false' iff present.

Verdict event: A gate event that is used to assign explicitly a verdict to the execution of a test description. The vedict can take on any value of the following kinds: none, pass, inconclusive, fail, error.

3.2 Symbols

Clause numbering depends on applicability.

For the purposes of the present document, the [following] symbols [given in ... and the following] apply:

Symbol format

<symbol>	<Explanation>
<2nd symbol>	<2nd Explanation>
<3rd symbol>	<3rd Explanation>

3.3 Abbreviations

Abbreviations should be ordered alphabetically.

Clause numbering depends on applicability.

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

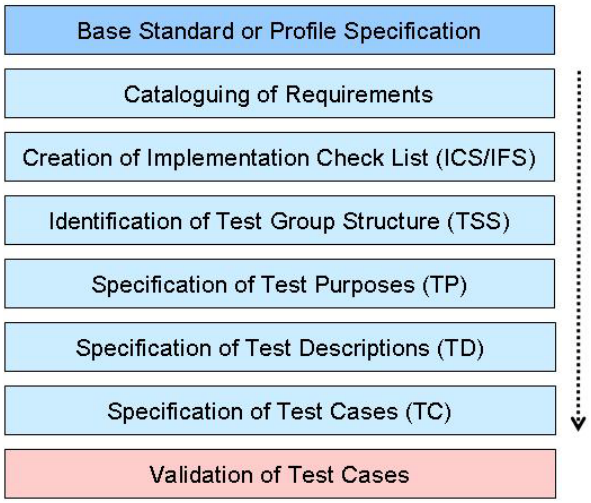
MOF	UML Meta-Object Facility
MSC	Message Sequence Chart
SUT	System Under Test
TDL	Test Description Language
UML	Unified Modeling Language

BELOW: LINKS TO EXTERNAL FILES FOR THE VARIOUS CLAUSES

4 Introduction to TDL

4.1 Motivation

The trend towards a higher degree of system integration such as in case of cyber-physical systems as well as the emergence of service-oriented architectures leads to a growing importance of asynchronous communication paradigms, concurrency, and distribution in system development. In the testing domain, a variety of programming and scripting languages are used, e.g. TTCN-3. Further to this, Message Sequence Charts (MSC) and derivates, such as Unified Message Language (UML) sequence diagram (SD) are used commonly for designing and visualizing tests. TDL attempts to bridge the gap between high-level test purpose descriptions, e.g. expressed in TPLan, and executable test test cases, e.g. epressed in TTCN-3, by providing a formal language for the specification of test descriptions. The following figure details the test specification development process at ETSI.



In addition to supporting the ETSI process, TDL has advantages in a number of application areas such as:

- Manual design of tests for the purpose of manual or automated test implementation;
- Documentation of tests derived from other sources, e.g. MBT tools, system simulators;
- Representation of execution traces from simulation or test runs.

4.2 General Design Principles

- Lanugage key elements of TDL
  - Abstract syntax: common meta-model that forms the core of TDL; an instance of the meta-model is the TDL model (also called: TDL specification);
  - Concrete syntax: different syntaxes for end-users of TDL can be provided such as textual or UML-like syntaxes;
  - Semantics: providing a formal meaning to the different language elements of the meta-model;
- TDL abstract syntax is defined using UML MOF [Complete MOF vs. Essential MOF not completely discussed; preference for EMOF];
- A test description expresses the the expected behaviour of a test case and makes no or only little assumptions about deviating execution paths;
- Simplistic test data type model is deployed that relies on data type definitions that are external to a TDL specification.

5 TDL Specification Structure

5.1 Foundation

Overview

Foundation is responsible of specifying the very fundamental concepts of a TDL model (aka test description specification). All other features of TDL rely on the concepts defined in Foundation, but may refine or restrict them, if required.

Abstract syntax

[Not yet available.]

Semantics

An *Element* represents an abstract constituent of a TDL model that is subclassed (either directly or transitively) by more specific TDL concepts (metaclasses). Element has no further semantics than providing an optional name for each specific concept. When names are required by a TDL concept, an according constraint will be explicitly stated in that context.

The root of each test description specification is called *TDLSpecification*. A TDLSpecification does not prescribe its structure.

Test description specifications may have varying levels of completeness. They might be self-contained or partially specified. Furthermore, test description specifications might be isolated from other test descriptions or import parts of other test description specifications.

Constraints

1) Each TDLSpecification must have a non-empty name.

Classifier description

[Not yet available.]

5.2 Packaging

Overview

Packaging introduces the general facilities for grouping and scoping of elements.

Abstract syntax

[Not yet available.]

Semantics

A *Package* represents a container for certain packagebale elements that shall be grouped together. A TDL model may contain any number of Packages which may have any arbirtrary substructure.

Packages hide their contents from the outside, thus, a Package is the scope for its packaged elements. A Package does not restrict a priori what elements it may contain. A Package provides the ability to add further information about the

Package as viewpoint. Tool vendors might use the viewpoint of a Package for tool-specific functionality. [Editorial note: Might also be expressed as Annotation]

A Package may import elements from other Packages. When a Package imports another Package, all elements in the imported Package are visible within the importing Package. This holds also true for Packages that are contained in imported Packages, meaning that each element contained by a Package that is contained in the acutally imported Package are visible within the importing Package. [Editorial note: the concrete algorithm for calculcating visibility have not been discussed to all extent]

A packageable element must provide a qualified name. The qualified name is a compound name derived from all directly and indirectly parent packages by concatenating the names each Package. [Editorial note: We have not yet decided about the separator between each Package name in a qualified name] The name of TDLspecification that (transitively) owns the packageable element constitutes always the first segment of the qualified name. The qualified name of a packageable elements must be distinctive within the (containing and importing) Package.

[Editorial note: We have not yet discussed the situation that we import an element from another TDL model that has the same name as the importing TDL model and where the structure with respected to the importing Package is exactly the same. This may lead to name clashes nevertheless.]

Constraints

2) Reflexive import of Packages is not permitted.

3) A Package must not import elements from itself.

4) It is not allowed for a Package to contain any two elements with identic qualified names (i.e., uniqueness of identifiers)

5) The first segment of the qualified name is always the name of the TDLSpecification the packageable elment is (transitively) contained in.

Classifier description

[Not yet available.]

6 Data Values

In the TDL Meta-model data values of a TDL specification are represented as literals. As no data type system is defined in the current version of TDL interpretation of data values may be done only external to a TDL specification model.

NOTE: Semantics of data value instances in a TDL specification may be given by use of an annotation indicating the origin of the data notation used for data values, e.g. TTCN-3 or ASN.1.

[**Editors note:** Current meta-model notation “ElementInstance” renamed to “DataInstance” in this draft, and correspondingly “SimpleInstance” and “StructuredInstance” are renamed to “SimpleDataInstance” and “StructuredInstance”.]

6.1 Data Instance

Overview

A data instance is either a simple data value instance or a structured data value instance.

Abstract syntax

[Not yet available.]

Semantics

None.

Constraints

None.

Classifier description

Generalizations:

- PackageableElement

6.2 Simple Data Instance

Overview

Simple data values are represented as a literal.

Abstract syntax

[Not yet available.]

Semantics

None.

Constraints

Classifier description

Generalizations:

- DataInstance

Attributes:

- literal : String
  Representation of the data value

6.3 Structured Data Instance

Overview

A structured data value is an ordered optionally empty list of Data value instances, hence each element in the list is either a simple data value or a structured data value.

Abstract syntax

[Not yet available.]

Semantics

None

Constraints

None

Classifier description

Generalizations:

- DataInstance

7 Test Architecture

In TDL, each test description sall refer to a test configuration that describe the actors of that test description and the connections amog them.

The fundamental units of a TDL test description are the component instances. Each component instance models a functional entity of a test system. A component instance may either be (a part of) the Tester or (a part of) the SUT. (That is both the Tester and the SUT can be decomposed.) The communication among the components takes place through gate instances, that are interconnected by connections. To support the reusability, TDL introduces component type and gate type. A component type describes the number and the type of the gates the instances of that component type have, while the gate type describes the [types of the] element instances [(messages)] that can be sent and received through the instances of that gate type.

7.1 Component Type

Overview

A component type is an element that refers to one or more gate type. A component defines which gates are associated with a component. Instances of components model the functional entities of a test system.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

6) A component shall refer to at least one gate.

Classifier description

[Not yet available.]

7.2 Gate Type

Overview

A gate type is an element that specifies the element instances [(data values, “messages”)] that can be sent and received via the instances of the gate. Each gate shall be referred by one or more component.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

[Not yet available.]

Classifier description

[Not yet available.]

7.3 Test Configuration

Overview

A test configuration is a packageable element. A test configuration contains two or more component instances. The component instances contain gate instances that are the endpoints of connections.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

7) Among the component instances at least one shall act as a ‘Tester’ component instance and at least one shall act as an ‘SUT’ component instance.

8) A connection can only be established between two gate instances of two different component instances.

9) In between any two gate instances maximum one connection can exist.

10) A test configuration shall be referenced by at least one test description.

Classifier description

[Not yet available.]

7.3.1 Component instances

Overview

A component instance is an element in a test configuration. A component instance models a functional entity of a test system, and contains one or more gate instances. A component insctance may act either as a ‘Tester’ or as an ‘SUT’ component instance.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

11) A component instance shall have as many gate instances of gates type as is referred to in the corresponding component type.

Classifier description

A component instance has an attribute 'role', of ComponentKind. The role attribute indicates whether the component instance acts as a 'Tester' or 'SUT' component instance in the corresponding test configuration.

7.3.2 Gate instances

Overview

A gate instance is an element is an instance of gate type. A gate instance shall be contained by a component instance. Gate instances are the active elements in a test configuration, at which gate interaction events can be specified.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

[Not yet available.]

Classifier description

[Not yet available.]

7.3.3 Connections

Overview

A connection is an element contained in a test configuration. The connections are the communicational paths in test configurations. Each connection refers to two different gate instances of two different component instances as its endpoints.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

12) A connection shall refer to two different gate instances. These gate instances shall belong to two different component instances.

13) In between any two gate instances at maximum one connection shall exist.

14) The endpoint gate instances of a connection shall be compatible, that is data values sent by one of the endpoints shall be able to receive by the other endpoint, and vice versa.

Classifier description

[Not yet available.]

8 Test Behaviour

8.1 Test Description

Overview

The *TestDescription* element is one of the key elements of a TDL model. It is used to describe the test behaviour based on tester/SUT interactions. Test descriptions can be grouped into packages. A test description contains exactly one interaction flow that defines the behaviour of that test description.

A test description is associated with a test configuration that serves in the definition of the interaction flow and a set of test objectives that allow for expressing coverage information of that test description. Further, it contains the *isTestCase* flag as an attribute to indicate whether this test description is a starting test description. A further attribute is a possibly empty ordered set of parameters.

Abstract Syntax

[Not yet available.]

Semantics

The test description defines the test behaviour that can be executed on a given test configuration in terms of an interaction flow.

Constraints

15) A test description shall be associated exactly with one test configuration.

Classifier description

Attributes:

- isTestCase: Boolean
  A flag indicating whether the test description is a starting test description

- parameter: String
  A possibly empty list of parameters given as data instances

8.2 Interaction Flow

Overview

An *InteractionFlow* element is part of a test description and contains a single, unguarded *Block* element that in turn defines the behaviour of the test description it belongs to. The behaviour defined inside a block is assumed to be executed sequentially.

Abstract Syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

16) The *Block* element of the interaction flow shall not declare a guard condition.

17) The behaviour contained in a *Block* element is executed sequentially.

Classifier description

- Block::guard: String [0..1]
  An optional expression that guards the execution of the block and shall be missing since the block is contained within an interaction flow.

8.3 Atomic Behaviour

Overview

Atomic behaviour is part of a *Block* element of an interaction flow. It defines the simplest form of behaviour that cannot be decomposed further. There are two types of atomic behaviour:

- **Gate events** that occur exactly at one single gate instance of a provided test configuration; and
- **Multiple gate events** that occur over all gate instances defined within a test configuration.

Gate events are classified further into:

- **Gate interaction events** that refer to send or receive events of an interaction, e.g. a message exchange between two gate instances of two components;
- **Local action events** that allow to describe any test related activity as free, unstructured text [**editor note**: might be moved to a multiple gate event to cover all gate instances that refer to tester components];
- **Verdict events** that allow setting an explicit test verdict within a test description. The verdict kind can be anyone from the following list. No test verdict arbiter is associated with setting a verdict.
  - None
  - Pass
  - Inconclusive
  - Fail
  - Error
- **Timeout events** (or quiescence events) that are treated similarly to receive events, but without associated send events. A timeout event denotes the end point of a time interval, in which no gate interaction events at the associated gate instance occur that could be received by a tester component.

Multiple gate events are classified further into:

- **Test description reference events** that reference another test description for execution together with a potentially empty list of actual parameters that need to fit to the formal parameter list of the referenced test description;

- **Exit events** that terminate the execution of a test description. No behaviour can be executed after an exit event.

Abstract Syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

18) Timeout events occur only at gate instances that are associated to tester components.

Classifier description

[Not yet available.]

8.4 Combined Behaviour

Overview

The *CombinedBehaviour* element is a behaviour that involves all gate instances defined in an associated test configuration. It contains one or many ordered and potentially guarded *Block* elements in accordance with the different types of combined behaviour defined:

- **Alternative**: It consists of two to many blocks, where each block shall start with a gate interaction event that refers to a receive event at a gate instance of a tester component. Each block can be optionally guarded.
- **Conditional**: It consists of one to many blocks, where each block shall start with a gate interaction event that refers to a send event at a gate instance of a tester component. Each block shall be guarded.
- **Parallel**: It consists of two to many blocks. Each block shall not be guarded.
- **Unbounded loop**: It consists of exactly one block that can be optionally guarded.
- **Bounded loop**: It consists of exactly one block that shall not be guarded. Additional attributes of *min* and *max* of Integer allow to specify the number of execution cycles.

Abstract Syntax

[Not yet available.]

Semantics

[Not yet available.]

The operational semantics when executing the above types of combined behaviour are as follows:

- **Alternative**: Depending on the receive event that occurs during execution the associated block that contains this receive event is executed provided that the optional guard of the block evaluates to true. All other blocks are skipped. If no matching receive event occurs, the execution stops.
- **Conditional**: The execution continues with the block that contains a guard that evaluates to true. All other blocks are skipped. If no guard evaluates to true, the execution stops.
- **Parallel**: All specified blocks are executed in parallel.
- **Unbounded loop**: The block is executed an infinite number of times. If the block is guarded, the execution terminates if the guard evaluates to false.

- **Bounded loop**: The block is executed a given number of times specified through the two attributes *min* (minimum number of cycles) and *max* (maximum number of cycles). That is, the loop terminates after at least *min* cycles, but at most *max* cycles.

Constraints

19) A combined behaviour covers all gate instances defined within a test configuration.

20) The first gate interaction event of an alternative shall be a receive event that occurs at a gate instance of a tester component.

Classifier description

[Not yet available.]

8.5 Exceptional and Periodic Behaviour

Overview

The *ExceptionalBehaviour* element is behaviour that consists of one to many blocks, where each block shall start with a gate interaction event that refers to a receive event at a gate instance of a tester component. It is optionally contained within a *CombinedBehaviour* element. Exceptional behaviour can be either of the kind 'default' or 'interrupt'. A default exceptional behaviour defines a behaviour that is executed once if one of its blocks becomes executable and terminates the combined behaviour it is contained in. An interrupt exceptional behaviour defines behaviour that is executed if one of its blocks becomes executable and returns execution afterwards to the combined behaviour it is contained in.

The Periodic element is behaviour that consists of one to many blocks and is optionally contained within a *CombinedBehaviour* element. A periodic behaviour defines behaviour that is executed periodically in parallel to the enclosing combined behaviour. The attribute 'period' specifies the frequency of the execution. The *period* attribute is a time expression describing the cycle time for this behaviour.

Abstract Syntax

[Not yet available.]

Semantics

- **Default exceptional behaviour**: If a receive event occurs during execution that matches the first event of one of the defined blocks, execution of the enclosing combined behaviour continues with the execution of the enabled block. After this block is executed the behaviour that follows the enclosing combined behaviour is executed.

- **Interrupt exceptional behaviour**: If a receive event occurs during execution that matches the first event of one of the defined blocks, execution of the enclosing combined behaviour continues with the execution of the enabled block. After this block is executed the behaviour of the enclosing combined behaviour that follows after its interruption is executed.

- **Periodic behaviour**: Independent from the execution of the enclosing combined behaviour, the periodic behaviour is repeatedly executed as long as the enclosing combined behaviour is executed. The repetition period (cycle time) is defined in the *period* attribute.

Constraints

[Not yet available.]

Classifier description

[Not yet available.]

9 Time

TDL provides a means to observe a global time and to specify time constraints on expected or observed behaviour.

[**Editors note:** The following is taken from the Feature list document:

TDL models assumes a global clock with a tick function that increases the value of the clock (discrete time); clock values can be of type Integer or Float of arbitrary precision; no assumptions are made about the time scale (e.g. sec, msec, usec).

Shall this refined data type model be used for time in the current TDL version? (the current TDL meta-model does not reflect this refined data type model.)]

In the current version of TDL time specifications are represented as literals.

9.1 Time Observations

Overview

A *TimeObservation* is a time stamp associated to a *GateEvent*. It can either be an observed time value, e.g. "@5" or a reference to an observation operation storing the observed time when the associated *GateEvent* occurred, e.g. "@T1". In the latter case the time observation may be referred to in a time constraint expression.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

None.

Classifier description

Generalizations:

- TimeSpecification

Attributes:

- expression: String
  The literal representing the time observation

9.2 Time Constraints

Overview

A *TimeConstraint* is an expression. It can be used to represent a timing requirement on a behaviour, e.g. the reception of a message before a specified time, a tester shall wait a specified amount of time before executing a next interaction, or quiescence.

Abstract syntax

[Not yet available.]

Semantics

[Not yet available.]

Constraints

21) A time constraint expression may refer to any number of time observations.

Classifier description

Generalizations:

- TimeSpecification

Attributes:

- expression : String]

10 Miscellaneous

10.1 Comments

Overview

A comment is an element that provides a non-empty value attribute. Any element except a comment itself can contain any number of comments. Comments can be attached to elements for documentation or other informative purposes. The contents of comments shall not be used for adding semantics to elements.

Abstract Syntax

[Not yet available.]

Semantics

None.

Constraints

22) A comment canot contain another comment. [**Editor note**: These may fall under guidelines]

23) The contents of comments shall not be used for adding semantics to elements.

24) A concrete textual syntax implementation is needed to ensure that comments are properly attached to the corresponding owner elements and that this relationship is maintained in case of modifications.

Classifier Description

Generalizations:

- Element

Attributes:

- value : String
  The comment content body

Associations:

- owner : Element
  Owning element to which the comment is attached

10.2 Annotations

Overview

Annotations allow to attach user- or tool-defined semantics to element instances of a TDL specification.

An *Annotation* is an element that provides pair of a key attribute and an optionally empty value attribute. Any element except an annotation itself can contain any number of annotations. In contrast to comments, annotations can be attached to elements to provide added semantics and user-defined extensions to TDL which may be interpreted by tooling. The annotation concept is similar to the tag (or rather tagged value) concept in UML.

[**Editor note**: Currently there are no standardized predefined key attributes of annotations, but some standardized predefined key attributes may be added in the finalized draft for TDL v1.0 or in future versions of TDL.to supply light-weight extensions.]

[**Editor note**: Scope of application and overwriting of annotations is not yet defined.]

Abstract Syntax

[Not yet available.]

Semantics

An annotation provides additional, structured information to an instance of a meta-model element that can be accessed and properly interpreted by a tool processing a TDL specification.

Constraints

25) An annotation cannot contain another annotation.

Classifier Description

Generalizations:

- Element

Attributes:

- key : String
  User defined key attribute
- value : String [0..1]
  Optionally empty value mapped to the key

Associations:

- annotatedElement : Element
  Element to which the annotation is attached

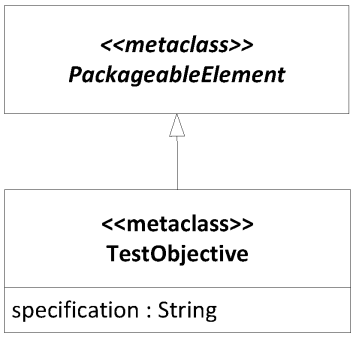
10.3 Test Objectives

Overview

A *TestObjective* is a packageable element that contains the specification of a test objective as unstructured free text. Test objectives can be referenced by test descriptions (e.g. for traceability reasons). Test objectives are typically system requirements, test purposes, or system use cases. The specification attribute of a test objective may contain a reference to an external document (e.g. clause reference in a requirement specification document) or a verbatim copy of content from an external document (e.g. full text of a clause from a requirement specification document).

[**Editor note**: More refined structuring is possible by means of annotations, or a refinement of the test objective concept itself (if required and if a standardized structure can be agreed on).]

Abstract Syntax



Semantics

None.

Constraints

None.

Classifier Description

Generalizations:

- PackageableElement

Attributes:

- specification : String

Annex A (normative): TDL Meta-Model

This annex describes the meta-model of a TDL specification. The meta-model also commonly refers to the abstract syntax of a language.

Annex B (normative): TDL Concrete Syntaxes

Concrete syntaxes to be used by TDL end users to write TDL specifications shall be based on the TDL meta-model (abstract syntax) and can be created according to individual needs. This annex suggests two possible concrete syntaxes: a textual one and a graphical, UML-like syntax.

B.1 TDL Textual Syntax

B.2 TDL Graphical Syntax

Annex C (informative): Examples of TDL Specifications

This annex serves only illustrative purpose to demonstrate how test descriptions can be specified using the concepts and proposed syntaxes of TDL. It might be deleted for the final draft standard on TDL.

C.1 Example: 3GPP

[to be provided]

C.2 Example: IMS

[to be provided]

ABOVE: LINKS TO EXTERNAL FILES FOR THE VARIOUS CLAUSES

History

This clause shall be the last one in the document and list the main phases (all additional information will be removed at the publication stage).

Document history		
V0.0.1	2013-04-29	Submission for MTS review