ETSI ES 203 119 V1.1.1 (2014-04)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Specification of the Abstract Syntax and Associated Semantics

Reference
DES/MTS-140_TDL

Keywords
language, MBT, methodology, 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

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

1 Scope

The present document specifies the abstract syntax of the Test Description Language (TDL) in the form of a meta-model based on the OMG Meta Object Facility (MOF) [1] and also specifies the semantics of the individual elements of the TDL meta-model. The intended use of the present document is to serve as the basis for the development of TDL concrete syntaxes aimed at TDL users and enable TDL tools such as documentation generators, specification analyzers, and code generators.

The specification of concrete syntaxes for TDL is outside the scope of the present document. However, for illustrative purposes, an example of a possible textual syntax together with its application on some existing ETSI test descriptions are provided.

2 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 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

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

[1] "OMG Meta Object Facility (MOF) Core Specification V2.4.1", formal/2013-06-01.

NOTE: Available at http://www.omg.org/spec/MOF/2.4.1/.

[2] "OMG Unified Modeling Language™ (OMG UML) Superstructure, Version 2.4.1", formal/2011-08-06.

[3] ISO/IEC 9646-1:1994: "Information technology - Open Systems Interconnection -- Conformance testing methodology and framework -- Part 1: General concepts".

2.2 Informative references

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 ES 201 873-1 (V4.5.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[i.2] ETSI TS 136 523-1 (V10.2.0): "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 1: Protocol conformance specification (3GPP TS 36.523-1 version 10.2.0 Release 10)".

[i.3] ETSI TS 186 011-2: "Technical Committee for IMS Network Testing (INT); IMS NNI Interoperability Test Specifications; Part 2: Test descriptions for IMS NNI Interoperability".

3 Definitions and abbreviations

3.1 Definitions

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

abstract syntax: graph structure representing a TDL specification in an independent form of any particular encoding

NOTE: The TDL abstract syntax is defined in terms of the TDL meta-model.

action: any procedure carried out by a component of a test configuration or an actor during test execution that could result in changes to the test verdict

actor: abstraction of entities outside a test configuration that interact directly with the components of that test configuration

component: active element of a test configuration that is either in the role tester or system under test

concrete syntax: particular representation of a TDL specification, encoded in a textual, graphical, tabular or any other format suitable for the users of this language

interaction: any form of communication between components that is accompanied with an exchange of data

NOTE: An interaction can be a point-to-point or a point-to multipoint communication.

meta-model: modelling elements representing the abstract syntax of a language

System Under Test (SUT): role of a component within a test configuration whose behaviour is validated when executing a test description

TDL model: instance of the TDL meta-model

TDL specification: representation of a TDL model given in a concrete syntax

test configuration: specification of a set of components that contains at least one tester component and one system under test component plus their interconnections via gates and connections

test description: specification of test behaviour that runs on a given test configuration

test verdict: Result from executing a test description [3].

tester: role of a component within a test configuration that controls the execution of a test description against the components in the role system under test

3.2 Abbreviations

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

EBNF	Extended Backus-Naur Form
IMS	IP Multimedia Subsystem
MBT	Model-Based Testing
MOF	Meta-Object Facility
SUT	System Under Test
TDD	Test Driven Development
TDL	Test Description Language
TTCN-3	Testing and Test Control Notation version 3
UML	Unified Modelling Language
URI	Unified Resource Identifier
UTP	UML Testing Profile

4 Basic Principles

4.1 What is TDL?

TDL is a language that supports the design and documentation of formal test descriptions that can be the basis for the implementation of executable tests in a given test framework, such as TTCN-3 [i.1]. Application areas of TDL that will benefit from this homogeneous approach to the test design phase include:

- Manual design of test descriptions from a test purpose specification, user stories in test driven development (TDD) or other sources.
- Representation of test descriptions derived from other sources such as MBT test generation tools, system simulators, or test execution traces from test runs.

TDL supports the design of black-box tests for distributed, concurrent real-time systems. It is applicable to a wide range of tests including conformance tests, interoperability tests, tests of real-time properties and security tests based on attack traces.

Being a formal notation, TDL clearly separates the specification of tests from their implementation by providing an abstraction level that lets users of TDL focus on the task of describing tests that cover the given test objectives rather than getting involved in implementing these tests to ensure their fault detection capabilities onto an execution framework.

TDL is designed to support different abstraction levels of test specifications. On one hand, the concrete syntax of the TDL meta-model can hide meta-model elements that are not needed for a declarative (more abstract) style of specifying test descriptions. For example, a declarative test description could work with the time operations *wait* and *quiescence* instead of explicit timers and operations on timers (see clause 9).

On the other hand, an imperative (less abstract or refined) style of a test description supported by a dedicated concrete syntax could provide additional means necessary to derive executable test descriptions from declarative test descriptions. For example, an imperative test description could include timers and timer operations necessary to implement the reception of SUT output at a tester component and further details. It is expected that most details of a refined, imperative test description can be generated automatically from a declarative test description. Supporting different shades of abstraction by a single TDL meta-model offers the possibility of working within a single language and using the same tools, simplifying the test development process that way.

4.2 Applicability of the present document

The TDL language design is centered around the three separate concepts of *abstract syntax*, *concrete syntax*, and *semantics* (see figure 4.1). The present document covers the TDL abstract syntax given as the TDL meta-model and its associated semantics.

The TDL concrete syntax is application or domain specific and is not specified in the present document. However, for information, see annex B for an example of a concrete textual syntax.

The semantics of the meta-model elements are captured in the individual clauses describing the meta-model elements defined in the present document.

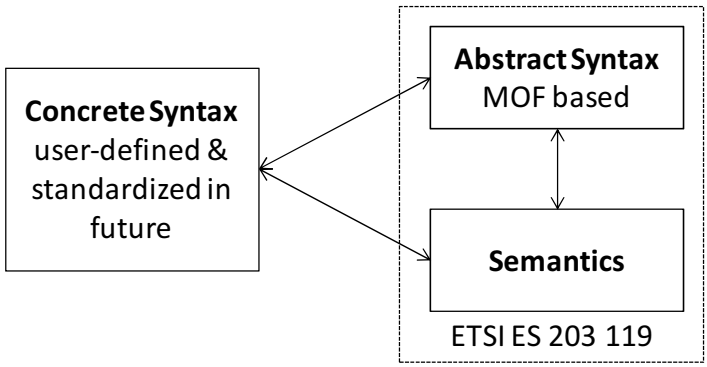

Figure 4.1: TDL language concepts

The abstract syntax (TDL meta-model) and semantics defined in the present document serve as the basis for the development of TDL tools such as editors for TDL specifications in graphical, textual or other forms of concrete syntaxes, analyzers of TDL specifications that check the consistency of TDL specifications, test documentation generators that retransform a TDL specification into another concrete syntax that is more appealing to some stakeholders, and (test) code generators to derive executable tests.

4.3 Design Considerations

TDL makes a clear distinction between concrete syntax that is adjustable to different application domains and a common abstract syntax, which a concrete syntax is mapped to (an example concrete syntax can be found in annex B). The definition of the abstract syntax for a TDL specification plays the key role in offering interchangeability and unambiguous semantics of test descriptions. It is defined in this TDL standard in terms of a MOF meta-model.

A TDL specification consists of the following major parts that are also reflected in the meta-model:

- A test configuration consisting of at least one tester and at least one SUT component and connections among them reflecting the test environment.
- A set of test descriptions, each of them describing one test scenario based on interactions between the components of a given test configuration and actions of components or actors. The control flow of a test description is expressed in terms of sequential, alternative, parallel, iterative, etc. behaviour.
- A set of data instances grouped into data sets that are used in interactions and as parameters of test description invocations.
- Behavioural elements used in test descriptions that operate on time.

Using these major ingredients, a TDL specification is abstract in the following sense:

- Interactions between tester and SUT components of a test configuration are considered to be atomic and not detailed further. For example, an interaction can represent a message exchange, a remote function/procedure call, or a shared variable access.
- All behavioural elements within a test description are totally ordered, unless it is specified otherwise. That is, there is an implicit synchronization mechanism assumed to exist between the components of a test configuration.
- The behaviour of a test description represents the expected, foreseen behaviour of a test scenario assuming an implicit test verdict mechanism, if it is not specified otherwise. If the specified behaviour of a test description is executed, the 'pass' test verdict is assumed. Any deviation from this expected behaviour is considered to be a failure of the SUT, therefore the 'fail' verdict is assumed. There is a possibility for explicit verdict assignment if in a certain case there is a need to override this implicit verdict setting mechanism (e.g. to assign 'inconclusive' or any user-defined verdict values). However, there is no assumption about verdict arbitration, which is implementation-specific.
- The data exchanged via interactions and used in parameters of test descriptions are represented as name tuples without further details of their underlying semantics, which is implementation-specific.

A TDL specification represents a closed system of tester and SUT components. That is, each interaction of a test description refers to one source component and at least one target component that are part of the underlying test configuration a test description runs on. The actions of the actors (entities of the environment of the given test configuration) can be indicated in an informal way.

Time in TDL is considered to be global and progresses in discrete quantities of arbitrary granularity. Progress in time is expressed as a monotonically increasing function. Time starts with the execution of an unreferenced ('base') test description.

TDL can be extended with tool, application, or framework specific information by use of annotations.

4.4 Document Structure

The present document defines the TDL abstract syntax expressed as a MOF meta-model. The TDL meta-model offers language features to express:

- Fundamental concepts such as structuring of TDL specifications and tracing of test objectives to test descriptions (clause 5).
- Abstract representations of data used in test descriptions (clause 6).
- Test configurations, on which test descriptions are executed (clause 7).
- A number of behavioural operations to specify the control flow of test descriptions (clause 8).
- Concepts of time, time constraints over behavioural elements of a test description, and timers; as well as their related operations (clause 9).
- Predefined values for verdicts and time units that can be extended by a user (clause 10).

4.5 Notational Conventions

In the present document, the following notational conventions are applied:

'element'	The name of an element or of the property of an element from the meta-model, e.g. the name of a meta-class.
«metaclass»	Indicates an element of the meta-model, which corresponds to a node of the abstract syntax, i.e. an intermediate node if the element name is put in *italic* or a terminal node if given in plain text.
«Enumeration»	Denotes an enumeration type.
/name	The value with this name of a property or relation is derived from other sources within the meta-model.
[1]	Multiplicity of 1, i.e. there exists exactly one element of the property or relation.
[0..1]	Multiplicity of 0 or 1, i.e. there exists an optional element of the property or relation.
[*] or [0..*]	Multiplicity of 0 to many, i.e. there exists a possibly empty set of elements of the property or relation.
[1..*]	Multiplicity of one to many, i.e. there exists a non-empty set of elements of the property or relation.
{unique}	All elements contained in a set of elements shall be unique.
{ordered}	All elements contained in a set of elements shall be ordered, i.e. the elements form a list.

Furthermore, the definitions and notations from the MOF 2 core framework [1] and the UML class diagram definition [2] apply.

4.6 Conformance

For an implementation claiming to conform to this version of the TDL meta-model, all features specified in the present document shall be implemented consistently with the requirements given in the present document. The electronic attachement in annex A can serve as a starting point for a TDL meta-model implementation conforming to the present document.

5 Foundation

5.1 Overview

The 'Foundation' package specifies the fundamental concepts of the TDL meta-model. All other features of the TDL meta-model rely on the concepts defined in this 'Foundation' package.

5.2 Abstract Syntax

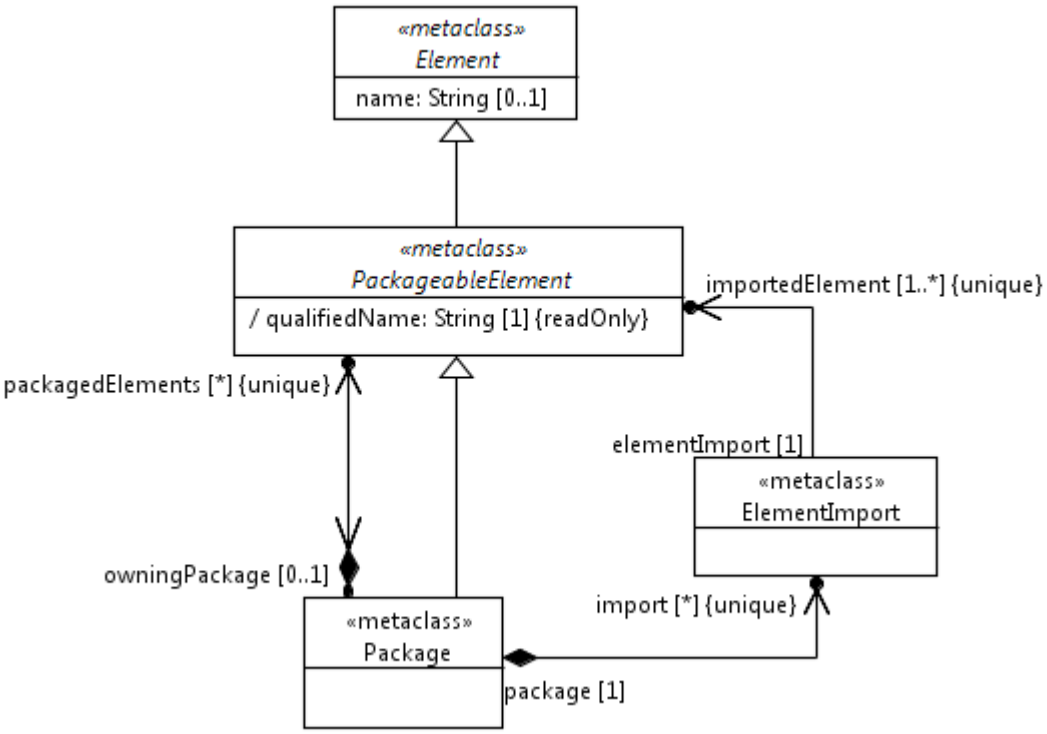

Figure 5.1: Foundational Language Concepts

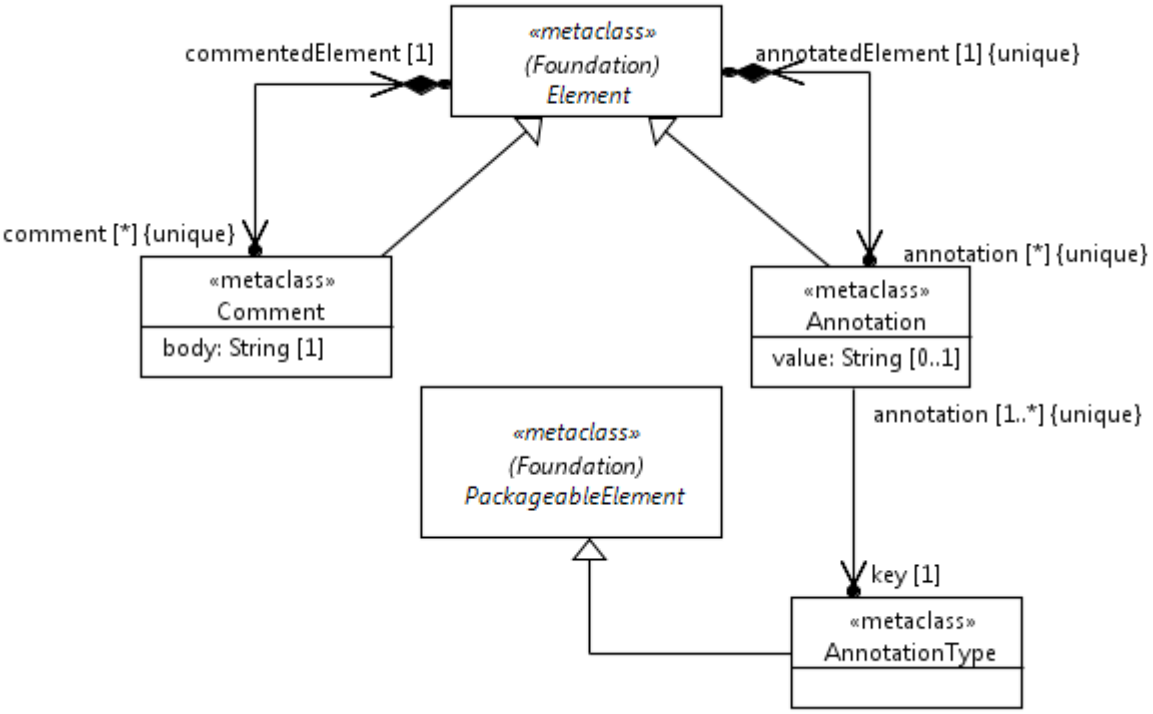

Figure 5.2: Comments and Annotations

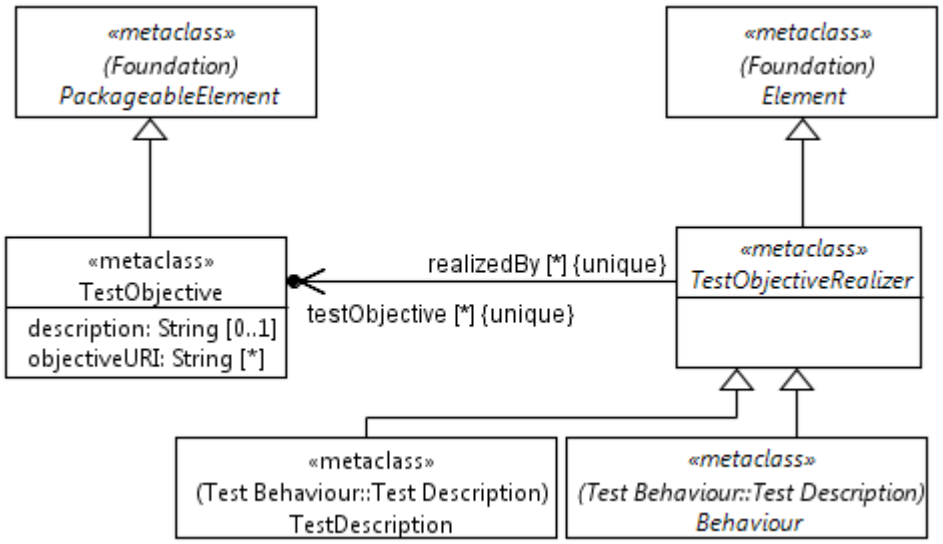

Figure 5.3: Test Objective Concepts

5.3 Classifier Description

5.3.1 Element

Semantics

An 'Element' is a constituent of a TDL Specification. It is the super-class of all other classes in the meta-model. It provides the ability to add comments and annotation to each 'Element'.

Generalizations

There are no generalizations specified.

Properties

- name : String [0..1]
  The optional name of the 'Element'. The 'name' can contain any character, including white-spaces. Having no name specified is different from the empty name (which is represented by an empty string).

- comment : Comment [0..*] {unique}
  A possibly empty set of 'Comment's attached to the 'Element'.

- annotation : AnnotationType [0..*] {unique}
  A possibly empty set of 'Annotation's attached to the 'Element'.

Constraints

There are no constraints specified.

5.3.2 PackageableElement

Semantics

A 'PackageableElement' is contained in a 'Package'. It has a mandatory 'qualifiedName' that shall be unique throughout the owning and any imported 'Package'. The 'qualifiedName' distinguishes equally named 'PackageableElement's of different 'Package's from each other. This makes it possible that a 'PackageableElement' with the same name can be imported into a 'Package' without causing a name clash.

The 'qualifiedName' is a compound name derived from the directly and all indirectly enclosing parent 'Package's by concatenating the names of each 'Package'. As a separator between the segments of a 'qualifiedName' the string '::' shall be used. The name of the root 'Package' that (transitively) owns the 'PackageableElement' shall always constitute the first segment of the 'qualifiedName'.

The visibility of a 'PackageableElement' is restricted to the 'Package' in which it is directly contained. A 'PackageableElement' may be imported into other 'Package's by using 'ElementImport'. A 'PackageableElement' has no means to actively increase its visibility.

Generalizations

- Element

Properties

- /qualifiedName : String [1] {read-only}
  A derived property that represents the unique name of a package within a TDL model.

- owningPackage : Package [0..1]
  The 'Package' that owns the 'PackageableElement'.

Constraints

Distinguishable qualified names
All qualified names shall be distinguishable within a TDL model.

5.3.3 Package

Semantics

A 'Package' represents a container for 'PackageableElement's. A TDL model may contain any number of 'Package's, which may have any arbitrary substructure. A 'Package' may contain any number of 'PackageableElement's.

A 'Package' builds a scope of visibility for its contained 'PackageableElement's. A 'PackageableElement' is only accessible within its owning 'Package' and within any 'Package' that directly import it.

A 'Package' may import any 'PackageableElement' from any other 'Package' with the means of 'ElementImport'. By importing a 'PackageableElement', the imported 'PackageableElement' becomes visible and accessible within the importing 'Package'. Cyclic imports of packages are not permitted.

Generalizations

- PackageableElement

Properties

- packagedElements : PackageableElements [0..*] {unique}
  The 'PackageableElement's that are directly contained in the 'Package'.

- import : ElementImport [0..*] {unique}
  The list of import declarations.

Constraints

No cyclic imports
Cyclic imports are not allowed. That is, a package shall not import itself directly or indirectly via other packages.

5.3.4 ElementImport

Semantics

An 'ElementImport' allows importing 'PackageableElement's from other 'Package's into the scope of an importing 'Package'. By establishing an import, the imported 'PackageableElement's become accessible within the importing 'Package'.

Only those 'PackageableElement's can be imported via 'ElementImport' that are directly contained in the exporting 'Package', that is, the import is not transitive. If an 'ElementImport' declares the import of a 'Package', all the directly contained 'PackageableElement's in that imported 'Package' become accessible within the importing 'Package'.

Generalizations

- Element

Properties

- importedElement : PackageableElement [1..*] {unique}
  The 'PackageableElement's that are imported into the context package via this 'ElementImport'.

Constraints

Consistency of imported elements
The imported 'PackageableElement' shall be directly owned by the exporting 'Package'.

5.3.5 Comment

Semantics

'Comment's may be attached to elements for documentation or for other informative purposes. Any 'Element' except for a 'Comment' may contain any number of 'Comment's. The contents of comments shall not be used for adding semantics to elements.

Generalizations

- Element

Properties

- commentedElement : Element [1]
  The 'Element' to which the 'Comment' is attached.
- body : String [1]
  The content of the 'Comment'.

Constraints

There are no constraints specified.

5.3.6 Annotation

Semantics

An 'Annotation' is a means to attach user- or tool-defined semantics to any 'Element' of a TDL model, except 'Comment' and 'Annotation' itself. An 'Annotation' provides a pair of a 'key'/'value' attributes, whereas the 'value' might be empty.

Generalizations

- Element

Properties

- annotatedElement : Element [1]
  The 'Element' to which the 'Annotation' is attached.
- key : AnnotationType [1]
  Reference to a user-defined 'AnnotationType'.
- value : String [0..1]
  The 'value' mapped to the 'key'.

Constraints

There are no constraints specified.

5.3.7 AnnotationType

Semantics

An 'AnnotationType' is a 'PackageableElement' used to define the 'key' attributes of an 'Annotation'.

Generalizations

- PackageableElement

Properties

There are no properties specified.

Constraints

There are no constraints specified.

5.3.8 TestObjective

Semantics

A 'TestObjective' specifies the reason for designing either a 'TestDescription' or a particular 'Behaviour' of a 'TestDescription'. A 'TestObjective' may either contain a 'description' of the objective directly or refer to an external resource for further information about the objective.

The 'description' of a 'TestObjective' should be in natural language, however, it may be provided as structured (i.e. machine-readable) text.

Generalizations

- PackageableElement

Properties

- description : String [0..1]
  A textual description of the 'TestObjective'.

- objectiveURI : String [0..*]
  A set of URIs locating resources that provide further information about the 'TestObjective'. These resources are usually external to a TDL model such as requirements specifications or TPLan artefacts.

Constraints

There are no constraints specified.

5.3.9 TestObjectiveRealizer

Semantics

A 'TestObjectiveRealizer' establishes traces to 'TestObjectives'. The semantics of such a trace are that a 'TestObjectiveRealizer' realizes the objective stated or referred to by the 'TestObjective'.

Generalizations

- Element

Properties

- testObjective : TestObjective [0..*] {unique}
  The set of 'TestObjective's that is realized by the 'TestObjectiveRealizer'.

Constraints

There are no constraints specified.

6 Data

6.1 Overview

The 'Data' package defines the various meta-model elements to represent data within a TDL specification. TDL does not feature a complete data type system. Instead, it relies on parameterizable data instances that are grouped into data sets. Data sets, data proxies, and data instances, denoted data elements in TDL, are abstract representations of corresponding data-related concepts in a concrete type system. Data elements in TDL can be mapped to the corresponding elements in a concrete type system by means of data element mappings, which represent the location of the concrete data definition within a resource, e.g. the concrete element name. The resource in which the mapped concrete type system element is located is represented by a data resource mapping which defines a resource where the concrete data definition is located, e.g. a file name. In addition to static data defined in files or other resources, data elements can also be mapped to dynamic data which is only available at runtime, e.g. a data instance can represent a session ID. The execution environment shall provide an interface to such dynamic data by means of a runtime element URI. There are no specific restrictions on the mapping imposed by TDL. In particular, a TDL data set could also map to several concrete data types.

Since data sets are packageable elements, they can be reused in other TDL specifications. Data element specifications shall be complete up to the level that all data sets and data instances used in a test description, as well as in data instance definitions are defined and accessible.

EXAMPLE 1: Consider the data set STATUS of data instances 200, 4xx, 5xx, and RetVal. A realization of these data definitions in a concrete data type system could be defined such that STATUS maps to the Java type int, while data instance 200 maps to the Java int value 200 and 4xx and 5xx map to a range of int values 400..404 and 500..502, respectively. The data instance RetVal could be mapped to a Java int variable of the same name.

Data instances may be related to other data instances by means of parameters. Data instance parameters can be either other data instances or data sets. In the latter case, a data set parameter serves as a wildcard referring to any of the data instances that are grouped into that data set.

EXAMPLE 2: Consider the following data set STATUS of data instances 200, 4xx, 5xx and data set HTTP-MSG of data instances HttpRequest and HttpResponse(STATUS). The latter is a shorthand notation for the data instances HttpResponse(200), HttpResponse(4xx), and HttpResponse(5xx).

TDL does not support variables. Data instances that are mapped to a runtime URI cover the basic scenarios where a concrete data instance that is only available at runtime needs to be shared among several interactions. In order to express formal parameters of a test description, TDL provides the data proxy construct. A data proxy, which is associated to a data set to determine the set of allowed data elements that it can be bound to, acts as a placeholder in the test description and can be used as the argument of interactions, as a parameter for the arguments of interactions, or as an actual parameter in a test description reference. In the operational semantics, this data proxy is bound to the data element (data instance, data set, or a data proxy) specified as the actual parameter of a reference to this test description.

6.2 Abstract Syntax

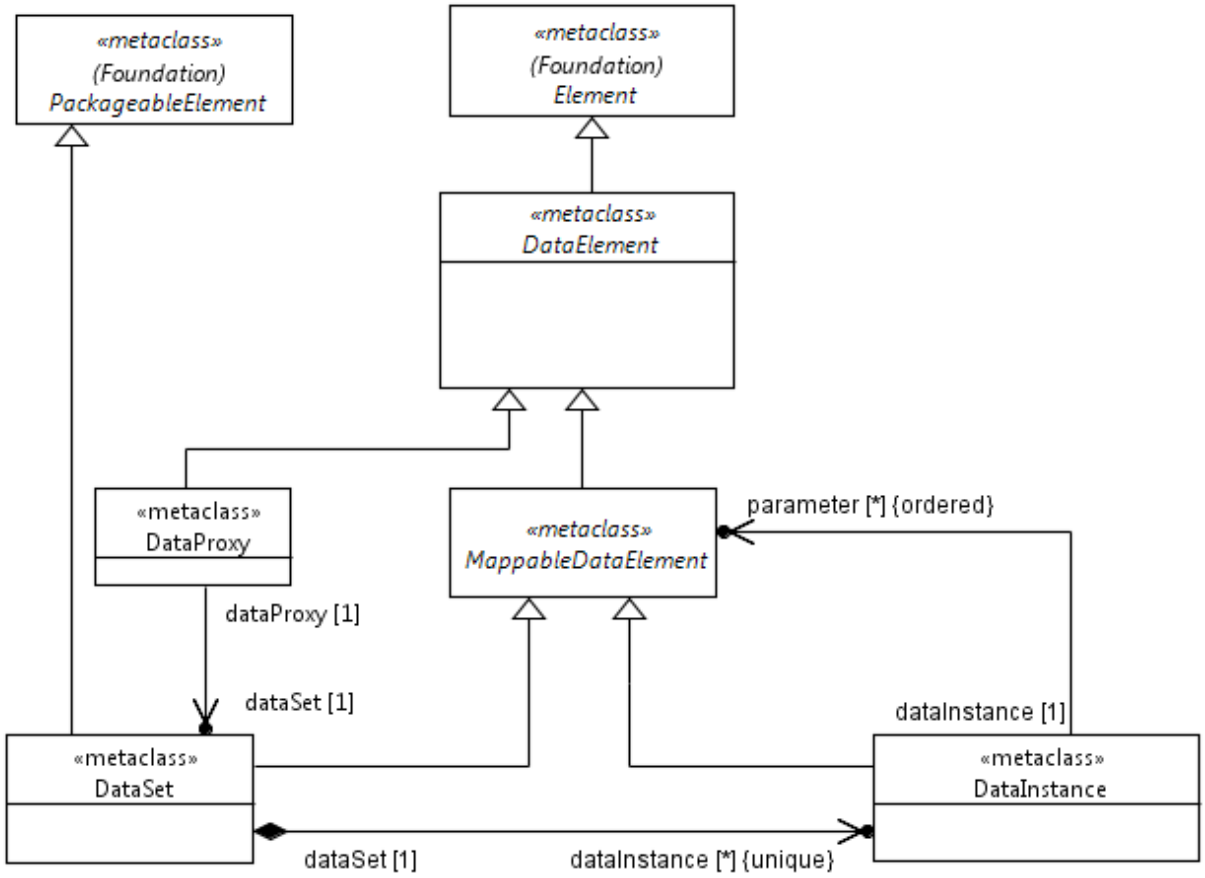

Figure 6.1: Basic Data Concepts

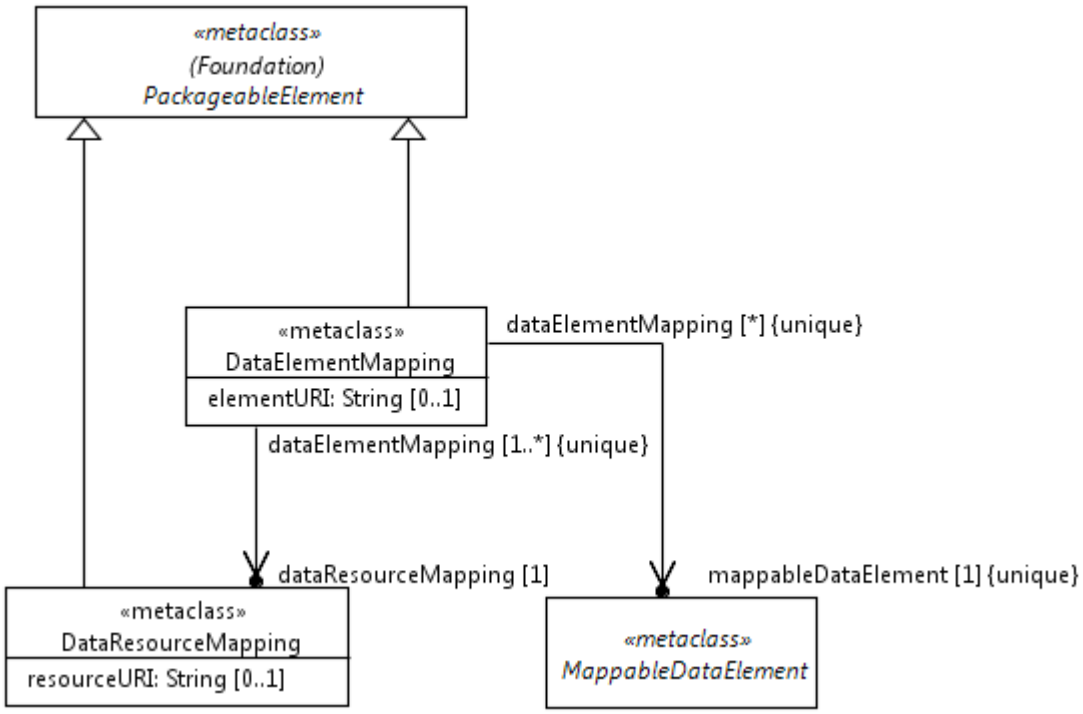

Figure 6.2: Data Mapping Concepts

6.3 Classifier Description

6.3.1 DataElement

Semantics

A 'DataElement' is an abstraction of 'MappableDataElement' and 'DataProxy' used to express data in TDL specifications.

Generalizations

- Element

Properties

There are no properties specified.

Constraints

There are no constraints specified.

6.3.2 MappableDataElement

Semantics

A 'MappableDataElement' is an abstraction of 'DataSet' and 'DataInstance' used to express data elements that can be referenced as parameters of data instance definitions in TDL specifications.

Generalizations

- DataElement

Properties

There are no properties specified.

Constraints

There are no constraints specified.

6.3.3 DataSet

Semantics

A 'DataSet' is a 'MappableDataElement' and by extension a 'DataElement, and a 'PackageableElement', that contains a set of 'DataInstance' elements. It may correspond to and be mapped to a concrete data type specification outside of TDL. Mapping is done by means of a 'DataMappingElement'.

Generalizations

- PackageableElement
- MappableDataElement

Properties

- dataInstance : DataInstance [0..*] {unique}
  The set of contained data instances.

Constraints

There are no constraints specified.

6.3.4 DataInstance

Semantics

A 'DataInstance' is a 'MappableDataElement' contained in a 'DataSet' element. It is linked to a data set and may reference other data elements as parameters. This provides basic data composition and structuring facilities in TDL. The exact data composition mechanism is implementation specific. A TDL data instance with parameters only implies that the provided parameters are related in some way or are a part of the concrete type system data element. Data instances defined with parameters shall always be referenced with the same number of parameters when used as the arguments of interactions and as actual parameters in test description references.

A data instance may be mapped to a concrete data value by means of a 'DataElementMapping'.

Generalizations

- MappableDataElement

Properties

- parameter : MappableDataElement [0..*] {ordered}
  References to further mappable data elements which are attached to the given data instance.

Constraints

No self-reference
The same 'DataInstance' element shall not occur as a reference in its data instance parameter set, i.e. no recursive structure shall be defined.

6.3.5 DataProxy

Semantics

A 'DataProxy' is a 'DataElement' that serves as a placeholder for instances of 'DataInstance', 'DataSet', or 'DataProxy' in the definition of formal parameters of 'TestDescription' elements (see clause 8.2.1). 'DataProxy' elements defined as formal parameters of a 'TestDescription' element are bound to the 'DataInstance', 'DataSet', or 'DataProxy' elements specified as actual parameters of a 'TestDescriptionReference' element referencing the 'TestDescription' element.

Generalizations

- DataElement

Properties

- dataset : DataSet [1]
  References the data set of acceptable data instances, to which the 'DataProxy' element can be bound in actual test description parameters. The data set itself as well as 'DataProxy' elements having the same data set can also be bound to the 'DataProxy' element in actual test description parameters.

Constraint

There are no constraints specified.

6.3.6 DataResourceMapping

Semantics

A 'DataResourceMapping' is a 'PackageableElement'. It defines the resource where the concrete data definitions from an underlying concrete data type system are located as identified by the 'resourceURI' property.

Generalizations

- PackageableElement

Properties

- resourceURI : String [0..1]
  Location of the resource that contains concrete data definitions. The location shall resolve to an unambiguous name.

Constraints

There are no constraints specified.

6.3.7 DataElementMapping

Semantics

A 'DataElementMapping' is a 'PackageableElement'. It represents the location of a single concrete data definition within the resource referred to in the referenced 'DataResourceMapping' element. The location within the resource is described by means of the 'elementURI' property.

Generalizations

- PackageableElement

Properties

- mappableDataElement : MappableDataElement [1] {unique}
  Refers to the mappable data element (data set or data instance) to be mapped to a concrete data definition.
- elementURI : String [0..1]
  Location of a concrete data element within the resource it refers to in the 'DataResourceMapping' element. The location shall resolve to an unambiguous name within the resource.
- dataResourceMapping : DataResourceMapping [1]
  References the data resource that contains the actual data definition of the referred mappable data element.

Constraints

There are no constraints specified.

7 Test Architecture

7.1 Overview

The 'Test Architecture' package describes the elements needed to define a 'TestConfiguration' consisting of tester and SUT components, gates, and their interconnections represented as 'Connection's. A 'TestConfiguration' specifies the structural foundations on which test descriptions can be built upon. The fundamental units of a 'TestConfiguration' are the 'ComponentInstance's. Each 'ComponentInstance' specifies a functional entity of the test system. A 'ComponentInstance' may either be a (part of a) tester or a (part of an) SUT. That is, both the tester and the SUT can be decomposed, if required. The communication exchange between 'ComponentInstance's is established through interconnected 'GateInstance's. To offer reusability, TDL introduces 'ComponentType's and gate types.

7.2 Abstract Syntax

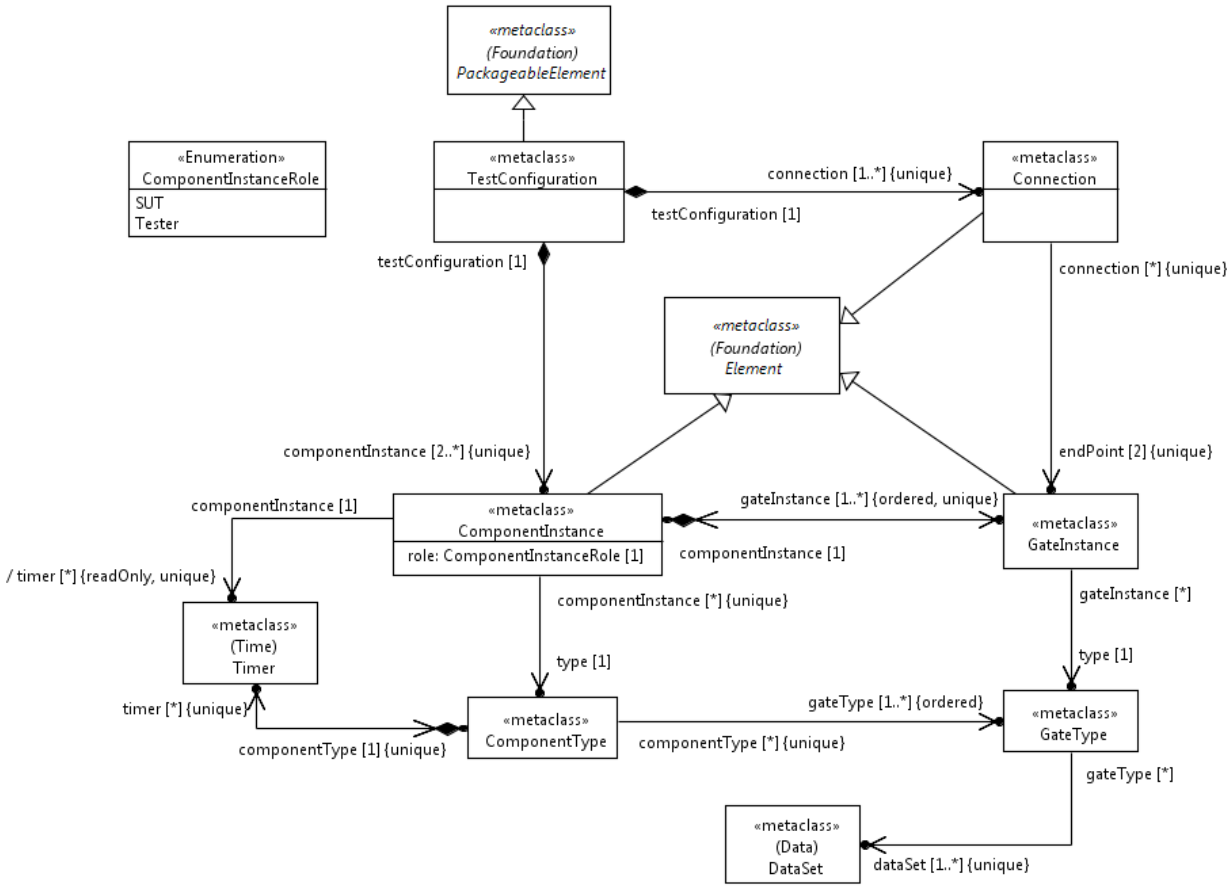

Figure 7.1: Test Architecture Concepts

7.3 Classifier Description

7.3.1 TestConfiguration

Semantics

A 'TestConfiguration' specifies the communication infrastructure necessary to build 'TestDescription's upon. As such, it contains all the elements required for information exchange, such as 'ComponentInstance's and 'Connection's.

It is not necessary that all 'ComponentInstance's contained in a 'TestConfiguration' are actually connected via 'Connection's but for any 'TestConfiguartion' at least the semantics of a minimal 'TestConfiguration' shall apply.

Generalizations

- PackageableElement

Properties

- componentInstance : ComponentInstance [2..*] {unique}
  The 'ComponentInstance's of the 'TestConfiguration'.

- connection : Connection [1.*] {unique}
  The 'Connection's of the 'TestConfiguration' over which 'Interaction's are sent and received.

Constraints

'TestConfiguration' and components roles
A 'TestConfiguration' shall contain at least one 'Tester' and one 'SUT' 'ComponentInstance'.

Minimal 'TestConfiguration'
Each 'TestConfiguration' shall specify at least one 'Connection' that connects a 'GateInstance' of a 'Tester' 'ComponentInstance' with a 'GateInstance' of an 'SUT' 'ComponentInstance'.

7.3.2 GateType

Semantics

A 'GateType' represents a point of communication for exchanging information between 'ComponentInstance's. A 'GateType' specifies the 'DataSets' (and, thus, the DataProxies and DataInstances that belong to that 'DataSet') that can be exchanged via that 'GateType'.

The same 'GateType' can be shared among multiple 'ComponentType's, thus, allowing for reuse.

Generalizations

- PackageableElement

Properties

- dataset : DataSet [1..*] {unique}
  The 'DataSet's that can be exchanged via that 'GateType'. The arguments of 'Interactions' shall adhere to the 'DataSet' that are allowed to be exchanged via the 'GateType'.

Constraints

There are no constraints specified.

7.3.3 GateInstance

Semantics

A 'GateInstance' represents an instance of a 'GateType'. 'GateInstance's are the means to exchange information by specifying an 'Interaction' and to execute 'TimeOperation's.

A 'GateInstance' may, but need not, be connected by 'Connection'. Additionally, a 'GateInstance' may be the endpoint of more than one 'Connection'.

Generalizations

- Element

Properties

- type : GateType [1]
  The 'GateType' of the 'GateInstance'.
- componentInstance : ComponentInstance [1]
  The 'ComponentInstance' that owns this 'GateInstance'.

Constraints

No additional constraints specified.

7.3.4 ComponentType

Semantics

A 'ComponentType' specifies the type of one or multiple 'ComponentInstance's that participate in a 'TestConfiguration'. A 'ComponentType' refers to at least one 'GateType' and may contain any number of 'Timer's. Instances of 'ComponentType's represent the functional entities of a test system.

An instance of a 'ComponentType' has local copies of the 'Timer's that are listed in the 'ComponentType' definition.

Instances of 'ComponentType's may either act as a 'Tester' or as an 'SUT' within a 'TestConfiguration'. It is possible that the very same 'ComponentType' participates in one 'TestConfiguration' as a 'Tester' and in another 'TestConfiguration' as an 'SUT'.

Generalizations

- PackageableElement

Properties

- gateType : GateType [1..*] {ordered}
  The 'GateType's used for the purpose of attaching 'GateInstance's to a 'ComponentInstance' of that 'ComponentType'.
- timer : Timer [0..*] {unique}
  The 'Timer's owned by a 'ComponentType'.

Constraints

There are no constraints specified.

7.3.5 ComponentInstanceRole

Semantics

'ComponentInstanceRole' specifies the role of a 'ComponentInstance', whether it acts as a 'Tester' or as an 'SUT' component.

Generalizations

There are no generalizations specified.

Literals

- SUT
  The 'ComponentInstance' assumes the role 'SUT' in the enclosing 'TestConfiguration'.
- Tester
  The 'ComponentInstance' assumes the role 'Tester' in the enclosing 'TestConfiguration'.

Constraints

There are no constraints specified.

7.3.6 ComponentInstance

Semantics

A 'ComponentInstance' represents a communication entity of the test system. It acts either as a 'Tester' or as an 'SUT' 'ComponentInstance'. A 'ComponentInstance' contains one or more 'GateInstance's that are the endpoints of 'Connection's. The number and the types of 'GateInstance's contained in a 'ComponentInstance' shall be the same as the number and the types of 'GateType's referred to by the 'ComponentType' this 'ComponentInstance' is an instance of.

The 'Timer's that can be utilized by timer-related 'Behaviour's of a 'TestDescription' are derived from the 'Timer's that the corresponding 'ComponentType's possesses. Thus, only 'Timer's that are known by the 'ComponentInstance's can be used within a 'TestDescription'.

Generalizations

- Element

Properties

- type : ComponentType [1]
  The 'ComponentType' of this 'ComponentInstance'.

- role : ComponentInstanceRole [1]
  The role of the 'ComponentInstance' plays within the given 'TestConfiguration'. It can be either 'Tester' or 'SUT'.

- gateInstance : GateInstance [1..*] {unique, ordered}
  The 'GateInstance's this 'ComponentInstance' contains.

- /timer : Timer [0..*] {unique, read-only}
  The set of timers is derived from the set of timers contained in the 'ComponentType' of the 'ComponentInstance'. That is, a 'ComponentInstance' shall have the same set of timers as specified in the 'ComponentType', which the 'ComponentInstance' belongs to.

Constraints

Named component instances
A component instance shall have a name.

Number of 'GateInstance's in a 'ComponentInstance'
A 'ComponentInstance' shall contain as many 'GateInstance's of corresponding types as the number of gate types declared in the associated 'ComponentType'.

Timers of 'ComponentInstance's
A 'ComponentInstance' shall refer exactly to the same set of timers as contained in the associated 'ComponentType'.

7.3.7 Connection

Semantics

A 'Connection' is a communication channel for exchanging information by specifying an 'Interaction' between 'GateInstance's and, thus, 'ComponentInstance's. 'Connection's do not specify or restrict the nature of the communication channel that is eventually used in an implementation.

'Connection's are always binary. A 'Connection' can be established between 'GateInstance's contained in any kind of 'ComponentInstance's as long as there is at most one 'Connection' between any two 'GateInstance's. The 'GateInstance's connected by a 'Connection' shall not be identical, i.e. self-looped 'Connection's are not permitted.

Multiple 'Connection's can refer to the very same 'GateInstance'. This allows for specifying multicast communication exchange capability.

Generalizations

- Element

Properties

- endPoint : GateInstance [2] {unique}
  The two 'GateInstance's that are the endpoints of this 'Connection'.

Constraints

Self-looped connections not allowed
The two 'GateInstance's (endpoints) of any 'Connection' shall be different from each other, i.e. no self-loop 'Connection' is allowed.

Only one connection allowed
Between any two 'GateInstance's at most one 'Connection' can exist.

8 Test Behaviour

8.1 Overview

The 'Test Behaviour' package defines all elements needed to describe the behaviour of a test description.

8.2 Abstract Syntax

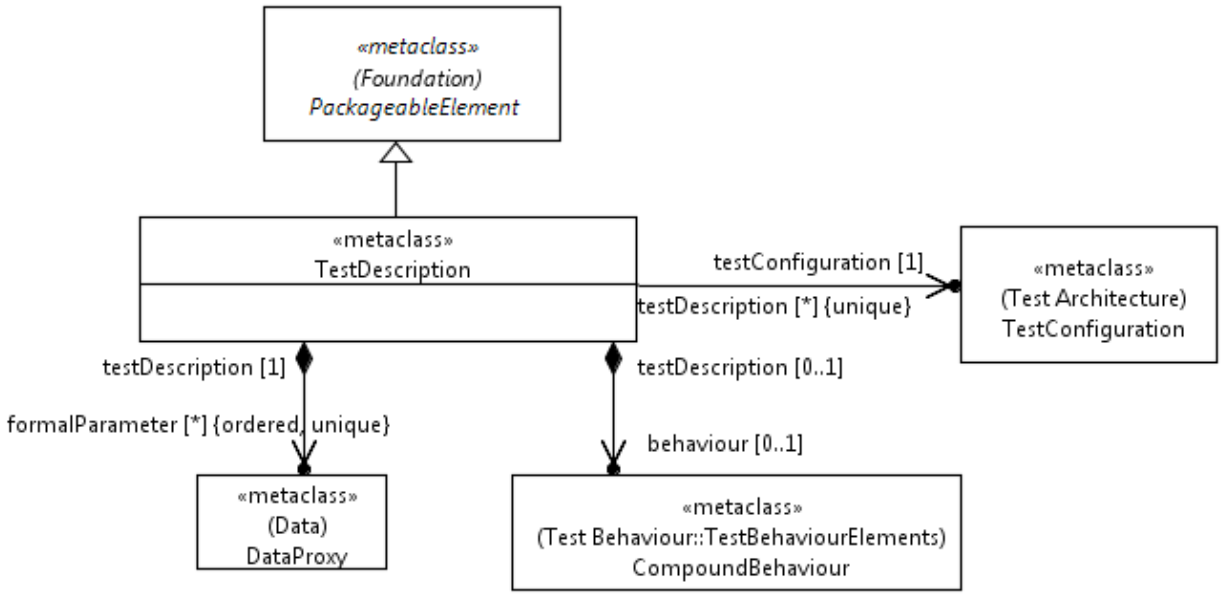

Figure 8.1: Test Description Concepts

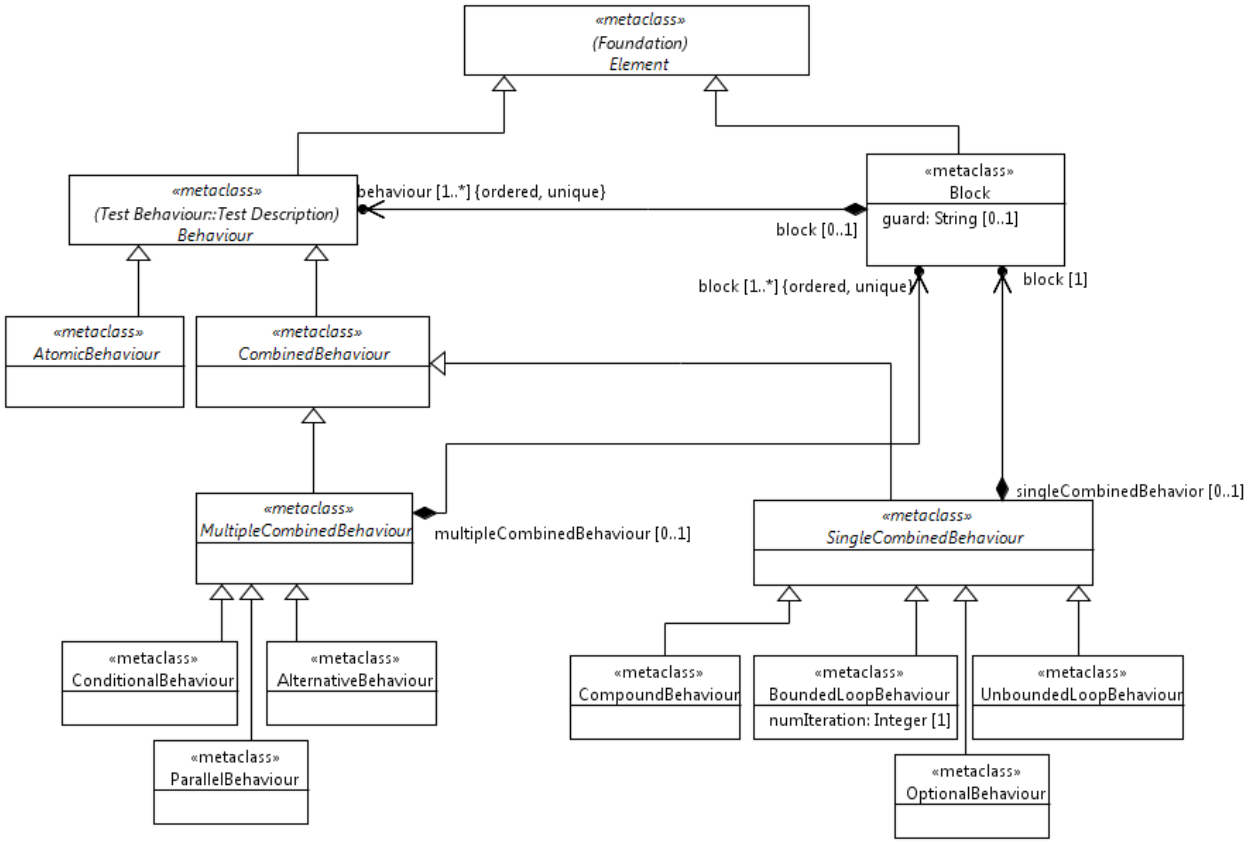

Figure 8.2: Behaviour Concepts

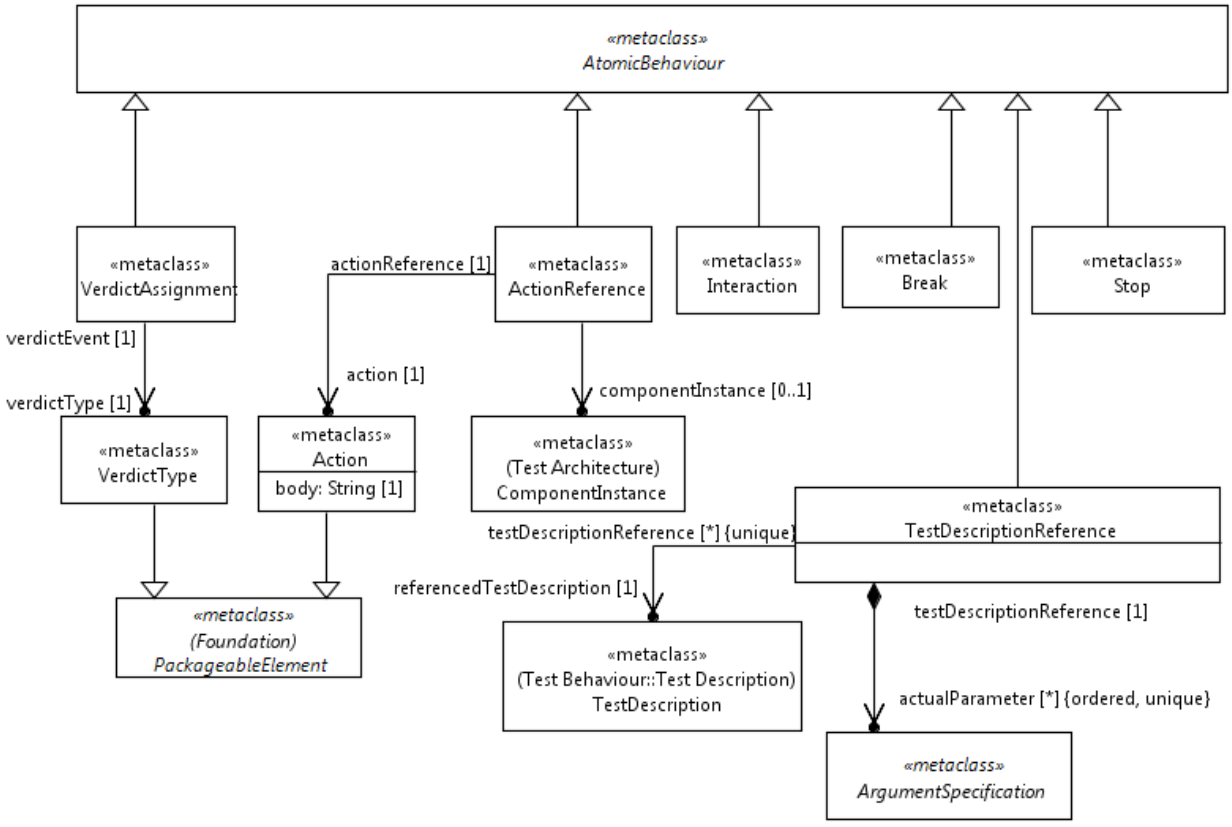

Figure 8.3: Atomic Behaviour Concepts

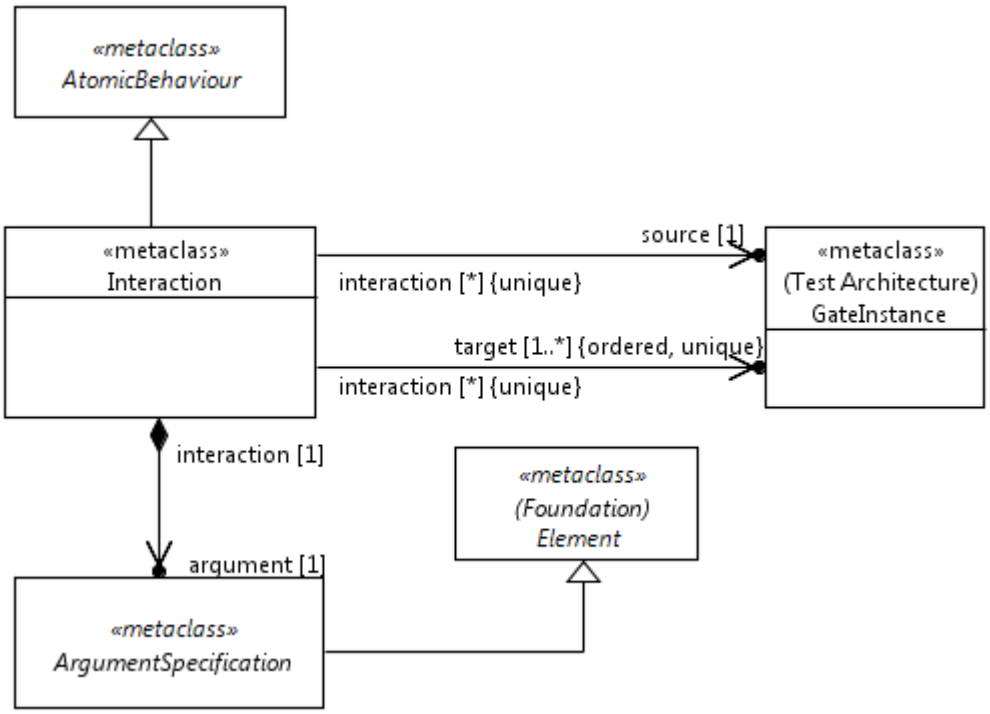

Figure 8.4: Interaction Concepts

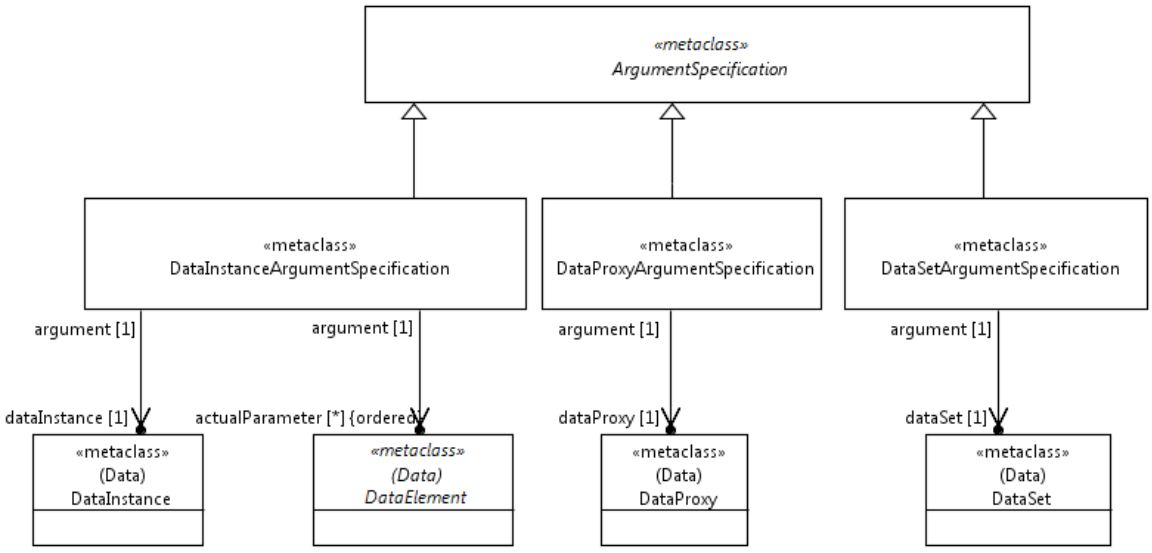

Figure 8.5: Argument Specification Concepts

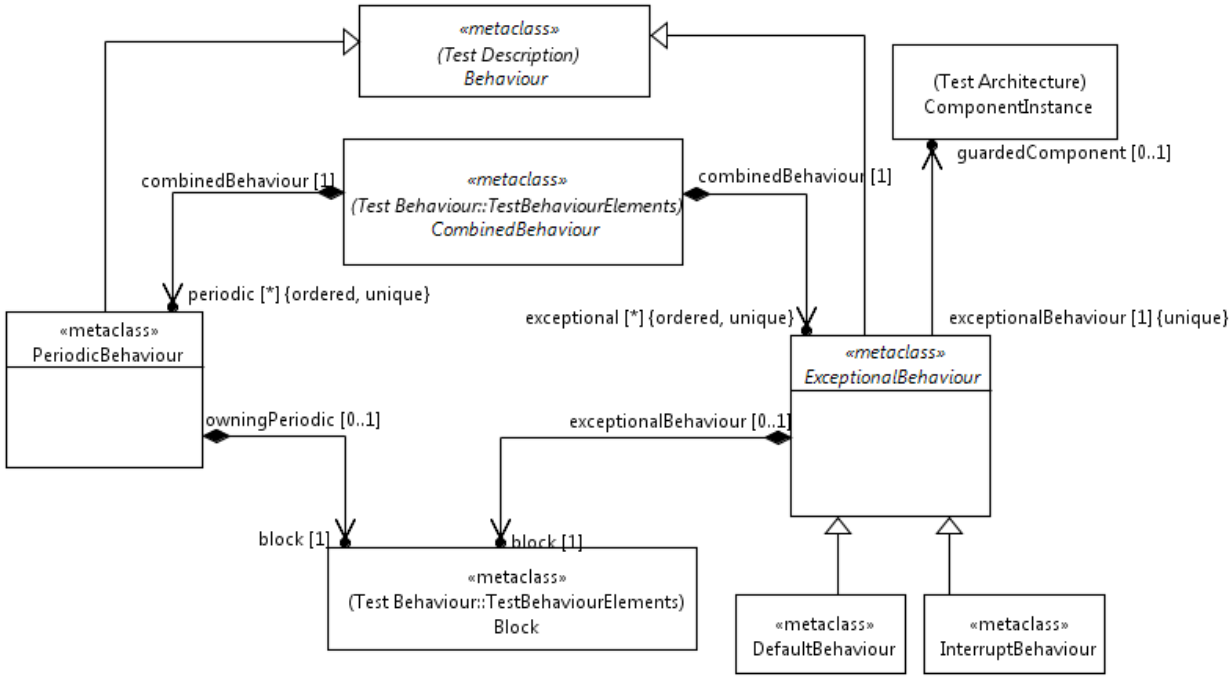

Figure 8.6: Exceptional Behaviour and Periodic Behaviour Concepts

8.3 Classifier Description

8.3.1 TestDescription

Semantics

The 'TestDescription' is a 'PackageableElement' and a 'TestObjectiveRealizer'. A 'TestDescription' defines the test behaviour based eventually on ordered 'AtomicBehaviour' elements. A test description may contain a 'CompoundBehaviour' defining the behaviour of the test description. It may also contain a list of 'DataProxy' elements that act as formal parameters of this test description.

A test description is associated with exactly one 'TestConfiguration' and may be associated with any number of 'DataProxy' elements. These 'DataProxy' elements represents the formal parameters of the 'TestDescription' used for parameterization.

If a test description has formal parameters, it shall be invoked within another test description that is possibly parameter-free in order to be able to provide actual parameters.

Generalizations

- PackageableElement
- TestObjectiveRealizer

Properties

- testConfiguration : TestConfiguration [1]
  The property 'testConfiguration' refers to the 'TestConfiguration' that is associated with the 'TestDescription'.

- behaviour : CompoundBehaviour [0..1]
  The property 'behaviour', if present, defines the actual behaviour of the test description in terms of a 'CompoundBehaviour' element.

- timeConstraint : TimeConstraint [0..*] {unique}
  This property, if present, refers to the expressions that shall be evaluated to Boolean. It is used to express time constraints between behavioural elements within this test description.

- formalParameter : DataProxy [0..*] {ordered}
  This property refers to 'DataProxy' elements. The property, if present, defines the formal parameters that shall be substituted by actual data elements when the 'TestDescription' is executed.

Constraints

Named test description
A test description shall have a name.

8.3.2 Block

Semantics

A 'Block' element is a container for 'Behaviour' elements that are executed in a strictly sequential way. A 'Block' element may have a 'guard' defined. If a 'Block' has a 'guard', it shall only be executed if its guard condition evaluates to true.

Generalizations

- Element

Properties

- behaviour : Behaviour [1..*] {unique, ordered}
  This property is a list of 'Behaviour' elements describing the behaviour of the 'Block' element. The 'Behaviour' elements shall be executed in their definition order.

- guard : String [0..1]
  The 'guard' property is a String that represents a Boolean expression. The 'guard' when present determines if the behaviour of the 'Block' element shall be executed or not. If the 'guard' expression evaluates to true, the behaviour is executed, otherwise not. If a 'Block' element has no 'guard' it is unconditionally executed.

Constraints

Guard shall evaluate to Boolean

The 'guard' property is a String that shall represent a Boolean expression.

8.3.3 Behaviour

Semantics

A 'Behaviour' is a generic, abstract 'Element' that is refined into 'AtomicBehaviour' and 'CombinedBehaviour'. Some 'Behaviour's operate on gate instances or component instances and some are completely independent.

Generalizations

- TestObjectiveRealizer

Properties

There are no properties specified.

Constraints

No additional constraints specified.

8.3.4 CombinedBehaviour

Semantics

A 'CombinedBehaviour' element is a 'Behaviour' that involves all gate instances defined in the associated test configuration. It can contain a single 'Block' element (in case of 'SingleCombinedBehaviour') or a list of ordered and potentially guarded 'Block' elements (in case of 'MultipleCombinedBehaviour'). A 'CombinedBehaviour' may have any number of ordered 'PeriodicBehaviour' and 'ExceptionalBehaviour' elements. The 'PeriodicBehaviour' and the 'ExceptionalBehaviour' elements shall be evaluated in their definition order.

Generalizations

- Behaviour

Properties

- periodic : PeriodicBehaviour [0..*] {unique, ordered}
  A 'CombinedBehaviour' element can contain any number of 'PeriodicBehaviour' elements. The 'periodic' property refers to the list of 'PeriodicBehaviour' elements.

- exceptional : ExceptionalBehaviour [0..*] {unique, ordered}
  A 'CombinedBehaviour' element can contain any number of 'ExceptionalBehaviour' elements. The 'exceptional' property refers to the list of 'ExceptionalBehaviour' elements.

Constraints

There are no constraints specified.

8.3.5 SingleCombinedBehaviour

Semantics

A 'SingleCombinedBehaviour' element is a 'CombinedBehaviour' that contains a single potentially guarded 'Block' element. It can be further refined to a 'CompoundBehaviour', a 'BoundedLoopBehaviour', an 'UnboundedLoopBehaviour', or an 'OptionalBehaviour'.

Generalizations

- CombinedBehaviour

Properties

- block : Block [1]
  The 'block' property refers to a 'Block' element that specifies the behaviour of the 'SingleCombinedBehaviour'.

Constraints

There are no constraints specified.

8.3.6 CompoundBehaviour

Semantics

A 'CompoundBehaviour' element groups any positive number of 'Behaviour' elements together in an ordered list of 'Behaviour' elements contained in a single 'Block' element. The list defines the execution order of the 'Behaviour' elements of the 'CompoundBehaviour'. As a derived 'CombinedBehaviour' element, a 'CompoundBehaviour' element can have also exceptional or periodic behaviour attached to it.

Generalizations

- SingleCombinedBehaviour

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.7 OptionalBehaviour

Semantics

An 'OptionalBehaviour' element is a 'SingleCombinedBehaviour', therefore it shall contain one 'Block' element, which may have a guard condition and shall start with one of the following 'AtomicBehaviour' elements: 'Interaction', 'TimeOut', or 'Quiescence'. For the 'Interaction' element it is required that the target 'GateInstance' shall be associated to a 'ComponentInstance' with the 'role' of 'Tester'.

If a 'Block' has no guard condition, it is equivalent to a 'Block' with a guard condition of true. If the guard condition of the 'Block' element is evaluated to true and the first event of the 'Block' occurs, the 'OptionalBehaviour' will be executed, otherwise the execution continues with the next behaviour after the 'OptionalBehaviour' element.

Generalizations

- SingleCombinedBehaviour

Properties

There are no properties specified.

Constraints

First event allowed
The contained 'Block' element shall start with one of the following 'AtomicBehaviour' elements: 'Interaction' received by a 'Tester' component, 'TimeOut', or 'Quiescence'.

8.3.8 BoundedLoopBehaviour

Semantics

A 'BoundedLoopBehaviour' element is a 'SingleCombinedBehaviour', therefore it shall contain one 'Block' element, which shall not have a guard condition. The 'Block' element shall be executed as many times as is determined by the 'numIteration' attribute.

Generalizations

- SingleCombinedBehaviour

Properties

- numIteration : Integer [1]
  This property of a 'BoundedLoopBehaviour' element determines how many times the 'Block' element of a 'BoundedLoopBehaviour' element shall be executed.

Constraints

No guard constraint
The 'Block' element of a 'BoundedLoopBehaviour' element shall not have a guard condition.

Iterations number shall be positive
The 'numIteration' attribute of a 'BoundedLoopBehaviour' element shall be a positive Integer value.

8.3.9 UnboundedLoopBehaviour

Semantics

An 'UnboundedLoopBehaviour' element is a 'SingleCombinedBehaviour', therefore it shall contain one 'Block' element, which may have a guard condition. The 'Block' element shall be executed as long as the guard condition of the 'Block' element evaluates to true. If the 'Block' element has no guard condition, it shall be executed an infinite number of times.

Generalizations

- SingleCombinedBehaviour

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.10 MultipleCombinedBehaviour

Semantics

A 'MultipleCombinedBehaviour' element is a 'CombinedBehaviour' that contains at least one potentially guarded 'Block' element (in case of 'ConditionalBehaviour') or at least two ordered and potentially guarded 'Block' elements (in case of 'AlternativeBehaviour' or 'ParallelBehaviour').

Generalizations

- CombinedBehaviour

Properties

- block : Block [1..*] {unique, ordered}
  The 'block' property refers to a list of 'Block' elements that specify the behaviour of the 'MultipleCombinedBehaviour' element.

Constraints

There are no constraints specified.

8.3.11 AlternativeBehaviour

Semantics

An 'AlternativeBehaviour' element is a 'MultipleCombinedBehaviour' that shall contain at least two 'Block' elements. Each block of an 'AlternativeBehaviour' element may have a guard condition and shall start with one of the following 'AtomicBehaviour' elements: 'Interaction', 'TimeOut' or 'Quiescence'. For the 'Interaction' element it is required that the target 'GateInstance' shall be associated to a 'ComponentInstance' with the 'role' of 'Tester'.

If a 'Block' element has no guard condition, it is equivalent to a 'Block' element with a guard condition of true. The first 'Block' element whose guard condition is evaluated to true and whose first event occurs, will be executed.

Generalizations

- MultipleCombinedBehaviour

Properties

There are no properties specified.

Constraints

Blocks
An 'AlternativeBehaviour' element shall contain at least two 'Block' elements. Each block of an 'AlternativeBehaviour' element may have a guard condition and shall start with one of the following 'AtomicBehaviour' elements: 'Interaction' received by a 'Tester' component, 'TimeOut' or 'Quiescence'.

8.3.12 ConditionalBehaviour

Semantics

A 'ConditionalBehaviour' element is a 'MultipleCombinedBehaviour' that can contain one or more 'Block' elements. All the 'Block' elements shall have a guard condition except for the last 'Block' element, which may have no guard if the 'ConditionalBehaviour' element contains more than one 'Block' elements. In this case, the last 'Block' element is equivalent to a 'Block' element with a guard condition of true ("*else*" block). The guard conditions of the 'Block' elements are evaluated in the order of their definition. The first 'Block' element, whose guard condition is evaluated to true, will be executed. If none of the guard conditions are evaluated to true, the execution continues with the next behaviour after the 'ConditionalBehaviour' element.

Generalizations

- MultipleCombinedBehaviour

Properties

There are no properties specified.

Constraints

Guards required
All the 'Block' elements shall have a guard condition except for the last 'Block' element, which may have no guard if the 'ConditionalBehaviour' element contains more than one 'Block' elements. In this case, the last 'Block' element is equivalent to a 'Block' element with a guard condition of true (else block).

8.3.13 ParallelBehaviour

Semantics

A 'ParallelBehaviour' element is a 'MultipleCombinedBehaviour' that shall contain at least two 'Block' elements. None of the 'Block' elements of a 'ParallelBehaviour' element shall have a guard condition. The 'Block' elements are executed in parallel (i.e. the relative execution order of the behaviours of the different 'Block' elements of a 'ParallelBehaviour' element is not specified). The execution of a 'ParallelBehaviour' element shall terminate when all its 'Block' elements are terminated.

Generalizations

- MultipleCombinedBehaviour

Properties

There are no properties specified.

Constraints

Blocks
A 'ParallelBehaviour' element shall contain at least two 'Block' elements. None of the 'Block' elements of a 'ParallelBehaviour' element shall have a guard condition.

8.3.14 AtomicBehaviour

Semantics

'AtomicBehaviour' is a 'Behaviour'. An 'AtomicBehaviour' element defines the simplest form of behaviour that cannot be decomposed further. An 'AtomicBehaviour' can be: 'TimerOperation', 'TimeOperation', 'VerdictAssignment', 'Break', 'Stop', 'ActionReference', ' Interaction', or 'TestDescriptionReference'.

Generalizations

- Behaviour

Properties

- timeConstraint : TimeConstraint [0..*] {unique}
  This property specifies a set of 'TimeConstraint' elements that determines the execution of the given 'AtomicBehaviour' element with respect to the passed time.

Constraints

There are no constraints specified.

8.3.15 Break

Semantics

'Break' terminates the execution of the enclosing 'CombinedBehaviour' element. Execution continues with the 'Behaviour' element that follows the enclosing 'CombinedBehaviour' element.

Generalizations

- AtomicBehaviour

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.16 Stop

Semantics

'Stop' is used to describe an explicit stop of the execution of a test description. No further behaviour shall be executed beyond a 'Stop'. In particular, a 'Stop' element in a referenced (called) test description shall also stop the behaviour of the referencing (calling) test description(s) recursively.

Generalizations

- AtomicBehaviour

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.17 VerdictAssignment

Semantics

The 'VerdictAssignment' is used to set the verdict of the test run explicitly. This might be necessary if the implicit verdict mechanism described below is not sufficient.

By default, the test description specifies the expected behaviour of the system. If an execution of a test description performs the expected behaviour, the verdict is set to 'pass' implicitly. If a test run deviates from the expected behaviour, the verdict 'fail' will be assigned to the test run implicitly. Other verdicts, including 'inconclusive' and user-definable verdicts, need to be set explicitly within a test description.

Generalizations

- AtomicBehaviour

Properties

- verdictType : VerdictType [1]
  Stores the value of the verdict to be set.

Constraints

There are no constraints specified.

8.3.18 VerdictType

Semantics

'VerdictType' is a 'PackageableElement' that specifies the possible verdicts of a test description. At minimum, the 'VerdictType' shall define the following instances: 'pass', 'inconclusive', 'fail'. (See Clause 10.1) This list can be extended by the user according to the needs of concrete implementations.

Generalizations

- PackageableElement

Properties

There are no properties specified.

Constraints

Minimum set of values of the VerdictType
The 'VerdictType' shall contain at least the following three instances: 'pass', 'inconclusive', 'fail'.

8.3.19 Interaction

Semantics

An 'Interaction' element is an abstract representation of any information exchanged between gate instances assuming that they are connected via a connection. The 'ArgumentSpecification' of an 'Interaction' element contained within refers to the data being exchanged between the components participating in the interaction via their connected gates. It can also carry parameters for this data.

In a concrete realization, an interaction can represent, among others, one of the following options:

- Message-based communication: The data of an interaction argument represents a message being sent (from source) and received (from target).
- Procedure-based communication: The data of an interaction argument represents a remote function call being initiated (from source) and invoked (at target) or its return values being transmitted back.
- Shared variable access: The data of an interaction argument represents a shared variable being read (source is the gate of the component that owns this variable, target is the gate of the reading component) or updated (source is the gate of the component that wants to change the value of a variable, target is the gate of the component that owns this variable).

The data description provided as an 'ArgumentSpecification' can be a (possibly parameterized) data instance, a data set, or a data proxy that is bound to either a data instance or a data set at runtime. Using a data set as part of the argument specification in an interaction enables the specification of a set of data instances that are all acceptable within this interaction.

EXAMPLE: Consider the data set STATUS of data instances 200, 4xx, 5xx and the data instance HttpResponse(STATUS) of another data set. An interaction that refers to the latter data instance in its argument specification provides a shorthand notation for the data instances HttpResponse(200), HttpResponse(4xx), and HttpResponse(5xx) that are all accepted and valid within the given interaction. This notation comes in handy for specifying SUT output (source of an interaction is the gate of a SUT component) by a tester component gate (target) when the exact output is not known or irrelevant.

Generalizations

- AtomicBehaviour

Properties

- argument : ArgumentSpecification [1]
  This property refers to an 'ArgumentSpecification' element that is taken as the argument (data) of this interaction.
- source : GateInstance [1]
  This property refers to a 'GateInstance' element that acts as the source of this interaction. That is, the associated component instance outputs the data of this interaction via its gate instance.
- target : GateInstance [1..*] {unique, ordered}
  This property refers to a list of 'GateInstance' elements that act as the target(s) of this interaction. That is, the associated component instance(s) input the data of this interaction via their gate instance(s). In case of point-to-point communication there is exactly one target. In case of point-to-multipoint communication there are multiple target gate instances.

Constraints

Gate instances of an interaction shall be different
All gate instances that act as source or target(s) of an interaction shall be different from each other.

Gate instances of an interaction shall be connected
The gate instances that act as source or a target(s) of an interaction shall be interconnected by a connection.

Typing of interaction arguments
The 'DataElement' referred to in the 'argument' shall match the 'DataSet' referenced in the 'GateType' definition of the source and target gate instances of an interaction. Matching is defined in the following manner:

- If the 'argument' refers to a (possibly parameterized) 'DataInstance', then the 'GateType' definition shall reference a 'DateSet' that contains this 'DataInstance'.
- If the 'argument' refers to a 'DataProxy', then this 'DataProxy' shall refer to the same 'DataSet' as referenced in the 'GateType' definition. If the 'argument' refers to a 'DataSet', then this 'DataSet' shall be referenced in the 'GateType' definition.

8.3.20 Action

Semantics

An 'Action' element is a 'PackageableElement' that can be used to specify any procedure (e.g. local computation, function call, physical setup, etc.) in an informal way. The interpretation of the action is outside the scope of TDL.

Generalizations

- PackageableElement

Properties

- body : String [1]
  This property describes the action as an informal String.

Constraints

Named action
An action shall have a name.

8.3.21 ActionReference

Semantics

An 'ActionReference' element can be used to refer to an 'Action' element to be executed. It may have a 'componentInstance' attribute that specifies the component instance on which the action is to be performed.

Generalizations

- AtomicBehaviour

Properties

- componentInstance : ComponentInstance [0..1]
  This property refers to a 'ComponentInstance' element on which the action is to be performed.
- action : Action [1]
  This property is refers to the 'Action' element to be executed.

Constraints

There are no constraints specified.

8.3.22 TestDescriptionReference

Semantics

A 'TestDescriptionReference' is used to describe the invocation of the behaviour of a test description within another test description. The invoked behaviour is executed in its entirety before the behaviour of the invoking test description is executed further. A 'TestDescriptionReference' also has a possibly empty list of actual parameters which are passed to the referenced 'TestDescription'.

Generalizations

- AtomicBehaviour

Properties

- referencedTestDescription : TestDescription [1]
  Refers the test description whose behaviour is invoked.

- actualParameter : ArgumentSpecification [0..*] {ordered}
  Refers to a list of actual parameters passed to the referenced test description.

Constraints

Test configuration of referenced test description
The referenced test description shall use the same test configuration as the referencing test description.

Number of actual parameters
The number of actual parameters in the 'TestDescriptionReference' shall be equal with the number of formal parameters of the referenced 'TestDescription'.

Matching parameters
The actual parameter *AP*[*i*] of index *i* in the ordered list of 'actualParameter's of the 'dataInstance' shall match the formal parameter *FP*[*i*] of index *i* in the ordered list of formal parameters of the referenced 'TestDescription'. Matching is defined in the following terms:

- If *AP*[*i*] is a 'DataInstance' then the 'DataSet' to which *AP*[*i*] refers to shall be the same the 'DataProxy' *FP*[*i*] refers to.

- If *AP*[*i*] is a 'DataSet' then it shall refer to the same 'DataSet' the 'DataProxy' *FP*[*i*] refers to.

- If *AP*[*i*] is a 'DataProxy' then the 'DataSet' to which *AP*[*i*] refers to shall be the same the 'DataProxy' *FP*[*i*] refers to.

8.3.23 ArgumentSpecification

Semantics

The abstract 'ArgumentSpecification' element specifies the 'DataElement' and its parameters (if any) used in interactions or test description references, i.e. the data that is exchanged in an 'Interaction' or passed to a 'TestDescription' via a 'TestDescriptionReference'. The 'ArgumentSpecification' element shall relate to 'DataElement's that are already defined elsewhere.

The effect of an argument specification is that it enables data sets and data instances being passed to interactions and between test descriptions. In this process, a data set defined in the invoked instance of an interaction or test description can also be refined to a specific data instance by means of a 'DataInstanceArgumentSpecification'.

Generalizations

- Element

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.24 DataInstanceArgumentSpecification

Semantics

The 'DataInstanceArgumentSpecification' element is a refinement of the 'ArgumentSpecification' element that deals with an argument provided as a (possibly parameterized) 'DataInstance'. A valid data instance argument specification shall match the data instance definition it refers to. The matching conditions are stated as constraints below.

Generalizations

- ArgumentSpecification

Properties

- dataInstance : DataInstance [1]
  The 'DataInstance' that is referenced as argument by the 'DataInstanceArgumentSpecification'.

- actualParameter : DataElement [0..*] {ordered}
  The set of 'DataElement's that are used as actual parameters to the referenced 'DataInstance' according to the parameter matching conditions stated as constraints below.

Constraints

Matching argument
The 'dataInstance' in the 'DataInstanceArgumentSpecification' shall refer to a 'DataInstance' that is a contained element of the 'DataSet' referenced in the related 'Interaction' or 'TestDescription'.

Equal number of parameters
The number of the 'actualParameter's attached to the 'DataInstance' in the 'DataInstanceArgumentSpecification' shall be equal to the number of 'parameter's used in the definition of this 'DataInstance'.

Matching parameters
The actual parameter *AP*[*i*] of index *i* in the ordered list of 'actualParameter's of the 'dataInstance' shall match the parameter *P*[*i*] of index *i* in the ordered list of the definition of this 'DataInstance'. Matching is defined in the following terms:

- If *P*[*i*] refers to a 'DataInstance' then *AP*[*i*] shall refer to the very same 'DataInstance'.

- If *P*[*i*] refers to a 'DataSet' then (i) *AP*[*i*] shall refer to the very same 'DataSet' or (ii) *AP*[*i*] shall refer to a 'DataProxy' that refers, in turn, to the same 'DataSet' or (iii) *AP*[*i*] shall refer to a 'DataInstance' that is contained within the definition of the 'DataSet'. In the latter case the data set is refined to this data instance.

Data proxies as parameters
If actual parameter *AP*[*i*] of index *i* refers to a 'DataProxy' then the same 'DataProxy' shall be referenced as 'formalParameter' of the containing 'TestDescription'.

8.3.25 DataProxyArgumentSpecification

Semantics

The 'DataProxyArgumentSpecification' element is a refinement of the 'ArgumentSpecification' element that deals with an argument provided as a 'DataProxy'. A valid data proxy argument specification shall match the data proxy definition (formal parameter) of the test description that contains this data proxy argument specification. The matching condition is stated as constraint below.

Generalizations

- ArgumentSpecification

Properties

- dataProxy: DataProxy [1]
  The 'DataProxy' that is used as argument in the 'DataProxyArgumentSpecification'.

Constraints

Matching argument
The 'dataProxy' in the 'DataProxyArgumentSpecification' shall be referenced as 'formalParameter' of the containing 'TestDescription'.

8.3.26 DataSetArgumentSpecification

Semantics

The 'DataSetArgumentSpecification' element is a refinement of the 'ArgumentSpecification' element that deals with an argument provided as a 'DataSet'. A valid data set argument specification shall match the data element definition of the invoked interaction or test description.

Generalizations

- ArgumentSpecification

Properties

- dataSet: DataSet [1]
  The 'DataSet' that is used as argument in the 'DataSetArgumentSpecification'.

Constraints

There are no constraints specified.

8.3.27 ExceptionalBehaviour

Semantics

'ExceptionalBehaviour' is optionally contained within a 'CombinedBehaviour' element. It is a 'Behaviour' that consists of one 'Block' element that shall have no guard and shall start with one of the following 'AtomicBehaviour' elements: 'Interaction', 'TimeOut', or 'Quiescence'. For the 'Interaction' element the target 'GateInstance' shall be associated to a 'ComponentInstance' with the 'role' of 'Tester'.

An 'ExceptionalBehaviour' element may have a 'guardedComponent' attribute.

An 'ExceptionalBehaviour' element defines a behaviour that is an alternative to any 'Interaction' element in the enclosing 'CombinedBehaviour' element:

- whose target 'GateInstance' is associated to a 'ComponentInstance' with the role of 'Tester' (provided the 'guardedComponent' attribute is not present);
- whose target 'GateInstance' is associated to that 'ComponentInstance' with the role of 'Tester', which is referenced by the 'guardedComponent' attribute (provided the 'guardedComponent' attribute is present).

A 'CombinedBehaviour' can have several 'ExceptionalBehaviour' elements. It can be interpreted as a shorthand of replacing every 'Interaction' targeted to the 'Tester'(or if the 'guardedComponent' is present, then every 'Interaction' targeted to that 'Tester' component) of the containing 'CombinedBehaviour' element by an 'AlternativeBehaviour' that contains the given 'Interaction' targeted to the 'Tester'(or if the 'guardedComponent' is present, then to the referenced 'Tester' component) as its first alternative and the block(s) of the 'ExceptionalBehaviour' element(s) as the following alternative(s), in the order of which the 'ExceptionalBehaviour' elements are specified in the containing 'CombinedBehaviour' element.

An 'ExceptionalBehaviour' can be either a 'DefaultBehaviour' or an 'InterruptBehaviour'.

Generalizations

- Behaviour

Properties

- block : Block [1]
  This property refers to a 'Block' element that specifies the behaviour of the 'ExceptionalBehaviour'.
- guardedComponent : ComponentInstance [0..1]
  This optional property refers to a 'Tester' component instance for which the behaviour specified by the 'ExceptionalBehaviour' element is to be applied.

Constraints

First element in block allowed
The 'Block' element referred to by the 'block' property shall have no guard and shall start with one of the following 'AtomicBehaviour' elements: 'Interaction', 'TimeOut', or 'Quiescence'. For the 'Interaction' element it is required that the target9 'GateInstance' shall be associated to a 'ComponentInstance' with the 'role' of 'Tester'.

Guarded component role
The 'guardedComponent' can refer only to a 'Tester' component.

8.3.28 DefaultBehaviour

Semantics

A 'DefaultBehaviour' is an 'ExceptionalBehaviour'. If it is executed, the execution continues with the next 'Behaviour' element of the enclosing 'CombinedBehaviour' element following the one that caused the execution of the 'DefaultBehaviour' element.

Generalizations

- ExceptionalBehaviour

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.29 InterruptBehaviour

Semantics

An 'InterruptBehaviour' is an 'ExceptionalBehaviour'. If it is executed, the execution continues with the same 'Behaviour' element of the enclosing 'CombinedBehaviour' element, at which the execution of the 'InterruptBehaviour' started.

Generalizations

- ExceptionalBehaviour

Properties

There are no properties specified.

Constraints

There are no constraints specified.

8.3.30 PeriodicBehaviour

Semantics

A 'PeriodicBehaviour' element is optionally contained within a 'CombinedBehaviour' element. It is a 'Behaviour' that consists of one 'Block' element that shall not start with one of the following 'AtomicBehaviour' elements: 'Interaction', 'TimeOut', 'Quiescence' or 'Interaction' whose target 'GateInstance' is associated to a 'ComponentInstance' with the 'role' of 'Tester'.

A 'PeriodicBehaviour' element defines a behaviour that is executed periodically in parallel with the enclosing 'CombinedBehaviour' element. The frequency of the execution is specified by its 'period' attribute.

Generalizations

- Behaviour

Properties

- block : Block [1]
  This property refers to a 'Block' element, whose behaviour will be executed periodically in parallel with the behaviour of the enclosing 'CombinedBehaviour' element.

- period : Time [1]
  This property defines the frequency of the execution of the behaviour of the 'Block' element specified by the 'block' attribute. The 'period' is a 'Time' value.

Constraints

First event allowed
The 'Block' element referred by the 'block' attribute shall start with an 'Interaction' initiated by a 'Tester' component.

9 Time

9.1 Overview

The 'Time' package defines the elements to express time, time requirements, and timer operations in TDL.

9.2 Abstract Syntax

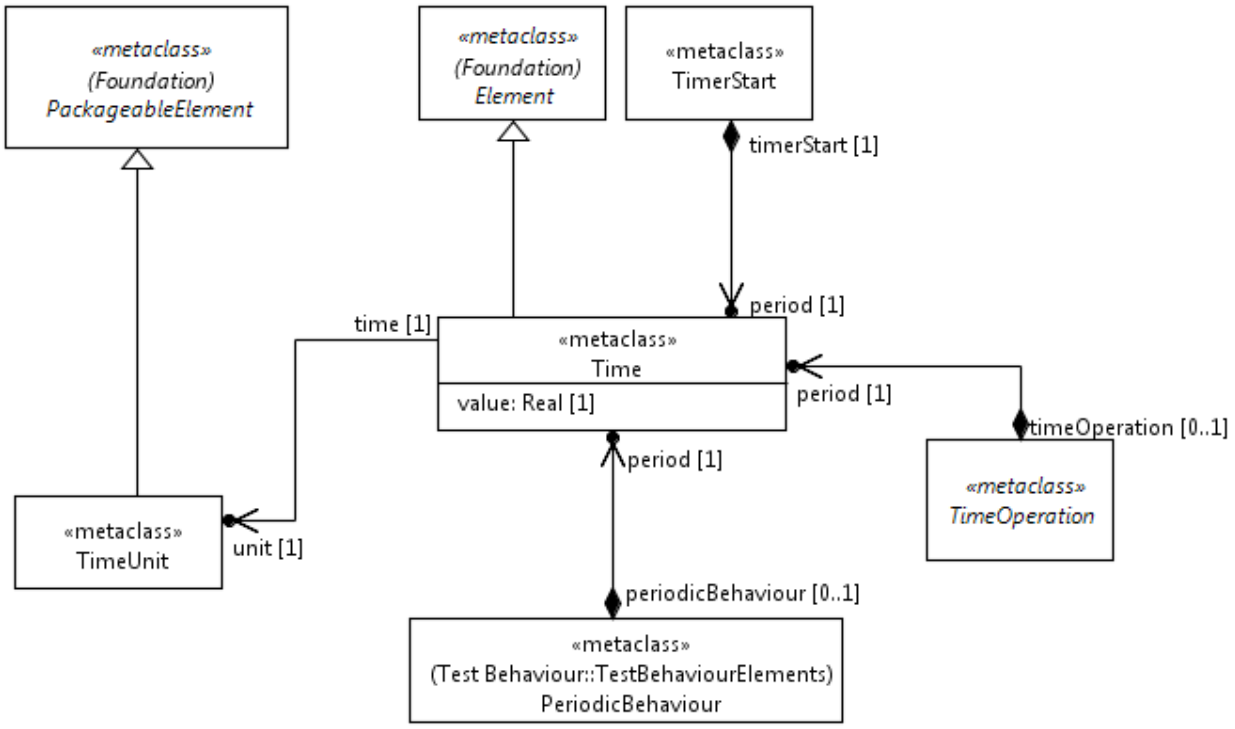

Figure 9.1: Time Concept

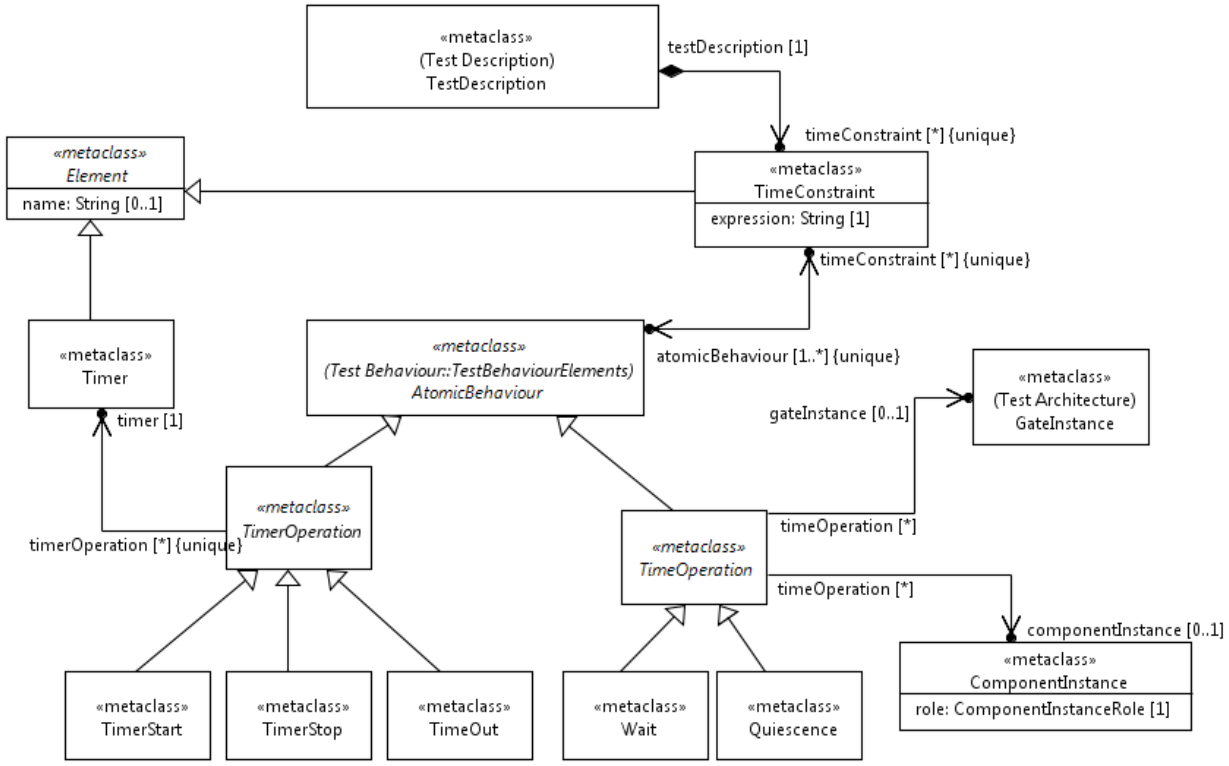

Figure 9.2: Timer and Time Operations

9.3 Classifier Description

9.3.1 Time

Semantics

'Time' represents the concept of time. Time in TDL is considered to be global and progresses in discrete quantities of arbitrary granularity. Progress in time is expressed as a monotonically increasing function. A time instance or a time duration are expressed by a positive 'value' of type 'Real'. The measurement unit is set by the property 'unit' which defines a predefined and user-extensible set of measurement units.

Time starts with the execution of an unreferenced 'TestDescription' (with time value 0.0).

Generalizations

- Element

Properties

- value : Real [1]
  Represents a 'value' of a time instance or a time duration.
- unit : TimeUnit [1]
  Defines the measurement unit of a time value or a time duration value.

Constraints

Time value shall be non-negative
The attribute 'value' shall be non-negative.

9.3.2 TimeUnit

Semantics

'TimeUnit' describes the unit of a 'Time' value. There exists a list of predefined instances for the time unit (see clause 10.2), which can be extended by the user.

Generalizations

- PackageableElement

Properties

There are no properties specified.

Constraints

There are no constraints specified.

9.3.3 TimeOperation

Semantics

A 'TimeOperation' summarizes the two possible time operations that can occur at a 'GateInstance' of a 'Tester' 'ComponentInstance': 'Wait' and 'Quiescence'. A 'TimeOperation' references either a 'GateInstance' or a 'ComponentInstance'. In the former case it means that the 'TimeOperation' applies only on the referenced 'GateInstance', while in the latter case it applies on all the 'GateInstance's of the referenced 'Tester' 'ComponentInstance'.

Generalizations

- AtomicBehaviour

Properties

- period : Time [1]
  The 'period' defines the time duration of the 'TimeOperation'.
- gateInstance : GateInstance [0..1]
  The 'GateInstance' element, to which the 'TimeOperation' is associated.
- componentInstance : ComponentInstance [0..1]
  The 'ComponentInstance' element, to which the 'TimeOperation' is associated.

Constraints

Time operations on tester components only
A 'TimeOperation' shall be performed only on a 'Tester' 'ComponentInstance' or on a 'GateInstance' that is associated with a 'ComponentInstance' in the role 'Tester'.

Association to gate or component instance is mutually exclusive
It is mandatory to set an association of a 'TimeOperation' to a 'GateInstance' or a 'ComponentInstance'. However the association shall be mutually exclusive, i.e. both associations shall not be given at the same time.

9.3.4 Wait

Semantics

'Wait' defines the time duration that a 'Tester' 'ComponentInstance' instance waits at a given 'GateInstance' (if 'Wait' is associated to a gate instance) or waits at all the 'GateInstance's the 'Tester' 'ComponentInstance' contains (if 'Wait' is associated to a component instance) before performing the next behaviour.

Generalizations

- TimeOperation

Properties

There are no properties specified.

Constraints

There are no constraints specified.

9.3.5 Quiescence

Semantics

'Quiescence' defines the time duration during which a 'Tester' component instance shall expect no input from a 'SUT' component instance at a given gate instance (if 'Quiescence' is associated to a gate instance) or at all the gate instances the tester component instance contains of (if 'Quiescence' is associated to a component instance).

Generalizations

- TimeOperation

Properties

There are no properties specified.

Constraints

There are no constraints specified.

9.3.6 TimeConstraint

Semantics

A 'TimeConstraint' is used to express a timing requirement over 'AtomicBehaviour' elements. The time constraint shall be formulated over two or more 'AtomicBehaviour' elements. A 'TimeConstraint' element limits the execution time of the 'AtomicBehaviour' element that is to be executed at latest, relative to other 'AtomicBehaviour' elements in that time constraint. Time constraints reside within a test description.

Currently, there is no formal notation to express time constraints. However it is understood that the names of affected 'AtomicBehaviour' elements, i.e. their labels, are used in a Boolean expression together with one or more 'Time' elements expressing time values.

EXAMPLE: In the expression "|L2 – L1| < 5,0 sec" the names L2 and L1 shall be the labels of two atomic behaviour elements. Then the expression states that atomic behaviour element L2 shall occur in less than 5,0 sec after L1.

Generalizations

- Element

Properties

- expression : String [1]
  Defines the time constraint as an informal string.

- atomicBehaviour : AtomicBehaviour [1..*] {ordered}
  The 'AtomicBehaviour' elements that are affected by this time constraint.

Constraints

There are no constraints specified.

9.3.7 Timer

Semantics

A 'Timer' element defines a timer that is used by different timer operations. A 'Timer' is contained within a 'ComponentType' element assuming that each 'ComponentInstance' of the given 'ComponentType' has its local copy of that timer in the operational mode.

Generalizations

- Element

Properties

- componentType : ComponentType [1]
  The 'ComponentType' element, in which the 'Timer' is defined.

Constraints

Named timer
A timer shall have a name.

Initial state of a timer
When a timer is defined, it is operationally in the state *idle*.

9.3.8 TimerOperation

Semantics

A 'TimerOperation' operates on an associated 'Timer'. 'TimerOperation' is an abstract element that summarizes the operations on timers: timer start, timeout and timer stop.

Generalizations

- AtomicBehaviour

Properties

- timer : Timer [1]
  This property refers to the 'Timer' element on which the 'TimerOperation' operates.

Constraints

There are no constraints specified.

9.3.9 TimerStart

Semantics

A 'TimerStart' operation starts a specific timer. The state of that timer becomes then *running*. If a running timer is started, the timer is stopped first implicitly and then (re-)started.

Generalizations

- TimerOperation

Properties

- period : Time [1]
  Defines the duration of the timer from start to timeout.

Constraints

There are no constraints specified.

9.3.10 TimeOut

Semantics

A 'TimeOut' element is used to specify the occurrence of a timeout event when the period set by the 'TimerStart' operation of that timer has elapsed. In operational mode, the timer changes then from *running* state to the *idle* state.

Generalizations

- TimerOperation

Properties

There are no properties specified.

Constraints

Running timer
The state of the timer shall be in state *running* in order to allow the 'TimeOut' to occur in the operational mode.

9.3.11 TimerStop

Semantics

A 'TimerStop' operation stops a running timer. If an idle timer is stopped, then no action shall be taken. After performing a 'TimerStop' operation, the state of that timer becomes *idle*.

Generalizations

- TimerOperation

Properties

There are no properties specified.

Constraints

There are no constraints specified.

10 Predefined Types

10.1 Overview

This clause lists the predefined element instances for the meta-model elements 'VerdictType' and 'TimeUnit' that shall be a part of a standard-compliant TDL implementation. It is not specified how these predefined instances are made available to the user.

Clause 10.2 lists the basic predefined instances of 'VerdictType' elements that shall be used either in explicit 'VerdictAssignment' elements or in the implicit verdict setting mechanism of TDL. Clause 10.3 afterwards lists the basic predefined instances of 'TimeUnit' elements that shall be used in any instances of 'Time' elements of TDL.

10.2 Predefined Element Instances of 'VerdictType'

10.2.1 pass

The predefined instance 'pass' of the element 'VerdictType' indicates the valid behaviour of the SUT as observed by the tester. See definition in ISO 9646-1 [3].

10.2.2 fail

The predefined instance 'fail' of the element 'VerdictType' indicates the invalid behaviour of the SUT as observed by the tester. See definition in ISO 9646-1 [3].

10.2.3 inconclusive

The predefined instance 'inconclusive' of the element 'VerdictType' can be used if neither 'pass' nor 'fail' verdict can be given. See definition in ISO 9646-1 [3].

10.3 Predefined Element Instances of 'TimeUnit'

10.3.1 tick

The time unit instance 'tick' represents an arbitrary, but fixed duration of time.

10.3.2 nanosecond

The time unit instance 'nanosecond' represents the duration of a nanosecond (i.e. 10^{-9} seconds).

10.3.3 microsecond

The time unit instance 'microsecond' represents the duration of a microsecond (i.e. 10^{-6} seconds).

10.3.4 millisecond

The time unit instance 'millisecond' represents the duration of a millisecond (i.e. 10^{-3} seconds).

10.3.5 second

The time unit instance 'second' represents the duration of a seconds.

10.3.6 minute

The time unit instance 'minute' represents the duration of a minute (i.e. 60 seconds).

10.3.7 hour

The time unit instance 'hour' represents the duration of an hour (i.e. 3 600 seconds).

Annex A (informative): Technical Representation of the TDL Meta-Model

The technical representation of the TDL meta-model is included as an electronic attachment es_203119v010101p0.zip which accompanies the present document. The purpose of this annex is to serve as a possible starting point for implementing the TDL meta-model conforming to the present document. See the readme contained in the zip file for details.

Annex B (informative): Examples of a TDL Concrete Syntax

B.1 Introduction

The applicability of the TDL meta-model that is described in the main part of the present document depends on the availability of TDL concrete syntaxes that implement the meta-model (abstract syntax). Such a TDL concrete syntax can then be used by end users to write TDL specifications. Though a concrete syntax will be based on the TDL meta-model, it can implement only parts of the meta-model if certain TDL features are not necessary to handle a user's needs.

This annex illustrates an example of a possible TDL concrete syntax in a textual format that supports all features of the TDL meta-model, called "TDLan". Three examples are outlined below - two examples translated from existing test descriptions taken from [i.2] and [i.3], as well as an example illustrating some of the TDL data parameterization and mapping concepts. The examples are accompanied by a complete reference description of the textual syntax of TDLan given in EBNF.

B.2 A 3GPP Conformance Example in Textual Syntax

This example describes one possible way to translate clause 7.1.3.1 from TS 136 523-1 [i.2] into the proposed TDL textual syntax, by mapping the concepts from the representation in the source document to the corresponding concepts in the TDL meta-model by means of the proposed textual syntax. The example has been enriched with additional information, such as explicit data definitions and test configuration details for completeness where applicable.

```
//Translated from [i.2], Section 7.1.3.1
TDLan Specification Layer_2_DL_SCH_Data_Transfer {
   //Procedures carried out by a component of a test configuration
   //or an actor during test execution
   Action preCondition : "Pre-test Conditions:
        RRC Connection Reconfiguration" ;
   Action preamble : "Preamble:
            The generic procedure to get UE in test state Loopback
            Activated (State 4) according to TS 36.508 clause 4.5
            is executed, with all the parameters as specified in the
            procedure except that the RLC SDU size is set to return no
            data in uplink.
            (reference corresponding behaviour once implemented" ;

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

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

   //User-defined time units
   Time Unit seconds;

   //Test objectives (copied verbatim from source document)
   Test Objective TP1 {
      from : "36523-1-a20_s07_01.doc::7.1.3.1.1 (1)" ;
      description : "with { UE in E-UTRA RRC_CONNECTED state }
                  ensure that {
                   when { UE receives downlink assignment on the PDCCH
                       for the UE's C-RNTI and receives data in the
                             associated subframe and UE performs HARQ
                             operation }
                    then { UE sends a HARQ feedback on the HARQ
                             process }
                  }" ;
   }
```

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

//Relevant data set and data instance definitions
Data Set PDU {
   instance MAC_PDU ;
}
Data Set ACK {
   instance HARQ_ACK ;
}
Data Set C_RNTI {
   instance UE_C_RNTI ;
   instance unknown_C_RNTI ;
}
Data Set OTHER {
   instance PDCCH ( C_RNTI ) ;
   instance RRCConnectionReconfiguration ;
}

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

//Component type definitions
Component Type defaultCT {
   gate types : defaultGT ;
}

//Test configuration definition
Test Configuration defaultTC {
   instantiate SS as Tester of type defaultCT having {
      gate SSgate of type defaultGT ;
   }
   instantiate UE as SUT of type defaultCT having {
      gate UEgate of type defaultGT ;
   }
   connect UEgate to SSgate ;
}

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

      //Test sequence
      SSgate sends instance PDCCH ( UE_C_RNTI ) to UEgate with {
         STEP "1" ;
         PROCEDURE "SS transmits a downlink assignment
                  including the C-RNTI assigned to
                  the UE" ;
      } ;
      SSgate sends instance MAC_PDU to UEgate with {
         STEP "2" ;
         PROCEDURE "SS transmits in the indicated
                  downlink assignment a RLC PDU in
                  a MAC PDU" ;
      } ;
      UEgate sends instance HARQ_ACK to SSgate with {
         STEP "3" ;
         PROCEDURE "Check: Does the UE transmit an
                     HARQ ACK on PUCCH?" ;
         test objectives : TP1 ;
      } ;
      set verdict to PASS ;
      SSgate sends instance PDCCH ( unknown_C_RNTI ) to UEgate with {
         STEP "4" ;
         PROCEDURE "SS transmits a downlink assignment
```

```
                    to including a C-RNTI different from
                    the assigned to the UE" ;
          } ;
          SSgate sends instance MAC_PDU to UEgate with {
             STEP "5" ;
             PROCEDURE "SS transmits in the indicated
                    downlink assignment a RLC PDU in
                    a MAC PDU" ;
          } ;

          //Interpolated original step 6 into an alternative behaviour,
          //covering both the incorrect and the correct behaviours of the UE
          alternatively {
             UEgate sends instance HARQ_ACK to SSgate ;
             set verdict to FAIL ;
          } or {
             gate SSgate is quiet for (5.0 seconds);
             set verdict to PASS ;
          } with {
             STEP "6" ;
             PROCEDURE "Check: Does the UE send any HARQ ACK
               on PUCCH?" ;
             test objectives : TP2 ;
          }
       }
    } with {
       Note : "Note 1: For TDD, the timing of ACK/NACK is not
              constant as FDD, see Table 10.1-1 of TS 36.213." ;
    }
} with {
    Note : "Taken from 3GPP TS 36.523-1 V10.2.0 (2012-09)" ;
    TITLE "Correct handling of DL assignment / Dynamic case" ;
}
```

B.3 An IMS Interoperability Example in Textual Syntax

This example describes one possible way to translate clause 4.5.1 from TS 186 011-2 [i.3] into the proposed TDL textual syntax, by mapping the concepts from the representation in the source document to the corresponding concepts in the TDL meta-model by means of the proposed textual syntax. The example has been enriched with additional information, such as explicit data definitions and test configuration details for completeness where applicable.

```
//Translated from [i.3], Clause 4.5.1.
TDLan Specification IMS_NNI_General_Capabilities {
  //Procedures carried out by a component of a test configuration
  //or an actor during test execution
  Action preConditions : "Pre-test conditions:
         - HSS of IMS_A and of IMS B is configured according to table 1
         - UE_A and UE_B have IP bearers established to their respective
          IMS networks as per clause 4.2.1
         - UE_A and IMS_A configured to use TCP for transport
         - UE_A is registered in IMS_A using any user identity
         - UE_B is registered user of IMS_B using any user identity
         - MESSAGE request and response has to be supported at II-NNI
         see tables 6.1 and 6.3)" ;

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

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

  //Test objectives (copied verbatim from source document)
  Test Objective TP_IMS_4002_1 {
    //Location in source document
    from : "ts_18601102v030101p.pdf::4.5.1.1 (CC 1)" ;
    //Further reference to another document
    from : "ts_124229v081000p.pdf, clause 4.2A, paragraph 1" ;
```

```
  description : "ensure that {
            when { UE_A sends a MESSAGE to UE_B
               containing a Message_Body greater than 1 300
               bytes }
            then { IMS_B receives the MESSAGE containing the
               Message_Body greater than 1 300 bytes }
           }" ;
}
Test Objective UC_05_I {
  //Only a reference to corresponding clause in the source document
  from : "ts_18601102v030101p.pdf::4.4.4.2" ;
}

//Relevant data set and data instance definitions
Data Set MSG {
  instance MESSAGE ;
  instance MESSAGE_TCP ( TCP ) ;
  instance TCP ;
  instance DING ;
  instance DELIVERY_REPORT ;
  instance M_200_OK ;
}

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

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

//Test configuration definition
Test Configuration CF_INT_CALL {
  instantiate USER_A as Tester of type USER having {
    gate gUSER_A of type defaultGT ;
  }
  instantiate UE_A as Tester of type UE having {
    gate gUE_A of type defaultGT ;
  }
  instantiate IMS_A as Tester of type IMS having {
    gate gIMS_A of type defaultGT ;
  }
  instantiate IBCF_A as Tester of type IBCF having {
    gate gIBCF_A of type defaultGT ;
  }
  instantiate IBCF_B as Tester of type IBCF having {
    gate gIBCF_B of type defaultGT ;
  }
  instantiate IMS_B as SUT of type IMS having {
    gate gIMS_B of type defaultGT ;
  }
  instantiate UE_B as Tester of type UE having {
    gate gUE_B of type defaultGT ;
  }
  instantiate USER_B as Tester of type USER having {
    gate gUSER_B of type defaultGT ;
  }
  connect gUSER_A to gUE_A ;
  connect gUE_B to gUSER_B ;
  connect gUE_A to gIMS_A ;
  connect gIMS_A to gIBCF_A ;
  connect gIBCF_A to gIBCF_B ;
  connect gIBCF_B to gIMS_B ;
  connect gIMS_B to gUE_B ;
}

//Test description definition
Test Description TD_IMS_MESS_0001 {
```

```
    use configuration : CF_INT_CALL ;
    {
      //Pre-conditions from the source document
      perform action preConditions with { PRECONDITION ; };

      //Test sequence
      gUSER_A sends instance MESSAGE to gUE_A with { STEP "1" ; } ;
      gUE_A sends instance MESSAGE to gIMS_A with { STEP "2" ; } ;
      gIMS_A sends instance MESSAGE to gIBCF_A with { STEP "3" ; } ;
      gIBCF_A sends instance MESSAGE to gIBCF_B with { STEP "4" ; } ;
      gIBCF_B sends instance MESSAGE_TCP to gIMS_B with { STEP "5" ; } ;
      gIMS_B sends instance MESSAGE to gUE_B with { STEP "6" ; } ;
      gUE_B sends instance DING to gUSER_B with { STEP "7" ; } ;
      gUE_B sends instance M_200_OK to gIMS_B with { STEP "8" ; } ;
      gIMS_B sends instance M_200_OK to gIBCF_B with { STEP "9" ; } ;
      gIBCF_B sends instance M_200_OK to gIBCF_A with { STEP "10" ; } ;
      gIBCF_A sends instance M_200_OK to gIMS_A with { STEP "11" ; } ;
      gIMS_A sends instance M_200_OK to gUE_A with { STEP "12" ; } ;

      optionally {
           gUE_A sends instance DELIVERY_REPORT to gUSER_A with { STEP "13" ; } ;
      } ;
    }
  } with {
    SUMMARY "IMS network shall support SIP messages greater than
             1 500 bytes" ;
  }
} with {
  Note : "Taken from ETSI TS 186 011-2 V3.1.1 (2011-06)" ;
  TITLE "SIP messages longer than 1 500 bytes" ;
}
```

B.4 An Example Demonstrating TDL Data Concepts

This example describes some of the concepts related to data and data mapping in TDL by means of the proposed TDL textual syntax. It illustrates how data instances can be parameterized, mapped to concrete data entities specified in an external resource, e.g. a TTCN-3 file, or to a runtime URI where dynamic concrete data values might be stored by the execution environment during runtime in order to facilitate some basic data flow of dynamic values between different interactions. The example considers a scenario where the SUT is required to generate and maintain a session ID between subsequent interactions using a similar test configuration as defined for the first example in clause B.2.

```
//A manually constructed example illustrating the data mapping concepts
TDLan Specification DataExample {

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

   //Test objectives
   Test Objective CHECK_SESSION_ID_IS_MAINTAINED {
      //Only a description
      description : "Check whether the session id is maintained
                     after the first response." ;
   }

   //Data definitions
   Data Set SESSION_ID {
      instance SESSION_ID_1 ;
      instance SESSION_ID_2 ;
   }

   Data Set MSG {
      instance REQUEST_SESSION_ID ;
      instance RESPONSE ( SESSION_ID ) ;
      instance MESSAGE ( SESSION_ID ) ;
   }

   //Data mappings

   //Load resource.ttcn3 containing the concrete data representations
   Use "resource.ttcn3"
      as TTCN_RESOURCE;

   //Map sets and instances to TTCN-3 records and templates, respectively
```

```
//(located in the loaded TTCN-3 file)
Map MSG
   to "record_message"
   in TTCN_ RESOURCE
   as MSG_mapping ;

Map REQUEST_SESSION_ID
   to "template_request"
   in TTCN_ RESOURCE
   as REQUEST_mapping ;

Map RESPONSE
   to "template_response"
   in TTCN_ RESOURCE
   as RESPONSE_mapping ;

Map MESSAGE
   to "template_message"
   in TTCN_MAPPING
   as MESSAGE_mapping ;

//Use a runtime URI for dynamic data available at runtime, such as
//session IDs
Use "runtime://sessions/"
   as RUNTIME_RESOURCE ;

//Map session ID data instances to locations within the runtime URI
Map SESSION_ID_1
   to "id_1"
   in RUNTIME_RESOURCE
   as SESSION_ID_1_mapping ;

Map SESSION_ID_2
   to "id_2"
   in RUNTIME_RESOURCE
   as SESSION_ID_2_mapping ;

//Gate type definitions
Gate Type defaultGT accepts MSG ;

//Component type definitions
Component Type defaultCT {
   gate types : defaultGT ;
}

//Test configuration definition
Test Configuration defaultTC {
   instantiate UE as SUT of type defaultCT having {
      gate UEgate of type defaultGT ;
   }

   instantiate SS as Tester of type defaultCT having {
      gate SSgate of type defaultGT ;
   }

   connect SSgate to UEgate ;
}

//Test description definition
Test Description exampleTD {
   use configuration : defaultTC ;
   {
      //Tester requests a session id
      SSgate sends instance REQUEST_SESSION_ID to UEgate ;

      //SUT responds with a session id that is assigned to the
      //runtime URI provided by the execution environment
      UEgate sends instance RESPONSE ( SESSION_ID_1 ) to SSgate ;

      //Tester sends a message with the session id
      //from the runtime URI
      SSgate sends instance MESSAGE ( SESSION_ID_1 ) to UEgate ;

      alternatively {

         //SUT responds with the same session id
         UEgate sends instance RESPONSE ( SESSION_ID_1 ) to SSgate ;
         set verdict to PASS;
```

```
        } or {

           //SUT responds with a different session id
           UEgate sends instance RESPONSE ( SESSION_ID_2 ) to SSgate ;
           set verdict to FAIL;

        } with {
           test objectives : CHECK_SESSION_ID_IS_MAINTAINED ;
        }
     }
  }
}
```

B.5 TDL Textual Syntax Reference

B.5.1 Conventions for the TDLan Syntax Definition

This annex describes the grammar of the used concrete textual syntax in the Extended Backus-Naur Form (EBNF) notation. The EBNF representation is generated from a reference implementation of the TDL meta-model. The EBNF representation can be used either as a concrete syntax reference for TDL end users or as input to a parser generator tool. Table B.1 defines the syntactic conventions that are to be applied when reading the EBNF rules. To distinguish this concrete textual syntax from other possible concrete textual syntax representations, it is referred to as "TDLan". This proposed syntax is complete in the sense that it covers the whole TDL meta-model.

Table B.1: Syntax definition conventions used

::=	is defined to be
abc	the non-terminal symbol abc
abc xyz	abc followed by xyz
abc | xyz	alternative (abc or xyz)
[abc]	0 or 1 instance of abc
{abc}+	1 or more instances of abc
{abc}	0 or more instances of abc
'a'-'z'	all characters from a to z
(...)	denotes a textual grouping
'abc'	the terminal symbol abc
;	production terminator
\	the escape character

B.5.2 TDL Textual Syntax EBNF Production Rules

```
TDLSpec                                  ::=   ( 'TDLan Specification' Identifier '{'
                                               [ ElementImport { ElementImport } ]
                                               [ PackageableElement { PackageableElement
                                               } ]
                                               '}'
                                               [ 'with' '{'
                                               [ Comment { Comment } ]
                                               [ Annotation { Annotation } ]
                                               '}' ] ) ;
Package                                  ::=   ( 'Package' Identifier '{'
                                               [ ElementImport { ElementImport } ]
                                               [ PackageableElement { PackageableElement
                                               } ]
                                               '}'
                                               [ 'with' '{'
                                               [ Comment { Comment } ]
                                               [ Annotation { Annotation } ]
```

```
                                        '}' ] ) ;
Identifier                          ::= ID ;
PackageableElement                  ::= ( AnnotationType
                                        | Package
                                        | TestObjective
                                        | DataSet
                                        | DataResourceMapping
                                        | DataElementMapping
                                        | ComponentType
                                        | GateType
                                        | TimeUnit
                                        | TestConfiguration
                                        | TestDescription
                                        | VerdictType
                                        | Action ) ;
TestObjectiveRealizer               ::= ( TimerStart
                                        | TimerStop
                                        | TimeOut
                                        | Wait
                                        | Quiescence
                                        | TestDescription
                                        | CompoundBehaviour
                                        | PeriodicBehaviour
                                        | AlternativeBehaviour
                                        | ParallelBehaviour
                                        | BoundedLoopBehaviour
                                        | UnboundedLoopBehaviour
                                        | ConditionalBehaviour
                                        | Stop
                                        | TestDescriptionReference
                                        | VerdictAssignment
                                        | ActionReference
                                        | Interaction
                                        | OptionalBehaviour
                                        | DefaultBehaviour
                                        | InterruptBehaviour
                                        | Break ) ;
DataElement                         ::= ( DataInstance | DataSet ) ;
ExceptionalBehaviour                ::= ( DefaultBehaviour | InterruptBehaviour )
                                        ;
AtomicBehaviour                     ::= ( TimerStart
                                        | TimerStop
                                        | TimeOut
                                        | Wait
                                        | Quiescence
                                        | Stop
                                        | TestDescriptionReference
                                        | VerdictAssignment
                                        | ActionReference
                                        | Interaction
                                        | Break ) ;
Behaviour                           ::= ( TimerStart
                                        | TimerStop
                                        | TimeOut
```

```
                                    | Wait
                                    | Quiescence
                                    | CompoundBehaviour
                                    | PeriodicBehaviour
                                    | AlternativeBehaviour
                                    | ParallelBehaviour
                                    | BoundedLoopBehaviour
                                    | UnboundedLoopBehaviour
                                    | ConditionalBehaviour
                                    | Stop
                                    | TestDescriptionReference
                                    | VerdictAssignment
                                    | ActionReference
                                    | Interaction
                                    | OptionalBehaviour
                                    | DefaultBehaviour
                                    | InterruptBehaviour
                                    | Break ) ;
Comment                         ::= ( 'Note' Identifier ':' String
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    '}' ] ';' ) ;
Annotation                      ::= ( Identifier String
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ] Identifier
                                    '}' ] ';' ) ;
String                          ::= STRING ;
ElementImport                   ::= ( 'Import' Identifier { ',' Identifier }
                                    ';'
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ] Identifier
                                    '}' ] ';' ) ;
AnnotationType                  ::= ( 'Annotation' Identifier
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    '}' ] ';' ) ;
TestObjective                   ::= ( 'Test Objective' Identifier '{'
                                    [ 'from' ':' String ';' { 'from' ':'
                                    String ';' } ]
                                    [ 'description' ':' String ';' ]
                                    '}'
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    '}' ] ) ;
DataSet                         ::= ( 'Data Set' Identifier '{'
                                    [ DataInstance { DataInstance } ]
                                    '}'
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
```

```
                                        '}' ] ) ;
DataResourceMapping                ::=  ( 'Use' String
                                        [ 'as' Identifier ]
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
DataElementMapping                 ::=  ( 'Map' Identifier
                                        [ 'to' String ] 'in' Identifier
                                        [ 'as' Identifier ]
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
ComponentType                      ::=  ( 'Component Type' Identifier '{'
                                        [ Timer { Timer } ] 'gate' 'types' ':'
                                        Identifier { ',' Identifier } ';'
                                        '}'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ) ;
GateType                           ::=  ( 'Gate Type' Identifier 'accepts'
                                        Identifier { ',' Identifier }
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
TimeUnit                           ::=  ( 'Time Unit' Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
TestConfiguration                  ::=  ( 'Test Configuration' Identifier '{'
                                        ComponentInstance { ComponentInstance }
                                        Connection { Connection }
                                        '}'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ) ;
TestDescription                    ::=  ( 'Test Description' Identifier
                                        [ '(' DataProxy { ',' DataProxy } ')' ]
                                        '{'
                                        use' 'configuration' ':' Identifier ';'
                                        CompoundBehaviour
                                        '}'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'time constraints' ':' TimeConstraint {
                                        ',' TimeConstraint } ';' ]
                                        '}' ] ) ;
VerdictType                        ::=  ( 'Verdict' Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
```

```
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
Action                          ::=     ( 'Action' Identifier ':' String
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
TimerStart                      ::=     ( 'start' Identifier 'for' '(' Time ')'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        '}' ] ';' ) ;
TimerStop                       ::=     ( 'stop' Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        '}' ] ';' ) ;
TimeOut                         ::=     ( Identifier 'times' 'out'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        '}' ] ';' ) ;
Wait                            ::=     (
                                        [ ( 'gate' | Identifier ) | ( 'component'
                                        | Identifier ) ] 'waits for' '(' Time ')'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        '}' ] ';' ) ;
Quiescence                      ::=     (
                                        [ ( 'gate' | Identifier ) | ( 'component'
                                        | Identifier ) ] 'is quiet for' '(' Time
                                        ')'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        '}' ] ';' ) ;
```

```
CompoundBehaviour                    ::=   ( Block
                                           [ 'with' '{'
                                           [ Comment { Comment } ]
                                           [ Annotation { Annotation } ]
                                           [ 'test objectives' ':' Identifier { ','
                                           Identifier } ';' ]
                                           [ 'name' Identifier ]
                                           [ PeriodicBehaviour { PeriodicBehaviour }
                                           ]
                                           [ ExceptionalBehaviour {
                                           ExceptionalBehaviour } ]
                                           '}' ] ) ;
PeriodicBehaviour                    ::=   ( 'every' '(' Time ')' Block
                                           [ 'with' '{'
                                           [ Comment { Comment } ]
                                           [ Annotation { Annotation } ]
                                           [ 'test objectives' ':' Identifier { ','
                                           Identifier } ';' ]
                                           [ 'name' Identifier ]
                                           '}' ] ) ;
AlternativeBehaviour                 ::=   ( 'alternatively' Block { 'or' Block }
                                           [ 'with' '{'
                                           [ Comment { Comment } ]
                                           [ Annotation { Annotation } ]
                                           [ 'test objectives' ':' Identifier { ','
                                           Identifier } ';' ]
                                           [ 'name' Identifier ]
                                           [ PeriodicBehaviour { PeriodicBehaviour }
                                           ]
                                           [ ExceptionalBehaviour {
                                           ExceptionalBehaviour } ]
                                           '}' ] ) ;
ParallelBehaviour                    ::=   ( 'run' Block { 'in' 'parallel' 'to' Block
                                           }
                                           [ 'with' '{'
                                           [ Comment { Comment } ]
                                           [ Annotation { Annotation } ]
                                           [ 'test objectives' ':' Identifier { ','
                                           Identifier } ';' ]
                                           [ 'name' Identifier ]
                                           [ PeriodicBehaviour { PeriodicBehaviour }
                                           ]
                                           [ ExceptionalBehaviour {
                                           ExceptionalBehaviour } ]
                                           '}' ] ) ;
BoundedLoopBehaviour                 ::=   ( 'repeat' Integer 'times' Block
                                           [ 'with' '{'
                                           [ Comment { Comment } ]
                                           [ Annotation { Annotation } ]
                                           [ 'test objectives' ':' Identifier { ','
                                           Identifier } ';' ]
                                           [ 'name' Identifier ]
                                           [ PeriodicBehaviour { PeriodicBehaviour }
                                           ]
                                           [ ExceptionalBehaviour {
                                           ExceptionalBehaviour } ]
                                           '}' ] ) ;
UnboundedLoopBehaviour               ::=   ( 'repeat' Block
                                           [ 'with' '{'
                                           [ Comment { Comment } ]
                                           [ Annotation { Annotation } ]
                                           [ 'test objectives' ':' Identifier { ','
                                           Identifier } ';' ]
```

```
                                    [ 'name' Identifier ]
                                    [ PeriodicBehaviour { PeriodicBehaviour }
                                    ]
                                    [ ExceptionalBehaviour {
                                    ExceptionalBehaviour } ]
                                    '}' ] ) ;
ConditionalBehaviour          ::=   ( 'if' Block
                                    [ ( ( 'else' Block ) ) | ( { 'else' 'if'
                                    Block } | ( 'else' Block ) ) ]
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    [ 'test objectives' ':' Identifier { ','
                                    Identifier } ';' ]
                                    [ 'name' Identifier ]
                                    [ PeriodicBehaviour { PeriodicBehaviour }
                                    ]
                                    [ ExceptionalBehaviour {
                                    ExceptionalBehaviour } ]
                                    '}' ] ) ;
Stop                          ::=   ( 'terminate'
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    [ 'test objectives' ':' Identifier { ','
                                    Identifier } ';' ]
                                    [ 'name' Identifier ]
                                    [ 'time constraints' '(' Identifier { ','
                                    Identifier } ')' ]
                                    '}' ] ';' ) ;
TestDescriptionReference      ::=   ( 'execute' Identifier
                                    [ '(' ArgumentSpecification { ','
                                    ArgumentSpecification } ')' ]
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    [ 'test objectives' ':' Identifier { ','
                                    Identifier } ';' ]
                                    [ 'time constraints' '(' Identifier { ','
                                    Identifier } ')' ]
                                    [ 'name' Identifier ]
                                    '}' ] ';' ) ;
VerdictAssignment             ::=   ( 'set verdict to' Identifier
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    [ 'test objectives' ':' Identifier { ','
                                    Identifier } ';' ]
                                    [ 'name' Identifier ]
                                    [ 'time constraints' '(' Identifier { ','
                                    Identifier } ')' ]
                                    '}' ] ';' ) ;
ActionReference               ::=   ( 'perform action' Identifier
                                    [ 'on component' Identifier ]
                                    [ 'with' '{'
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    [ 'test objectives' ':' Identifier { ','
                                    Identifier } ';' ]
                                    [ 'name' Identifier ]
                                    [ 'time constraints' '(' Identifier { ','
                                    Identifier } ')' ]
                                    '}' ] ';' ) ;
Interaction                   ::=   ( Identifier 'sends' ArgumentSpecification
```

```
                                        'to' Identifier { ',' Identifier }
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        '}' ] ';' ) ;
OptionalBehaviour                ::=    ( 'optionally' Block
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        '}' ]
                                        [ ExceptionalBehaviour {
                                        ExceptionalBehaviour } ]
                                        [ PeriodicBehaviour { PeriodicBehaviour }
                                        ] ';' ) ;
DefaultBehaviour                 ::=    ( 'default'
                                        [ 'on' Identifier ] Block
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        '}' ] ) ;
InterruptBehaviour               ::=    ( 'interrupt'
                                        [ 'on' Identifier ] Block
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'name' Identifier ]
                                        '}' ] ) ;
Break                            ::=    ( 'break'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'test objectives' ':' Identifier { ','
                                        Identifier } ';' ]
                                        [ 'time constraints' '(' Identifier { ','
                                        Identifier } ')' ]
                                        [ 'name' Identifier ]
                                        '}' ] ';' ) ;
TimeConstraint                   ::=    ( Identifier String
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ]
                                        '}' ) ;
Timer                            ::=    ( 'timer' Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
```

```
                                        '}' ] ';' ) ;
Time                                ::= ( Real Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'name' Identifier ]
                                        '}' ] ) ;
Real                                ::= (
                                        ['-'] {INT}+ '.' {INT}+ ) ;
GateInstance                        ::= ( 'gate' Identifier 'of type' Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
Block                               ::= (
                                        [ '
                                        [' String ']' ] '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        [ 'name' Identifier ] Behaviour {
                                        Behaviour }
                                        '}' ) ;
ComponentInstance                   ::= ( 'instantiate' Identifier 'as'
                                        ComponentInstanceRole 'of type' Identifier
                                        'having' '{'
                                        GateInstance { GateInstance }
                                        '}'
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ) ;
Integer                             ::= (
                                        ['-'] {INT}+ ) ;
DataInstance                        ::= ( 'instance' Identifier
                                        [ '(' Identifier { ',' Identifier } ')' ]
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ';' ) ;
Connection                          ::= ( 'connect' Identifier
                                        [ 'to' Identifier ] { 'and' Identifier }
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ] Identifier
                                        '}' ] ';' ) ;
ArgumentSpecification               ::= ( DataInstanceArgumentSpecification
                                        | DataSetArgumentSpecification
                                        | DataProxyArgumentSpecification ) ;
DataProxy                           ::= ( Identifier 'from' Identifier
                                        [ 'with' '{'
                                        [ Comment { Comment } ]
                                        [ Annotation { Annotation } ]
                                        '}' ] ) ;
DataInstanceArgumentSpecification   ::= ( 'instance' Identifier
                                        [ '(' Identifier { ',' Identifier } ')' ]
                                        [ 'with' '{'
```

```
                                    [ 'name' Identifier ]
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    '}' ] ) ;
DataProxyArgumentSpecification      ::= ( 'data bound to' Identifier
                                    [ 'with' '{'
                                    [ 'name' Identifier ]
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    '}' ] ) ;
DataSetArgumentSpecification        ::= ( 'any instance from data set' Identifier
                                    [ 'with' '{'
                                    [ 'name' Identifier ]
                                    [ Comment { Comment } ]
                                    [ Annotation { Annotation } ]
                                    '}' ] ) ;
ComponentInstanceRole               ::= ( 'SUT' | 'Tester' ) ;
ID                                  ::= (
                                    ['^'] ( 'a'-'z' | 'A'-'Z' | '_' ) { 'a'-
                                    'z' | 'A'-'Z' | '_' | '0'-'9' } ) ;
INT                                 ::= '0'-'9' ;
STRING                              ::= ( ( '"' | { ( '\\' | ( 'b' | 't' | 'n' |
                                    'f' | 'r' | 'u' | '"' | "'" | '\\' ) ) | (
                                    '\\' | '"' ) } | '"' ) | ( "'" | { ( '\\'
                                    | ( 'b' | 't' | 'n' | 'f' | 'r' | 'u' |
                                    '"' | "'" | '\\' ) ) | ( '\\' | "'" ) } |
                                    "'" ) ) ;
ML_COMMENT                          ::= ( '/*'  '*/') ;
SL_COMMENT                          ::= ( '//' ( '\\n' | '\\r' )
                                    [ ['\\r'] '\\n' ] ) ;
WS                                  ::= { ' '
                                    | '\\t'
                                    | '\\r'
                                    | '\\n' }+ ;
```

Annex C (informative): Bibliography

ETSI ES 202 553 (V1.2.1): "Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes".

ISO/IEC/IEEE 29119-3:2013: "Software and Systems Engineering - Software Testing; Part 3: Test Documentation".

OMG: "UML Testing Profile (UTP) V1.2", formal/2013-04-03.

History

Document history		
V1.1.1	February 2014	Membership Approval Procedure MV 20140418: 2014-02-17 to 2014-04-18
V1.1.1	April 2014	Publication