# ETSI ES 203 119-1 V1.6.1 (2022-05)


**Methods for Testing and Specification (MTS);**
**The Test Description Language (TDL);**
**Part 1: Abstract Syntax and Associated Semantics**

Reference
RES/MTS-1191v161

Keywords
language, MBT, methodology, model, 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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

# Contents

# Intellectual Property Rights

## Essential patents

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

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

## Trademarks

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

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



# Foreword

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



The present document is part 1 of a multi-part deliverable covering the Test Description Language, as identified below:

**Part 1: "Abstract Syntax and Associated Semantics";**

Part 2: "Graphical Syntax";

Part 3: "Exchange Format";

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

Part 5: "UML profile for TDL";

Part 6: "Mapping to TTCN-3";

Part 7: "Extended Test Configurations";

Part 8: "Textual Syntax".

# Modal verbs terminology

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

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

# 1 Scope

The present document specifies the abstract syntax of the Test Description Language (TDL) in the form of a meta-model based on the OMG® Meta Object Facility™ (MOF) [1]. It 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 to enable TDL tools such as documentation generators, specification analysers 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.

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

# 2 References

## 2.1 Normative references

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



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

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



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

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

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

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

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

[3] OMG® formal/2014-02-03: "Object Constraint Language™ (OCL), Version 2.4".

NOTE: Available at http://www.omg.org/spec/OCL/2.4/.

[4] Void.

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

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

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

[8] ETSI ES 203 119-8 (V1.1.1): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 8: Textual Syntax".

## 2.2 Informative references

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

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

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

[i.1] ETSI 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 (V3.1.1): "IMS Network Testing (INT); IMS NNI Interoperability Test Specifications; Part 2: Test Description for IMS NNI Interoperability".

[i.4] ETSI: The TDL Open Source Project Website (last visited 20.12.2021).

NOTE: Available at https://tdl.etsi.org/index.php/open-source.

# 3 Definition of terms, symbols and abbreviations



## 3.1 Terms

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



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

**action:** any procedure carried out by a component of a test configuration or an actor during test execution

**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

**effectively static:** any data specification that is statically determined (also recursively for all parameter bindings) and can be considered constant during the execution

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

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

**resolved data type:** data type of a data use, determined by a referenced data instance, data type, parameter, return type of a function, or the context where the data use is defined

**scope of behaviour:** components participating in a behaviour

**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

**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

**tester-input event:** event that occurs at a component in the role tester and determines the subsequent behaviour of this tester component

NOTE: Tester-input events in the present document are the following:

- Quiescence.
- TimeOut.
- An 'Interaction' with a 'Target' that in turn-via its 'GateReference'-refers to a 'ComponentInstance' in the role 'Tester'. If the source of an 'Interaction' is also a tester then it is not a tester-input event.

**<undefined>:** semantical concept denoting an undefined data value

## 3.2 Symbols

Void.



## 3.3 Abbreviations

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



| | |
|---|---|
| ADT | Abstract Data Type |
| EBNF | Extended Backus-Naur Form |
| ID | Identification |
| IEC | International Electrotechnical Commission |
| IMS | IP Multimedia Subsystem |
| ISO | International Organization for Standardization |
| MBT | Model-Based Testing |
| MOF | Meta Object Facility™ |
| OCL | Object Constraint Language™ |
| OMG | Object Management Group® |
| 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 |
| XML | eXtensible Markup Language |

# 4 Basic Principles

## 4.1 What is TDL?

TDL is a language that supports the design and documentation of formal test descriptions that may 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 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.

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 specification. On one hand, the concrete syntax of the TDL meta-model may 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 levels 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 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 is provided 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 the present document 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 definitions 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 and only one atomic behaviour is executing at any time.
- 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.
- An explicit verdict assignment may be used if in a certain case there is a need to override the implicit verdict setting mechanism (e.g. to assign 'inconclusive' or any user-defined verdict values).
- The data exchanged via interactions and used in parameters of test descriptions are represented as values of an abstract data type without further details of their underlying semantics, which is implementation-specific.
- There is no assumption about verdict arbitration, which is implementation-specific. If a deviation from the specified expected behaviour is detected, the subsequent behaviour becomes undefined. In this case an implementation might stop executing the TDL specification.

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) may 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 the first ('base') test description being invoked.

The elements in a TDL specification may be extended with tool, application, or framework specific information by means of annotations.



## 4.3 Principal Design Approach



The language TDL is designed following the meta-modelling approach which separates the language design into abstract syntax and concrete syntax on the one hand, and static semantics and dynamic semantics on the other hand. The abstract syntax of a language describes the structure of an expression defined in the language by means of abstract concepts and relationships among them, where a concrete syntax describes concrete representation of an expression defined in this language by means of textual, graphical, or tabular constructs which are mapped to concepts from the abstract syntax. The semantics describes the meaning of the individual abstract syntax concepts.

The realization of multiple representations by means of different syntactical notations for a single language requires a clear distinction between abstract syntax and concrete syntax. In a model-based approach to language design, the abstract syntax is defined by means of a meta-model. The meta-model of TDL defines the underlying structure of the abstract concepts represented by means of textual, graphical, or tabular constructs, without any restrictions on how these are expressed by means of e.g. keywords, graphical shapes, or tabular headings. The concrete syntax provides means for the representation of the abstract concepts in the form of textual, graphical, or tabular constructs and defines mappings between the concrete representations and the abstract concepts. This approach allows any concrete representation conforming to a given meta-model to be transformed into another representation conforming to that meta-model, such as graphical to textual, textual to tabular, tabular to graphical, etc. The transformations on the concrete syntax level have no impact on the semantics of the underlying abstract syntax concepts.

The semantics of a language is divided into static semantics and dynamic semantics. The static semantics defines further restrictions on the structure of abstract syntax concepts that cannot be expressed in syntax rules. The dynamic semantics defines the meaning of a syntactical concept when it is put into an execution environment.

The four pieces of the TDL design, concrete syntax, abstract syntax, static semantics, dynamic semantics, as well as extensions and mappings to other languages are represented in the standards series of TDL as follows (see figure 4.1):

- TDL-MM, part 1: Covers abstract syntax, static semantics and dynamic semantics.
- TDL-GR, part 2: Covers concrete syntax of graphical TDL.
- TDL-XF, part 3: Covers concrete syntax of the XML-based TDL exchange format.

- TDL-TO, part 4: Covers all parts of concrete/abstract syntax and static/dynamic semantics of the TDL Test Objective extension.
- TDL-UP4TDL, part 5: Covers the standardised mapping of TDL to UML with the UML Profile for TDL.
- TDL-T3, part 6: Covers the standardised mapping all TDL to TTCN-3.
- TDL-TC, part 7: Covers all parts of concrete/abstract syntax and static/dynamic semantics of the TDL Extended Test Configurations extension.
- TDL-TX, part 8: Covers the normative concrete syntax of textual TDL.

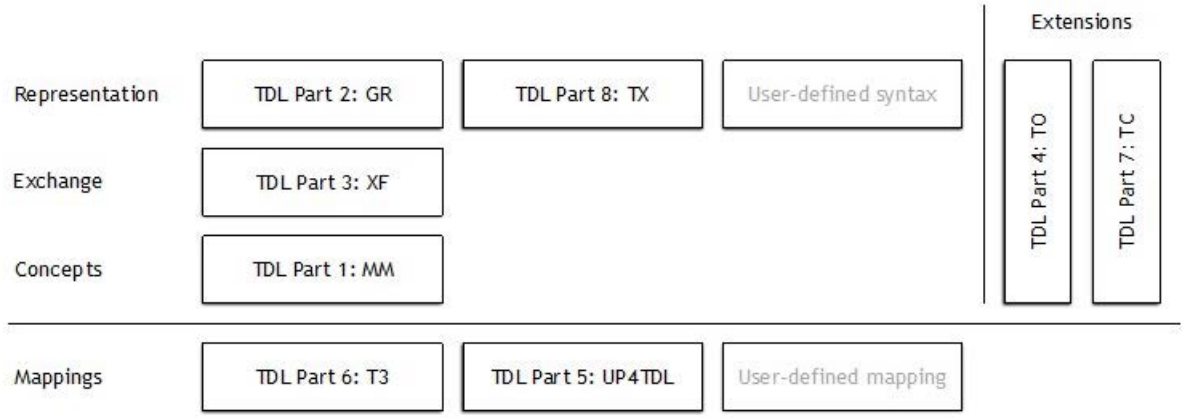


**Figure 4.1: The TDL standards and their positioning**



This decomposition of the TDL language design into the different standard parts allows for the development of integrated and stand-alone tools: editors for TDL specifications in graphical, textual, and user-defined concrete syntaxes, analysers of TDL specifications that check the consistency of TDL specifications, test documentation generators, test code generators to derive executable tests and others. In all cases the TDL exchange format [5] serves as the bridge between all TDL tools and to ensure tool interoperability (see figure 4.2).



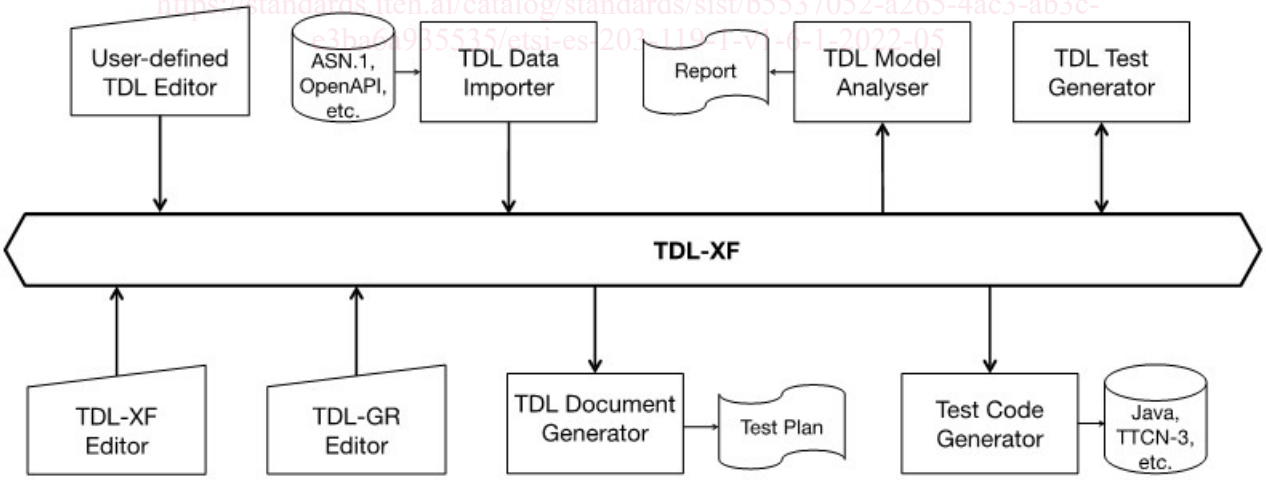


**Figure 4.2: A scalable TDL tool architecture**

## 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).

- Concepts of time, time constraints, and timers as well as their related operations (clause 7).
- Test configurations, on which test descriptions are executed (clause 8).
- A number of behavioural operations to specify the control flow of test descriptions (clause 9).
- A set of predefined instances of the TDL meta-model for test verdict, time, data types and functions over them that may be extended further by a user (clause 10).

Each language feature clause contains a brief introduction to the concepts defined in that clause. A set of class diagrams defines the concepts associated with the feature. For each concept, properties and relationships are specified and visualized in the diagrams (figures in the present document). The defining instance of a concept (with property compartment) appears only once in the set of diagrams. However, a concept may occur more than once in diagrams, in which case subsequent occurrences omit the icon and property compartment.

Besides the diagrams introducing the abstract syntax of the various TDL concepts formally, each clause is structured into the following paragraphs:

- Paragraph "Semantics" refers to the dynamic semantics of the concept defined in a declarative style hereafter. To emphasize the dynamic semantics aspect, sometimes the expression "at runtime" is used in the description. The description is augmented frequently with further explanations to ease reading interpretation of the document. These explanations are provided as NOTEs.
- Paragraph "Generalization" is derived from the abstract syntax diagram (figure) and lists the concept, which the defined concept is a specialization from. There is at most one generalization for any defined concept.
- Paragraph "Properties" is derived from the abstract syntax diagram (figure) and describes informally the meaning of the attributes that belong to the defined concept. For enumeration types, a paragraph "Literals" is used instead of the "Properties" paragraph to describe the enumeration literals and their meaning.
- Paragraph "Constraints" lists rules describing the static semantics of the concept, both in terms of informal descriptions and formally as OCL constraints.



## 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 the TDL concept in 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. |
| {readOnly} | The element shall be accessed read-only, i.e. shall not be modified. Used for derived properties. |
| Inv [Name]: | Formal definition of a constraint by means of OCL [3], where [Name] is a placeholder for the unique constraint name. |

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

## 4.6 Element Operations

The following operations shall be provided in an implementation of the TDL meta-model in order to ensure the semantic integrity of TDL models. The operations are also used as reusable shortcuts for the specification of the formalized constraints and are required for their interpretation, in addition to the operations provided by the standard library of OCL.

- Element **container**(): Element - applicable on any TDL 'Element', returns the 'Element' that contains the construct directly, except for the top level 'Element' which is not contained in any other 'Element'.
- Element **getParentTestDescription**(): TestDescription - applicable on any TDL 'Element', returns the 'TestDescription' that contains the construct directly or indirectly.
- DataUse **resolveDataType** (): DataType - applicable on any TDL 'DataUse', returns the 'DataType' resolved from the 'DataUse', or undefined if the data type cannot be resolved, which implies an invalid model.
- DataUse **isEffectivelyStatic** (): Boolean - applicable on any TDL 'DataUse', returns the 'Boolean' 'true' if the 'DataUse' is a 'StaticDataUse' or a 'DataElementUse' that refers to a 'DataType', a 'DataInstance', or does not specify a 'dataElement', and all 'ParameterBinding's fulfil the same condition recursively.
- Behaviour **isTesterInputEvent** (): Boolean - applicable on any TDL 'Behaviour', returns the 'Boolean' 'true' if the 'Behaviour' is a tester-input event as defined in the present document, and 'false' otherwise.
- Block **getParticipatingComponents** (): Set<ComponentInstance> - applicable on any TDL 'Block', returns all 'ComponentInstance's that participate in the 'Block' (as specified in clause 9.3.2).
- Behaviour **getParticipatingComponents** (): Set<ComponentInstance> - applicable on any TDL 'Behaviour', returns all 'ComponentInstance's that participate in the 'Behaviour'.
- Behaviour **getPermittedComponents** (): Set<ComponentInstance> - applicable on any TDL 'AlternativeBehaviour', 'OptionalBehaviour', or 'ExceptionalBehaviour', returns all 'ComponentInstance's that are permitted if the starting behaviour is specified. In case the starting behaviour is not specified all 'ComponentInstance's are returned.
- DataType **allConstraints** (): Set<Constraint> - applicable on any TDL 'DataType', returns all 'Constraints's that are associated with the 'DataType', including inherited ones.
- StructuredDataType **allMembers** (): Set<Member> - applicable on any TDL 'StructuredDataType', returns all 'Member's that are associated with the 'StructuredDataType', including inherited ones.
- GateType **allDataTypes** (): Set<DataType> - applicable on any TDL 'GateType', returns all 'DataTypes's that are associated with the 'GateType', including inherited ones.
- ComponentType **allGates** (): Set<GateInstance> - applicable on any TDL 'ComponentType', returns all 'GateInstance's that are associated with the 'ComponentType', including inherited ones.
- ComponentType **allTimers** (): Set<Timer> - applicable on any TDL 'ComponentType', returns all 'Timer's that are associated with the 'ComponentType', including inherited ones.
- ComponentType **allVariables** (): Set<Variable> - applicable on any TDL 'ComponentType', returns all 'Variable's that are associated with the 'ComponentType', including inherited ones.
- TestConfiguration **compatibleWith** (tc: TestConfiguration, cb: Set<ComponentInstanceBinding>): Boolean - applicable on any TDL 'TestConfiguration', returns 'true' if this 'TestConfiguration' is compatible with the provided 'TestConfiguration' 'tc' under the provided 'ComponentInstanceBinding's 'cb' between component instances of this 'TestConfiguration' and the TestConfiguration 'tc' (as specified in clause 9.4.11).
- Extension **isExtending** (e: PackageableElement): Boolean - applicable on any TDL 'Extension', returns 'true' if the value of 'extending' property of this 'Extension' is the same as the provided 'PackageableElement' 'e', or if the value of 'extending' property of this 'Extension' contains an 'Extension' that extends the provided 'PaclageableElement' 'e' (the condition is determined by calling this operation).