

**Date:** November 2024



# UML Testing Profile 2 (UTP 2)

*Version 2.3*

---

**OMG Document Number:** ptc/24-12-01

**Normative reference:** <https://www.omg.org/spec/UTP2>

---

Copyright © 2014-2023, Fraunhofer FOKUS  
Copyright © 2014-2023, Grand Software Testing  
Copyright © 2014-2023, Hamburg University of Applied Science  
Copyright © 2014-2023, KnowGravity Inc.  
Copyright © 2014-2024, Object Management Group, Inc.  
Copyright © 2014-2023, PTC Inc.  
Copyright © 2014-2023, Simula Research Lab  
Copyright © 2014-2023, SELEX  
Copyright © 2014-2023, SOFTEAM  
Copyright © 2014-2023, University of Cantabria

## USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

## LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

## PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

## GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

## DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

## RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 9C Medway Road, PMB 274, Milford, MA 01757, U.S.A.

## TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: [http://www.omg.org/legal/tm\\_list.htm](http://www.omg.org/legal/tm_list.htm). All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

## COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

## **OMG's Issue Reporting Procedure**

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page <https://www.omg.org>, under Specifications, Report a Bug/Issue.



# Table of Contents

1	Scope .....	1
2	Conformance .....	3
3	Terms and Definitions .....	5
4	References .....	9
4.1	Normative References .....	9
4.2	Informative References .....	9
5	Symbols .....	12
6	Additional Information .....	13
6.1	How to read this document .....	13
6.2	Typographical conventions .....	13
6.3	Typical Use Cases of UTP 2 .....	14
6.4	Relation to testing-relevant standards .....	17
6.5	Relation to model-based testing .....	19
6.6	Relation to keyword-driven testing .....	20
6.7	Relation to the MARTE Profile .....	20
6.8	Acknowledgements .....	21
7	(Informative) Conceptual Model .....	23
7.1	Test Planning.....	23
7.1.1	Test Analysis .....	23
7.1.1.1	Test Context Overview .....	23
7.1.1.2	Concept Descriptions.....	25
7.1.2	Test Design.....	26
7.1.2.1	Test Design Facility Overview .....	26
7.1.2.2	Concept Descriptions.....	27
7.2	Test Architecture.....	28
7.2.1	Test Architecture Overview.....	28
7.2.2	Concept Descriptions.....	28
7.3	Test Behavior .....	30
7.3.1	Test Cases.....	30
7.3.1.1	Test Case Overview .....	30
7.3.1.2	Concept Descriptions.....	30
7.3.2	Test-specific Procedures.....	31
7.3.2.1	Test Procedures.....	31
7.3.2.2	Concept Descriptions.....	32
7.3.3	Test-specific Actions .....	35
7.3.3.1	Overview of test-specific actions.....	35
7.3.3.2	Concept Descriptions.....	36
7.4	Test Data .....	38
7.4.1	Test Data Concepts.....	38
7.4.2	Concept Descriptions.....	39
7.5	Test Evaluation .....	41
7.5.1	Arbitration Specifications.....	41
7.5.1.1	Arbitration & Verdict Overview .....	41
7.5.1.2	Concept Descriptions .....	42
7.5.2	Test Logging.....	43
7.5.2.1	Test Log Overview .....	43
7.5.2.2	Concept Descriptions.....	45
8	Profile Specification .....	47
8.1	Language Architecture .....	47
8.2	Profile Summary .....	48
8.3	Test Planning.....	50

8.3.1	Test Analysis .....	50
8.3.1.1	Test Context Overview .....	51
8.3.1.2	Test-specific Contents of Test Context .....	51
8.3.1.3	Test Objective Overview .....	52
8.3.1.4	Stereotype Specifications .....	53
8.3.1.4.1	TestContext .....	53
8.3.1.4.2	TestObjective .....	55
8.3.1.4.3	TestObjectiveTrace .....	55
8.3.1.4.4	TestRequirement .....	56
8.3.1.4.5	TestSet .....	57
8.3.1.4.6	verifies .....	58
8.3.2	Test Design .....	58
8.3.2.1	Test Design Facility .....	59
8.3.2.2	Generic Test Design Capabilities .....	59
8.3.2.3	Predefined high-level Test Design Techniques .....	60
8.3.2.4	Predefined data-related Test Design Techniques .....	60
8.3.2.5	Predefined state-transition-based Test Design Techniques .....	61
8.3.2.6	Predefined experience-based Test Design Techniques .....	62
8.3.2.7	Stereotype Specifications .....	63
8.3.2.7.1	BoundaryValueAnalysis .....	63
8.3.2.7.2	CauseEffectAnalysis .....	63
8.3.2.7.3	ChecklistBasedTesting .....	63
8.3.2.7.4	ClassificationTreeMethod .....	64
8.3.2.7.5	CombinatorialTesting .....	64
8.3.2.7.6	DecisionTableTesting .....	64
8.3.2.7.7	EquivalenceClassPartitioning .....	65
8.3.2.7.8	ErrorGuessing .....	65
8.3.2.7.9	ExperienceBasedTechnique .....	65
8.3.2.7.10	ExploratoryTesting .....	65
8.3.2.7.11	GenericTestDesignDirective .....	66
8.3.2.7.12	GenericTestDesignTechnique .....	66
8.3.2.7.13	NSwitchCoverage .....	66
8.3.2.7.14	PairwiseTesting .....	67
8.3.2.7.15	StateCoverage .....	67
8.3.2.7.16	StateTransitionTechnique .....	67
8.3.2.7.17	TestDesignDirective .....	68
8.3.2.7.18	TestDesignDirectiveStructure .....	69
8.3.2.7.19	TestDesignInput .....	70
8.3.2.7.20	TestDesignTechnique .....	70
8.3.2.7.21	TestDesignTechniqueStructure .....	71
8.3.2.7.22	TransitionCoverage .....	71
8.3.2.7.23	TransitionPairCoverage .....	71
8.3.2.7.24	UseCaseTesting .....	72
8.4	Test Architecture .....	72
8.4.1	Test Architecture Overview .....	72
8.4.2	Stereotype Specifications .....	73
8.4.2.1	RoleConfiguration .....	73
8.4.2.2	TestComponent .....	74
8.4.2.3	TestComponentConfiguration .....	74
8.4.2.4	TestConfiguration .....	75
8.4.2.5	TestConfigurationRole .....	75
8.4.2.6	TestItem .....	76
8.4.2.7	TestItemConfiguration .....	76
8.5	Test Behavior .....	76
8.5.1	Test-specific Procedures .....	77
8.5.1.1	Test Case Overview .....	77

8.5.1.2	Stereotype Specifications.....	78
8.5.1.2.1	TestProcedure.....	78
8.5.1.2.2	TestCase.....	79
8.5.1.2.3	TestExecutionSchedule.....	81
8.5.2	Procedural Elements.....	82
8.5.2.1	Procedural Elements Overview.....	83
8.5.2.2	Compound Procedural Elements Overview.....	84
8.5.2.3	Stereotype Specifications.....	85
8.5.2.3.1	Alternative.....	85
8.5.2.3.2	AtomicProceduralElement.....	86
8.5.2.3.3	CompoundProceduralElement.....	86
8.5.2.3.4	Loop.....	87
8.5.2.3.5	Negative.....	87
8.5.2.3.6	OpaqueProceduralElement.....	88
8.5.2.3.7	Parallel.....	88
8.5.2.3.8	ProceduralElement.....	89
8.5.2.3.9	ProcedureInvocation.....	90
8.5.2.3.10	Sequence.....	91
8.5.2.4	Enumeration Specifications.....	91
8.5.3	Test-specific Actions.....	92
8.5.3.1	Test-specific actions Overview.....	92
8.5.3.2	Tester Controlled Actions.....	93
8.5.3.3	Test Item Controlled Actions.....	94
8.5.3.4	Stereotype Specifications.....	95
8.5.3.4.1	CheckPropertyAction.....	95
8.5.3.4.2	CreateLogEntryAction.....	96
8.5.3.4.3	CreateStimulusAction.....	97
8.5.3.4.4	ExpectResponseAction.....	98
8.5.3.4.5	SuggestVerdictAction.....	101
8.5.3.5	Enumeration Specifications.....	102
8.6	Test Data.....	102
8.6.1	Data Specifications.....	102
8.6.1.1	Data Specifications Overview.....	102
8.6.1.2	Stereotype Specifications.....	103
8.6.1.2.1	Complements.....	103
8.6.1.2.2	DataPartition.....	103
8.6.1.2.3	DataPool.....	104
8.6.1.2.4	DataProvider.....	104
8.6.1.2.5	DataSpecification.....	104
8.6.1.2.6	Extends.....	105
8.6.1.2.7	Morphing.....	105
8.6.1.2.8	Refines.....	106
8.6.2	Data Values.....	106
8.6.2.1	Data Value Extensions.....	106
8.6.2.2	Stereotype Specifications.....	107
8.6.2.2.1	AnyValue.....	107
8.6.2.2.2	overrides.....	108
8.6.2.2.3	RegularExpression.....	109
8.7	Test Evaluation.....	109
8.7.1	Arbitration Specifications.....	109
8.7.1.1	Test Procedure Arbitration Specifications.....	110
8.7.1.1.1	Arbitration Facility Overview.....	112
8.7.1.1.2	Stereotype Specifications.....	112
8.7.1.2	Procedural Element Arbitration Specifications.....	118
8.7.1.2.1	Arbitration of AtomicProceduralElements.....	118
8.7.1.2.2	Arbitration of CompoundProceduralElements.....	119

8.7.1.2.3	Stereotype Specifications .....	120
8.7.1.3	Test-specific Action Arbitration Specifications .....	124
8.7.1.3.1	Arbitration of Test-specific Actions .....	124
8.7.1.3.2	Stereotype Specifications .....	125
8.7.2	Test Logging .....	127
8.7.2.1	Test Logging Overview .....	127
8.7.2.2	Test Log Entries Overview .....	128
8.7.2.3	Test Log Entries Details .....	129
8.7.2.4	Invocation Test Log Entry Details .....	130
8.7.2.5	Stereotype Specifications .....	130
8.7.2.5.1	TestLogElement .....	130
8.7.2.5.2	TestLog .....	131
8.7.2.5.3	TestSetLog .....	132
8.7.2.5.4	TestCaseLog .....	132
8.7.2.5.5	TestLogStructure .....	133
8.7.2.5.6	TestLogEntry .....	133
8.7.2.5.7	AtomicProceduralElementLogEntry .....	134
8.7.2.5.8	InvocationLogEntryStructure .....	134
8.7.2.5.9	FormalParameterReference .....	134
8.7.2.5.10	InvocationLogEntry .....	135
8.7.2.5.11	ActualParameterValue .....	135
8.7.2.5.12	ProcedureInvocationLogEntryStructure .....	135
8.7.2.5.13	ProcedureInvocationLogEntry .....	136
8.7.2.5.14	MessageEventLogEntryStructure .....	136
8.7.2.5.15	MessageEventLogEntry .....	136
8.7.2.5.16	CreateStimulusLogEntry .....	137
8.7.2.5.17	ActualResponseLogEntry .....	137
8.7.2.5.18	CheckPropertyLogEntry .....	137
8.7.2.5.19	SuggestVerdictLogEntry .....	138
8.7.2.5.20	CreateLogEntryLogEntry .....	138
8.7.2.5.21	OpaqueProceduralElementLogEntry .....	138
8.7.2.5.22	TestLogStructureBinding .....	139
8.8	Test Directives .....	139
8.8.1	Test Directive Facility .....	140
8.8.2	Stereotype Specifications .....	140
8.8.2.1	TestDirective .....	140
8.8.2.2	TestDirectiveStructure .....	141
8.8.2.3	TestTechnique .....	141
8.8.2.4	TestTechniqueStructure .....	141
9	Model Libraries .....	143
9.1	UTP Types Library .....	143
9.1.1	Predefined types .....	143
9.1.2	Predefined verdict instances .....	143
9.2	UTP Auxiliary Library .....	144
9.2.1	UTP Auxiliary Library .....	144
9.2.1.1	The UTP auxiliary library .....	144
9.2.1.2	ISTQB Library .....	145
9.2.1.2.1	Overview of the ISTQB library .....	145
9.2.1.3	Test Design Facility Library .....	149
9.2.1.3.1	The UTP test design facility library .....	149
9.2.1.3.2	Predefined Test Design Techniques .....	149
9.2.1.3.3	Predefined Test Design Technique Structures .....	151
Annex A (Informative): Examples .....		153
A.1	Croissants Example .....	153
A.1.1	The Test Item .....	153
A.1.1.1	Given Requirements on the Test Item .....	153

A.1.2	Test Requirements .....	154
A.1.2.1	Given Test Objectives .....	154
A.1.2.2	Given Requirements .....	154
A.1.3	Test Design .....	155
A.1.3.1	Test Design Strategies shown on "Test Strategy" .....	155
A.1.3.2	Test Directives shown on "Test Strategy" .....	155
A.1.4	Test Configuration .....	155
A.1.5	Test Cases .....	156
A.1.5.1	Test Set "Manual croissants test" .....	156
A.2	LoginServer Example .....	158
A.2.1	Requirements Specification .....	159
A.2.2	Test Planning .....	159
A.2.3	Test Analysis .....	160
A.2.3.1	Derivation and Modeling of Test Requirements .....	160
A.2.3.2	Modeling the Type System and Logical Interfaces .....	162
A.2.3.3	Modeling Test Data .....	163
A.2.4	Test Design .....	164
A.2.4.1	Test Architecture and Test Configuration .....	164
A.2.4.2	Specification of Complex Test Data .....	165
A.2.4.3	Test Requirements Realization .....	166
A.2.4.4	Design of Test Case Procedures .....	167
A.2.5	Mapping to TTCN-3 .....	168
A.2.5.1	Mapping the Test Type System .....	169
A.2.5.2	Mapping Interface Descriptions .....	169
A.2.5.3	Mapping the Test Architecture .....	169
A.2.5.4	Mapping the Test Data Specification .....	170
A.2.5.5	Mapping Test Cases and Test Configuration .....	170
A.3	Videoconferencing Example .....	172
A.3.1	Given Requirements on the Test Item .....	172
A.3.2	Modeling the Structure of the System .....	172
A.3.3	Modeling the Behavior of the System .....	173
A.3.4	The TRUST Test Generator .....	175
A.3.5	Mapping to Code .....	176
A.3.6	References .....	176
A.4	Subsea Production System Example .....	177
A.4.1	Description of Case Study .....	177
A.4.2	Functionality to Test .....	177
A.4.3	Test Design Inputs .....	178
A.4.4	Generation of Test Sets and Abstract Test Cases .....	179
A.4.5	References .....	181
A.5	ATM Example .....	182
A.5.1	General .....	182
A.5.2	Unit Test Example .....	183
A.5.3	Integration Testing Example .....	187
A.5.4	System Test Example .....	192
A.5.5	References .....	197
Annex B (Informative): Mappings .....		199
B.1	Mapping between UTP 1 and UTP 2 .....	199
Annex C (Informative): Value Specification Extensions .....		203
C.1	Profile Summary .....	203
C.2	Non-normative data value extensions .....	203
C.1.1	Overview of non-normative ValueSpecification Extensions .....	203
C.2.2	Stereotype Specifications .....	204
C.2.2.1	ChoiceOfValues .....	204
C.2.2.2	CollectionExpression .....	205
C.2.2.3	ComplementedValue .....	205

C.2.2.4	MatchingCollectionExpression .....	205
C.2.2.5	RangeValue .....	206
C.2.3	Enumeration Specifications .....	206
Annex D:	Deprecated Elements .....	208
Annex E:	Index .....	209

# Preface

## OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel™); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at <http://www.omg.org/>.

## OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

<http://www.omg.org/spec>

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters  
9C Medway Road, PMB 274  
Milford, MA, 01757  
USA  
Tel: +1-781-444-0404  
Fax: +1-781-444-0320  
Email: [pubs@omg.org](mailto:pubs@omg.org)

Certain OMG specifications are also available as ISO standards. Please consult <http://www.iso.org>





# 1 Scope

In 2001, a working group at the OMG started developing a UML Profile dedicated to Model-based testing, called UML Testing Profile (UTP). It is a standardized language based on OMG's Unified Modeling Language (UML) for designing, visualizing, specifying, analyzing, constructing, and documenting the [artifacts](#) commonly used in and required for various testing approaches, in particular model-based testing (MBT) approaches. UTP has the potential to assume the same important role for model-based testing approaches as UML assumes for model-driven system engineering.

UTP is a part of the UML ecosystem (see figure below), and as such, it can be combined with other profiles of that ecosystem in order to associate test-related [artifacts](#) with other relevant system [artifacts](#), e.g. requirements, risks, use cases, business processes, system specifications etc. This enables requirements engineers, system engineers and test engineers to bridge the communication gap among different engineering disciplines.



**Figure 1.1 - The UML Ecosystem**

As the interest of industry in model-based testing approaches and languages increased, UTP attracted more and more users. UTP was the first standardized language for model-based approaches to help in the validation and verification of software-intensive systems. Model-based test specifications expressed with the UML Testing Profile are independent of any methodology, domain, environment, or type of system.

Eight years later, the UTP working group (WG) has agreed on consolidating the experiences and achievements of UTP in order to justify the move from UTP 1.2 to a successor specification. These efforts resulted in a Request For Information (RFI) for UML Testing Profile 2 (UTP 2), which was aimed at eliciting and gathering the shortcomings of the current UTP and the most urgent requirements for a successor specification from the OMG and model-based testing community.

Some of the main issues in the RFI responses are that UTP 2 should:

- be able to design test models of different [test levels](#)
- address testing of non-functional requirements
- be able to reuse [test logs](#) for further test evaluation and test generation
- meet industry-relevant standards
- integrate with SysML for requirements traceability
- and so forth

The UML Testing Profile 2 (UTP 2) was designed to meet the requirements derived from the RFI responses.

People may use the UML Testing Profile in addition to UML to:

- Specify the design and the configuration of a test system: Designing a test system includes the identification of the [test item](#) (also known as system under test or abbreviated as SUT), its boundaries, the derivation of [test components](#), and the identification of communication channels between interconnected [test items](#) [test components](#) over which data can be exchanged.
- Build the model-based test plans on top of already existing system models: The possibility to reuse already existing (system) [artifacts](#), e.g. requirements, interface definitions, type definitions etc.
- Model [test cases](#): The specification of [test cases](#) is an essential task of each test process in order to assess the quality of the [test item](#) and to verify whether the [test item](#) complies with its specification.
- Model test environments: A test environment contains hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test (according to IEEE 610).
- Model deployment specifications of test-specific [artifacts](#): By relying on the UML's deployment specification capabilities, the actual deployment of a test system can be done in a model-based way.
- Model [data](#): Modeling of [data](#) includes the data values being used as stimuli into the [test item](#) as well as for [responses](#) expected from the [test item](#) such as the test oracle.
- Provide necessary information pertinent to test scheduling optimization: Test scheduling optimization can be based on priorities, risk-related information, costs etc.
- Document [test case](#) execution results: To associate [test cases](#) with the actual outcome of their execution within the very same model in order to perform further analysis, calculate specific metrics, etc.
- Document traceability to requirements and other UML model [artifacts](#): Requirements traceability within test specification is important to document and evaluate test coverage and to calculate other metrics such as progress reports. Native traceability is given by the underlying UML capabilities. UTP does not offer different concepts for traceability other than that provided by UML.

The intended audience for the UML Testing Profile are users who are able to read model-based test specifications expressed within the UML Testing Profile models including:

- Test engineers
- Requirements Engineers
- System/Software Engineers
- Domain experts
- Customer/Stakeholder
- Certification authorities
- Testing tools ([test case](#) generators, [data](#) generators, schedulers, reporting engines, test script generators, +-\*etc.).

The intended audience of this UML Testing Profile specification itself includes, among others:

- People who want to implement UML Testing Profile-compliant tools.
- People who need to/want to/like to teach the UML Testing Profile.
- People who want to improve the UML Testing Profile specification.
- People who want to tailor the UML Testing Profile to satisfy needs of their specific project/domain/process.

## 2 Conformance

As a native profile specification of the UML, the UTP 2 has to abide by the conformance types declared for compliant UML profiles. The corresponding conformance types of UML can be found in section 2 "Conformance" of the current UML specification [\[UML\]](#). This guarantees that the underlying environment of any UTP 2 implementation is a UML modeling environment that is conformant with the UML. The UTP 2 adopted version of UML's conformance types are defined as follows:

- Abstract syntax conformance: All concrete stereotypes and tags are implemented in the profile implementation.
- Concrete syntax conformance: Support for the visual representation (i.e. icons) of the UTP concepts is provided by the profile implementation.
- Model interchange conformance: (delegated to underlying UML).
- Diagram interchange conformance: (delegated to underlying UML).
- Semantic conformance: All UTP [constraints](#) are enforced, either directly in the model with OCL (assuming underlying OCL support) or indirectly by any other suitable means of the underlying modeling environment.

In addition to the fundamental conformance types of the UML and its profiling mechanism, UTP 2 specifies two compliance levels for its respective concepts:

- Mandatory: concepts that are deemed mandatory have to be implemented in order to claim UTP 2 compliance.
- Optional: concepts that are deemed optional might be implemented. If they are implemented, they have to be implemented exactly how they have been specified by the UTP 2 specification - i.e., optional concepts are still normative, but when they are implemented, they have to abide by the conformance types imposed by the underlying UML and its profiling mechanism.

The decisions, which concepts are considered as mandatory and optional, have been based on the typical use cases of UTP 2 (see section 6.3 [Typical Use Cases of UTP 2](#)). The main objective of UTP 2 is to design [test cases](#), potentially in an automated manner, and to describe the test architecture in order to execute [test cases](#), potentially in an automated manner. Except from that, UTP 2 provides further helpful concepts for the design and implementation of a test environment that supports various activities of the test process, such as test analysis, manual and automated test design, test execution and evaluation. The concepts required for these activities are grouped by corresponding sections within this specification. The following relates the test process activities with the respective sections of the UTP 2 specification and indicates whether a feature (a set of concepts grouped in a section) is normative, mandatory, or optional:

Test Process Phase	Normative	Mandatory
• <a href="#">Test Analysis</a> Activities		
- Section 8.3.1 Test Analysis	X	-
• <a href="#">Test Design</a> Activities		
- Section 8.3.2 Test Design	X	-
- Section 8.4 Test Architecture	X	X
- Section 8.5.1 Test-specific Procedures	X	X
- Section 8.5.1 Procedural Elements	X	X
- Section 8.5.1 Test-specific Actions	X	X
- Section 8.6.1 Data Specifications	X	-
• Test Execution and Evaluation Activities		
- Section 8.6.2 Data Values	X	-
- Annex C Non-normative <a href="#">data</a> value extensions	-	-
- Section 8.7.1 Arbitration Specifications	X	-
- Section 8.7.2 Test Logging	X	-

In addition to these concepts, UTP 2 specifies three model libraries for UTP 2. The conformance considerations for the libraries are as follows:

UTP 2 Model Libraries	Normative	Mandatory
• Section 9.1 UTP Types Library	X	X
• Section 9.2 UTP Auxiliary Library	X	-

Any implementation that wants to claim conformance with UTP 2 specification has to abide by the adopted UTP 2 conformance types for each normative concept. If the concept is deemed mandatory in addition, any implementation that wants to claim conformance with the UTP 2 specification, has to provide those mandatory concepts to the user.

### 3 Terms and Definitions

The following terms and definitions are a summary of the Conceptual Model described in clause 7. For further examples and details refer to the respective sub-section in Clause 7.

Name	Description	Source
abstract test case	A <a href="#">test case</a> that declares at least one <a href="#">formal parameter</a> .	UTP 2
abstract test configuration	A <a href="#">test configuration</a> that specifies the <a href="#">test item</a> , <a href="#">test components</a> and their interconnections as well as configuration data that should be abstract test data.	UTP 2
actual data pool	A specification of an actual implementation of a <a href="#">data pool</a> .	UTP 2
actual parameter	A concrete value that is passed over to the <a href="#">procedure</a> and replaces the <a href="#">formal parameter</a> with its concrete value.	UTP 2
alternative	A <a href="#">compound procedural element</a> that executes only a subset of its contained <a href="#">procedural elements</a> based on the evaluation of a <a href="#">boolean expression</a> .	UTP 2
arbitration specification	A set of rules that calculates the eventual <a href="#">verdict</a> of an executed <a href="#">test case</a> , test set or procedural element.	UTP 2
artifact	An object produced or modified during the execution of a process.	UTP 2
atomic procedural element	A <a href="#">procedural element</a> that cannot be further decomposed.	UTP 2
boolean expression	An expression that may be evaluated to either of these values: "TRUE" or "FALSE".	UTP 2
check property action	A <a href="#">test action</a> that instructs the tester to check the conformance of a <a href="#">property</a> of the <a href="#">test item</a> and to set the <a href="#">procedural element verdict</a> according to the result of this check.	UTP 2
complement	A <a href="#">morphism</a> that inverts <a href="#">data</a> (i.e., that replaces the <a href="#">data items</a> of a given set of <a href="#">data items</a> by their opposites).	UTP 2
compound procedural element	A <a href="#">procedural element</a> that can be further decomposed.	UTP 2
concrete test case	A <a href="#">test case</a> that declares no <a href="#">formal parameter</a> .	UTP 2
concrete test configuration	A <a href="#">test configuration</a> that specifies the <a href="#">test item</a> , <a href="#">test components</a> and their interconnections as well as configuration data that should be concrete <a href="#">data</a> .	UTP 2
constraint	An assertion that indicates a restriction that must be satisfied by any valid realization of the model containing the <a href="#">constraint</a> .	[UML]
create log entry action	A <a href="#">test action</a> that instructs the tester to record the execution of a <a href="#">test action</a> , potentially including the outcome of that <a href="#">test action</a> in the <a href="#">test case log</a> .	UTP 2
create stimulus action	A <a href="#">test action</a> that instructs the tester to submit a <a href="#">stimulus</a> (potentially including <a href="#">data</a> ) to the <a href="#">test item</a> .	UTP 2
data	A usually named set of <a href="#">data items</a> .	UTP 2
data item	Either a value or an instance.	UTP 2
data partition	A role that some <a href="#">data</a> plays with respect to some other <a href="#">data</a> (usually being a subset of this other <a href="#">data</a> ) with respect to some <a href="#">data specification</a> .	UTP 2
data pool	Some <a href="#">data</a> that is an explicit or implicit composition of other <a href="#">data items</a> .	UTP 2
data provider	A <a href="#">test component</a> that is able to deliver (i.e., either select and/or generate) <a href="#">data</a> according to a <a href="#">data specification</a> .	UTP 2
data specification	A named <a href="#">boolean expression</a> composed of a <a href="#">data type</a> and a set of <a href="#">constraints</a> applicable to some <a href="#">data</a> in order to determine whether or not its <a href="#">data items</a> conform to this <a href="#">data specification</a> .	UTP 2
data type	A type whose instances are identified only by their value.	[UML]

Name	Description	Source
duration	The duration from the start of a <a href="#">test action</a> until its completion.	UTP 2
Error	An indication that an unexpected exception has occurred while executing a specific <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> .	UTP 2
executing entity	An <a href="#">executing entity</a> is a human being or a machine that is responsible for executing a <a href="#">test case</a> or a <a href="#">test set</a> .	UTP 2
expect response action	A <a href="#">test action</a> that instructs the tester to check the occurrence of one or more particular <a href="#">responses</a> from the <a href="#">test item</a> within a given time window and to set the <a href="#">procedural element verdict</a> according to the result of this check.	UTP 2
extension	A <a href="#">morphism</a> that increases the amount of <a href="#">data</a> (i.e., that adds more <a href="#">data items</a> to a given set of <a href="#">data items</a> ).	UTP 2
Fail	A <a href="#">verdict</a> that indicates that the <a href="#">test item</a> did not comply with the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> during execution.	UTP 2
formal parameter	A placeholder within a <a href="#">procedure</a> that allows for execution of the <a href="#">procedure</a> with different <a href="#">formal parameters</a> that are provided by the <a href="#">procedure invocation</a> .	UTP 2
Inconclusive	A <a href="#">verdict</a> that indicates that the compliance of a <a href="#">test item</a> against the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> could not be determined during execution.	UTP 2
loop	A <a href="#">compound procedural element</a> that repeats the execution of its contained <a href="#">procedural elements</a> .	UTP 2
main procedure invocation	A <a href="#">procedure invocation</a> that is considered as the main part of a <a href="#">test case</a> by the <a href="#">test case arbitration specification</a> .	UTP 2
morphism	A structure-preserving map from one mathematical structure to another.	[WikiM]
negative	A <a href="#">compound procedural element</a> that prohibits the execution of its contained <a href="#">procedural elements</a> in the specified structure.	UTP 2
None	A <a href="#">verdict</a> that indicates that the compliance of a <a href="#">test item</a> against the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> has not yet been determined (i.e., it is the initial value of a <a href="#">verdict</a> when a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> was started).	UTP 2
parallel	A <a href="#">compound procedural element</a> that executes its contained <a href="#">procedural elements</a> in parallel to each other.	UTP 2
Pass	A <a href="#">verdict</a> that indicates that the <a href="#">test item</a> did comply with the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> during execution.	UTP 2
PE end duration	The <a href="#">duration</a> between the end of the execution of a <a href="#">procedural element</a> and the end of the execution of the subsequent <a href="#">procedural element</a> .	UTP 2
PE start duration	The <a href="#">duration</a> between the end of the execution of a <a href="#">procedural element</a> and the beginning of the execution of the subsequent <a href="#">procedural element</a> .	UTP 2
postcondition	A <a href="#">boolean expression</a> that is guaranteed to be True after a <a href="#">test case</a> execution has been completed.	UTP 2
precondition	A <a href="#">boolean expression</a> that must be met before a <a href="#">test case</a> may be executed.	UTP 2
procedural element	An instruction to do, to observe, and/or to decide.	UTP 2
procedural element verdict	A <a href="#">verdict</a> that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test action</a> on a <a href="#">test item</a> .	UTP 2
procedure	A specification that constrains the execution order of a number of <a href="#">procedural elements</a> .	UTP 2

Name	Description	Source
procedure invocation	An atomic procedural element of a procedure that invokes another procedure and waits for its completion.	UTP 2
property	A basic or essential attribute shared by all members of a class of <a href="#">test items</a> .	UTP 2
refinement	A <a href="#">morphism</a> that decreases the amount of <a href="#">data</a> (i.e., that removes <a href="#">data items</a> from a given set of <a href="#">data items</a> ).	UTP 2
response	A set of <a href="#">data</a> that is sent by the <a href="#">test item</a> to its environment (often as a reaction to a <a href="#">stimulus</a> ) and that is typically used to assess the behavior of the <a href="#">test item</a> .	UTP 2
sequence	A <a href="#">compound procedural element</a> that executes its contained <a href="#">procedural elements</a> sequentially.	UTP 2
setup procedure invocation	A <a href="#">procedure invocation</a> that is considered as part of the setup by the <a href="#">arbitration specification</a> and that is invoked before any <a href="#">main procedure invocation</a> .	UTP 2
stimulus	A set of <a href="#">data</a> that is sent to the <a href="#">test item</a> by its environment (often to cause a <a href="#">response</a> as a reaction) and that is typically used to control the behavior of the <a href="#">test item</a> .	UTP 2
suggest verdict action	A <a href="#">test action</a> that instructs the tester to suggest a particular <a href="#">procedural element verdict</a> to the <a href="#">arbitration specification</a> of the <a href="#">test case</a> for being taken into account in the final <a href="#">test case verdict</a> .	UTP 2
teardown procedure invocation	A <a href="#">procedure invocation</a> that is considered as part of the teardown by the responsible <a href="#">arbitration specification</a> and that is invoked after any <a href="#">main procedure invocation</a> .	UTP 2
test action	An <a href="#">atomic procedural element</a> that is an instruction to the tester that needs to be executed as part of a test procedure of a test case within some time frame.	UTP 2
test case	A <a href="#">procedure</a> that includes a set of preconditions, inputs and expected results, developed to drive the examination of a <a href="#">test item</a> with respect to some <a href="#">test objectives</a> .	UTP 2
test case log	A <a href="#">test log</a> that captures relevant information on the execution of a <a href="#">test case</a> .	UTP 2
test case verdict	A <a href="#">verdict</a> that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test case</a> against a <a href="#">test item</a> .	UTP 2
test component	A role of an <a href="#">artifact</a> within a <a href="#">test configuration</a> that is required to perform a <a href="#">test case</a> .	UTP 2
test component configuration	A set of configuration options offered by an <a href="#">artifact</a> in the role of a <a href="#">test component</a> chosen to meet the requirements of a particular <a href="#">test configuration</a> .	UTP 2
test configuration	A specification of the test item and test components as well as their interconnection and configuration data.	UTP 2
test context	A set of information that is prescriptive for testing activities which can be organized and managed together for deriving or selecting <a href="#">test objectives</a> , <a href="#">test design techniques</a> , <a href="#">test design inputs</a> and eventually <a href="#">test cases</a> .	UTP 2
test design directive	A <a href="#">test design directive</a> is an instruction for a test designing entity to derive test <a href="#">artifacts</a> such as <a href="#">test sets</a> , <a href="#">test cases</a> , <a href="#">test configurations</a> , <a href="#">data</a> or <a href="#">test execution schedules</a> by applying <a href="#">test design techniques</a> on a <a href="#">test design input</a> . The set of assembled <a href="#">test design techniques</a> are referred to as the capabilities a test designing entity must possess in order to carry out the <a href="#">test design directive</a> , regardless whether it is carried out by a human tester or a test generator. A <a href="#">test design directive</a> is a means to support the achievement of a <a href="#">test objective</a> .	UTP 2

Name	Description	Source
test design input	Any piece of information that must or has been used to derive testing <a href="#">artifacts</a> such as <a href="#">test cases</a> , <a href="#">test configuration</a> , and <a href="#">data</a> .	UTP 2
test design technique	A specification of a method used to derive or select <a href="#">test configurations</a> , <a href="#">test cases</a> and <a href="#">data</a> . <a href="#">test design techniques</a> are governed by a <a href="#">test design directive</a> and applied to a <a href="#">test design input</a> . Such <a href="#">test design techniques</a> can be monolithically applied or in combination with other <a href="#">test design techniques</a> . Each <a href="#">test design technique</a> has clear semantics with respect to the <a href="#">test design input</a> and the <a href="#">artifacts</a> it derives from the <a href="#">test design input</a> .	UTP 2
test execution schedule	A <a href="#">procedure</a> that constrains the execution order of a number of <a href="#">test cases</a> .	UTP 2
test item	A role of an <a href="#">artifact</a> that is the object of testing within a <a href="#">test configuration</a> .	UTP 2
test item configuration	A set of configuration options offered by an <a href="#">artifact</a> in the role of a <a href="#">test item</a> chosen to meet the requirements of a particular <a href="#">test configuration</a> .	UTP 2
test level	A specification of the boundary of a <a href="#">test item</a> that must be addressed by a specific <a href="#">test context</a> .	UTP 2
test log	A <a href="#">test log</a> is the instance of a <a href="#">test log structure</a> that captures relevant information from the execution of a <a href="#">test case</a> or <a href="#">test set</a> . The least required information to be logged is defined by the <a href="#">test log structure</a> of the <a href="#">test log</a> .	UTP 2
test log structure	A <a href="#">test log structure</a> specifies the information that is deemed relevant during execution of a <a href="#">test case</a> or a <a href="#">test set</a> . There is an implicit default <a href="#">test log structure</a> that prescribes at least the start <a href="#">time point</a> , the <a href="#">duration</a> , the finally calculated verdict and the executing entity of a test case or test set execution which should be logged.	UTP 2
test objective	A desired effect that a <a href="#">test case</a> or <a href="#">test set</a> intends to achieve.	UTP 2
test procedure	A <a href="#">procedure</a> that constrains the execution order of a number of <a href="#">test actions</a> .	UTP 2
test requirement	A desired property on a <a href="#">test case</a> or <a href="#">test set</a> , referring to some aspect of the <a href="#">test item</a> to be tested.	UTP 2
test set	A set of <a href="#">test cases</a> that share some common purpose.	UTP 2
test set log	A <a href="#">test log</a> that captures relevant information from the execution of a <a href="#">test set</a> .	UTP 2
test set purpose	A statement that explains the rationale for grouping <a href="#">test cases</a> together.	UTP 2
test set verdict	A <a href="#">verdict</a> that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test set</a> against a <a href="#">test item</a> .	UTP 2
test type	A quality attribute of a <a href="#">test item</a> that must be addressed by a specific <a href="#">test context</a> .	UTP 2
time point	The <a href="#">time point</a> at which a <a href="#">test action</a> is initiated.	UTP 2
verdict	A statement that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test set</a> , a <a href="#">test case</a> , or a <a href="#">test action</a> against a <a href="#">test item</a> .	UTP 2



## 4 References

### 4.1 Normative References

[MOF]	<a href="http://www.omg.org/spec/MOF/">http://www.omg.org/spec/MOF/</a> Object Management Group: “ <b>Meta Object Facility™ (MOF™) - Version 2.5.1</b> ”, November 2016, formal/2016-11-01
[OCL]	<a href="http://www.omg.org/spec/OCL/">http://www.omg.org/spec/OCL/</a> Object Management Group: “ <b>Object Constraint Language™ (OCL™) - Version 2.4</b> ”, February 2014, formal/2014-02-03
[UML]	<a href="http://www.omg.org/spec/UML/">http://www.omg.org/spec/UML/</a> Object Management Group: “ <b>OMG Unified Modeling Language™ (OMG UML) - Version 2.5</b> ”, March 2015, formal/2015-03-01
[XMI]	<a href="http://www.omg.org/spec/XMI/">http://www.omg.org/spec/XMI/</a> Object Management Group: “ <b>XML Metadata Interchange (XMI) Specification - Version 2.5.1</b> ”, June 2015, formal/2015-06-07

### 4.2 Informative References

[BMM]	<a href="http://www.omg.org/spec/BMM/">http://www.omg.org/spec/BMM/</a> Object Management Group: “ <b>Business Motivation Model - Version 1.3</b> ”, May 2015, formal/2015-05-19
[DD]	<a href="http://www.omg.org/spec/DD/">http://www.omg.org/spec/DD/</a> Object Management Group: “ <b>Diagram Definition™ (DD) - Version 1.1</b> ”, June 2015, formal/2015-06-01
[ES20187301]	<a href="http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.07.01_60/es_20187301v040701p.pdf">http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.07.01_60/es_20187301v040701p.pdf</a> ETSI ES 201 873-1: “ <b>Methods for Testing and Specifications (MTS) - The Testing and Test Control Notation version 3 - Part 1: TTCN-3 Core Language</b> ”; V4.7.1 (2015-06)
[ES202951]	<a href="http://www.etsi.org/deliver/etsi_es/202900_202999/202951/01.01.01_60/es_202951v010101p.pdf">http://www.etsi.org/deliver/etsi_es/202900_202999/202951/01.01.01_60/es_202951v010101p.pdf</a> ETSI ES 202 951: “ <b>Requirements for Modeling Notations. ETSI Standard, Methods for Testing and Specification (MTS)</b> ”; Model-Based Testing (MBT). V1.1.1 (2011-07)
[ES20311901]	<a href="https://www.etsi.org/deliver/etsi_es/203100_203199/20311901/01.07.01_50/es_20311901v010701m.pdf">https://www.etsi.org/deliver/etsi_es/203100_203199/20311901/01.07.01_50/es_20311901v010701m.pdf</a> ETSI ES 203 119-1: “ <b>Methods for Testing and Specifications (MTS) - The Test Description Language (TDL) - Part 1: Abstract Syntax and Associated Semantics</b> ”; V1.7.1 (2023-10)
[ES20311902]	<a href="https://www.etsi.org/deliver/etsi_es/203100_203199/20311902/01.05.01_60/es_20311902v010501p.pdf">ES 203 119-2 - V1.5.1 - Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 2: Graphical Syntax (etsi.org)</a> ETSI ES 203 119-2: “ <b>Methods for Testing and Specifications (MTS) - The Test Description Language (TDL) - Part 2: Graphical Syntax</b> ”; V1.5.1 (2022-05)
[ES20311903]	<a href="https://www.etsi.org/deliver/etsi_es/203100_203199/20311903/01.05.01_60/es_20311903v010501p.pdf">https://www.etsi.org/deliver/etsi_es/203100_203199/20311903/01.05.01_60/es_20311903v010501p.pdf</a> ETSI ES 203 119-3: “ <b>Methods for Testing and Specifications (MTS) - The Test Description Language (TDL) - Part 3: Exchange Format</b> ”; V1.5.1 (2023-05)
[ES20311904]	<a href="https://www.etsi.org/deliver/etsi_es/203100_203199/20311904/01.05.01_60/es_20311904v010501p.pdf">https://www.etsi.org/deliver/etsi_es/203100_203199/20311904/01.05.01_60/es_20311904v010501p.pdf</a>

	ETSI ES 203 119-4: “ <b>Methods for Testing and Specifications (MTS) - The Test Description Language (TDL) - Part 4: Structured Test Objective Specification (Extension)</b> ”; V1.5.1 (2022-05)
[ES20311905]	<a href="http://www.etsi.org/deliver/etsi_es/203100_203199/20311905/01.01.01_60/es_20311905v010101p.pdf">http://www.etsi.org/deliver/etsi_es/203100_203199/20311905/01.01.01_60/es_20311905v010101p.pdf</a> ETSI ES 203 119-3: “ <b>Methods for Testing and Specifications (MTS) - The Test Description Language (TDL) - Part 5: UML profile for TDL</b> ”; V1.1.1 (2018-05)
[FUML]	<a href="http://www.omg.org/spec/FUML/">http://www.omg.org/spec/FUML/</a> Object Management Group: “ <b>Semantics of a Foundational Subset for Executable UML Models (fUML) - Version 1.2.1</b> ”, January 2016, formal/2016-01-05
[HWT2012]	R. Hametner, D. Winkler, and A. Zoitl, “ <b>Agile testing concepts based on keyword-driven testing for industrial automation systems</b> ” in IECON 2012-38 <sup>th</sup> Annual Conference on IEEE Industrial Electronics Society, 2012, pp. 3727-3732
[IEC61508]	<a href="https://webstore.iec.ch/publication/5515">https://webstore.iec.ch/publication/5515</a> IEC: “ <b>Functional safety of electrical/electronic/programmable electronic safety-related systems—Part 1: General Requirements</b> ”, Edition 2.0, IEC 61508-1, 2010-04
[ISO1087-1]	ISO: “ <b>Terminology work - Vocabulary - Part 1: Theory and application</b> ”, ISO 1087-1:2000(E/F), 15-OCT-2000
[ISO25010]	ISO/IEC: “ <b>System and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Systems and software quality models</b> ”, ISO/IEC 25010:2011, ISO, 2011-03-01
[ISO29119]	<a href="http://www.softwaretestingstandard.org/">http://www.softwaretestingstandard.org/</a> ISO/IEC/IEEE: “ <b>Software Testing - The International Software Testing Standard</b> ”
[ISO9126]	ISO/IEC: “ <b>Software engineering—Product quality—Part 1: Quality model</b> ”, ISO/IEC 9126-1:2001, ISO, 2001
[ISTQB]	<a href="http://www.istqb.org">http://www.istqb.org</a> ISTQB: “ <b>International Software Testing Qualifications Board</b> ”
[MDA]	<a href="http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf">http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf</a> Object Management Group: “ <b>MDA Guide - Version 1.0.1</b> ”, June 2003, omg/2003-06-01
[MDAa]	<a href="http://www.omg.org/mda/papers.htm">http://www.omg.org/mda/papers.htm</a> Object Management Group: “ <b>OMG Architecture Board, “Model Driven Architecture - A Technical Perspective”</b> ”
[MDAb]	<a href="http://www.omg.org/mda/papers.htm">http://www.omg.org/mda/papers.htm</a> Object Management Group: “ <b>Developing in OMG’s Model Driven Architecture (MDA)</b> ”
[MDAd]	<a href="http://www.omg.org/mda">http://www.omg.org/mda</a> Object Management Group: “ <b>MDA “The Architecture of Choice for a Changing World”</b> ”
[OSLC]	<a href="http://open-services.net/bin/view/Main/QmSpecificationV2">http://open-services.net/bin/view/Main/QmSpecificationV2</a> Open Services for Lifecycle Collaboration (OSLC): “ <b>Open Services for Lifecycle Collaboration Quality Management Specification Version 2.0</b> ”
[SBVR]	<a href="http://www.omg.org/spec/SBVR">http://www.omg.org/spec/SBVR</a> Object Management Group: “ <b>Semantics of Business Vocabularies and Business Rules (SBVR) - Version 1.3</b> ”, May 2015, formal/2015-05-07
[SEP2014a]	<a href="http://plato.stanford.edu/archives/win2015/entries/category-theory/">http://plato.stanford.edu/archives/win2015/entries/category-theory/</a> Marquis, Jean-Pierre, “ <b>Category Theory</b> ”, The Stanford Encyclopedia of Philosophy (Winter 2015 Edition), Edward N. Zalta (ed.)

[SysML]	<a href="http://www.omg.org/spec/SysML">http://www.omg.org/spec/SysML</a> Object Management Group: “ <b>OMG Systems Modeling Language (OMG SysML™) - Version 1.4</b> ”, September 2015, formal/2015-06-03
[TCM2008]	J. Tang, X. Cao, and A. Ma, “ <b>Towards adaptive framework of keyword driven automation testing</b> ” in Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on, 2008, pp. 1631-1636
[TestIF]	<a href="http://www.omg.org/spec/TestIF/">http://www.omg.org/spec/TestIF/</a> Object Management Group: “ <b>Test Information Interchange Format (TestIF) Specification - Version 1.0</b> ”, May 2015, formal/2015-05-05
[UL2007]	Utting, M., Legeard, B.: “ <b>Practical Model-Based Testing: A Tools Approach</b> ”, Morgan-Kaufmann, 2007
[UPL2012]	<a href="http://dx.doi.org/10.1002/stvr.456">http://dx.doi.org/10.1002/stvr.456</a> Utting, M., Pretschner, A., and Legeard, B.: “ <b>A taxonomy of model-based testing approaches</b> ”, in Softw. Test. Verif. Reliab. 22, 5, August 2012, p. 297-312
[UTP]	<a href="http://www.omg.org/spec/UTP">http://www.omg.org/spec/UTP</a> Object Management Group: “ <b>UML Testing Profile - Version 1.2</b> ”, April 2013, formal/2013-04-03
[WikiCT]	<a href="https://en.wikipedia.org/wiki/Category_theory">https://en.wikipedia.org/wiki/Category_theory</a> Wikipedia: “ <b>Category Theory</b> ”
[WikiM]	<a href="https://en.wikipedia.org/wiki/Morphism">https://en.wikipedia.org/wiki/Morphism</a> Wikipedia: “ <b>Morphism</b> ”

## **5 Symbols**

No special symbols have been used in this specification.

## 6 Additional Information

### 6.1 How to read this document

This specification is intended to be read by the audience listed below in order to learn, apply, implement and support UTP 2. To understand how UTP 2 relates to other testing standards, all readers are encouraged to read Clause 6 ([Additional Information](#)). In order to learn more about the conformance of UML and UTP 2 as well as the compliance levels between the UTP 2 specification and the UTP 2 tool implementation, please read Clause 2 ([Conformance](#)). Some references to other standards are listed in Clause 3 ([References](#)). For convenience, Clause 4 ([Terms and Definitions](#)) contains a brief summary of the concepts described in more detail in Clause 7 ((Informative) Conceptual Model [STUB]).

The definition of the UML Testing Profile itself can be found in the Chapters 7-9. Clause 7 ((Informative) Conceptual Model [STUB]) starts with the definition of a pure conceptual model of UTP 2 independent of any implementation measures. The conceptual model is informative (i.e. non-normative) but provides the big picture of the intended scope of UTP 2. The mapping of the conceptual model to the UML profile specification is described in Clause 8 (Profile Specification [STUB]). The stereotype mappings abide by the semantics of the conceptual elements in general. Only additional aspects of the semantics regarding the integration of a stereotype with related UML metaclasses will be added in Clause 8.

Clause 9 ([Model Libraries](#)) describes the predefined UTP 2 model libraries. The [UTP Auxiliary Library](#) provides predefined elements for reuse across multiple modeling projects. The UTP Types Library provides additional types that have been proven helpful for the definition of tests.

The Annex sections provide further informative material for UTP 2, in particular an examples section that shows different methodologies on how to apply UTP 2 technically and conceptually. The Annex sections are living sections that means they may change among future versions.

Modeling tool vendors should read the whole document, including the annex chapters. Modelers and engineers are encouraged to read Annex A to understand how the language is applied to examples.

This document may be read in both sequential and non-sequential manner.

### 6.2 Typographical conventions

A set of typographical conventions have been applied to the editorial part of this specification that should help the reader in understanding and relating things to their proper context. These conventions are subsequently explained:

- Concepts of the conceptual model are written in lower letters and colored blue, indicating a link to the section of the conceptual element. Example: [test context](#)
- UML metaclasses start with an upper-case letter and are written in camel-case and are set bold and blue. Example: **[Constraint](#)**, **[BehavioredClassifier](#)**
- Stereotypes are start with an upper-case letter and are written in camel-case, surrounded by guillemets. Example: «[TestContext](#)»
- Properties of metaclasses or tag definitions of stereotypes are stated in italic: Examples: *constrainedElement* (from UML metaclass Constraint), *arbitrationSpecification* (from stereotype «[ProceduralElement](#)»)
- Values of Properties or tagged values of tag definitions are stated italic: Examples: *false*, *true*
- OCL constraints as formalization of natural language Constraint descriptions are set in Courier. Example:  
context [TestComponent](#):  
not self.base\_Property.class.getAppliedStereotype('UTP::[TestItem](#)')->  
oclIsUndefined()

## 6.3 Typical Use Cases of UTP 2

This section briefly summarizes typical use cases of UML Testing Profile V2 (UTP 2) by means of a simple UML use case model. It is intended to give the interested reader an initial idea of who and what for UTP 2 may be used in the context of developing and testing complex systems.

The following use case diagram summarizes typical UTP 2 users and their use cases of UTP 2.

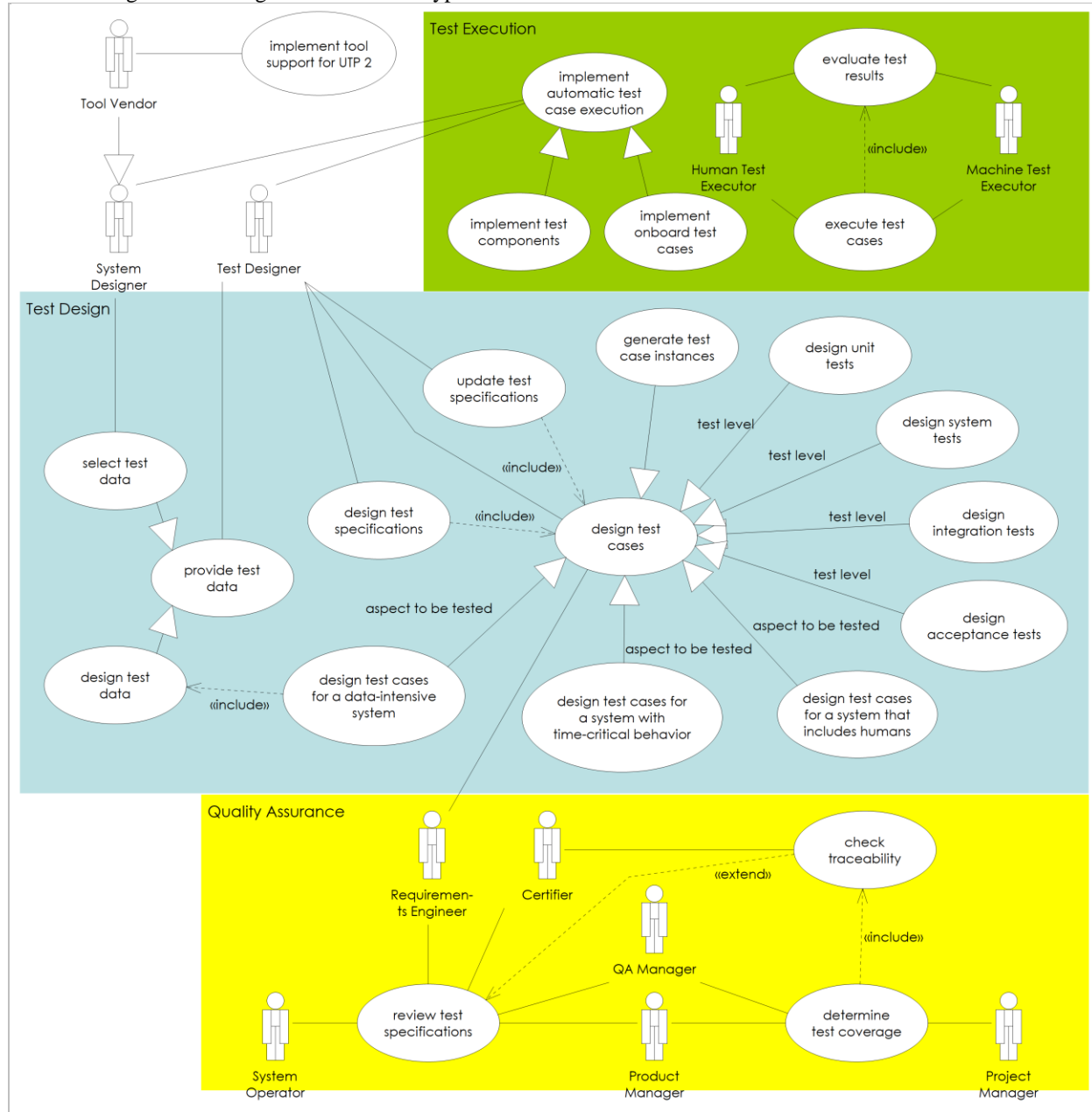


Figure 6.1 - UTP 2 Use Cases

The following table characterizes the users (represented as UML actors) introduced in the diagram above and lists for each user the use cases related to UTP 2 she or he may directly or indirectly carry-out.

**Table 6.1 - Typical UTP 2 Users**

User Type	Description	Use Cases
Certifier	A role of a person responsible for certifying a safety-critical or mission-critical system or product.	<ul style="list-style-type: none"> <li>• <a href="#">check traceability</a></li> <li>• <a href="#">review test specifications</a></li> </ul>
Human Test Executor	A role of a person responsible for executing <a href="#">test cases</a> and/or evaluating their outcomes.	<ul style="list-style-type: none"> <li>• <a href="#">evaluate test results</a></li> <li>• <a href="#">execute test cases</a></li> </ul>
Machine Test Executor	A machine or device that executes test cases and/or evaluates their outcomes.	<ul style="list-style-type: none"> <li>• <a href="#">evaluate test results</a></li> <li>• <a href="#">execute test cases</a></li> </ul>
Product Manager	A role of a person having the overall responsibility for a system or product.	<ul style="list-style-type: none"> <li>• <a href="#">determine test coverage</a></li> <li>• <a href="#">check traceability</a></li> <li>• <a href="#">review test specifications</a></li> </ul>
Project Manager	A role of a person having the overall responsibility for the development, procurement, implementation, or adaption of a system or product or a part of it.	<ul style="list-style-type: none"> <li>• <a href="#">determine test coverage</a></li> <li>• <a href="#">check traceability</a></li> </ul>
QA Manager	A role of a person responsible to guarantee the appropriate quality of a system or product.	<ul style="list-style-type: none"> <li>• <a href="#">determine test coverage</a></li> <li>• <a href="#">check traceability</a></li> <li>• <a href="#">review test specifications</a></li> </ul>
Requirements Engineer	A role of a person responsible for gathering, expression and managing the requirements on a system or product.	<ul style="list-style-type: none"> <li>• <a href="#">design test cases</a></li> <li>• <a href="#">design acceptance tests</a></li> <li>• <a href="#">design integration tests</a></li> <li>• <a href="#">design system tests</a></li> <li>• <a href="#">design test cases for a data-intensive system</a></li> <li>• <a href="#">design test data</a></li> <li>• <a href="#">design test cases for a system that includes humans</a></li> <li>• <a href="#">design test cases for a system with time-critical behavior</a></li> <li>• <a href="#">design unit tests</a></li> <li>• <a href="#">generate test case instances</a></li> <li>• <a href="#">review test specifications</a></li> <li>• <a href="#">check traceability</a></li> </ul>
System Designer	A role of a person that designs, builds, extends, maintains, or updates a system or product.	<ul style="list-style-type: none"> <li>• <a href="#">implement automatic test case execution</a></li> <li>• <a href="#">implement onboard test cases</a></li> <li>• <a href="#">implement test components</a></li> <li>• <a href="#">select test data</a></li> </ul>
System Operator	A role of a person that utilizes a system or product.	<ul style="list-style-type: none"> <li>• <a href="#">review test specifications</a></li> <li>• <a href="#">check traceability</a></li> </ul>
Test Designer	A role of a person that designs, builds, extends, maintains, or updates test specifications of a system.	<ul style="list-style-type: none"> <li>• <a href="#">design test cases</a></li> <li>• <a href="#">design acceptance tests</a></li> <li>• <a href="#">design integration tests</a></li> <li>• <a href="#">design system tests</a></li> <li>• <a href="#">design test cases for a data-intensive system</a></li> <li>• <a href="#">design test data</a></li> <li>• <a href="#">design test cases for a system that includes humans</a></li> <li>• <a href="#">design test cases for a system with time-critical behavior</a></li> </ul>

		<ul style="list-style-type: none"> <li>• <a href="#">design unit tests</a></li> <li>• <a href="#">generate test case instances</a></li> <li>• <a href="#">design test specifications</a></li> <li>• <a href="#">implement automatic test case execution</a></li> <li>• <a href="#">implement onboard test cases</a></li> <li>• <a href="#">implement test components</a></li> <li>• <a href="#">provide test data</a></li> <li>• <a href="#">select test data</a></li> <li>• <a href="#">update test specifications</a></li> </ul>
Tool Vendor	A role of a person that develops a tool implementing at least some aspects of the UTP 2 specification.	<ul style="list-style-type: none"> <li>• <a href="#">implement tool support for UTP 2</a></li> <li>• <a href="#">implement automatic test case execution</a></li> <li>• <a href="#">implement onboard test cases</a></li> <li>• <a href="#">implement test components</a></li> <li>• <a href="#">select test data</a></li> </ul>

The following table briefly describes the use cases introduced in the diagram above.

**Table 6.2 - Typical UTP 2 Use Cases**

Use Case	Description
check traceability	Verification of the traceability between requirements and test cases in order to determine the coverage of a system by a set of test cases.
design acceptance tests	The design of <a href="#">test cases</a> that are used to perform an acceptance test of a system or product, i.e. that the sponsor/customer may decide on the acceptance of that system or product.
design integration tests	The design of <a href="#">test cases</a> that are used to perform an integration test of a system or product, i.e. the verification of the interoperability among its internal components as well as with its environment conforms to its specification.
design system tests	The design of <a href="#">test cases</a> that are used to perform a system test of a system or product, i.e. the verification that the system or product (typically viewed as a black box) fulfills its requirements.
design test cases	The design, elaboration and adaptation of test sets comprising test cases in order to verify the requirements and/or to validate the goals of a system or product.
design test cases for a data-intensive system	The design of <a href="#">test cases</a> for a system whose functionality includes complex processing of data that is of a highly complex structure and/or of large data volumes.
design test cases for a system that includes humans	The design of <a href="#">test cases</a> for a sociotechnical system that includes technical systems as well as humans collaboratively performing complex processes.
design test cases for a system with time-critical behavior	The design of <a href="#">test cases</a> for a system that must comply to soft or hard real-time constraints on its behavior.
design test data	The design and production of data that is of a highly complex structure and/or of large data volumes.
design test specifications	The elaboration and compilation of all information necessary for carrying-out verification and validation procedures of a system or product. This includes specifying test objectives, test strategies, test procedures, test data, test configurations, evaluation criteria and more.
design unit tests	The design of <a href="#">test cases</a> that are used to perform functional tests of an individual component of a system or product.
determine test coverage	The examination of <a href="#">test sets</a> and <a href="#">test cases</a> with the focus on the coverage provided by of those <a href="#">test sets</a> and <a href="#">test cases</a> with respect to the requirements and/or implementation aspects of a system or product in order to determine the



	suitability of the <a href="#">test sets</a> and <a href="#">test cases</a> for a given purpose.
evaluate test results	The examination of the results of an executed test set or executed test case in order to determine the verdict of the test set or test case.
execute test cases	The manual or automatic execution of test procedures according to a given test specification composed of sets and/or test cases.
generate test case instances	The manual or automatic production of specific test case instances from a given test specification composed of generic sets and/or test cases.
implement automatic test case execution	The implementation, provisioning and configuration of test infrastructure required to perform and evaluate <a href="#">test sets</a> or <a href="#">test cases</a> automatically.
implement onboard test cases	The implementation of test components and test procedures as part of a system or product in order to make it able to perform self-tests while it is in operation.
implement test components	The implementation, provisioning, and configuration of auxiliary test components in order to automate or at least to simplify the execution of test sets or test cases.
implement tool support for UTP 2	The implementation, provisioning, or configuration of a tool in order to supports the utilization of UTP 2. This could e.g. be a UML Profile implementing UTP 2 for a particular UML modeling tool or a test execution tool that supports the concepts of UTP 2.
provide test data	The provisioning of dedicated data that is used to perform test sets or test cases.
review test specifications	The quality assurance of a particular test specification in order to fulfill given quality goals.
select test data	The selection and potentially transformation of available operational data in order to use this data during the execution of test sets or test cases.
update test specifications	The adaption of <a href="#">test objectives</a> , test strategies, <a href="#">test procedures</a> , test <a href="#">data</a> , <a href="#">test configurations</a> , evaluation criteria etc. according to changing requirements and goals of an already existing system or product.

## 6.4 Relation to testing-relevant standards

The landscape of software/system testing standards is diversified. Many domain-specific standards (e.g., [\[IEC61508\]](#)) set requirements on how a test process should be conducted. In addition, there are a number of domain- and methodology-independent testing-relevant standards (e.g., [\[ISO29119\]](#)), to which UTP 2 can define integration points. In the following section, the specification describes some of these standards and discusses how they can be integrated with UTP 2.

### ISO/IEC/IEEE 29119 Software Testing Standard

The ISO/IEC/IEEE 29119 Software Testing Standard is a family of standards for software testing, which consists of five parts:

- Concepts and definitions
- Test processes
- Test documentation
- Test techniques
- Keyword-driven testing

[\[ISO29119\]](#) is a conceptual standard, in the sense that it does not define technical solutions, specific languages or methodologies, in contrast to UTP 2. Instead, [\[ISO29119\]](#) standardizes a number of concepts and definitions, some of which have been adopted by UTP 2. [\[ISO29119\]-2](#) specifies the structure of test processes and distinguishes different levels for test processes: organizational, test management and dynamic test processes. The first two processes deal with management-related aspects of test processes, and the dynamic test process is mainly about deriving [test cases](#), implementing and executing [test cases](#) and evaluating executed [test cases](#).

UTP 2 is designed to support the dynamic test process. That means, it provides concepts that enable the derivation/generation, specification, visualization and documentation of test [artifacts](#) such as [test cases](#), [data](#), [test configurations](#), [test sets](#) and [test contexts](#). Furthermore, UTP 2 provides necessary concepts to generate [\[ISO29119\]](#)-3-compliant test reports and documentations out of a UTP 2 model.

A set of standardized [test design techniques](#), such as equivalence partitioning or state-based testing, has been adopted in [\[ISO29119\]](#)-4 made technically explicit as part of the UTP 2 language. Test engineers can utilize UTP 2 to specify [test design techniques](#) to be applied on a certain [test design input](#) (e.g., a description of the intended behavior of the [test item](#), which is represented as a state machine or interaction). In addition to these standardized [test design techniques](#), test engineers may define additional [test design techniques](#) if required.

The relation to [\[ISO29119\]](#)-5, which deals with standardizing the concepts of the keyword-driven testing paradigm, is of an implicit nature. UTP 2 can be effectively employed to setup and drive keyword-driving testing approaches. For further information on the relation of UTP 2 to keyword-driven testing see section [Relation to keyword-driven testing](#).

### **ISTQB and its glossary**

The ISTQB [\[ISTQB\]](#) and its glossary defines a set of globally standardized terminologies and definitions of testing-related concepts. The ISTQB nomenclature was deemed equally important for the definition of UTP 2 concepts as the [\[ISO29119\]](#) definitions. Hence, UTP 2 adopted a set of definitions, terminologies and even [test design techniques](#) from the ISTQB glossary and syllabi.

To keep the analogy with [\[ISO29119\]](#), UTP 2 is designed to support activities of test analysis and test design of the ISTQB fundamental test process. Test implementation and test execution are supported rather indirectly by means of [arbitration specifications](#), precise semantics of [test actions](#) and the definition of [test execution schedules](#).

Test evaluation activities are supported by means of the [test logging](#) capability of UTP 2, which enables a system-independent representation of a test execution. For example, UTP 2 [test logs](#) can be exploited for metrics calculations or supporting other analysis.

### **ETSI Testing and Test Control Notation 3 (TTCN-3)**

ETSI TTCN-3 [\[ES20187301\]](#) standardizes a test programming language and architecture of a test execution system. It enables a platform-independent implementation of executable [test cases](#). As such, it provides test engineers a set of language features that has been proven efficient in the development of large and complex test suites for software-intensive systems of various domains, including telecommunication, transportation, and automotive airborne software. In addition, TTCN-3 provides concepts that address reusability and simplicity in the specification of large test suites, such as using wildcard values to ease the definition of expected [responses](#) from the [test item](#).

UTP 2, as a successor of UTP 1, is influenced by the capabilities of TTCN-3. UTP 2 adopts some TTCN-3 concepts such as [test components](#), [test configurations](#) and [test actions](#). Moreover, some of the TTCN-3 wildcards definitions (e.g., regular expression, any value) have been adopted by UTP.

Although UTP 2 defines [test cases](#) (due to being dependent on UML) at a much higher level of abstraction than TTCN-3, it is possible (and has been done in numerous approaches) to generate TTCN-3 modules from UTP 2 test models.

### **ETSI Test Description Language (TDL)**

The Test Description Language (TDL) standardized by ETSI ([\[ES20311901\]](#), [\[ES20311902\]](#), [\[ES20311903\]](#), [\[ES20311904\]](#)) is a MOF-based graphical modeling language for describing test scenarios (not [test cases](#)) by a similar notation to Message [sequence](#) Charts (MSC) or UML [sequence](#) Diagrams (SD). TDL represents the next generation of testing languages in the ETSI testing technology stack and exploits the advantages of MBT. TDL is used primarily - but not exclusively - for functional testing.

According to ETSI, TDL can bring a number of benefits, including:

- higher quality tests through better design
- easier to review by non-testing experts
- better, faster test development
- seamless integration of methodology and tools

TDL and UTP 2 share a set of common concepts such as [test component](#), [test configuration](#) and [procedural elements](#). This is partially due to the same origin of TDL and UTP 2: TTCN-3. In that regard the two languages are compatible. However, UTP 2 has a bigger scope than TDL, which so far mainly focuses on functional testing and the manual definition of test scenarios. UTP 2 offers several features beyond the capability of TDL, such as specifying [test design techniques](#) and application thereof onto a [test design input](#). UTP 2 offers explicit concepts for test generation. Another feature of UTP 2 is the flexible handling of [arbitration specifications](#). Finally, UTP 2 offers concepts to organize testing activities based on test management concepts such as [test contexts](#), which resemble the semantics of [ISO29119](#) test process or test sub-process, [test types](#), [test objectives](#) and [test sets](#).

## 6.5 Relation to model-based testing

Model-Based Testing (MBT) is a testing technique that uses models of a software-intensive system under test to perform certain testing activities such as test analysis, test design and test implementation in both an automated (e.g., generation of [test cases](#) and [data](#)) and manual manner. Such a system under test is called a [test item](#) in the context of the UTP.

The UTP definition of MBT is adopted and slightly adjusted from the [\[ES202951\]](#) definition. "Model-based testing (MBT) is an umbrella of techniques that uses semi-formal models as engineering [artifacts](#) in order to specify and/or generate testing-relevant [artifacts](#), such as [test cases](#), test scripts, and reports." Other valid definitions of MBT are:

- "Testing based on or involving models" ([\[ISTOB\]](#), Glossary)
- "An umbrella of techniques that generates tests from models" [\[ES202951\]](#)

MBT has been thoroughly investigated in the academic literature and has also been of great interest in a variety of industry domains [\[UPL2012\]](#), [\[UL2007\]](#). The idea of MBT is to utilize models (so called test models in the context of UTP 2) that represent the expected behavior of the [test item](#) or [test cases](#) of the [test item](#) at a higher level of abstraction. Such abstraction enables test engineers to focus exclusively on the logical aspects of the [test item](#), instead of being bothered by technical details of the eventual implementation. Low level details of [test cases](#), for example, syntactical details of a scripting language or completeness of [data](#), can be taken care of by domain specific generators eventually producing executable [test cases](#), which can finally be executed against the [test item](#).

UTP 2 is an industrial standard that dedicatedly supports MBT by relying on UML. UTP covers a variety of concepts that are deemed mandatory such [test case](#), [data](#), and Arbitration & [verdict](#). It also dedicatedly and exclusively defines concepts to govern the derivation of test-relevant information (such as [test cases](#), [data](#) etc.) by means of test directives and [test design techniques](#). Additionally, it also provides a few test management-related concepts that are required for defining complete test specification documents (compatible with [\[ISO29119\]](#)) such as [test contexts](#) (called test process/test sub-process in [\[ISO29119\]](#)), [test level](#), [test type](#) and [test logs](#).

UTP 2 is agnostic of any MBT methodology, and thus, supports a variety of MBT approaches. Some of the key aspects include: 1) Modeling [test cases](#) for a [test item](#) using stereotypes from the profile; 2) Modeling the expected behavior of the [test item](#) for test derivation using stereotypes from the profile; 3) Modeling [test case](#) specifications in domain specific languages implementing UTP.

Based on the philosophy of (test) modeling, UTP allows creating test models at various levels of abstraction ranging from test models that have no concrete [data](#), test models that have some [data](#), and test models that have all concrete [data](#) available.

## 6.6 Relation to keyword-driven testing

Keyword-driven testing (KDT) is an industrial de-facto standard that is suitable for both manual and automated test execution. KDT methodologies define logical functions that can be performed on the [test item](#) in an implementation-independent format (i.e., keyword) at a higher level of abstraction. Keywords are used to design so called keyword [test cases](#) (see [\[ISO29119\]-5](#)). In order to execute the keyword [test cases](#) against the [test item](#), it is required that implementations of the keywords can be executed by a keyword-based test execution system. Keyword implementations are usually organized in a test library.

The keyword-based test execution system is responsible to establish a connection between the keyword implementations and the actual implementation of the [test item](#), run keyword [test cases](#), and execute the keyword implementations against the actual implementation of the [test item](#).

In the literature, there exist a number of keyword-driven testing frameworks. For example, Tang et al. [\[TCM2008\]](#) proposed a keyword-driven testing framework to transform keyword-based [test cases](#) into different kinds of test scripts. Hametner et al. [\[HWT2012\]](#) proposed a keyword-driven testing approach to specify keyword [test cases](#) in a high abstraction level, as tabular format using predefined keywords, and automatically generated executable [test cases](#) from the keyword [test case](#). There are a number of commercial and open-source tools available for KDT.

UTP 2 is defined to facilitate MBT, but it does not explicitly cope with the design and implementation of test execution systems. However, UTP 2 defines concepts such as, [abstract test cases](#) and [data specification](#) explicitly to enable automated generation of [concrete test cases](#) and [data](#) from abstract ones. This idea conforms to the idea of KDT in terms of raising the level of abstractions by defining keyword [test cases](#).

Keywords can be represented by numerous concepts of the underlying UML within UTP 2. For example, Operations of Interfaces may be interpreted as the logical functions that can be performed on the [test item](#). Additionally, UTP 2 can be used to define or generate [test cases](#) that are based on these UML-based keyword representations. UML behaviors such as Activities or Interactions are suitable means to represent keywords in [test cases](#) in UTP 2, which are eventually exported into the keyword format required by the utilized keyword-based test execution system. As such, UTP 2 is suitable to be used as a standardized and visual language for keywords and keyword [test cases](#).

UTP 2 could even go one step further. Due to the fact that UTP 2 is based on UML, it is even possible to provide an executable specification of the test library (i.e., the implementation of a keyword) by means of other standards such as fUML.

As a summary, UTP 2 can be efficiently leveraged as the language for the (automated or manual) design, visualization, documentation and communication of keywords, keyword [test cases](#) and even implementations thereof.

## 6.7 Relation to the MARTE Profile

Modeling and Analysis of Real-Time and Embedded Systems (MARTE) is a UML profile that is specifically designed for modelling and supporting analyses (e.g., performance and schedulability) for real-time and embedded systems. MARTE is developed to replace its predecessor UML profile, i.e., the UML profile for the Schedulability, Performance, and Time specification (SPTP).

At a very high level, the MARTE profile is organized into four main packages: MARTE foundations, MARTE design model, MARTE analysis model, and MARTE annexes including: MARTE model libraries, Value Specification Language, and Repetitive Structure Modeling. Out of these four packages MARTE analysis model is outside the scope of UTP since it doesn't aim to support analyses such as performance and schedulability but rather focuses on the test case generation. Nonetheless, UTP may be used for supporting model-based performance and schedulability testing and such modelling can be supported with MARTE foundation package on which MARTE analysis model relies on.

The most relevant packages for UTP from MARTE include Non-Functional Properties Modeling (NFP), Time Modelling (Time), and MARTE Library. The NFP package provides a generic framework for modelling NFPs using UML modeling elements. The package defines stereotypes such as «Nfp» to define new NFPs for a particular application and «Unit» for defining new measurement units by extending the existing ones provided in the MARTE

model library such as `TimeUnitKind` and `PowerUnitKind`. Notice that NFPs defined in MARTE can be used together with UTP to support test case generation.

The Time package is specifically designed for modelling time and its related concepts specifically for real-time and embedded systems. Since Time and behavior are tightly coupled, MARTE's Time modelling can be used in conjunction with the UTP for supporting model-based testing of real-time embedded software/system with a focus on time behavior. The extensive model library of MARTE provides extended basic data types such as `Real` and `DateTime` and a rich collection of operations on them. In addition, it also provides a wide variety of measurement units such as `TimeUnitKind` and `LengthUnitKind`, general data types such as `IntegerVector` and `IntegerInterval`, predefined data types such as `NFP_Percentage` and `NFP_DataSize` and `TimeLibrary` supporting modelling such as logical and ideal clocks. These types can be used for modelling [test items](#) and [test components](#) that require extended data types rather than the basic data types supported by the UML. In addition, the modelling support for a variety of clocks, i.e., logical and ideal clocks, can be used for modelling complex time behavior of [test items](#) and [test components](#).

## 6.8 Acknowledgements

The following OMG member organizations submitted this specification (in alphabetic order):

- Fraunhofer FOKUS, Germany.
- SOFTEAM, France.

The following OMG and non-OMG member organizations supported this specification (in alphabetic order):

- PTC Inc., United Kingdom and USA.
- Hamburg University of Applied Science, Germany.
- KnowGravity Inc., Switzerland.
- Simula Research Lab, Norway.

### Special Acknowledgments

The following persons were members of the core teams that contributed to the content of this document (in alphabetic order):

- Alessandra Bagnato, [alessandra.bagnato@softeam.fr](mailto:alessandra.bagnato@softeam.fr)
- Etienne Brosse, [etienne.brosse@softeam.fr](mailto:etienne.brosse@softeam.fr)
- Zhen Ru Dai, [dai@informatik.haw-hamburg.de](mailto:dai@informatik.haw-hamburg.de)
- Rolf Gubser, [rolf.gubser@knowgravity.com](mailto:rolf.gubser@knowgravity.com)
- Andreas Hoffmann, [andreas.hoffmann@fokus.fraunhofer.de](mailto:andreas.hoffmann@fokus.fraunhofer.de)
- Andreas Korff, [akorff@ptc.com](mailto:akorff@ptc.com)
- Markus Schacher, [markus.schacher@knowgravity.com](mailto:markus.schacher@knowgravity.com)
- Marc-Florian Wendland, [marc-florian.wendland@fokus.fraunhofer.de](mailto:marc-florian.wendland@fokus.fraunhofer.de)

This page intentionally left blank.

## 7 (Informative) Conceptual Model

This section is *informative*, i.e. non-normative and not relevant for actual profile implementations. However, it is included here to help the reader to get a better understanding of the concepts behind UTP 2. This section illustrates some of the semantics for the concepts defined in this document by means of a pragmatic application of the OMG specification "Semantics of Business Rules and Vocabularies" [\[SBVR\]](#). This pragmatic application of SBVR includes the following:

- A number of concept diagrams visualize the concepts as well as their interrelationships (in SBVR called "verb concepts") organized around different subject areas. Furthermore, any SBVR definitional rule related to the concepts shown is also visualized on the diagram.
- For each concept diagram, the rule statements of each definitional rule shown are listed. The styling of those rule statements is simplified compared to [\[SBVR\]](#) in the sense that no colors/formatting is used. The only styling that is shown is that concepts defined within the document are shown underlined and represent an intra-document hyperlink.
- For each concept diagram, the semantics of each concept shown on the diagram is defined, usually by means of an intensional definition as suggested by [\[ISO1087-1\]](#). Here underlined words also represent hyperlinks to the mentioned concepts. When defined, additional properties of concepts such as synonyms, examples, generalizations, specialization, etc. are also listed. Furthermore, for each concept the source of its definition is specified.

### 7.1 Test Planning

#### 7.1.1 Test Analysis

##### 7.1.1.1 Test Context Overview

The following concept diagram represents important semantic aspects of [test context](#) and associated other concepts such as [test set](#), [test case](#), [data](#) and [test design input](#).

A [test context](#) is defined as a hub for information that specifies [test type](#), [test level](#), prescribes [test design technique](#), and refers to [data](#), [data pool](#), [test design input](#), [arbitration specification](#), [test set](#) and [test case](#). A [test context](#) also refers to other important test model elements, such as the set of [test cases](#), [data](#) and the [test design input](#). A [test context](#) also provides information for test management, where planning and strategies for the test are defined.





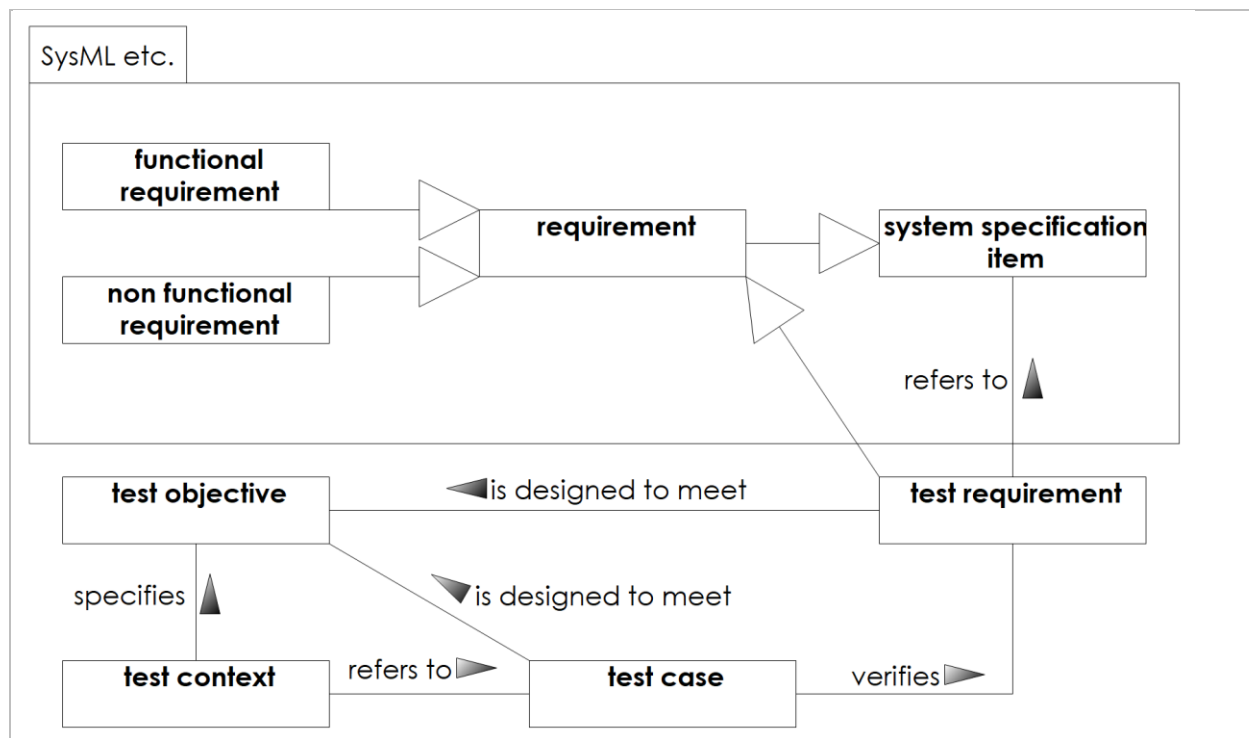


Figure 7.2 - Test Requirement and Test Objective Overview

#### 7.1.1.2 Concept Descriptions

test context	
Definition	A set of information that is prescriptive for testing activities which can be organized and managed together for deriving or selecting <a href="#">test objectives</a> , <a href="#">test design techniques</a> , <a href="#">test design inputs</a> and eventually <a href="#">test cases</a> .
Examples	acceptance test, smoke test, system test, ...
Source	UTP 2

test level	
Definition	A specification of the boundary of a <a href="#">test item</a> that must be addressed by a specific <a href="#">test context</a> .
Examples	integration test, system test, component test, ...
Source	UTP 2

test objective	
Definition	A desired effect that a <a href="#">test case</a> or <a href="#">test set</a> intends to achieve.
Examples	<ul style="list-style-type: none"> <li>Provision of information about the qualities of the product to a certification authority or other stakeholders</li> <li>Provision of information that the product has met stakeholder expectations</li> <li>Provision of information that requirements of a product are fulfilled (i.e. regulatory, design, contractual, etc.)</li> </ul>
Source	UTP 2

test requirement	
Definition	A desired property on a <a href="#">test case</a> or <a href="#">test set</a> , referring to some aspect of the <a href="#">test item</a> to be tested.
Synonyms	test condition

Examples	<ul style="list-style-type: none"> <li>• Test case must ensure 80% path coverage of use case XY.</li> <li>• Test case must check that an IPv6 multicast message is carried out over a GeoBroadcast message into the correct geographical area, with a GVL manually configured.</li> </ul>
Source	UTP 2
Is a	requirement

test set	
Definition	A set of <a href="#">test cases</a> that share some common purpose.
Source	UTP 2

test set purpose	
Definition	A statement that explains the rationale for grouping <a href="#">test cases</a> together.
Source	UTP 2

test type	
Definition	A quality attribute of a <a href="#">test item</a> that must be addressed by a specific <a href="#">test context</a> .
Examples	functionality test, usability test, conformance test, interoperability test, performance test, ...
Source	UTP 2

## 7.1.2 Test Design

### 7.1.2.1 Test Design Facility Overview

The following diagram summarizes the concepts of UTP 2 test design facility. The test design facility enables the specification of [test design techniques](#) that must be applied on a [test design input](#) in order to derive test [artifacts](#) such as [test sets](#), [test cases](#), [test configurations](#), required [data](#) or [test execution schedules](#). Whether the test derivation process according to the specified [test design techniques](#) is carried out manually or automatically does not matter whatsoever. Such [test design techniques](#) are assembled and governed by a [test design directive](#). Thus, the [test design directive](#) is a specification of the capabilities a test designing entity (e.g. a human tester or test generator) must offer in order to perform the derivation activities according to the assembled [test design techniques](#). The UTP 2 test design facility is agnostic of any implementation- or tool-specific details and simply offers the ability to describe, select and extend the set of potentially available and applicable [test design techniques](#).

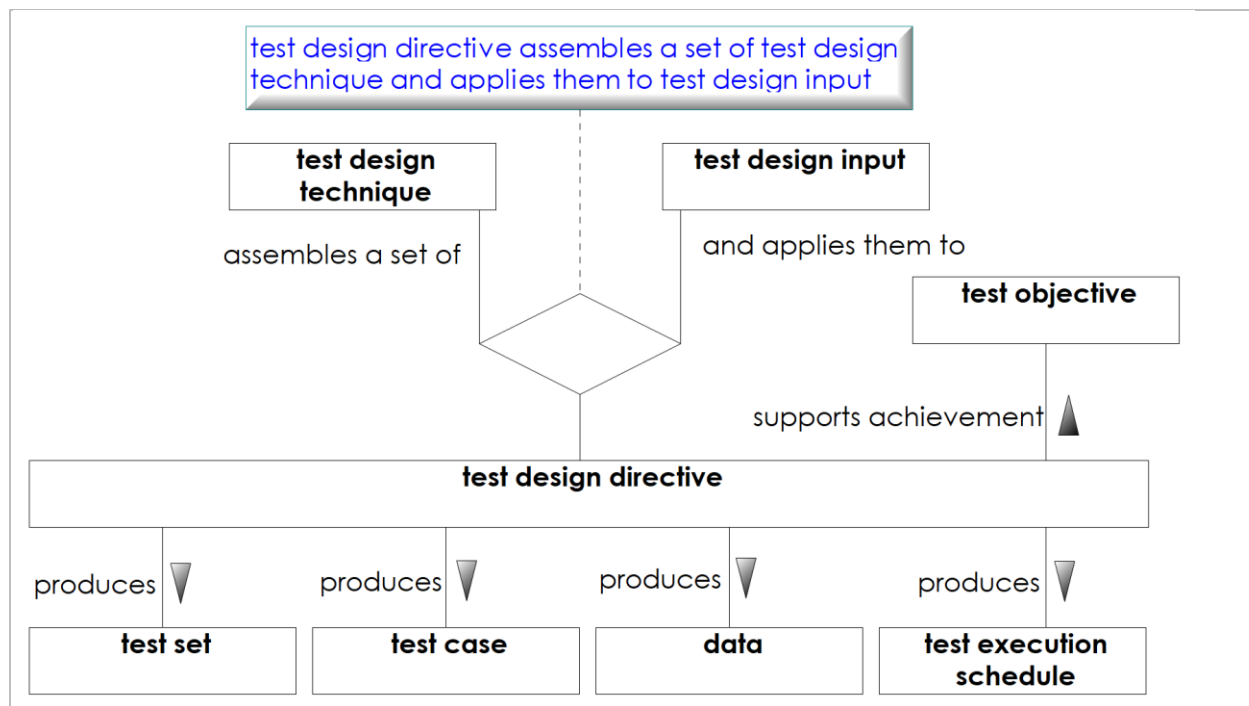


Figure 7.3 - Test Design Facility Overview

#### 7.1.2.2 Concept Descriptions

test design directive	
Definition	A <a href="#">test design directive</a> is an instruction for a test designing entity to derive test <a href="#">artifacts</a> such as <a href="#">test sets</a> , <a href="#">test cases</a> , <a href="#">test configurations</a> , <a href="#">data</a> or <a href="#">test execution schedules</a> by applying <a href="#">test design techniques</a> on a <a href="#">test design input</a> . The set of assembled <a href="#">test design techniques</a> are referred to as the capabilities a test designing entity must possess in order to carry out the <a href="#">test design directive</a> , regardless whether it is carried out by a human tester or a test generator. A <a href="#">test design directive</a> is a means to support the achievement of a <a href="#">test objective</a> .
Source	UTP 2

test design input	
Definition	Any piece of information that must or has been used to derive testing <a href="#">artifacts</a> such as <a href="#">test cases</a> , <a href="#">test configuration</a> , and <a href="#">data</a> .
Examples	a state machine specifying some expected behavior of the test item used to derive some test cases, a requirements catalog used to derive some test cases, ...
Source	UTP 2
Is a	model

test design technique	
Definition	A specification of a method used to derive or select <a href="#">test configurations</a> , <a href="#">test cases</a> and <a href="#">data</a> . <a href="#">test design techniques</a> are governed by a <a href="#">test design directive</a> and applied to a <a href="#">test design input</a> . Such <a href="#">test design techniques</a> can be monolithically applied or in combination with other <a href="#">test design techniques</a> . Each <a href="#">test design technique</a> has clear semantics with respect to the <a href="#">test design input</a> and the <a href="#">artifacts</a> it derives from the <a href="#">test design input</a> .
Examples	equivalence testing, structural coverage
Source	UTP 2

## 7.2 Test Architecture

### 7.2.1 Test Architecture Overview

The following concept diagram represents important semantic aspects in the context of [test configuration](#) and associated other concepts such as [test component](#), [test items](#) and [test cases](#). A [test case](#) relies on at least one [test configuration](#) to execute. A [test configuration](#) specifies how the [test item](#) and [test components](#) are interconnected and what configuration data are needed. Configuration data are specified as part of the [test item configuration](#) and [test component configuration](#) for the [test item](#) and each [test component](#).

We explicitly classify [test configuration](#) into two categories: [abstract test configuration](#) and [concrete test configuration](#) such that enabling the generation of [concrete test configurations](#) from an [abstract test configuration](#) would be possible.

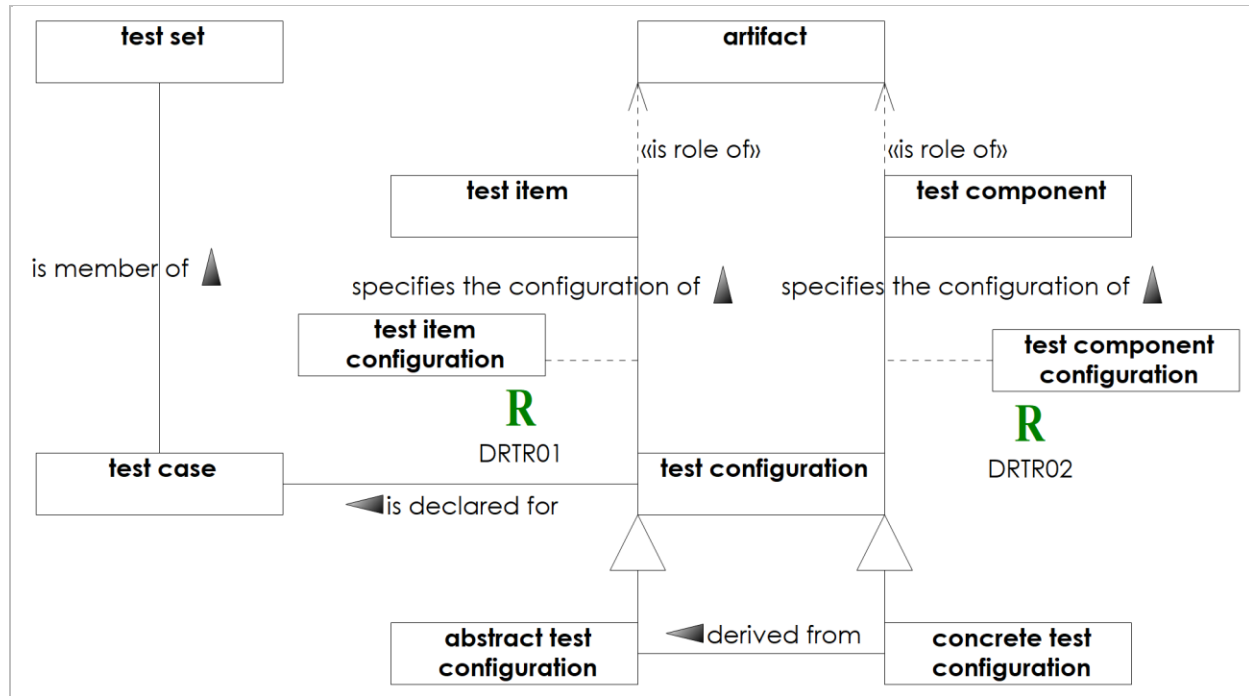


Figure 7.4 - Test Architecture Overview

Definitional Rules shown on "Test Architecture Overview"

Table 7.1 - Structural rules shown on Test Architecture Overview

Name	Rule statement
DRTR01	It is necessary that each <a href="#">test item configuration</a> specifies the configuration of at least one <a href="#">test item</a> .
DRTR02	It is necessary that each <a href="#">test component configuration</a> specifies the configuration of at least one <a href="#">test component</a> .

### 7.2.2 Concept Descriptions

abstract test configuration	
Definition	A <a href="#">test configuration</a> that specifies the <a href="#">test item</a> , <a href="#">test components</a> and their interconnections as well as configuration data that should be abstract test data.
Source	UTP 2
Is a	<a href="#">test configuration</a>

<b>artifact</b>	
Definition	An object produced or modified during the execution of a process.
Synonyms	work product
Examples	<ul style="list-style-type: none"> <li>• Software XY.</li> <li>• Software Requirements Specification.</li> <li>• Coffee machine.</li> <li>• Coffee bean.</li> </ul>
Source	UTP 2

<b>concrete test configuration</b>	
Definition	A <a href="#">test configuration</a> that specifies the <a href="#">test item</a> , <a href="#">test components</a> and their interconnections as well as configuration data that should be concrete <a href="#">data</a> .
Source	UTP 2
Is a	<a href="#">test configuration</a>

<b>test component</b>	
Definition	A role of an <a href="#">artifact</a> within a <a href="#">test configuration</a> that is required to perform a <a href="#">test case</a> .
Examples	<ul style="list-style-type: none"> <li>• A test driver</li> <li>• A test stub</li> <li>• Coffee machine that grinds the coffee beans to be tested.</li> </ul>
Source	UTP 2
Sub categories	<a href="#">data provider</a>
Is role of	<a href="#">artifact</a>

<b>test component configuration</b>	
Definition	A set of configuration options offered by an <a href="#">artifact</a> in the role of a <a href="#">test component</a> chosen to meet the requirements of a particular <a href="#">test configuration</a> .
Source	UTP 2

<b>test configuration</b>	
Definition	A specification of the test item and test components as well as their interconnection and configuration data.
Source	UTP 2
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">abstract test configuration</a></li> <li>• <a href="#">concrete test configuration</a></li> </ul>

<b>test item</b>	
Definition	A role of an <a href="#">artifact</a> that is the object of testing within a <a href="#">test configuration</a> .
Synonyms	System Under Test, SUT
Examples	<ul style="list-style-type: none"> <li>• Software XY to be tested.</li> <li>• Software Requirements Specification to be reviewed.</li> <li>• Coffee machine to be tested.</li> <li>• Coffee beans to be tested.</li> </ul>
Abbreviation	SUT
Source	UTP 2
Is role of	<a href="#">artifact</a>

<b>test item configuration</b>	
Definition	A set of configuration options offered by an <a href="#">artifact</a> in the role of a <a href="#">test item</a> chosen to meet the requirements of a particular <a href="#">test configuration</a> .
Source	UTP 2

## 7.3 Test Behavior

### 7.3.1 Test Cases

#### 7.3.1.1 Test Case Overview

The following concept diagram represents important semantic aspects in the context of what a [test case](#) is and what its components are. A [test case](#) invokes a [test procedure](#) describing the execution order of individual [test actions](#) (not shown here, see [Test Procedures](#) and [Test-specific Actions](#) for details). A [test case](#) is specialized into [abstract test case](#) and [concrete test case](#) depending on the availability of [data](#). If all the [data](#) required for a [test case](#) is available, it is classified as a [concrete test case](#) and [abstract test case](#) otherwise.

As shown in [Test Context Overview](#), [test cases](#) may be grouped into [test sets](#). A [test execution schedule](#) prescribes execution order of this set of [test cases](#). All, [test cases](#), [test procedure](#), and [test execution schedule](#) may require a [precondition](#) and may guarantee a [postcondition](#), each of which plays the role of [boolean expression](#).

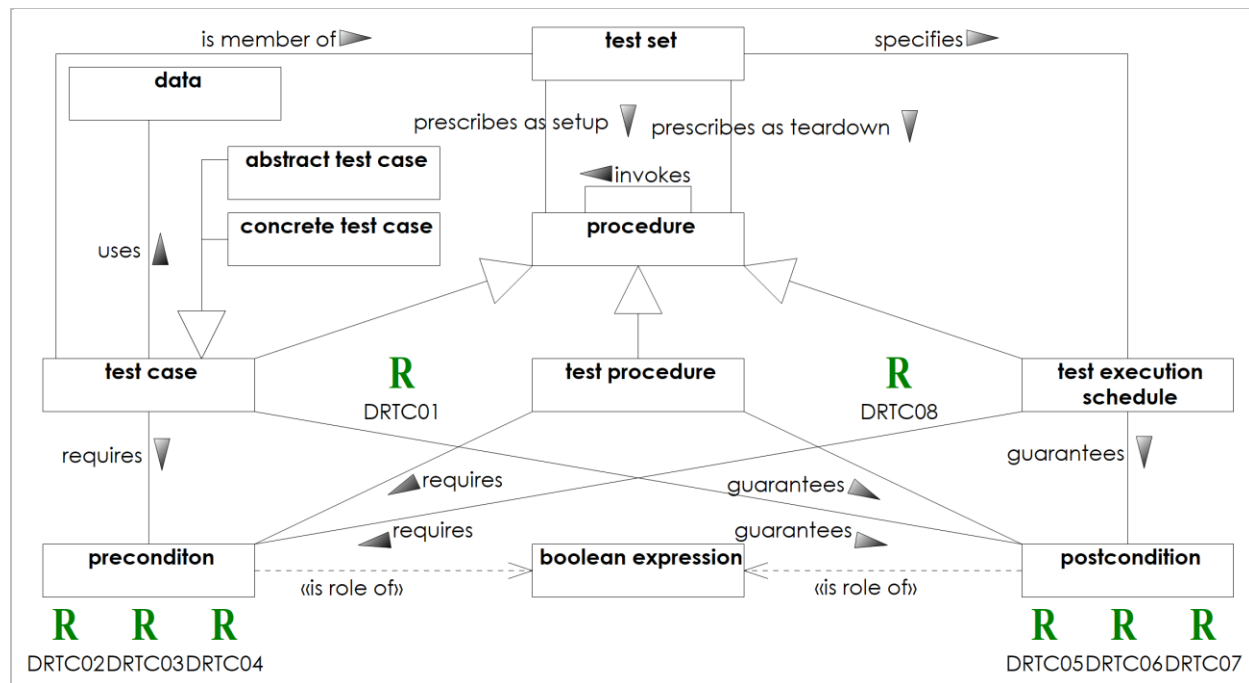


Figure 7.5 - Test Case Overview

Definitional Rules shown on "Test Case Overview"

Table 7.2 - Structural rules shown on Test Case Overview

Name	Rule statement
DRTC01	It is necessary that each <a href="#">test case</a> invokes at least one <a href="#">test procedure</a> .
DRTC08	It is impossible that a <a href="#">test execution schedule</a> invokes a <a href="#">test procedure</a> .

#### 7.3.1.2 Concept Descriptions

abstract test case	
Definition	A <a href="#">test case</a> that declares at least one <a href="#">formal parameter</a> .
Source	UTP2
Is a	<a href="#">test case</a>
boolean expression	
Definition	An expression that may be evaluated to either of these values: "TRUE" or "FALSE".

Synonyms	predicate
Source	UTP2

<b>concrete test case</b>	
Definition	A <a href="#">test case</a> that declares no <a href="#">formal parameter</a> .
Source	UTP2
Is a	<a href="#">test case</a>

<b>postcondition</b>	
Definition	A <a href="#">boolean expression</a> that is guaranteed to be True after a <a href="#">test case</a> execution has been completed.
Source	UTP2
Is role of	<a href="#">boolean expression</a>

<b>precondition</b>	
Definition	A <a href="#">boolean expression</a> that must be met before a <a href="#">test case</a> may be executed.
Source	UTP2
Is role of	<a href="#">boolean expression</a>

<b>test case</b>	
Definition	A <a href="#">procedure</a> that includes a set of preconditions, inputs and expected results, developed to drive the examination of a <a href="#">test item</a> with respect to some <a href="#">test objectives</a> .
Source	UTP2
Is a	<a href="#">procedure</a>
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">abstract test case</a></li> <li>• <a href="#">concrete test case</a></li> </ul>

<b>test execution schedule</b>	
Definition	A <a href="#">procedure</a> that constrains the execution order of a number of <a href="#">test cases</a> .
Source	UTP2
Is a	<a href="#">procedure</a>

## 7.3.2 Test-specific Procedures

### 7.3.2.1 Test Procedures

The following concept diagram represents important semantic aspects of [procedures](#) as they are used in UTP. UTP distinguishes three different types of [procedures](#): [test execution schedules](#), [test cases](#) and [test procedures](#), which are all special forms of [procedures](#). In general, [procedures](#) may invoke other [procedures](#). Furthermore, all [procedures](#) may declare one or more [formal parameters](#) which are replaced by [actual parameters](#) upon [procedure invocation](#).

A [procedure](#) prescribes the execution order of a set of [procedural elements](#), which are either [atomic procedural elements](#) (such as [procedure invocations](#) or individual [test actions](#)) or [compound procedural elements](#).

A [compound procedural element](#) is a container that groups a set of [procedural elements](#) into [sequences](#), [loops](#), and other control structures.

Any [procedural element](#) may be constrained by time which is expressed by its possible fact statements of [time points](#) and [duration](#)s. A [procedural element](#) may be constrained on when it is to be performed as well as how long it is to be performed by the tester.

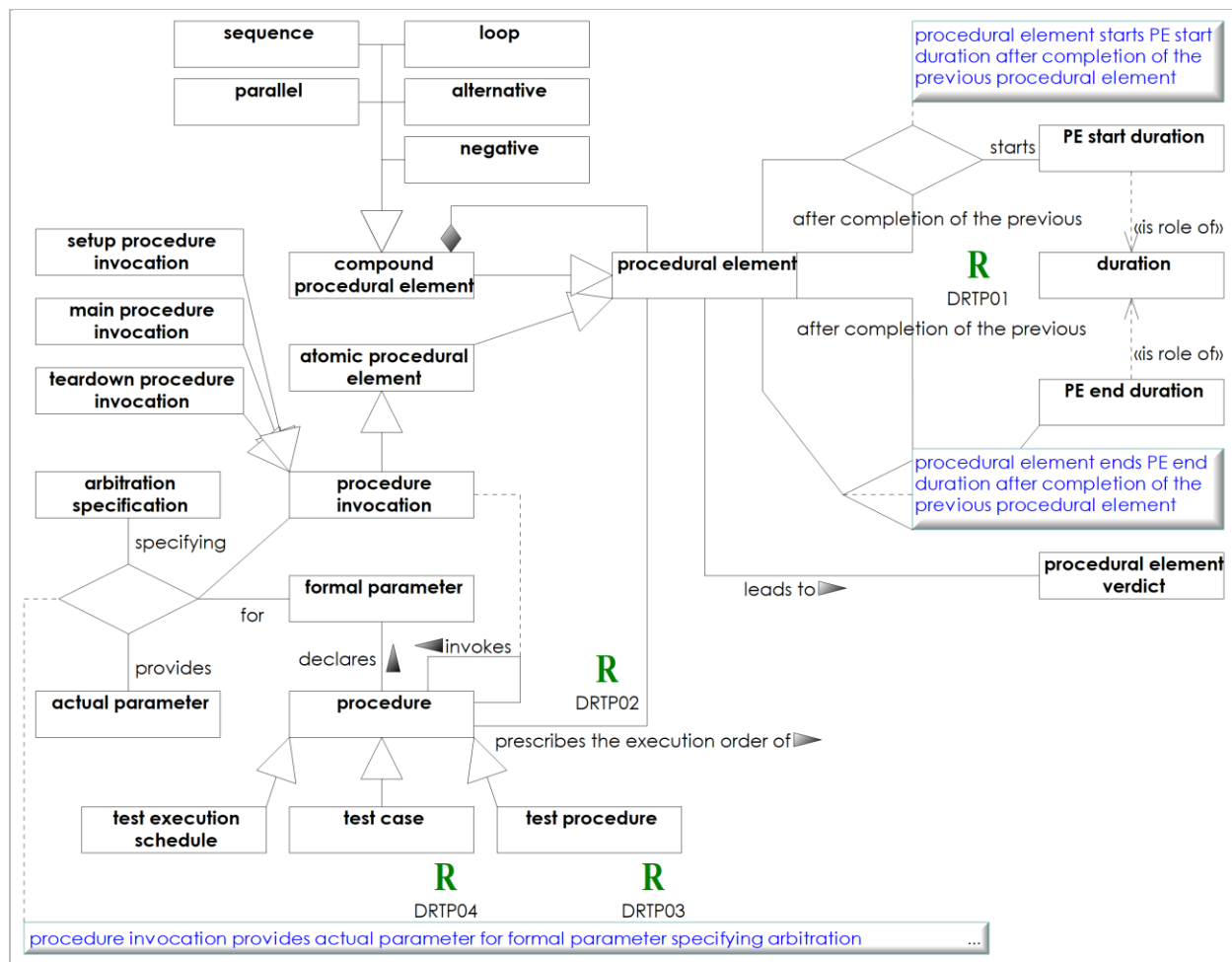


Figure 7.6 - Test Procedures

## Definitional Rules shown on "Test Procedures"

Table 7.3 - Structural rules shown on Test Procedures

Name	Rule statement
DRTP01	It is necessary that the <u>PE start duration</u> of a <u>procedural element</u> is smaller than the <u>PE end duration</u> of the same <u>procedural element</u> .
DRTP02	It is necessary that each <u>procedure</u> prescribes the execution order of at least one <u>procedural element</u> .
DRTP03	It is necessary that each <u>test procedure</u> prescribes the execution order of at least one <u>test action</u> .
DRTP04	It is necessary that each test case invokes at least one <u>test procedure</u> as a <u>main procedure invocation</u> .

### 7.3.2.2 Concept Descriptions

actual parameter	
Definition	A concrete value that is passed over to the <u>procedure</u> and replaces the <u>formal parameter</u> with its concrete value.
Source	UTP 2
alternative	
Definition	A <u>compound procedural element</u> that executes only a subset of its contained



	<a href="#">procedural elements</a> based on the evaluation of a <a href="#">boolean expression</a> .
Source	UTP 2
Is a	<a href="#">compound procedural element</a>

#### atomic procedural element

Definition	A <a href="#">procedural element</a> that cannot be further decomposed.
Source	UTP 2
Is a	<a href="#">procedural element</a>
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">procedure invocation</a></li> <li>• <a href="#">test action</a></li> </ul>

#### compound procedural element

Definition	A <a href="#">procedural element</a> that can be further decomposed.
Source	UTP 2
Is a	<a href="#">procedural element</a>
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">alternative</a></li> <li>• <a href="#">loop</a></li> <li>• <a href="#">negative</a></li> <li>• <a href="#">parallel</a></li> <li>• <a href="#">sequence</a></li> </ul>

#### UTP 2

duration	
Definition	The duration from the start of a test action until its completion.
Source	UTP 2 WG
Is a	duration

#### formal parameter

Definition	A placeholder within a <a href="#">procedure</a> that allows for execution of the <a href="#">procedure</a> with different <a href="#">formal parameters</a> that are provided by the <a href="#">procedure invocation</a> .
Source	UTP 2

#### loop

Definition	A <a href="#">compound procedural element</a> that repeats the execution of its contained <a href="#">procedural elements</a> .
Source	UTP 2
Is a	<a href="#">compound procedural element</a>

#### main procedure invocation

Definition	A <a href="#">procedure invocation</a> that is considered as the main part of a <a href="#">test case</a> by the <a href="#">test case arbitration specification</a> .
Source	UTP 2
Is a	<a href="#">procedure invocation</a>

#### negative

Definition	A <a href="#">compound procedural element</a> that prohibits the execution of its contained <a href="#">procedural elements</a> in the specified structure.
Source	UTP 2
Is a	<a href="#">compound procedural element</a>

#### parallel

Definition	A <a href="#">compound procedural element</a> that executes its contained <a href="#">procedural elements</a> in parallel to each other.
------------	--

Source	UTP 2
Is a	<a href="#">compound procedural element</a>

<b>PE end duration</b>	
Definition	The <a href="#">duration</a> between the end of the execution of a <a href="#">procedural element</a> and the end of the execution of the subsequent <a href="#">procedural element</a> .
Source	UTP 2
Is role of	<a href="#">duration</a>

<b>PE start duration</b>	
Definition	The <a href="#">duration</a> between the end of the execution of a <a href="#">procedural element</a> and the beginning of the execution of the subsequent <a href="#">procedural element</a> .
Source	UTP 2
Is role of	<a href="#">duration</a>

<b>procedural element</b>	
Definition	An instruction to do, to observe, and/or to decide.
Source	UTP 2
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">atomic procedural element</a></li> <li>• <a href="#">compound procedural element</a></li> </ul>

<b>procedure</b>	
Definition	A specification that constrains the execution order of a number of <a href="#">procedural elements</a> .
Source	UTP 2
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">test case</a></li> <li>• <a href="#">test execution schedule</a></li> <li>• <a href="#">test procedure</a></li> </ul>

<b>procedure invocation</b>	
Definition	An atomic procedural element of a procedure that invokes another procedure and waits for its completion.
Source	UTP 2
Is a	<a href="#">atomic procedural element</a>
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">main procedure invocation</a></li> <li>• <a href="#">setup procedure invocation</a></li> <li>• <a href="#">teardown procedure invocation</a></li> </ul>

<b>sequence</b>	
Definition	A <a href="#">compound procedural element</a> that executes its contained <a href="#">procedural elements</a> sequentially.
Source	UTP 2
Is a	<a href="#">compound procedural element</a>

<b>setup procedure invocation</b>	
Definition	A <a href="#">procedure invocation</a> that is considered as part of the setup by the <a href="#">arbitration specification</a> and that is invoked before any <a href="#">main procedure invocation</a> .
Source	UTP 2
Is a	<a href="#">procedure invocation</a>

<b>teardown procedure invocation</b>	
Definition	A <a href="#">procedure invocation</a> that is considered as part of the teardown by the responsible <a href="#">arbitration specification</a> and that is invoked after any <a href="#">main procedure invocation</a> .

Source	UTP 2
Is a	<a href="#">procedure invocation</a>

<b>test procedure</b>	
Definition	A <a href="#">procedure</a> that constrains the execution order of a number of <a href="#">test actions</a> .
Source	UTP 2
Is a	<a href="#">procedure</a>

<b>time point</b>	
Definition	The <a href="#">time point</a> at which a <a href="#">test action</a> is initiated.
Source	UTP 2
Is a	time point

### 7.3.3 Test-specific Actions

#### 7.3.3.1 Overview of test-specific actions

The following concept diagram represents important semantic aspects of [test actions](#) as parts of [test procedures](#). A [test action](#) is a specialization of an [atomic procedural element](#) and is to be interpreted as an instruction to the tester responsible for executing a [test case](#). Any [test action](#) leads to a [procedural element verdict](#) (i.e., influences the final [test case verdict](#)).

Most [test actions](#) check certain aspects of the [test item](#). The most important aspects of the [test item](#) are its observable behavior (i.e., its [responses](#)) and its measurable properties.

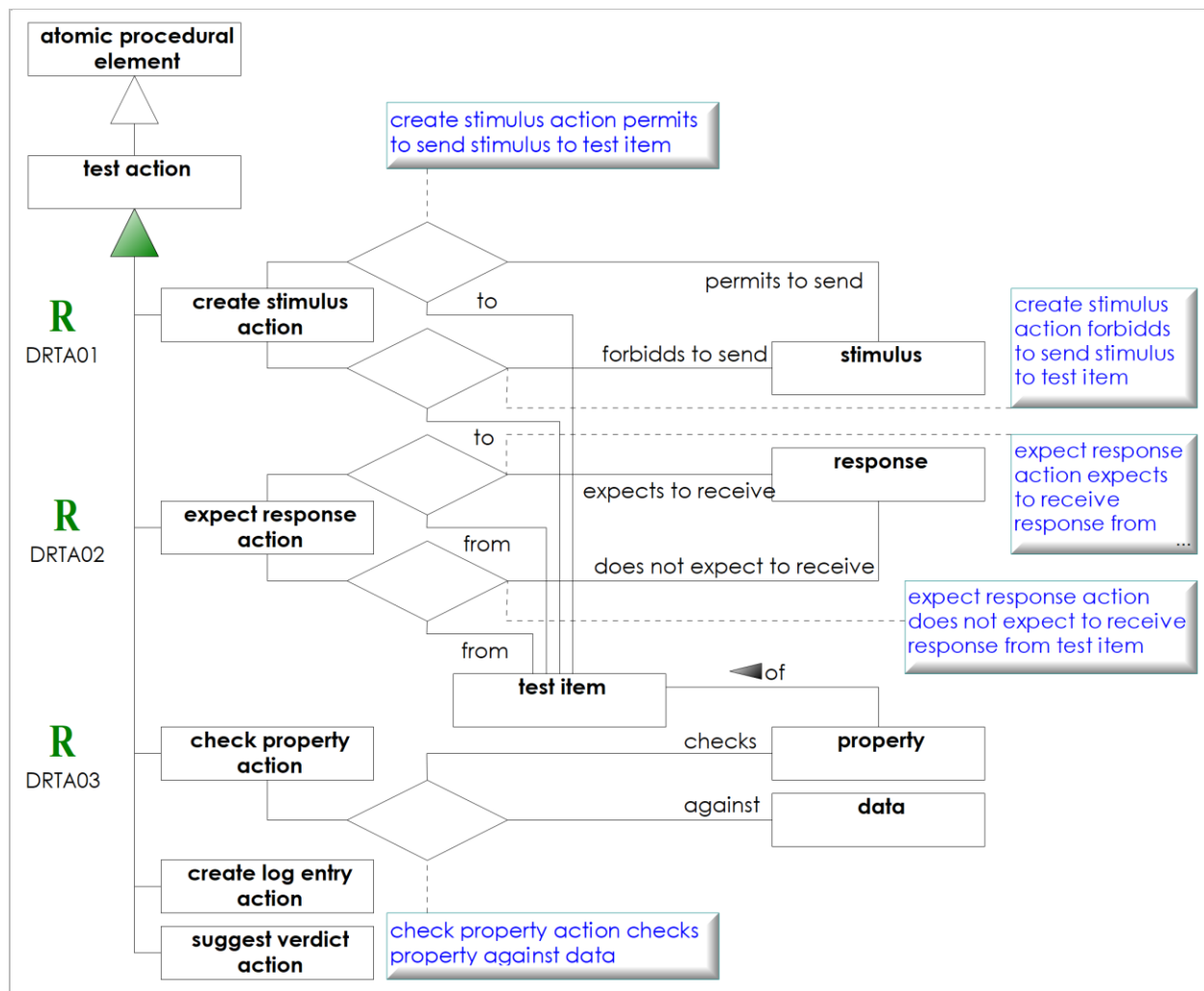


Figure 7.7 - Overview of test-specific actions

Definitional Rules shown on "Overview of test-specific actions"

Table 7.4 - Structural rules shown on Overview of test-specific actions

Name	Rule statement
DRTA01	It is necessary that a <a href="#">create stimulus action</a> permits to send at least one <a href="#">stimulus</a> .
DRTA02	It is necessary that a <a href="#">expect response action</a> expects to receive at least one <a href="#">response</a> .
DRTA03	It is necessary that a <a href="#">check property action</a> checks at least one <a href="#">property</a> of the <a href="#">test item</a> against the <a href="#">data</a> .

### 7.3.3.2 Concept Descriptions

check property action	
Definition	A <a href="#">test action</a> that instructs the tester to check the conformance of a <a href="#">property</a> of the <a href="#">test item</a> and to set the <a href="#">procedural element verdict</a> according to the result of this check.
Source	UTP 2
Is a	<a href="#">test action</a>

<b>create log entry action</b>	
Definition	A <a href="#">test action</a> that instructs the tester to record the execution of a <a href="#">test action</a> , potentially including the outcome of that <a href="#">test action</a> in the <a href="#">test case log</a> .
Source	UTP 2
Is a	<a href="#">test action</a>

<b>create stimulus action</b>	
Definition	A <a href="#">test action</a> that instructs the tester to submit a <a href="#">stimulus</a> (potentially including <a href="#">data</a> ) to the <a href="#">test item</a> .
Source	UTP 2
Is a	<a href="#">test action</a>

<b>expect response action</b>	
Definition	A <a href="#">test action</a> that instructs the tester to check the occurrence of one or more particular <a href="#">responses</a> from the <a href="#">test item</a> within a given time window and to set the <a href="#">procedural element verdict</a> according to the result of this check.
Source	UTP 2
Is a	<a href="#">test action</a>

<b>property</b>	
Definition	A basic or essential attribute shared by all members of a class of <a href="#">test items</a> .
Source	UTP 2

<b>response</b>	
Definition	A set of <a href="#">data</a> that is sent by the <a href="#">test item</a> to its environment (often as a reaction to a <a href="#">stimulus</a> ) and that is typically used to assess the behavior of the <a href="#">test item</a> .
Source	UTP 2

<b>stimulus</b>	
Definition	A set of <a href="#">data</a> that is sent to the <a href="#">test item</a> by its environment (often to cause a <a href="#">response</a> as a reaction) and that is typically used to control the behavior of the <a href="#">test item</a> .
Source	UTP 2

<b>suggest verdict action</b>	
Definition	A <a href="#">test action</a> that instructs the tester to suggest a particular <a href="#">procedural element verdict</a> to the <a href="#">arbitration specification</a> of the <a href="#">test case</a> for being taken into account in the final <a href="#">test case verdict</a> .
Source	UTP 2
Is a	<a href="#">test action</a>

<b>test action</b>	
Definition	An <a href="#">atomic procedural element</a> that is an instruction to the tester that needs to be executed as part of a test procedure of a test case within some time frame.
Synonyms	test step
Source	UTP 2
Is a	<a href="#">atomic procedural element</a>
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">check property action</a></li> <li>• <a href="#">create log entry action</a></li> <li>• <a href="#">create stimulus action</a></li> <li>• <a href="#">expect response action</a></li> <li>• <a href="#">suggest verdict action</a></li> </ul>

## 7.4 Test Data

### 7.4.1 Test Data Concepts

The following concept diagram represents important semantic aspects of test [data](#). Test [data](#) or more generally just [data](#) may be modeled at two different levels:

- **Extensional level:** model elements that actually represent some [data](#) composed as a set of individual [data items](#).
- **Intensional level:** model elements that specify some criteria that some [data](#) must comply with, i.e. the specification of the meaning of [data](#).

At the *extensional level* [data](#) always represents a specific set of [data items](#) and is covered by concepts such as [data pool](#), [actual data pool](#), and [data partition](#). The concepts [data pool](#) and [actual data pool](#) represent containers of [data](#), the former is a logical container, the latter a physical container such as a concrete database. A [data partition](#) represents a subset of another set of [data items](#) in which all [data item](#) are conformant to a particular [data specification](#).

In contrast, at the *intensional level*, [data](#) is represented by a boolean expression that may be used to qualify [data items](#) as member of [data](#), i.e., it represents the intended meaning of [data](#) and is covered by concepts such as [data specification](#), [data type](#), and [constraint](#). A [data specification](#) is composed of a basic [data type](#) plus a set of [constraints](#) on that [data type](#). The entire concept of a [data specification](#) may be considered as a category in the sense of "Category Theory" in mathematics (see for example [\[WikiCT\]](#) or [\[SEP2014a\]](#)). Thus, two [data specifications](#) might be interpreted as categories that are related to each other by means of different dependencies called "[morphisms](#)". These may be considered as structure-preserving maps supporting the following three informal semantics:

- A [morphism](#) of type "[extension](#)" increases the amount of [data](#), i.e. they add more [data items](#) to a given set of [data items](#).
- A [morphism](#) of type "[refinement](#)" decreases the amount of [data](#), i.e. they remove [data items](#) from a given set of [data items](#).
- A [morphism](#) of type "[complement](#)" inverts [data](#), i.e. it replaces the [data items](#) of a given set of [data items](#) by their opposites.

A [data provider](#) is a [test component](#) that is able to deliver (i.e. either select and/or generate) [data](#) according to a [data specification](#).

In the context of a [test case](#), different places of a [test case](#) typically refer to different levels of test [data](#):

- [Test cases](#) typically refer to [data](#) used as preconditions as well as [data](#) to be supplied with stimuli to be sent to the [test item](#).
- [Test cases](#) typically refer to [data specifications](#) in postconditions or [data](#) returned by responses in order to determine or influence the [verdict](#) of the [test case](#).

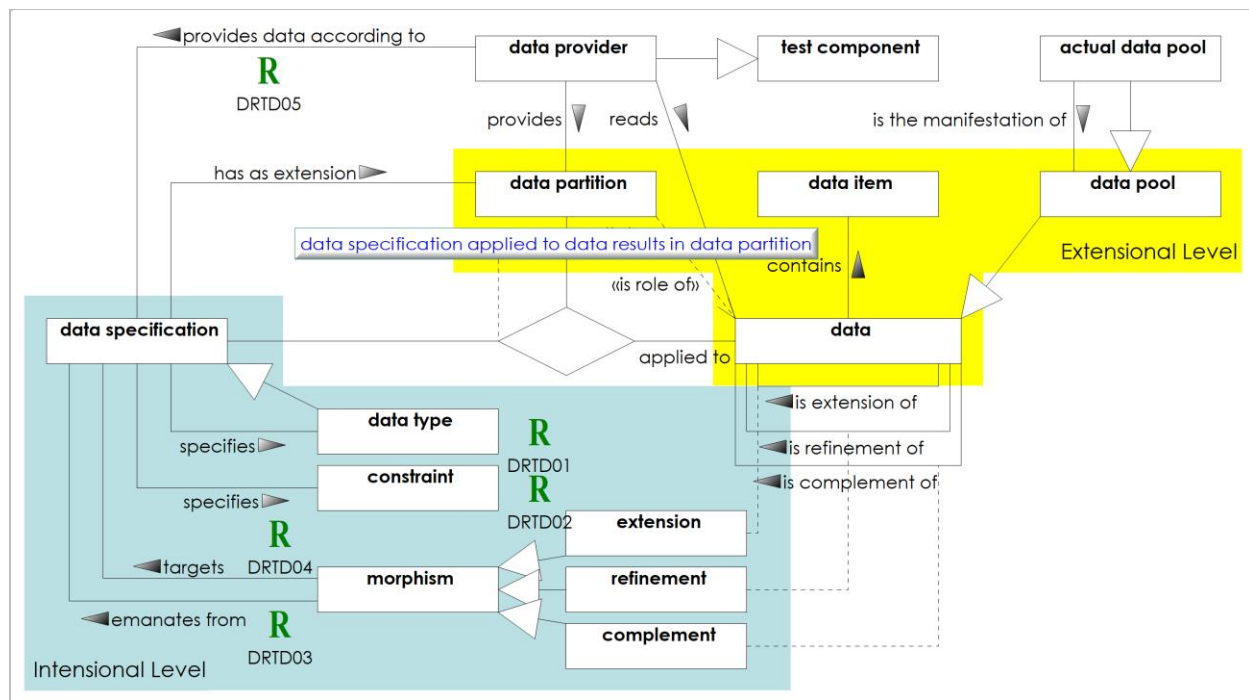


Figure 7.8 - Test Data Concepts

Definitional Rules shown on "Test Data Concepts"

Table 7.5 - Structural rules shown on Test Data Concepts

Name	Rule statement
DRTD01	It is necessary that each <a href="#">data specification</a> specifies at least one <a href="#">data type</a> .
DRTD02	It is necessary that each <a href="#">data specification</a> specifies at least one <a href="#">constraint</a> .
DRTD03	It is necessary that a <a href="#">morphism</a> emanates from exactly one <a href="#">data specification</a> .
DRTD04	It is necessary that a <a href="#">morphism</a> targets exactly one <a href="#">data specification</a> .
DRTD05	It is necessary that each <a href="#">data provider</a> provides data according to at least one <a href="#">data specification</a> .

## 7.4.2 Concept Descriptions

actual data pool	
Definition	A specification of an actual implementation of a <a href="#">data pool</a> .
Examples	<ul style="list-style-type: none"> <li>the specification of the database of type "Customers" on disk DK13 on machine XYZ.</li> </ul>
Source	UTP 2
Is a	<a href="#">data pool</a>
complement	
Definition	A <a href="#">morphism</a> that inverts <a href="#">data</a> (i.e., that replaces the <a href="#">data items</a> of a given set of <a href="#">data items</a> by their opposites).
Source	UTP 2
Is a	<a href="#">morphism</a>
constraint	
Definition	An assertion that indicates a restriction that must be satisfied by any valid realization of the model containing the <a href="#">constraint</a> .
Source	<a href="#">[UML]</a>

<b>data</b>	
Definition	A usually named set of <a href="#">data items</a> .
Synonyms	concrete data
Examples	<ul style="list-style-type: none"> <li>• 42.</li> <li>• "John".</li> <li>• "Some people": {"John", "Greg", "Barb", "Aline"}</li> <li>• "Example customer": Sherlock Holmes, living at Baker Street in London</li> <li>• The contents of a database "CUST-PRD" containing customers.</li> </ul>
Source	UTP 2
Subcategories	<a href="#">data pool</a>
Is instance of	data structure

<b>data item</b>	
Definition	Either a value or an instance.
Source	UTP 2

<b>data partition</b>	
Definition	A role that some <a href="#">data</a> plays with respect to some other <a href="#">data</a> (usually being a subset of this other <a href="#">data</a> ) with respect to some <a href="#">data specification</a> .
Source	UTP 2
Is role of	<a href="#">data</a>

<b>data pool</b>	
Definition	Some <a href="#">data</a> that is an explicit or implicit composition of other <a href="#">data items</a> .
Examples	<ul style="list-style-type: none"> <li>• the specification of a database type named "Customers"</li> </ul>
Source	UTP 2
Is a	<a href="#">data</a>
Subcategories	<a href="#">actual data pool</a>

<b>data provider</b>	
Definition	A <a href="#">test component</a> that is able to deliver (i.e., either select and/or generate) <a href="#">data</a> according to a <a href="#">data specification</a> .
Source	UTP 2
Is a	<a href="#">test component</a>

<b>data specification</b>	
Definition	A named <a href="#">boolean expression</a> composed of a <a href="#">data type</a> and a set of <a href="#">constraints</a> applicable to some <a href="#">data</a> in order to determine whether or not its <a href="#">data items</a> conform to this <a href="#">data specification</a> .
Synonyms	abstract data
Examples	<ul style="list-style-type: none"> <li>• 40...50.</li> <li>• "Jo(h)?n".</li> <li>• "odd numbers", i.e. numbers where <math>\text{self} \bmod 2 = 1</math></li> <li>• "right-angled triangles", i.e. triangles where <math>a^2 + b^2 = c^2</math></li> <li>• "young, German-speaking customers" i.e., customers, where language='German' and age &lt; 18</li> <li>• any/all/295 customers having the forename "John" and living in London.</li> </ul>
Source	UTP 2
Subcategories	<a href="#">data type</a>



data type	
Definition	A type whose instances are identified only by their value.
Source	<a href="#">[UML]</a>
Is a	<a href="#">data specification</a>

extension	
Definition	A <a href="#">morphism</a> that increases the amount of <a href="#">data</a> (i.e., that adds more <a href="#">data items</a> to a given set of <a href="#">data items</a> ).
Source	UTP 2
Is a	<a href="#">morphism</a>

morphism	
Definition	A structure-preserving map from one mathematical structure to another.
Source	<a href="#">[WikiM]</a>
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">complement</a></li> <li>• <a href="#">extension</a></li> <li>• <a href="#">refinement</a></li> </ul>

refinement	
Definition	A <a href="#">morphism</a> that decreases the amount of <a href="#">data</a> (i.e., that removes <a href="#">data items</a> from a given set of <a href="#">data items</a> ).
Source	UTP 2
Is a	<a href="#">morphism</a>

## 7.5 Test Evaluation

### 7.5.1 Arbitration Specifications

#### 7.5.1.1 Arbitration & Verdict Overview

The following concept diagram represents important semantic aspects of [verdicts](#) and how they are derived.

An [arbitration specification](#) is defined as a set of rules that should be followed to determine the instance of a [verdict](#) of an executed [test case](#). An [arbitration specification](#) should be specified for a [procedure](#) which describes the behavior of [test case](#) ([test procedure](#)) or a [test execution schedule](#) (associated to the execution of a set of [test cases](#)). An [arbitration specification](#) calculates a [verdict](#) which can be [Fail](#), [Pass](#), [Inconclusive](#) and [None](#).

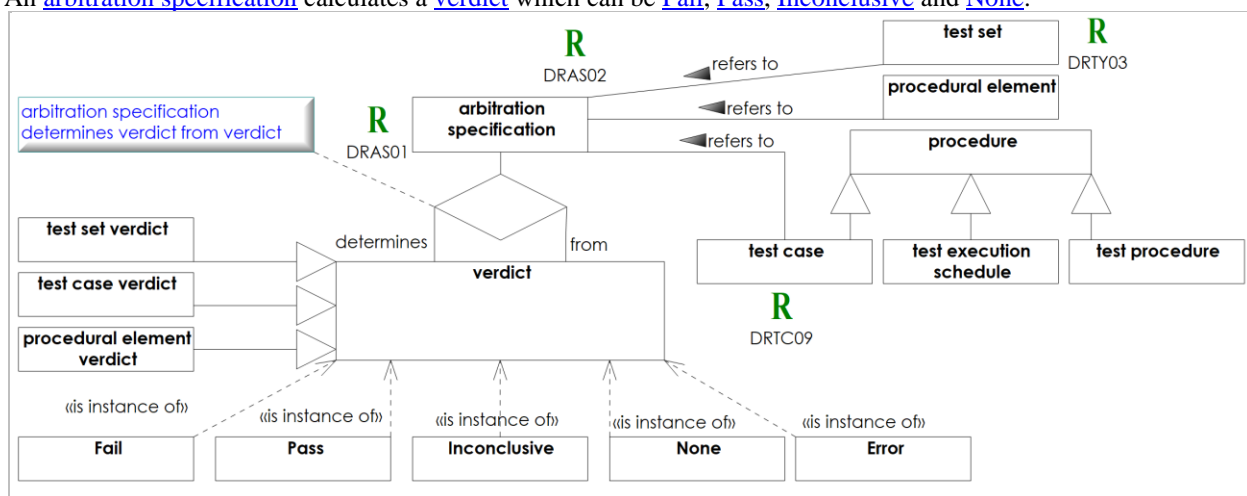


Figure 7.9 - Arbitration & Verdict Overview

## Definitional Rules shown on "Arbitration & Verdict Overview"

Table 7.6 - Structural rules shown on Arbitration & Verdict Overview

Name	Rule statement
DRAS01	It is necessary that an <a href="#">arbitration specification</a> determines exactly one type of <a href="#">verdict</a> .
DRAS02	It is necessary that an <a href="#">arbitration specification</a> determines exactly one of a <a href="#">test set verdict</a> , a <a href="#">test case verdict</a> or a <a href="#">procedural element verdict</a> .

### 7.5.1.2 Concept Descriptions

arbitration specification	
Definition	A set of rules that calculates the eventual <a href="#">verdict</a> of an executed <a href="#">test case</a> , test set or procedural element.
Source	UTP 2
Error	
Definition	An indication that an unexpected exception has occurred while executing a specific <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> .
Source	UTP 2
Is instance of	<a href="#">verdict</a>
Fail	
Definition	A <a href="#">verdict</a> that indicates that the <a href="#">test item</a> did not comply with the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> during execution.
Source	UTP 2
Is instance of	<a href="#">verdict</a>
Inconclusive	
Definition	A <a href="#">verdict</a> that indicates that the compliance of a <a href="#">test item</a> against the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> could not be determined during execution.
Source	UTP 2
Is instance of	<a href="#">verdict</a>
None	
Definition	A <a href="#">verdict</a> that indicates that the compliance of a <a href="#">test item</a> against the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> has not yet been determined (i.e., it is the initial value of a <a href="#">verdict</a> when a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> was started).
Source	UTP 2
Is instance of	<a href="#">verdict</a>
Pass	
Definition	A <a href="#">verdict</a> that indicates that the <a href="#">test item</a> did comply with the expectations defined by a <a href="#">test set</a> , <a href="#">test case</a> , or <a href="#">test action</a> during execution.
Source	UTP 2
Is instance of	<a href="#">verdict</a>
procedural element verdict	
Definition	A <a href="#">verdict</a> that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test action</a> on a <a href="#">test item</a> .
Source	UTP 2
Is a	<a href="#">verdict</a>
test case verdict	
Definition	A <a href="#">verdict</a> that indicates the result (i.e., the conformance of the actual properties of

	the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test case</a> against a <a href="#">test item</a> .
Source	UTP 2
Is a	<a href="#">verdict</a>

<b>test set verdict</b>	
Definition	A <a href="#">verdict</a> that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test set</a> against a <a href="#">test item</a> .
Source	UTP 2
Is a	<a href="#">verdict</a>

<b>verdict</b>	
Definition	A statement that indicates the result (i.e., the conformance of the actual properties of the <a href="#">test item</a> with its expected properties) of executing a <a href="#">test set</a> , a <a href="#">test case</a> , or a <a href="#">test action</a> against a <a href="#">test item</a> .
Source	UTP 2
Subcategories	<ul style="list-style-type: none"> <li>• <a href="#">procedural element verdict</a></li> <li>• <a href="#">test case verdict</a></li> <li>• <a href="#">test set verdict</a></li> </ul>
Instances	<ul style="list-style-type: none"> <li>• <a href="#">Pass</a></li> <li>• <a href="#">Inconclusive</a></li> <li>• <a href="#">None</a></li> <li>• <a href="#">Error</a></li> <li>• <a href="#">Fail</a></li> </ul>

## 7.5.2 Test Logging

### 7.5.2.1 Test Log Overview

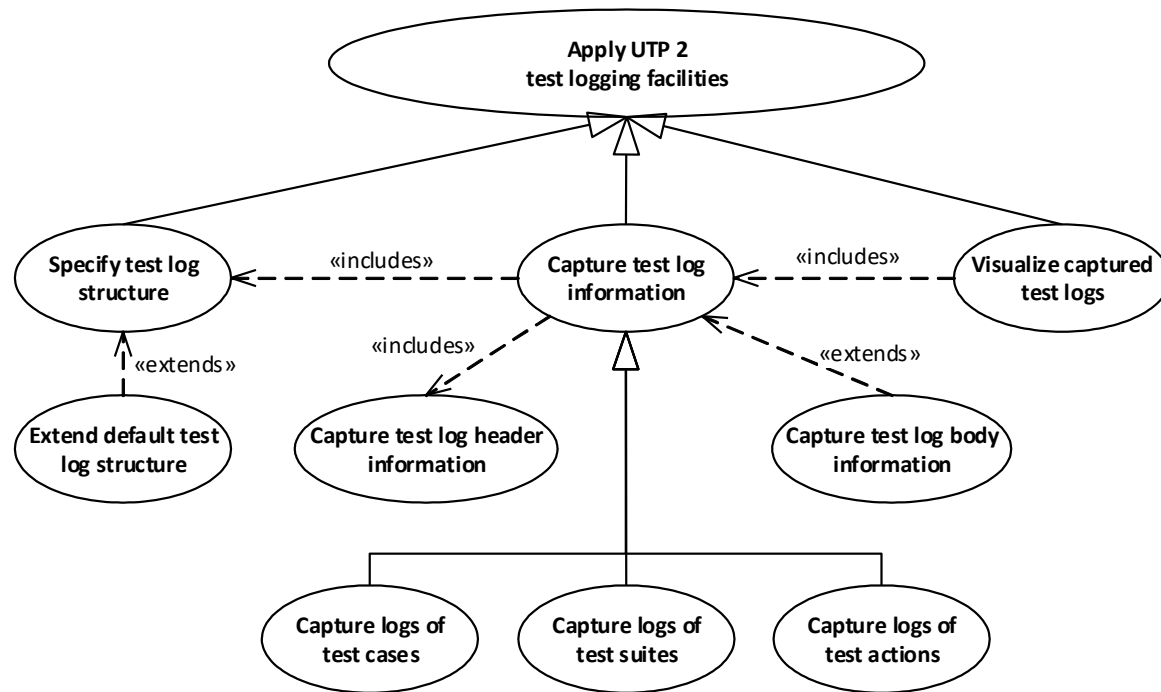
As defined by [\[ISTQB\]](#) a [test log](#) is “a chronological record of relevant details about the execution of tests” and as such is an important means for test evaluation and reporting activities. Thus, the purpose of the UTP 2 [test logging](#) facility is twofold:

- 1.) It helps establish a trace link between a [test case](#) or an entire [test set](#) and one or potentially more executions thereof. Essential information of a [test log](#) are, for example, the date and the [duration](#) when the corresponding [test case](#) was executed; the [executing entity](#) (i.e., a human tester or automated test execution system) or entities (in some domains it is not uncommon that [test cases](#) are executed over several days by potentially more than one [executing entity](#)), and finally, the [test case verdict](#). These so called [test log](#) header information are the minimal required information in order to achieve full traceability between [test objectives](#), [test requirements](#), [test cases/test sets](#) and finally the execution thereof. Full traceability among those [artifacts](#) enables the computation of test metrics such as the status of test execution (how many [test cases](#) have eventually been executed at a certain point in time), coverage of requirements (not part of UTP), [test requirements](#) or [test objectives](#), etc.
- 2.) It supports a deeper analysis of what was going on during the execution of a [test case](#) or [test set](#). Since the execution of [test case](#) or [test set](#) is a transient set of [test actions](#) performed by an [executing entity](#) against the [test item](#), the capturing of detailed information about the performed [test actions](#) in a [test log](#) is the only way for a stakeholder, usually a test analyst or test manager, to be able to comprehend what has really happened during execution without being part of the executing entities. Such a chronological record of detailed information of an executed [test case](#) or [test set](#) is in UTP 2 called [test log](#) body information. They optionally supplement the [test log](#) header information of UTP.

Since the understanding of what information is really relevant during the execution of a [test case](#) or [test set](#) heavily depends on domain- and/or project-specific requirements, UTP 2 enables the definition of user-defined [test log structures](#) that specify what information or data deemed relevant in the respective (test) context and additionally the

minimal required header information mentioned above.

Representing [test logs](#) on model level contributes to a harmonized and homogeneous view on relevant [test log](#) information in the dynamic test process. Usually, a test execution toolscape comprises more than just one tool. Tools for functional testing might be [complemented](#) by specialized tools such as those for performance testing (stress, load etc.), security testing or UI testing. The [test logs](#) of such heterogeneous toolscapes are basically heterogeneous, too. Thus, a comprehensive, detailed analysis (e.g., for the calculation of metrics over tools etc.) requires access to the proprietary structures of each tool's [test log](#) format. The UTP 2 [test logging](#) facility mitigates the heterogeneity of [test logs](#) by offering an extensible framework to describe arbitrary complex and structured [test log](#) formats. The following use cases depict the scenarios the UTP 2 [test logging](#) facility was intended to cope with:



**Figure 7.10 - Use Cases of UTP 2 test log XE "test log" ging Facility**

The use case “Specify [test log structure](#)” enables testers to specify which information is deemed relevant during the execution of in the given test process in addition to the predefined minimal required information. If no additional information is desired, the tester can rely on the implicit default [test log structure](#). This ensures that testers can employ the UTP 2 [test logging](#) facilities immediately out of the box.

The use case “Capture [test log](#) information” is about capturing the information deemed as relevant that actually appeared during the execution of a [test case](#), [test set](#) or even a [test action](#) in accordance with the [test log structure](#). Incorporating the [test log](#) header information is mandatory, while representing the body part, in contrast, is optional.

The use case “Visualize captured [test logs](#)” deals with exposing the captured [test log](#) information in an appropriate representation. Since there is no common definition of the most appropriate format of [test logs](#), UTP 2 does not prescribe how that information must be visualized. Thus, it is up to tool vendors to decide about the most appropriate and helpful visual representation(s) of captured [test log](#) information.

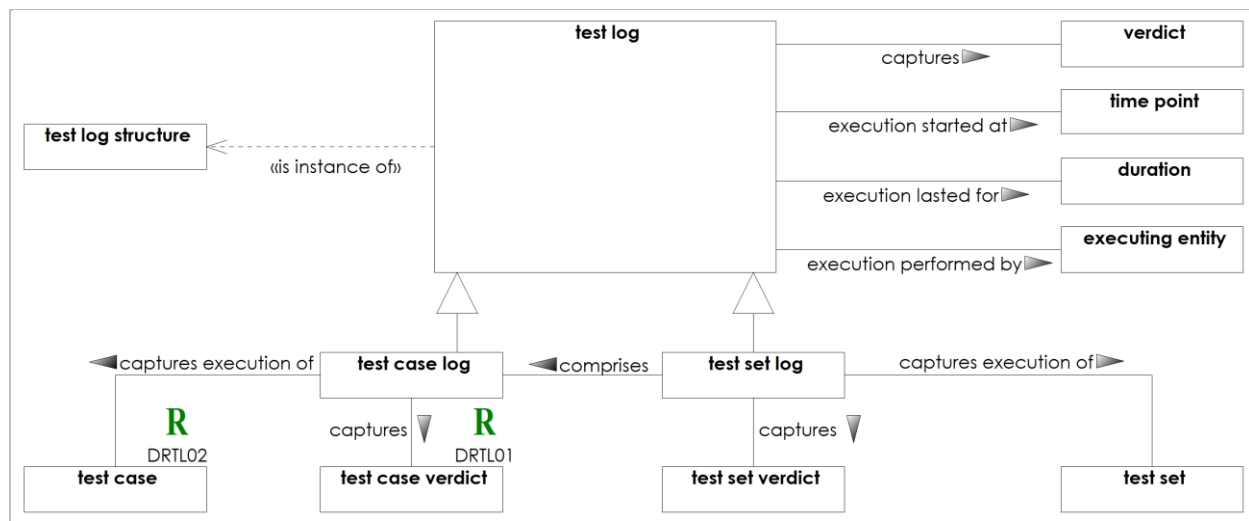


Figure 7.11 - Test Log Overview

Definitional Rules shown on "Test Log Overview"

Table 7.7 - Structural rules shown on Test Log Overview

Name	Rule statement
DRTL01	It is necessary that each <a href="#">test case log</a> captures exactly one <a href="#">test case verdict</a> .
DRTL02	It is necessary that each <a href="#">test case log</a> captures execution of exactly one <a href="#">test case</a> .

### 7.5.2.2 Concept Descriptions

executing entity	
Definition	An <a href="#">executing entity</a> is a human being or a machine that is responsible for executing a <a href="#">test case</a> or a <a href="#">test set</a> .
Source	UTP 2

test case log	
Definition	A <a href="#">test log</a> that captures relevant information on the execution of a <a href="#">test case</a> .
Source	UTP 2
Is a	<a href="#">test log</a>

test log	
Definition	A <a href="#">test log</a> is the instance of a <a href="#">test log structure</a> that captures relevant information from the execution of a <a href="#">test case</a> or <a href="#">test set</a> . The least required information to be logged is defined by the <a href="#">test log structure</a> of the <a href="#">test log</a> .
Source	UTP 2
Subcategories	<ul style="list-style-type: none"> <li><a href="#">test case log</a></li> <li><a href="#">test set log</a></li> </ul>
Is instance of	<a href="#">test log structure</a>

test log structure	
Definition	A <a href="#">test log structure</a> specifies the information that is deemed relevant during execution of a <a href="#">test case</a> or a <a href="#">test set</a> . There is an implicit default <a href="#">test log structure</a> that prescribes at least the start <a href="#">time point</a> , the <a href="#">duration</a> , the finally calculated verdict and the executing entity of a test case or test set execution which should be logged.
Source	UTP 2
Instances	<a href="#">test log</a>

<b>test set log</b>	
Definition	A <a href="#">test log</a> that captures relevant information from the execution of a <a href="#">test set</a> .
Source	UTP 2
Is a	<a href="#">test log</a>

## 8 Profile Specification

This section specifies the stereotypes that are defined by the UML Testing Profile.

### 8.1 Language Architecture

The UML Testing Profile consists of the profile definition and three normative model libraries, which can be imported and applied if required. The profile itself is independent of these libraries and is a self-contained package. The normative model library [UTP Auxiliary Library](#) uses concepts from UTP and defines concepts that can be used, extended or specialized by the users.

The UTP Types Library offers helpful types and values, in particular the default verdict type and the default verdict instances. Since some of the definitions and constraints in the profile are based on predefined types, the profile imports the UTP Types Library.

The [UTP Auxiliary Library](#) offers the following concepts:

- ISTQB terms for test levels and test set purposes
- Predefined test design techniques and test design technique structures

Overview of the technical, high-level UML Testing Profile language architecture is given next.

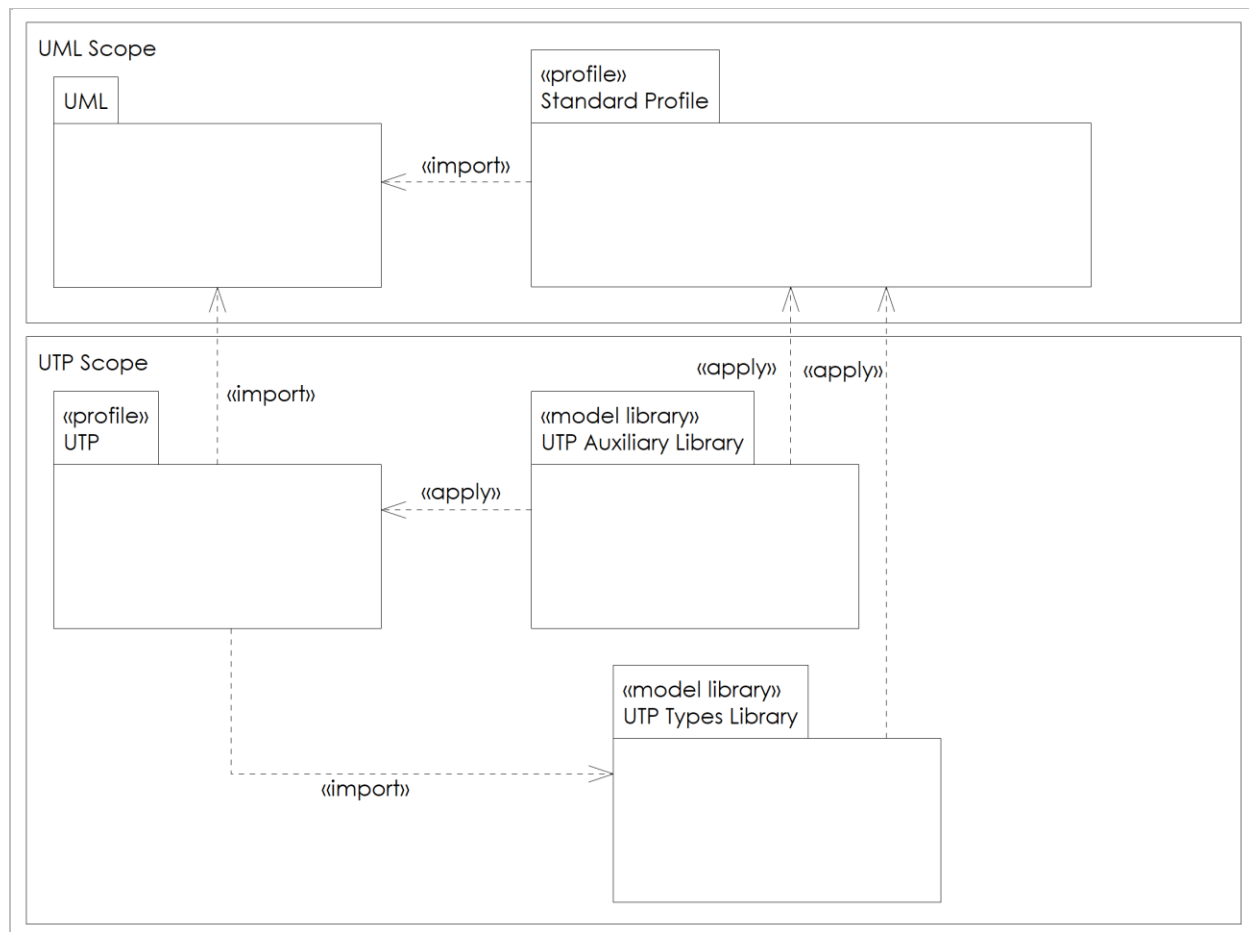


Figure 8.1 - Language Architecture

## 8.2 Profile Summary

The following table gives a brief summary on the stereotypes introduced by the UML Testing Profile 2 (listed in the second column of the table). The first column specifies the mapping to the conceptual model shown in the previous section and the third column specifies the UML 2.5 metaclasses that are extended by the stereotypes.

Stereotype	UML 2.5 Metaclasses	Concepts
ActualParameterValue	<b>Slot</b>	<a href="#">actual parameter</a>
ActualResponseLogEntry	<b>InstanceSpecification</b>	
Alternative	<b>CombinedFragment, StructuredActivityNode</b>	<a href="#">alternative</a>
AlternativeArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
AnyValue	<b>Expression</b>	<a href="#">data specification</a>
ArbitrationResult	<b>InstanceSpecification</b>	
ArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
ArbitrationSpecificationBinding	<b>InstanceSpecification</b>	
ArbitrationTarget		
AtomicProceduralElement		<a href="#">atomic procedural element</a>
AtomicProceduralElementArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
AtomicProceduralElementLogEntry	<b>InstanceSpecification</b>	
BoundaryValueAnalysis	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
CauseEffectAnalysis	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
ChecklistBasedTesting	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
CheckPropertyAction	<b>Constraint, ObjectFlow</b>	<a href="#">check property action</a>
CheckPropertyArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
CheckPropertyLogEntry	<b>InstanceSpecification</b>	
ClassificationTreeMethod	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
CombinatorialTesting	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
Complements	<b>Dependency</b>	<a href="#">complement</a>
CompoundProceduralElement	<b>CombinedFragment, StructuredActivityNode</b>	<a href="#">compound procedural element</a>
CompoundProceduralElementArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
CreateLogEntryAction	<b>InvocationAction</b>	<a href="#">create log entry action</a>
CreateLogEntryArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
CreateLogEntryLogEntry	<b>InstanceSpecification</b>	
CreateStimulusAction	<b>InvocationAction, Message</b>	<a href="#">create stimulus action</a>
CreateStimulusArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
CreateStimulusLogEntry	<b>InstanceSpecification</b>	
DataPartition	<b>Classifier</b>	<a href="#">data pool</a>
DataPool	<b>Classifier</b>	<a href="#">data pool</a>
DataProvider	<b>Classifier, Property</b>	<a href="#">data provider</a>
DataSpecification	<b>Constraint</b>	<a href="#">data specification</a>
DecisionTableTesting	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
EquivalenceClassPartitioning	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
ErrorGuessing	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
ExpectResponseAction	<b>Message, Trigger</b>	<a href="#">expect response action</a>
ExpectResponseArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
ExperienceBasedTechnique	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
ExploratoryTesting	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
Extends	<b>Dependency</b>	<a href="#">extension</a>



FormalParameterReference	<b>Property</b>	<a href="#">formal parameter</a>
GenericTestDesignDirective	<b>InstanceSpecification</b>	<a href="#">test design directive</a>
GenericTestDesignTechnique	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
InvocationLogEntry	<b>InstanceSpecification</b>	
InvocationLogEntryStructure	<b>Classifier</b>	
Loop	<b>CombinedFragment, StructuredActivityNode</b>	<a href="#">loop</a>
LoopArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
MessageEventLogEntry	<b>InstanceSpecification</b>	
MessageEventLogEntryStructure	<b>Classifier</b>	
Morphing	<b>Dependency</b>	<a href="#">morphism</a>
Negative	<b>CombinedFragment, StructuredActivityNode</b>	<a href="#">negative</a>
NegativeArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
NSwitchCoverage	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
OpaqueProceduralElement	<b>NamedElement</b>	<a href="#">procedural element</a>
OpaqueProceduralElementLogEntry	<b>InstanceSpecification</b>	
overrides	<b>Dependency</b>	<a href="#">morphism</a>
PairwiseTesting	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
Parallel	<b>CombinedFragment, StructuredActivityNode</b>	<a href="#">parallel</a>
ParallelArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
ProceduralElement		<a href="#">procedural element</a>
ProceduralElementArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
ProcedureInvocation	<b>CallBehaviorAction, InteractionUse</b>	<a href="#">procedure invocation</a>
ProcedureInvocationArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
ProcedureInvocationLogEntry	<b>InstanceSpecification</b>	
ProcedureInvocationLogEntryStructure	<b>Classifier</b>	
Refines	<b>Dependency</b>	<a href="#">refinement</a>
RegularExpression	<b>Expression</b>	<a href="#">data specification</a>
RoleConfiguration	<b>Constraint</b>	<a href="#">test configuration</a>
Sequence	<b>CombinedFragment, StructuredActivityNode</b>	<a href="#">sequence</a>
SequenceArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
StateCoverage	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
StateTransitionTechnique	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
SuggestVerdictAction	<b>InvocationAction</b>	<a href="#">suggest verdict action</a>
SuggestVerdictArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
SuggestVerdictLogEntry	<b>InstanceSpecification</b>	
TestCase	<b>Behavior, BehavioredClassifier</b>	<ul style="list-style-type: none"> <li>• <a href="#">test case</a></li> <li>• <a href="#">abstract test case</a></li> <li>• <a href="#">concrete test case</a></li> </ul>
TestCaseArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
TestCaseLog	<b>InstanceSpecification</b>	<a href="#">test case log</a>
TestComponent	<b>Classifier, Property</b>	<a href="#">test component</a>
TestComponentConfiguration	<b>Constraint</b>	<a href="#">test component configuration</a>
TestConfiguration	<b>StructuredClassifier</b>	<a href="#">test configuration</a>
TestConfigurationRole	<b>Classifier, Property</b>	<a href="#">test configuration</a>
TestContext	<b>Package</b>	<a href="#">test context</a>

TestDesignDirective	<b>InstanceSpecification</b>	<a href="#">test design directive</a>
TestDesignDirectiveStructure	<b>Classifier</b>	<a href="#">test design directive</a>
TestDesignInput	<b>NamedElement</b>	<a href="#">test design input</a>
TestDesignTechnique	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
TestDesignTechniqueStructure	<b>Classifier</b>	<a href="#">test design technique</a>
TestDirective	<b>InstanceSpecification</b>	
TestDirectiveStructure	<b>Classifier</b>	
TestExecutionSchedule	<b>Behavior</b>	<a href="#">test execution schedule</a>
TestItem	<b>Classifier, Property</b>	<a href="#">test item</a>
TestItemConfiguration	<b>Constraint</b>	<a href="#">test item configuration</a>
TestLog	<b>InstanceSpecification</b>	<a href="#">test log</a>
TestLogElement	<b>InstanceSpecification</b>	
TestLogEntry	<b>InstanceSpecification</b>	
TestLogStructure	<b>Classifier</b>	<a href="#">test log structure</a>
TestLogStructureBinding	<b>Dependency</b>	<a href="#">test log structure</a>
TestObjective	<b>Class</b>	<a href="#">test objective</a>
TestObjectiveTrace	<b>Dependency</b>	
TestProcedure	<b>Behavior</b>	<a href="#">test procedure</a>
TestRequirement	<b>Class</b>	<a href="#">test requirement</a>
TestSet	<b>Package</b>	<a href="#">test set</a>
TestSetArbitrationSpecification	<b>BehavioredClassifier</b>	<a href="#">arbitration specification</a>
TestSetLog	<b>InstanceSpecification</b>	<a href="#">test set log</a>
TestTechnique	<b>InstanceSpecification</b>	
TestTechniqueStructure	<b>Classifier</b>	
TransitionCoverage	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
TransitionPairCoverage	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
UseCaseTesting	<b>InstanceSpecification</b>	<a href="#">test design technique</a>
verifies	<b>Dependency</b>	

## 8.3 Test Planning

Test analysis and test design deals with determining the identifying test basis for specific testing activities, determination of [test objectives](#), and eventually the selection and application of appropriate the [test design techniques](#) to achieve those [test objectives](#). UTP organizes concepts provided for carrying out test analysis and design activities into two parts: concepts for describing [test contexts](#), [test objectives](#), [test requirements](#), and concepts to specify test design activities.

### 8.3.1 Test Analysis

The test analysis concepts are means to argue and justify why certain testing activities have to be carried out as well as how these testing activities with all required or helpful [artifacts](#) are organized.

In order to group [artifacts](#) and information that are deemed necessary for certain testing activities, the [test context](#) concept (represented by the stereotype «[TestContext](#)») is introduced. It offers the capability to bundle [artifacts](#) (e.g., any PackageableElement) in a shared scope (e.g., the Namespace), to hide information from other scopes and to import elements from other scopes. This enables a high degree of organizational reusability of information.

In dynamic testing, [test cases](#) are eventually produced by the test design activities in order to execute them. For certain reasons, [test cases](#) are often assembled and executed together in a [test set](#) (or test suite, which is a synonym of a [test set](#)). In UTP, a [test set](#) is represented by the stereotype «[TestSet](#)» which has the ability to assemble, import and reuse [test cases](#).

The definition of certain coverage criteria and/or objectives that the testing activities have to meet is essential for

test planning. In UTP, the planning activities are supported by means of the concepts [test objective](#) (implemented by the stereotype «[TestObjective](#)»), [test requirement](#) (implemented by the stereotype «[TestRequirement](#)»), a verification dependency among development [artifacts](#) and [test objectives](#) or [test requirements](#) (represented by the stereotype «[verifies](#)»). In order to stay as close as possible to the SysML definition of requirements [\[SysML\]](#), both [test objective](#) and [test requirements](#) are designed as [extensions](#) to the UML metaclass **Class**. Such a stereotyped **Class** is capable of defining new properties solely, whereas most of the capabilities of the metaclass **Class** are forbidden by [constraint](#), such as owning Ports, **Operations**, **Behavior**s etc.. The stereotype «[verifies](#)» extends the UML metaclass Dependency in order to be technically compatible with SysML [\[SysML\]](#), too.

These concepts enable testers to adhere to well-known and established industrial testing standards such as ISTQB [\[ISTQB\]](#) or ISO 29119 [\[ISO29119\]](#) when creating model-based test specifications. Whereas [test objectives](#) are intended to describe higher level goals the testing activities have to achieve in a certain context (e.g., coverage of all high priority requirements at system level testing), [test requirements](#) are intended to pinpoint a single and testable aspect of the [test item](#). As such, [test objectives](#) describe often the test ending criteria for the testing activities in a certain context (e.g., system level testing), and [test requirements](#) leverage the development of [test design input](#) definitions or [test cases](#). Eventually, [test requirements](#) are realized by [test cases](#), which is similar to the coverage of [test requirements](#). Test requirements contribute to the fulfilment of [test objectives](#).

Both [test objectives](#) and [test requirements](#) can be used independently of each other or in joint manner or not at all. This is contextually up to the respective testing methodology. UTP does not prescribe the use of these concepts.

### 8.3.1.1 Test Context Overview

The stereotypes «[TestContext](#)» and «[TestSet](#)» are defined in UTP. Both represent a container for dedicated elements, thus, they are [extensions](#) of the UML Package. As such they inherit the concept of nested Packages, Package templates, owned and imported members as well as visibility. However, it is not prescribed that the visibility concepts have to be respected by any conforming UTP tooling. The decision whether or not to utilize the visibility and import mechanism of UML is up to the tool implementation. However, the derived associations of «[TestContext](#)» and «[TestSet](#)», however, are based on UML visibility and import.

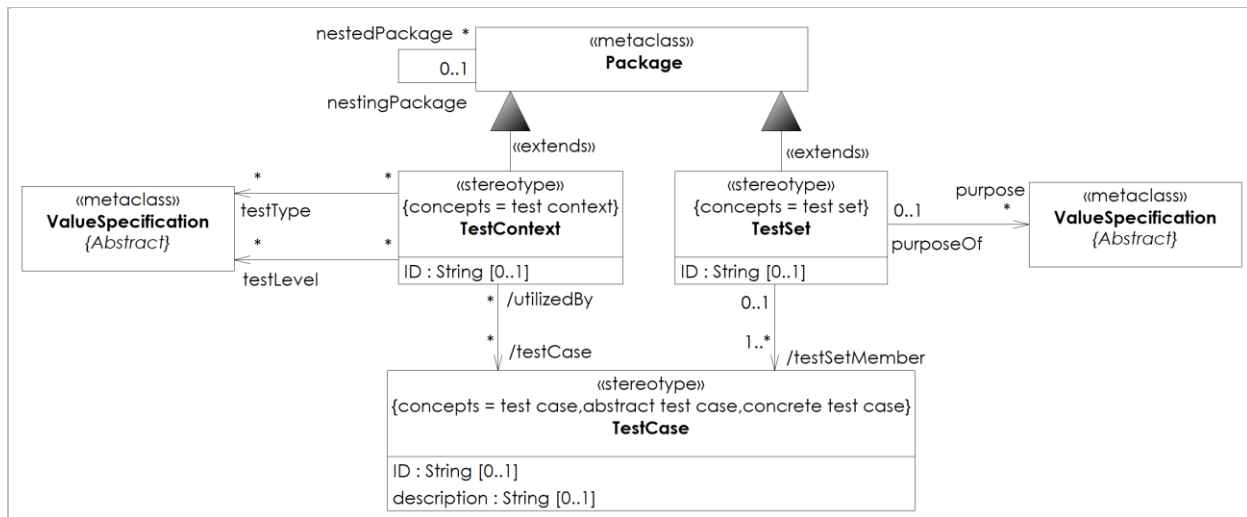
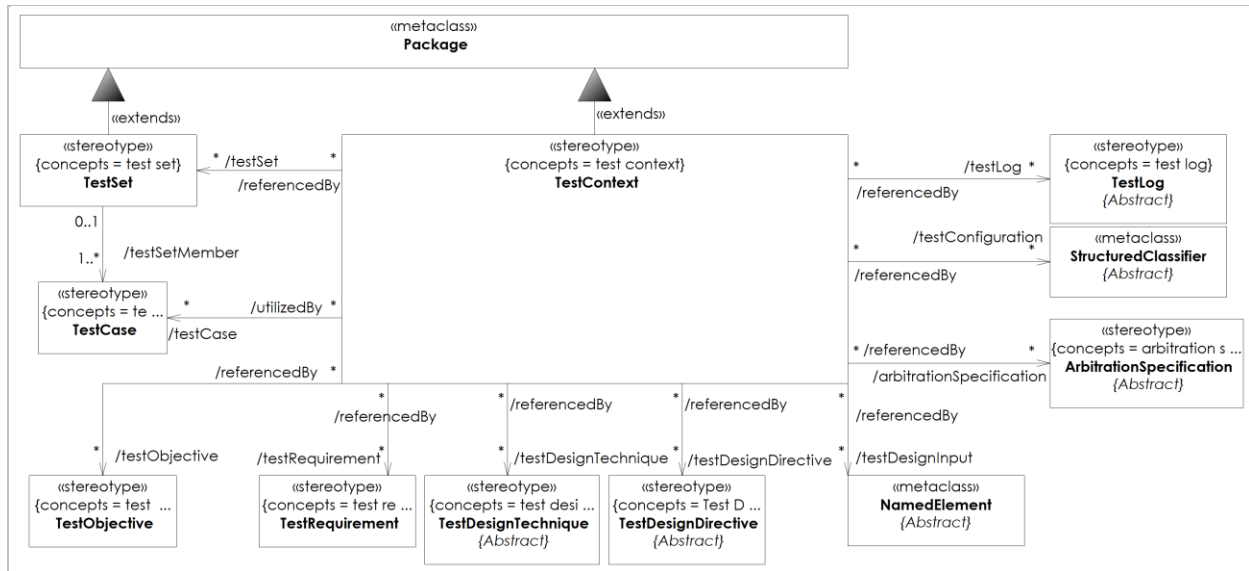


Figure 8.2 - Test Context Overview

### 8.3.1.2 Test-specific Contents of Test Context

The UML profile specification for the [test context](#) concepts is shown in the following diagram. Most of the relationships among the concepts of the Conceptual Model are already covered by the underlying UML metamodel. In order to allow users of the UTP an easy access to related elements, a set of derived associations is defined that retrieves the desired element for a currently processed stereotype. As an example for the design decision, please see the derived associations between «[TestContext](#)» and «[TestCase](#)». In the Conceptual Model it is stated that a [test](#)



### Figure 8.3 - Test-specific Contents of Test Context

### 8.3.1.3 Test Objective Overview

The following diagram shows the abstract syntax for the [test objectives](#) concepts.

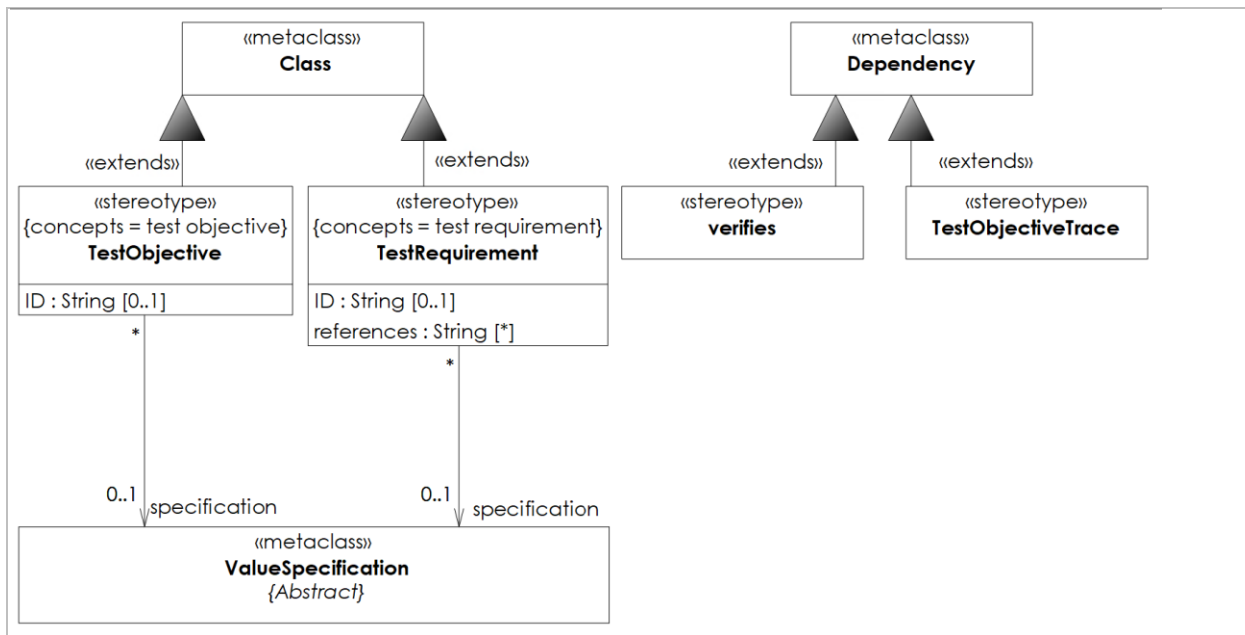


Figure 8.4 - Test Objective Overview

### 8.3.1.4 Stereotype Specifications

#### 8.3.1.4.1 TestContext

Description	<p><b>TestContext:</b> A set of information that is prescriptive for testing activities which can be organized and managed together for deriving or selecting <a href="#">test objectives</a>, <a href="#">test design techniques</a>, <a href="#">test design inputs</a> and eventually <a href="#">test cases</a>.</p> <p>A <a href="#">test context</a> may import the packaged elements of another <a href="#">test context</a> in order to access and reuse visible elements of the imported <a href="#">test context</a>. This is inherently given by the native UML concepts PackageImport or ElementImport. Whether or not the visibility of elements contained in a <a href="#">test context</a> is respected is up to the tool implementation.</p> <p>Since a «<a href="#">TestContext</a>» is an extended Package, it is possible to decompose <a href="#">test contexts</a> into more fine-grained <a href="#">test contexts</a>. For example, a <a href="#">test context</a> defined for the <a href="#">test level</a> 'System testing' might be decomposed in accordance to the <a href="#">test types</a> that are addressed at that <a href="#">test level</a> (e.g., functional system testing, security system testing etc.).</p>
Extension	<b>Package</b>
Attributes	<p>ID : String [0..1]</p> <p>An optional identifier to unambiguously distinguish between any two <a href="#">test contexts</a>. If it is set, it has to be unique for all the test contexts in the scope of the model.</p>
Associations	<p>/testCase : TestCase [*]</p> <p>The test cases that are accessible by the given «<a href="#">TestContext</a>». This feature is derived by the set of directly owned or via ElementImport or PackageImport for imported test cases.</p> <p>testLevel : ValueSpecification [*]</p> <p>The test levels that the testing activities within the given «<a href="#">TestContext</a>» have to cope with.</p>

	<p><code>testType : ValueSpecification [*]</code></p> <p>The test types that the testing activities within the given «<a href="#">TestContext</a>» have to cope with.</p> <p><code>/testSet : TestSet [*]</code></p> <p>Refers to the test sets that are known by this test context. It is derived from both contained and imported Packages with «TestSet» applied.</p> <p><code>/testObjective : TestObjective [*]</code></p> <p>Refers to the test objectives that are known by this test context. It is derived from both contained and imported Classes with «TestObjective» applied.</p> <p><code>/testRequirement : TestRequirement [*]</code></p> <p>Refers to the test requirements that are known by this test context. It is derived from both contained and imported Classes with «TestRequirement» applied.</p> <p><code>/testConfiguration : StructuredClassifier [*]</code></p> <p>Refers to the test configurations that are known by this test context. It is derived from both contained and imported StructuredClassifier with «TestConfiguration» applied.</p> <p><code>/testDesignInput : NamedElement [*]</code></p> <p>Refers to the test design inputs that are known by this test context. It is derived from both contained and imported NamedElements with «TestDesignInput» applied and the NamedElements that are referenced by all known «TestDesignDirective» as their test design input (i.e., referenced by the tag definition <i>testDesignInput</i>). The latter part of the derivation algorithm is necessary, because the use of the «TestDesignInput» stereotype is not mandatory, and sometimes even not possible.</p> <p><code>/testDesignDirective : TestDesignDirective [*]</code></p> <p>Refers to the test design directives that are known by this test context. It is derived from both contained and imported InstanceSpecifications with a concrete subclass of «TestDesignDirective» applied.</p> <p><code>/testDesignTechnique : TestDesignTechnique [*]</code></p> <p>Refers to the test design techniques that are known by this test context. It is derived from both contained and imported InstanceSpecifications with a concrete subclass of «TestDesignTechnique» applied.</p> <p><code>/arbitrationSpecification : ArbitrationSpecification [*]</code></p> <p>Refers to the arbitration specifications that are known by this test context. It is derived from both contained and imported BehavioredClassifiers with «TestDesignTechnique» applied.</p> <p><code>/testLog : TestLog [*]</code></p> <p>Refers to the test logs that are known by this test context. It is derived from both contained and imported InstanceSpecification with a concrete subclass of «TestLog» applied.</p>
Constraints	<p>Restriction of extendable metaclasses</p> <p>«<a href="#">TestContext</a>» shall not be applied to instances of the metaclass Profile.</p>
Change from UTP 1.2	<p>Changed from UTP 1.2. In UTP 1.2 «TestContext» extended StructuredClassifier and BehavioredClassifier as well as incorporated the concepts TestSet, TestExecutionSchedule and TestConfiguration into a single concept.</p>

#### 8.3.1.4.2 TestObjective

Description	<p><b>TestObjective:</b> A desired effect that a <a href="#">test case</a> or <a href="#">test set</a> intends to achieve.</p> <p>The stereotype «<a href="#">TestObjective</a>» extends <b>Class</b>. <a href="#">test objective</a> enables tester to define the test ending criteria for the testing activities in a certain <a href="#">test context</a>. A <a href="#">test objective</a> can be expressed with detail or very abstractly, depending on the underlying methodology.</p> <p>As pure test analysis concept, it is very likely that <a href="#">test objectives</a> have to be traceable to and from test environment tools, which first and foremost would be test management tools. Therefore, <a href="#">test objectives</a> have the ability to specify a unique identifier represented by the tag definition ID. However, the use of the explicit identifier is optional and simply enables the most primitive kind of traceability within a test environment.</p> <p>The specification of a <a href="#">test objective</a>, i.e., the reason why <a href="#">test cases</a> are created and eventually executed, is expressed by means of the tag definition specification.</p> <p>If a BMM profile (see <a href="#">BMM</a>) is also loaded into a model containing the UTP 2.0 profile, this stereotype may be considered as a BMM objective (i.e., merged with a BMM objective).</p>
Extension	<b>Class</b>
Attributes	<p>ID : String [0..1]</p> <p>A unique identifier that unambiguously identifies the <a href="#">test objective</a>.</p> <p>specification : ValueSpecification [0..1]</p> <p>The specification of the test objective. It might be represented in both unstructured and structured text or any other concrete sub-class of ValueSpecification.</p>
Constraints	<p>Restriction of extendable metaclasses</p> <p>«TestObjective» shall only be applied to instances of the metaclass Class.</p>
Change from UTP 1.2	Changed from UTP 1.2. In UTP 1.2, «TestObjective» was called «TestObjectiveSpecification».

#### 8.3.1.4.3 TestObjectiveTrace

Description	<p>A test objective trace enables tester to express relationships between <a href="#">test objectives</a> and any element that is considered to contribute to the fulfillment of those <a href="#">test objectives</a>.</p> <p>A test objective trace is a means to establish traceability among the fulfilling elements and the test objectives they are bound to.</p> <p>A test objective trace must only allow client elements that are stereotyped with «TestObjective». The types of supplier elements are not restricted, meaning, any suitable element that may contribute to the fulfillment of the «TestObjective» is allowed as supplier element.</p>
Extension	<b>Dependency</b>
Constraints	<p>Only «TestObjective» as client elements allowed</p> <p>Client elements of the underlying extended metaclass Dependency have to be stereotyped with «TestObjective».</p>
Change from UTP 1.2	«TestObjectiveTrace» was newly introduced by UTP 2.3.

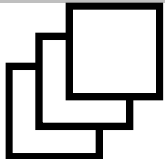


#### 8.3.1.4.4 TestRequirement

Description	<p><b>TestRequirement</b>: A desired property on a <a href="#">test case</a> or <a href="#">test set</a>, referring to some aspect of the <a href="#">test item</a> to be tested.</p> <p>The stereotype «<b>TestRequirement</b>» extends <b>Class</b> (for integration with the SysML stereotype «requirement»). A <a href="#">test requirement</a> enables testers to decompose single and distinct testable aspects of the <a href="#">test item</a> prior to test design. As such, it is part of the test analysis facility of UTP. <a href="#">test requirements</a> are deemed helpful for both the derivation of <a href="#">test cases</a>, <a href="#">test procedures</a> and in particular <a href="#">test design input</a> definitions. <a href="#">test requirements</a> are said to be realized by <a href="#">test design input</a> definitions, <a href="#">test case</a> or <a href="#">test procedures</a>. The default UML metaclass Realize is intended to be utilized to express this relationship.</p> <p>As a pure test analysis concept, it is very likely that <a href="#">test requirements</a> have to be traceable to and from test environment tools, first and foremost test management tools. Therefore, <a href="#">test requirements</a> have the ability to specify a unique identifier represented by the tag definition ID. However, the use of the explicit identifier is optional and simply enables the most primitive kind of traceability within a test environment.</p> <p>The specification of a <a href="#">test requirement</a> (i.e., the textual description of a single testable aspect of a <a href="#">test requirement</a>) is expressed by means of the tag definition specification. Although it is typed by the PrimitiveType String, the <a href="#">test requirement</a> might be specified by means of a more formal or structured language (e.g., using the Test Purpose Language (TPlan) standardized by ETSI).</p> <p>Additional references to external resources (e.g., relevant standards, guidelines, documents, websites etc.) can be added via the tag definition references.</p> <p>If SysML <a href="#">[SysML]</a> is also loaded into a model containing the UTP 2.0 profile, this stereotype may be considered as (i.e., merged with) the SysML stereotype «requirement».</p>
Extension	<b>Class</b>
Attributes	<p>ID : String [0..1]</p> <p>A unique identifier that unambiguously identifies the <a href="#">test requirement</a>.</p> <p>references : String [*]</p> <p>Includes any additional references that are deemed relevant for the definition of the test requirement (such as relevant standards, papers, or any other meaningful artifact)</p>
Associations	<p>/realizedBy : TestCase [*]</p> <p>References the <a href="#">test cases</a> that realize the given <a href="#">test requirement</a>. They are derived from the set of UML Realization dependencies that point to the base Class of this stereotype and stem from a BehaviorClassifier or Behavior stereotyped with «<b>TestCase</b>».</p> <p>specification : ValueSpecification [0..1]</p> <p>The specification of the test requirement. It might be represented in both unstructured and structured text or any other concrete sub-class of ValueSpecification.</p>
Constraints	<p>Restriction of extendable metaclasses</p> <p>«TestRequirement» shall only be applied to instances of the metaclass Class.</p>
Change from UTP 1.2	« <b>TestRequirement</b> » has been newly introduced into UTP 2.



### 8.3.1.4.5 TestSet

Description	<p><b>TestSet</b>: A set of <a href="#">test cases</a> that share some common purpose.</p> <p>A <a href="#">test set</a> assembles <a href="#">test cases</a> either via ownership or import. These <a href="#">test cases</a> are called the members of the <a href="#">test set</a>. Ownership assembly is based on the ability of UML <b>Packages</b> to nest any PackageableElement. Import assembly is based on the ability of UML <b>Packages</b> to import PackageableElements either directly or indirectly by importing the <b>Package</b> that contains the PackageableElement to be imported. A <a href="#">test case</a> is transitively an <a href="#">extension</a> of PackageableElement, thus, the import mechanisms given by UML can be reused to group <a href="#">test cases</a> in <a href="#">test sets</a> by either assembly kind.</p> <p>Visibility of <a href="#">test cases</a> within a <a href="#">test set</a> is defined in accordance with the visibility of NamedElement in Namespaces as defined by UML. Since the use of visibility is not mandatory by UML, it is also not mandatory to utilize visibility in UTP. However, if visibility is desired, it must comply with the UML semantics.</p> <p>A <a href="#">test set</a> can have an arbitrary number of <a href="#">test execution schedules</a> (extends <b>Behavior</b>) either by ownership or import, similar to <a href="#">test case</a> assembly. A <a href="#">test execution schedule</a> must only schedule the execution of <a href="#">test cases</a> that are members of the respective <a href="#">test sets</a>. If a <a href="#">test set</a> does not contain an explicit <a href="#">test execution schedule</a>, it is semantically equivalent to an implicitly owned <a href="#">test execution schedule</a> that schedules the execution of all <a href="#">test cases</a> assembled by the current <a href="#">test set</a> in an arbitrary order. If a <a href="#">test set</a> is supposed to be executed, the decision which <a href="#">test execution schedule</a> will be taken into account for scheduling is not defined UTP, since a <a href="#">test set</a> may have more than just one <a href="#">test execution schedule</a> defined. A viable method is to use the UML deployment specification to implement the desired <a href="#">test execution schedule</a> for eventual execution by an <a href="#">executing entity</a>.</p> <p>If a <a href="#">test set</a> assembles another <a href="#">test set</a>, the assembling <a href="#">test set</a> has access to all visible <a href="#">test cases</a> assembled by the assembled <a href="#">test set</a>. In addition, the assembling <a href="#">test set</a> has access to all visible <a href="#">test execution schedules</a> of the assembled <a href="#">test set</a>. This enables the composition and decomposition of <a href="#">test sets</a> and their respective <a href="#">test execution schedules</a>.</p> <p>The purpose of a <a href="#">test set</a> is set of a ValueSpecifications that can be shared with other <a href="#">test sets</a>. If a <a href="#">test set</a> has more than one purpose, the purposes are logically combined by AND (i.e., if a <a href="#">test set</a> has the two purposes 'Manual Testing' and 'Regression Testing' it should be read as follows 'The <a href="#">test set</a>'s purpose is 'manual regression testing').</p>
Graphical syntax	
Extension	<b>Package</b>
Super Class	<a href="#">ArbitrationTarget</a>
Attributes	<p>ID : String [0..1]</p> <p>An optional identifier to unambiguously distinguish between any two <a href="#">test sets</a>. If it is set, it has to be unique for all the <a href="#">test sets</a> in the scope of the model.</p>
Associations	<p>purpose : ValueSpecification [*]</p> <p>Denotes the purposes why the test set has been assembled.</p> <p>/testSetMember : TestCase [1..*]</p> <p>Refers to the TestCases that are assembled, either via ownership or import, by the</p>

	given TestSet, and thus, are members of that TestSet. A TestCase can be a member of more than one TestSet.
	testSetAS : TestSetArbitrationSpecification [0..1]
Constraints	Restriction of extendable metaclass «TestSet» shall not only be applied to instances of the metaclass Profile.
Change from UTP 1.2	«TestSet» has been newly introduced by UTP 2. It was part of the TestContext in UTP 1.2.

#### 8.3.1.4.6 verifies

Description	<p>The stereotype «<a href="#">verifies</a>» extends Dependency and is intended to express relationships among elements that are supposed to be verified (e.g., a requirement, an interface operation, a use case, a user story, a single transition or state, and so forth) and elements that support the verification thereof (e.g., a <a href="#">test objective</a>, a <a href="#">test requirement</a>, a <a href="#">test case</a>, a <a href="#">test set</a>).</p> <p>A «<a href="#">verifies</a>» Dependency as a means to establish traceability within UML-based model elements. It weakens the <a href="#">constraints</a> applied on SysML «Verify» in a sense that UTP «<a href="#">verifies</a>» allows targeting elements different than SysML «requirement». This limitation is too restrictive for UTP, in particular in setups where, for example, use cases are the elements to be verified.</p> <p>Since the semantics of Dependencies with respect to n:m-ary in contrast to binary, 1:m-ary, or n:1-ary Dependencies are not precisely defined, UTP considers by default no difference among all the different ways on how «<a href="#">verifies</a>» Dependencies can be expressed between more than two elements.</p> <p>If a SysML profile (see <a href="#">[SysML]</a>) is also loaded into a model containing the UTP 2.0 profile, this stereotype may be considered as the SysML «Verify» stereotype (i.e. merged with the SysML «Verify» stereotype).</p>
Extension	<b><a href="#">Dependency</a></b>
Change from UTP 1.2	«verifies» has been newly introduced into UTP 2. In UTP 1.2 the «verify» stereotype from SysML was recommended.

### 8.3.2 Test Design

The UTP 2 test design facility describes a language framework for the specification of [test design techniques](#) and their application to a [test design input](#) element. This includes behavioral descriptions (e.g., UML state machines), or structural information (e.g., interface definitions). [test design techniques](#) are usually assembled by so called [test design directive](#) which is responsible for establishing the associations between a set of [test design techniques](#) and the [test design input](#) element those [test design techniques](#) must operate on. A [test design directive](#) may also link the test design outputs elements that have been generated or derived by the set of applied [test design techniques](#). This allows for a more comprehensible test design phase and is the key to comprehensive traceability among [test objectives](#)/[test requirements](#), [test design techniques](#), [test design input](#) and eventually test design output elements.

The UTP 2 test design facility only represents the very core of the language framework. Since the stereotypes of the core framework are based on abstract stereotypes and mostly derived (and read-only unions) associations, it is possible to concretize and extend the test design facility as required by using stereotype specialization and [property](#) subsetting. A built-in concretization of the core framework was done by means of the generic test design capabilities and the predefined [test design techniques](#). It enables test engineers to immediately utilize the test design facility or develop proprietary [test design directives](#) and [test design techniques](#). Tailoring of the UTP test design facility can be done at metalevel M1 (model level) and metalevel M2 (metamodel level).

The different mechanisms for tailoring are:

- Tailoring through structural features: Both «[TestDesignTechnique](#)» and «[TestDesignDirective](#)» extend the UML metaclass InstanceSpecification with implicit attributes predefined by the respective stereotypes. In addition to these predefined attributes, user may add additional attributes to these two elements by using the genuine InstanceSpecification-Classifier association. Since both stereotypes extend InstanceSpecification, it is possible to classify these InstanceSpecifications with multiple Classifiers. For this purpose, UTP provides the stereotypes «[TestDesignDirectiveStructure](#)» and «[TestDesignTechniqueStructure](#)». As a result, the user may add as many additional attributes as desired or required to a «[TestDesignDirective](#)» and «[TestDesignTechnique](#)».
- Tailoring through use of «[GenericTestDesignDirective](#)» and «[GenericTestDesignTechnique](#)»: By means of the predefined stereotypes «[GenericTestDesignTechnique](#)» and «[GenericTestDesignDirective](#)», users can build on proprietary [test design directives](#) and [test design techniques](#) by simply providing dedicated names to the underlying InstanceSpecification (i.e., the InstanceSpecification with «[GenericTestDesignDirective](#)» or «[GenericTestDesignTechnique](#)» applied. In combination with the [extension](#) through structural features as just described above, the use of «[GenericTestDesignTechnique](#)» and «[GenericTestDesignDirective](#)» provides a flexible and powerful mechanism to tailor the UTP test design facility for user-specific purposes. For example, an InstanceSpecification with «[TestDesignTechnique](#)» applied and name set to 'PathCoverage' is one way to provide the test engineer with a new [test design techniques](#) that represents path coverage.
- Profile [extension](#): The third and most powerful tailoring to user-specific needs comes along with profile [extension](#). Similar to the provision of specialized stereotypes of the abstract stereotypes «[TestDesignTechnique](#)» and «[TestDesignDirective](#)» as predefined concepts of the language itself, users or vendors may introduce proprietary stereotypes that specialize the abstract stereotypes provided by the test design facility of UTP.

### 8.3.2.1 Test Design Facility

The following picture shows the abstract syntax of the very core of the UTP test design facility.

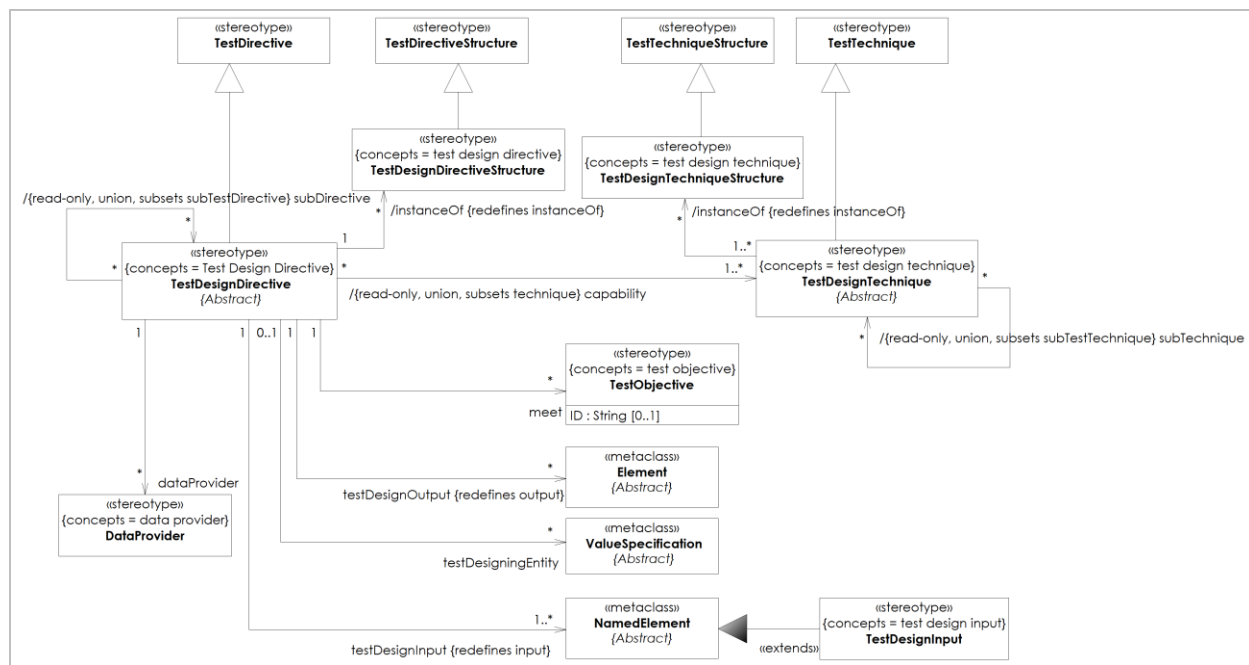


Figure 8.5 - Test Design Facility

### 8.3.2.2 Generic Test Design Capabilities

The generic test design capabilities of UTP 2 enable tester to immediately start off with specifying [test design directives](#) and defining proprietary, user-defined or project-specific [test design techniques](#), if the predefined [test design techniques](#) does not suffice.

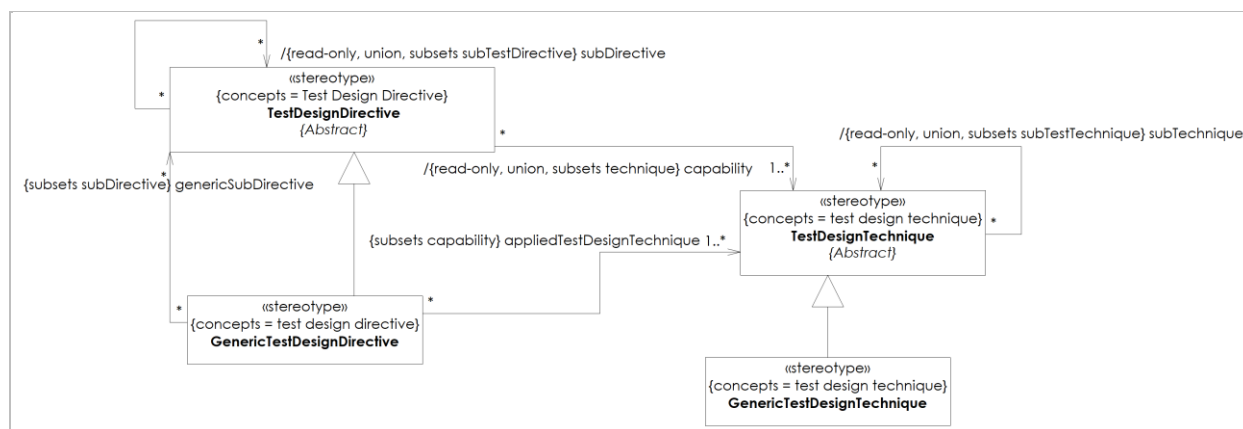


Figure 8.6 - Generic Test Design Capabilities

### 8.3.2.3 Predefined high-level Test Design Techniques

The following diagram shows the predefined high-level [test design techniques](#). They belong to the so called specification-based [test design techniques](#) as categorized by [\[ISO29119\]-4](#).

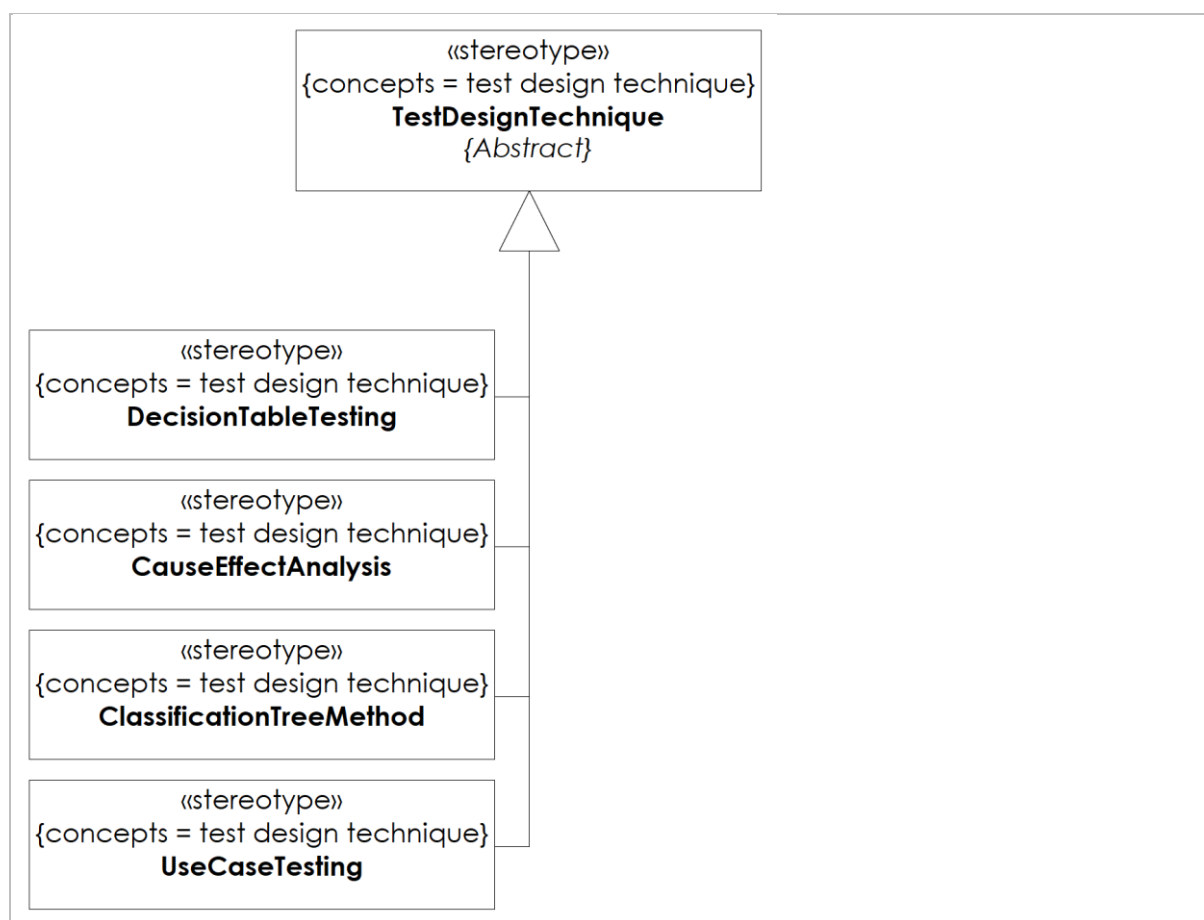


Figure 8.7 - Predefined high-level Test Design Techniques

### 8.3.2.4 Predefined data-related Test Design Techniques

The following diagram shows the predefined [data-related test design techniques](#). They belong to the so called specification-based [test design techniques](#) as categorized by [\[ISO29119\]-4](#).

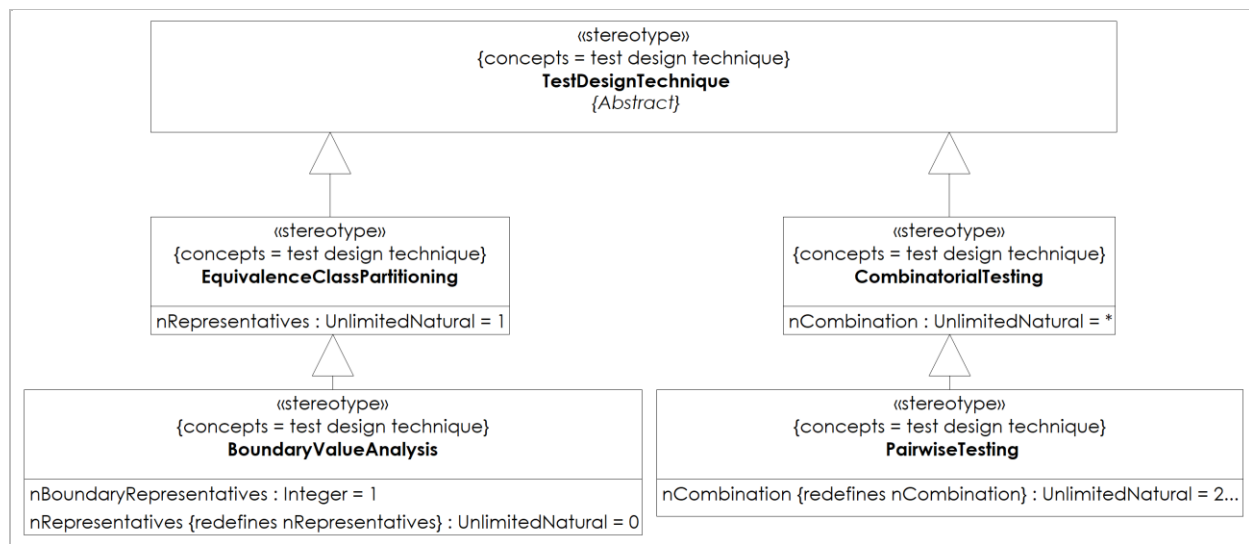


Figure 8.8 - Predefined data-related Test Design Techniques

### 8.3.2.5 Predefined state-transition-based Test Design Techniques

The following diagram shows the predefined state-transition based [test design techniques](#). They belong to the so called specification-based [test design techniques](#) as categorized by [ISO29119](#)-4.

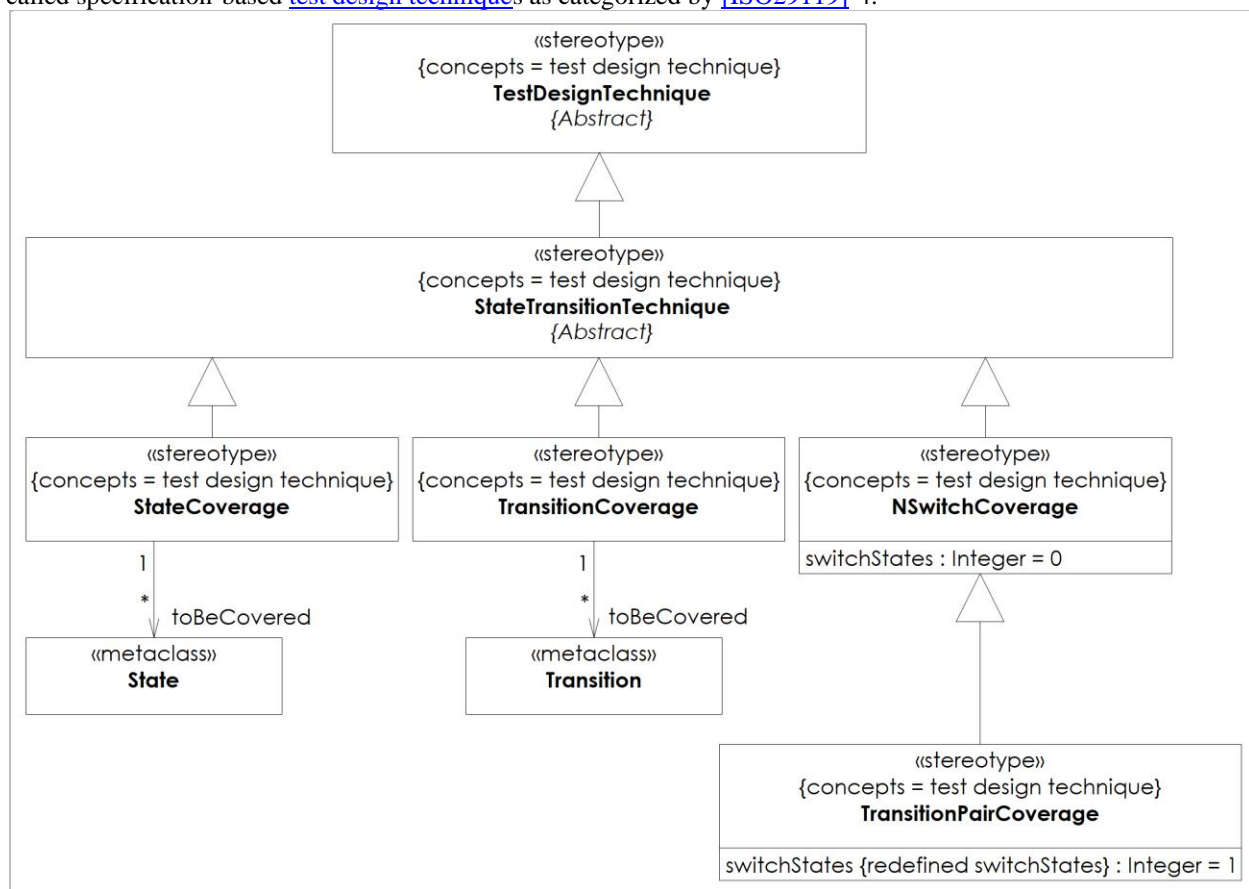


Figure 8.9 - Predefined state-transition-based Test Design Techniques

### 8.3.2.6 Predefined experience-based Test Design Techniques

The following diagram shows the predefined experienced-based [test design techniques](#) as categorized by [ISO29119]-4.

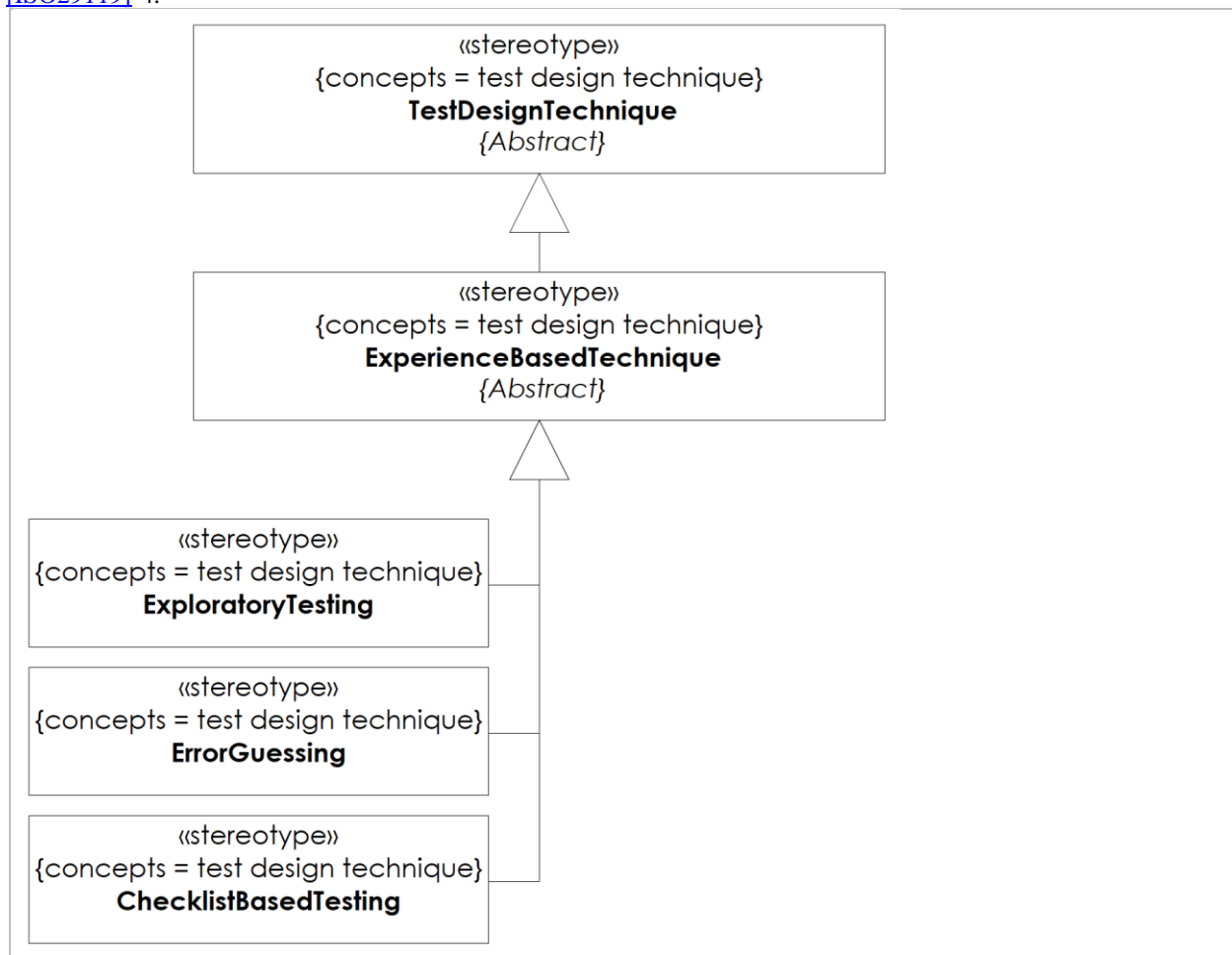


Figure 8.10 - Predefined experience-based Test Design Techniques

## 8.3.2.7 Stereotype Specifications

### 8.3.2.7.1 BoundaryValueAnalysis

Description	<p>According to <a href="#">[ISTQB]</a>: Black box testing is a <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed based on boundary values.</p> <p>«<a href="#">BoundaryValueAnalysis</a>» is an <a href="#">extension</a> of «<a href="#">EquivalenceClassPartitioning</a>» that takes also values at the boundaries (left and right or upper and lower boundary) into account. A boundary value is defined by ISTQB as "an input value or output value which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum and maximum value of a range."</p> <p>Since the boundary values already define representatives of an equivalence class, the ordinary (i.e. non-boundary) representatives are usually of less interest. Therefore, the inherited <a href="#">property</a> <code>nRepresentatives</code> is redefined to obtain the default value 0. This ensures that no additional ordinary representatives of the equivalence class are selected. However, it is still possible to specify that in addition to the boundary values, ordinary representatives of the corresponding equivalence class will be selected by setting the value of <code>nRepresentatives</code> to a value greater than 0.</p> <p>See <a href="#">[ISO29119]</a>-4 clause 5.2.3 <a href="#">BoundaryValueAnalysis</a> for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">EquivalenceClassPartitioning</a>
Attributes	<p><code>nBoundaryRepresentatives</code> : Integer [1] = 1</p> <p>Specifies the number of boundary representatives that have to be covered by the resulting test cases. Default is 1.</p> <p><code>nRepresentatives</code> {redefines <code>nRepresentatives</code>} : UnlimitedNatural [1] = 0</p> <p>Redefines the number of representatives to 0, in addition to the boundary values, meaning that by default only the boundary values will be selected.</p>
Change from UTP 1.2	« <a href="#">BoundaryValueAnalysis</a> » has been newly introduced by UTP 2.

### 8.3.2.7.2 CauseEffectAnalysis

Description	<p>According to <a href="#">[ISTQB]</a>: A black box <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed from cause-effect graphs.</p> <p>See also <a href="#">[ISO29119]</a>-4, clause 5.2.7 Cause-Effect Graphing for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Change from UTP 1.2	« <a href="#">CauseEffectAnalysis</a> » has been newly introduced by UTP 2.

### 8.3.2.7.3 ChecklistBasedTesting

Description	<p>According to <a href="#">[ISTQB]</a>: An experience-based <a href="#">test design technique</a> whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">ExperienceBasedTechnique</a>
Change from UTP 1.2	« <a href="#">ChecklistBasedTesting</a> » has been newly introduced by UTP 2.

#### 8.3.2.7.4 ClassificationTreeMethod

Description	<p>According to <a href="#">[ISTQB]</a>: A black box <a href="#">test design technique</a> in which <a href="#">test cases</a>, described by means of a classification tree, are designed to execute combinations of representatives of input and/or output domains. A classification tree is a tree showing equivalence partitions hierarchically ordered, which are used to design <a href="#">test cases</a> in the classification tree method.</p> <p>See also <a href="#">[ISO29119]</a>-4, clause 5.2.2 Classification Tree Method for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Change from UTP 1.2	«ClassificationTreeMethod» has been newly introduced by UTP 2.

#### 8.3.2.7.5 CombinatorialTesting

Description	<p>According to <a href="#">[ISTQB]</a>: A means to identify a suitable subset of test combinations to achieve a predetermined level of coverage when testing an object with multiple input parameters and where those parameters themselves each have several values.</p> <p>The Property nCombinations specifies the number of how many parameters must be combined with each other. The higher the number of combinations, the higher the number of derived <a href="#">test cases</a>. By default, all combinations of input parameters will be covered, which is indicated by the asterisk (*). However, the value of the Property nCombination has to be less than the number of the input parameters.</p> <p>See <a href="#">[ISO29119]</a>-4 clause 5.2.5 Combinatorial Test Design Techniques for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Sub Class	<a href="#">PairwiseTesting</a>
Attributes	<p>nCombination : UnlimitedNatural [1] = *</p> <p>The number of combinations of input parameters</p>
Change from UTP 1.2	«CombinatorialTesting» has been newly introduced by UTP 2.

#### 8.3.2.7.6 DecisionTableTesting

Description	<p>According to <a href="#">[ISTQB]</a>: A black box <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed to execute combinations of inputs and/or stimuli (causes) shown in a decision table. A decision table is a table showing combinations of inputs and/or stimuli (causes) with their associated outputs and/or actions (effects), which can be used to design <a href="#">test cases</a>.</p> <p>See also <a href="#">[ISO29119]</a>-4, clause 5.2.6 Decision Table Testing for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Change from UTP 1.2	«DecisionTableTesting» has been newly introduced by UTP 2.



#### 8.3.2.7.7 EquivalenceClassPartitioning

Description	<p>According to <a href="#">[ISTQB]</a>: A black box <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed to execute representatives from equivalence partitions. In principle <a href="#">test cases</a> are designed to cover each partition at least once.</p> <p>Usually, the number of the representatives of each equivalence class that will be used to derive the <a href="#">test cases</a> is set to 1 in order to keep the number of <a href="#">test cases</a> as low as possible. In certain situations, it might be, for whatever reason, desired to select more than just one representative per equivalence class. The <a href="#">property nRepresentatives</a> enables the tester to set any number desired number of representatives per equivalence class. By default, the value is set to 1 (reflecting the usual application of that <a href="#">test design technique</a>). If the value is set to unlimited (i.e., the asterisk (*)), all possible representatives of an equivalence class have to be selected.</p> <p>See <a href="#">[ISO29119]</a>-4 clause 5.2.1 Equivalence Partitioning for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Sub Class	<a href="#">BoundaryValueAnalysis</a>
Attributes	<p><a href="#">nRepresentatives</a> : UnlimitedNatural [1] = 1</p> <p>Indicates the desired number of minimal representatives that should be derived for a given equivalence class.</p>
Change from UTP 1.2	«EquivalenceClassPartitioning» has been newly introduced by UTP 2.

#### 8.3.2.7.8 ErrorGuessing

Description	<p>According to <a href="#">[ISTQB]</a>: A <a href="#">test design technique</a> where the experience of the tester is used to anticipate what defects might be present in the component or <a href="#">test item</a> as a result of <a href="#">Errors</a> made and to design tests specifically to expose them.</p> <p>See <a href="#">[ISO29119]</a>-4 clause 5.4 <a href="#">Error</a> Guessing for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">ExperienceBasedTechnique</a>
Change from UTP 1.2	«ErrorGuessing» has been newly introduced by UTP 2.

#### 8.3.2.7.9 ExperienceBasedTechnique

Description	<p>According to <a href="#">[ISTQB]</a>: A <a href="#">procedure</a> to derive and/or select <a href="#">test cases</a> based the tester's experience, knowledge and intuition.</p> <p>Experienced-based <a href="#">test design techniques</a> are usually informal techniques potentially supported by checklists or <a href="#">Error</a> taxonomies.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Sub Class	<a href="#">ChecklistBasedTesting</a> , <a href="#">ErrorGuessing</a> , <a href="#">ExploratoryTesting</a>
Change from UTP 1.2	«ExperienceBasedTechnique» has been newly introduced by UTP 2.

#### 8.3.2.7.10 ExploratoryTesting

Description	<p>According to <a href="#">[ISTQB]</a>: An informal <a href="#">test design technique</a> where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">ExperienceBasedTechnique</a>
Change from UTP 1.2	«ExploratoryTesting» has been newly introduced by UTP 2.

#### 8.3.2.7.11 GenericTestDesignDirective

Description	<p>A predefined <a href="#">test design directive</a> that is able to assemble any <a href="#">test design technique</a> available or known in a certain context, including any user-defined «<a href="#">GenericTestDesignTechnique</a>». As such, the generic <a href="#">test design directive</a> makes no assumptions about the capabilities of a test designing entity a priori.</p> <p>Additional required information can be introduced by utilizing the <a href="#">test design directive</a> structure concept.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignDirective</a>
Associations	<p>{subsets capability} appliedTestDesignTechnique : TestDesignTechnique [1..*]</p> <p>Enables a generic test design directive to apply any known test design technique for the test design activity.</p> <p>{subsets subDirective} genericSubDirective : TestDesignDirective [*]</p> <p>Enables a generic test design directive to be potentially refined by any other known test design directive.</p>
Change from UTP 1.2	«GenericTestDesignDirective» has been newly introduced by UTP 2.

#### 8.3.2.7.12 GenericTestDesignTechnique

Description	<p>The predefined generic <a href="#">test design technique</a> is a semantic-free test design technique that is intended to be used to specify proprietary test design techniques that are not part of the predefined UTP 2 test design facility. The name of the underlying InstanceSpecification determines the name of the test design technique, potentially extended by structural information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Change from UTP 1.2	«GenericTestDesignTechnique» has been newly introduced by UTP 2.

#### 8.3.2.7.13 NSwitchCoverage

Description	<p>According to <a href="#">[ISTQB]</a>: A form of state transition testing in which <a href="#">test cases</a> are designed to execute all valid <a href="#">sequences</a> of N+1 transitions.</p> <p>N-Switch coverage was initially developed by [Chow], where n defines the number of switch states among a <a href="#">sequence</a> of consecutive transitions. The default is 0, meaning that a <a href="#">test case</a> may only consist of a single transition. However, the entirety of all transitions will be captured by the resulting <a href="#">test cases</a>.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">StateTransitionTechnique</a>
Sub Class	<a href="#">TransitionPairCoverage</a>
Attributes	<p>switchStates : Integer [1] = 0</p> <p>Specifies the number of switch states, and thus, implicitly the sequence of transitions that will at least be covered by the resulting test cases.</p>
Change from UTP 1.2	«NSwitchCoverage» has been newly introduced by UTP 2.

#### 8.3.2.7.14 PairwiseTesting

Description	<p>According to <a href="#">[ISTQB]</a>: A black box <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed to execute all possible discrete combinations of each pair of input parameters.</p> <p>«<a href="#">PairwiseTesting</a>» is a specialized «<a href="#">CombinatorialTesting</a>» test design technique whose <a href="#">property</a> nCombination is refined and set to the read-only value 2, meaning, that at least each pair of input parameters will be covered in the resulting <a href="#">test cases</a>.</p> <p>See <a href="#">[ISO29119]</a>-4 clause 5.2.5.4 Pair-wise Testing for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">CombinatorialTesting</a>
Attributes	<p>nCombination {redefines nCombination} : UnlimitedNatural [1] = 2</p> <p>The number of combinations for each input parameter is set to exactly 2 (i.e., each combination of every pair of input parameters must at least be covered).</p>
Change from UTP 1.2	«PairwiseTesting» has been newly introduced by UTP 2.

#### 8.3.2.7.15 StateCoverage

Description	<p>A coverage criterion for state-based testing that requires that test cases are designed that execute a certain set of states.</p> <p>If no State is referenced by the <a href="#">property</a> toBeCovered, all States in the related state machine will be covered.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">StateTransitionTechnique</a>
Associations	<p>toBeCovered : State [*]</p> <p>Refers to a set of States that will at least be covered by the test designer.</p>
Change from UTP 1.2	«StateCoverage» has been newly introduced by UTP 2.

#### 8.3.2.7.16 StateTransitionTechnique

Description	<p>According to <a href="#">[ISTQB]</a>: A black box <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed to execute valid and invalid state transitions.</p> <p>Test design directives that assemble a concrete state-transition technique must refer to at least one state machine as its <a href="#">test design input</a>. If more than one state machine is referenced as <a href="#">test design input</a>, the concrete state-transition techniques are applied to all state machines.</p> <p>See also <a href="#">[ISO29119]</a>-4, clause 5.2.8 State-Transition Testing for further information.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Sub Class	<a href="#">NSwitchCoverage</a> , <a href="#">StateCoverage</a> , <a href="#">TransitionCoverage</a>
Change from UTP 1.2	«StateTransitionTechnique» has been newly introduced by UTP 2.

### 8.3.2.7.17 TestDesignDirective

Description	<p><b>TestDesignDirective:</b> A <a href="#">test design directive</a> is an instruction for a test designing entity to derive test <a href="#">artifacts</a> such as <a href="#">test sets</a>, <a href="#">test cases</a>, <a href="#">test configurations</a>, <a href="#">data</a> or <a href="#">test execution schedules</a> by applying <a href="#">test design techniques</a> on a <a href="#">test design input</a>. The set of assembled <a href="#">test design techniques</a> are referred to as the capabilities a test designing entity must possess in order to carry out the <a href="#">test design directive</a>, regardless whether it is carried out by a human tester or a test generator. A <a href="#">test design directive</a> is a means to support the achievement of a <a href="#">test objective</a>.</p> <p>The abstract stereotype «<a href="#">TestDesignDirective</a>» extends InstanceSpecification and brings all relevant information together that is required for automatically or manually derive test <a href="#">artifacts</a> from a <a href="#">test design input</a>. The derivation process is steered by the set of <a href="#">test design techniques</a>, which the current <a href="#">test design directives</a> refers to.</p> <p>Each <a href="#">test design directive</a> has a basic set of structural elements, given by the tag definitions of the «<a href="#">TestDesignDirective</a>» stereotype. The fundamental and implicit structure can be extended by means of UML. Since «<a href="#">TestDesignDirective</a>» extends InstanceSpecification, it is possible to add Classifiers to the underlying InstanceSpecification which then define additional structural information deemed necessary in a specific context. This is the easiest and UML native mechanism to tailor <a href="#">test design directive</a> to specific needs.</p> <p>The <a href="#">test design techniques</a> that will be applied on the <a href="#">test design input</a> are captured in the association end capabilities. This is a derived union, since it cannot be foreseen which <a href="#">test design techniques</a> are required. Concrete subtypes have to subset the derived union capabilities (see for example «<a href="#">GenericTestDesignDirective</a>») in order to enable certain <a href="#">test design techniques</a> for a <a href="#">test design directive</a>. Those <a href="#">test design techniques</a> can be combined with each other by a <a href="#">test design directive</a>.</p> <p>A <a href="#">test design directive</a> refers to a set of NamedElements as the input for the eventual test design activities performed by a test designing entity. This input yields the association end <a href="#">TestDesignInput</a>. It is not required that a referenced NamedElement has the stereotype «<a href="#">TestDesignInput</a>» applied. The assembled <a href="#">test design techniques</a> by the given <a href="#">test design directive</a> are then applied on the <a href="#">test design input</a> in order to produce the test design output <a href="#">artifacts</a>.</p> <p>A <a href="#">test design directive</a> may provide sub-directives by means of the association end subDirective. Providing a sub <a href="#">test design directive</a> enables testers to refine the test design activities for certain elements contained in the <a href="#">test design input</a>. As an example, this specification assumes a parent <a href="#">test design directive</a> refers to a StateMachine as its <a href="#">test design input</a>. The <a href="#">test design directive</a> also assembles a set of state-transition and <a href="#">data</a>-related <a href="#">test design techniques</a> that will be applied to the StateMachine by a test designing entity. This specification further assume that the StateMachine contains a submachine State (i.e., a reference of another StateMachine that is considered to be copied to the location of the submachine State) which is referred to as <a href="#">test design input</a> by a sub <a href="#">test design directive</a>. This enables the composition of different kinds of <a href="#">test design directives</a> in order to meet different <a href="#">test objectives</a>.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDirective</a>
Sub Class	<a href="#">GenericTestDesignDirective</a>
Associations	<p>meet : TestObjective [*]</p> <p>The test objectives that have to be fulfilled by putting the given test design directive into effect.</p>

	<pre>/{read-only, union, subsets technique} capability : TestDesignTechnique [1..*]</pre> <p>Refers to the set test design techniques that are assembled by the given test design directive. The set is referred to as the capabilities a test designing entity (e.g., a generator in automated test design or human tester in manual test design) has to offer in order to be able to perform the test design activities imposed by the test design directive.</p>
	<pre>/{read-only, union, subsets subTestDirective} subDirective : TestDesignDirective [*]</pre> <p>Refers to one or more test design directives that further refine the instructions given by the parent test design directive.</p>
	<pre>testDesignOutput {redefines output} : Element [*]</pre> <p>The outcome of the test design activities produced by the given test design directives.</p>
	<pre>testDesigningEntity : ValueSpecification [*]</pre> <p>Identifies the test designing entity (e.g. a generator in automated test design or a human tester in manual test design) that has produced (parts of) the test design output.</p>
	<pre>/instanceOf {redefines instanceOf} : TestDesignDirectiveStructure [*]</pre> <p>Refers to the test design directive structure of which the given test design directive is an instance of. The test design directive structure is derived from all Classifiers with «TestDesignDirectiveStructure» applied that are referred as classifiers by the underlying InstanceSpecification.</p>
	<pre>testDesignInput {redefines input} : NamedElement [1..*]</pre> <p>Refers to the model elements that have to be incorporated by the test designer (e.g. a generator in automated test design or a human tester in manual test design) as input to the derivation process.</p>
	<pre>dataProvider : DataProvider [*]</pre> <p>References the data providers that are supposed to deliver or produce the required test data.</p>
Change from UTP 1.2	«TestDesignDirective» has been newly introduced by UTP 2.

#### 8.3.2.7.18 TestDesignDirectiveStructure

Description	A <a href="#">TestDesignDirectiveStructure</a> describes user-defined or context-specific additional information that may augment any given <a href="#">TestDesignDirective</a> . A Classifier with « <a href="#">TestDesignDirectiveStructure</a> » applied might be of arbitrary complexity. It enables the provision of information that are deemed relevant in a certain context but not required in a different context.
Extension	<a href="#">Classifier</a>
Super Class	<a href="#">TestDirectiveStructure</a>
Change from UTP 1.2	«TestDesignDirectiveStructure» has been newly introduced by UTP 2.

### 8.3.2.7.19 TestDesignInput

Description	<p><b>TestDesignInput:</b> Any piece of information that must or has been used to derive testing <a href="#">artifacts</a> such as <a href="#">test cases</a>, <a href="#">test configuration</a>, and <a href="#">data</a>.</p> <p>The stereotype «<a href="#">TestDesignInput</a>» is an explicit, yet optional means to indicate that the purpose of a given model element is to use it for test design activities (i.e., usually the derivation of <a href="#">test cases</a>, test <a href="#">data</a>, <a href="#">test configurations</a> etc.). The application of this stereotype is declared as optional, because in general any kind of model element might be used as input for the test design activities.</p>
Extension	<b>NamedElement</b>
Change from UTP 1.2	«TestDesignInput» has been newly introduced by UTP 2.

### 8.3.2.7.20 TestDesignTechnique

Description	<p><b>TestDesignTechnique:</b> A specification of a method used to derive or select <a href="#">test configurations</a>, <a href="#">test cases</a> and <a href="#">data</a>. <a href="#">test design techniques</a> are governed by a <a href="#">test design directive</a> and applied to a <a href="#">test design input</a>. Such <a href="#">test design techniques</a> can be monolithically applied or in combination with other <a href="#">test design techniques</a>. Each <a href="#">test design technique</a> has clear semantics with respect to the <a href="#">test design input</a> and the <a href="#">artifacts</a> it derives from the <a href="#">test design input</a>.</p> <p>The abstract stereotype «<a href="#">TestDesignTechnique</a>» extends InstanceSpecification and integrates <a href="#">test design techniques</a> with <a href="#">test design directives</a>. A <a href="#">test design technique</a> is a concrete action, technique or <a href="#">procedure</a> to derive test design output from a <a href="#">test design input</a>. A <a href="#">test design technique</a> is basically independent of a dedicated <a href="#">test design input</a> element, but can be reused across multiple <a href="#">test design input</a> elements. Some <a href="#">test design techniques</a> only make sense if a certain <a href="#">test design input</a> element was selected (e.g., state-transition <a href="#">test design techniques</a> make only sense if the <a href="#">test design input</a> element is a StateMachine).</p> <p>Each <a href="#">test design technique</a> has a basic set of structural elements given by the tag definitions of the «<a href="#">TestDesignTechnique</a>» stereotype. The fundamental (and implicit) structure can be extended by means of UML. Since «<a href="#">TestDesignTechnique</a>» extends InstanceSpecification, it is possible to add Classifiers to the underlying InstanceSpecification which then define additional structural information deemed necessary in a specific context. This is the easiest and UML native mechanism to tailor <a href="#">test design techniques</a> to specific needs.</p> <p>A <a href="#">test design technique</a> may provide sub-techniques by means of the association end subTechnique. Providing a sub <a href="#">test design technique</a> enables testers to refine the <a href="#">test design techniques</a> for certain elements contained in the <a href="#">test design input</a> and also to enrich existing (potentially pre-defined) <a href="#">test design techniques</a> in a certain context.</p>
Extension	<b>InstanceSpecification</b>
Super Class	<a href="#">TestTechnique</a>
Sub Class	<a href="#">CauseEffectAnalysis</a> , <a href="#">ClassificationTreeMethod</a> , <a href="#">CombinatorialTesting</a> , <a href="#">DecisionTableTesting</a> , <a href="#">EquivalenceClassPartitioning</a> , <a href="#">ExperienceBasedTechnique</a> , <a href="#">GenericTestDesignTechnique</a> , <a href="#">StateTransitionTechnique</a> , <a href="#">UseCaseTesting</a>
	<pre>/{read-only, union, subsets subTestTechnique} subTechnique : TestDesignTechnique [*]</pre> <p>Refers to one or more test design techniques that may further refine the parent test design technique.</p>

	<pre>/instanceOf {redefines instanceOf} : TestDesignTechniqueStructure [*]</pre> <p>Refers to additional structural information of the given test design technique. The test design technique structures are derived from all Classifiers with «TestDesignTechniqueStructure» applied that are referred to as classifiers by the underlying InstanceSpecification.</p>
Change from UTP 1.2	«TestDesignTechnique» has been newly introduced by UTP 2.

#### 8.3.2.7.21 TestDesignTechniqueStructure

Description	A <a href="#">test design technique</a> structure describes user-defined or context-specific additional information that may augment any given <a href="#">test design technique</a> . A Classifier with « <a href="#">TestDesignTechniqueStructure</a> » applied might be of arbitrary complexity. It enables the provision of information that is deemed relevant in a certain context but not required in a different context.
Extension	<a href="#">Classifier</a>
Super Class	<a href="#">TestTechniqueStructure</a>
Change from UTP 1.2	«TestDesignTechniqueStructure» has been newly introduced by UTP 2.

#### 8.3.2.7.22 TransitionCoverage

Description	<p>A coverage criterion for state-based testing that requires that test cases are designed that execute a certain set of transitions.</p> <p>If no Transition is referenced by the <a href="#">property</a> toBeCovered, all States in the related state machine will be covered.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">StateTransitionTechnique</a>
Associations	<pre>toBeCovered : Transition [*]</pre> <p>Refers to a set of Transitions that will at least be covered by the test designer.</p>
Change from UTP 1.2	«TransitionCoverage» has been newly introduced by UTP 2.

#### 8.3.2.7.23 TransitionPairCoverage

Description	<p>The «<a href="#">TransitionPairCoverage</a>» <a href="#">test design technique</a> is a specific (and often used) «<a href="#">NSwitchCoverage</a>» <a href="#">test design technique</a> that redefines the Property switchStates to the read-only value 1. That means that the resulting <a href="#">test cases</a> should at least cover all <a href="#">sequences</a> of any two consecutive Transitions.</p> <p>The semantics of transition pair coverage and N-Switch coverage with nSwitches set to 1 is semantically equivalent.</p>
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">NSwitchCoverage</a>
Attributes	<pre>switchStates {redefined switchStates} : Integer [1] = 1</pre> <p>Restricts the number of switch states to exactly one, meaning that every pair of subsequent Transitions will at least be covered.</p>
Change from UTP 1.2	«TransitionPairCoverage» has been newly introduced by UTP 2.

#### 8.3.2.7.24 UseCaseTesting

Description	According to <a href="#">[ISTQB]</a> : A black box <a href="#">test design technique</a> in which <a href="#">test cases</a> are designed to execute scenarios of use cases.  See also <a href="#">[ISO29119]</a> -4, clause 5.2.9 Scenario Testing for further information.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestDesignTechnique</a>
Change from UTP 1.2	«UseCaseTesting» has been newly introduced by UTP 2.

## 8.4 Test Architecture

Test architecture concepts specify structural aspects of a test environment, including a [test configuration](#), necessary to eventually execute [test cases](#) against the [test item](#)(s). The test environment comprises everything that is necessary to execute [test cases](#) (e.g., [test components](#), hardware, simulators, test execution tools etc.). The [test configuration](#) describes how those parts of the test environment and represented [test components](#), are connected with the [test item](#).

Building a reliable [test configuration](#) is required for any [test case](#), because it determines the [test item](#)(s) and how the test environment (in UTP represented by [test components](#)) interfaces to the [test item](#)(s).

Test architectures are mainly expressed by means of UML class and composite structure diagrams. In contrast to UTP 1.2, both [test components](#) and [test items](#) can be represented either as a standalone type or as a role that a certain type may assume in a specific [test configuration](#). However, UTP does not prescribe which option to use for describing test architecture and both have advantages and disadvantages.

The test architecture concepts consist of:

- [test configuration](#), implemented by the stereotype «[TestConfiguration](#)»
- [test configuration](#) role, implemented by the abstract stereotype «[TestConfigurationRole](#)» as a superclass for any known (even future) role a [test configuration](#) may assume
- role configuration, implemented by the abstract stereotype «[RoleConfiguration](#)» as superclass for configurations of concrete roles
- [test component](#), implemented by the stereotype «[TestComponent](#)» that specializes «[TestConfigurationRole](#)»
- [test component configuration](#), implemented by the stereotype «[TestComponentConfiguration](#)» that specializes «[RoleConfiguration](#)»
- [test item](#), implemented by the stereotype «[TestItem](#)» that specializes «[TestConfigurationRole](#)»
- [test item configuration](#), implemented by the stereotype «[TestItemConfiguration](#)» that specializes «[RoleConfiguration](#)»

### 8.4.1 Test Architecture Overview

The diagram below shows the abstract syntax of the test architecture concepts.



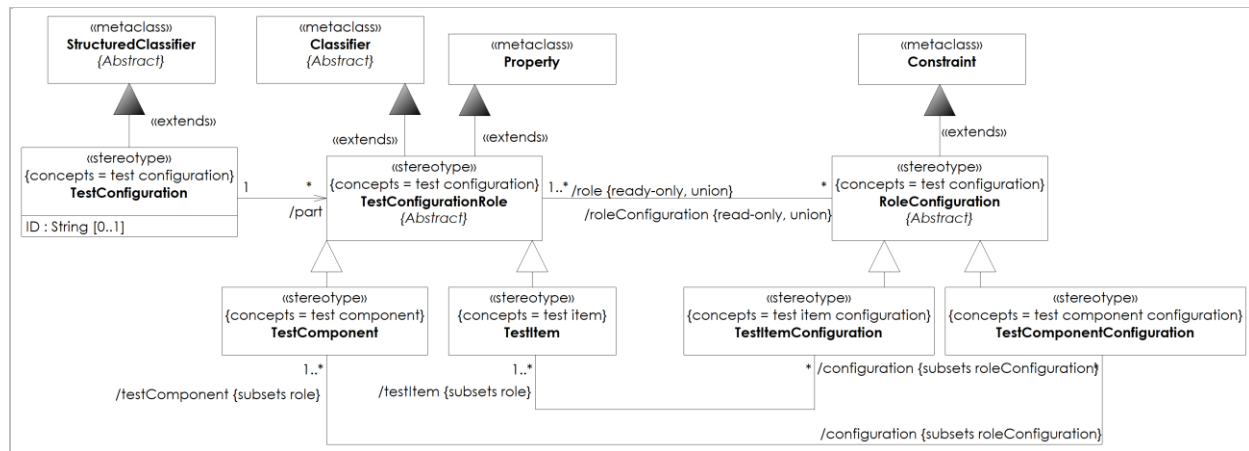


Figure 8.11 - Test Architecture Overview

## 8.4.2 Stereotype Specifications

### 8.4.2.1 RoleConfiguration

Description	<p>The abstract stereotype «<a href="#">RoleConfiguration</a>» extends the metaclass <a href="#">Constraint</a> and is used to specify the configuration of <a href="#">test configuration</a> role within a certain <a href="#">test configuration</a>.</p> <p>There are at least two ways a role configuration can be associated with a <a href="#">test configuration</a> role, both stemming from the underlying UML Constraints metamodel:</p> <ul style="list-style-type: none"> <li>• Classifier-oriented: A <a href="#">Constraint</a> with a concrete subs stereotype of «<a href="#">RoleConfiguration</a>» applied is contained by a <a href="#">Classifier</a> as its context with a concrete subs stereotype of «<a href="#">TestConfigurationRole</a>» applied, or it refers to a set of such <a href="#">Classifiers</a> by means of the meta-association <a href="#">constrainedElement</a>.</li> <li>• Property-oriented: A <a href="#">Constraint</a> with a concrete subs stereotype of «<a href="#">RoleConfiguration</a>» applied refers to one or more <a href="#">Properties</a> with «<a href="#">TestConfigurationRole</a>» applied by means of the meta-association <a href="#">constrainedElement</a>.</li> </ul> <p>The Classifier-oriented way has the advantage that all parts of <a href="#">test configurations</a> which are typed by a <a href="#">Classifier</a> with a concrete subs stereotype of «<a href="#">TestConfigurationRole</a>» applied, must abide by the configurations defined for that <a href="#">Classifier</a>. On the downside, this might prevent reuse, because it is not possible to get rid of configurations (similar to the handling of <a href="#">Constraints</a> in UML) expressed on <a href="#">Classifier</a> level.</p> <p>The Property-oriented way has the advantage that it enables the dedicated configuration of single <a href="#">test component</a> roles within a <a href="#">test configuration</a>.</p>
Extension	<a href="#">Constraint</a>
Sub Class	<a href="#">TestComponentConfiguration</a> , <a href="#">TestItemConfiguration</a>
Associations	<p>/role {ready-only, union} : <a href="#">TestConfigurationRole</a> [1..*]</p> <p>Refers to the set of at least one test configuration roles.</p>
Change from UTP 1.2	« <a href="#">RoleConfiguration</a> » is newly introduced in UTP 2.

### 8.4.2.2 TestComponent

Description	<p><b>TestComponent</b>: A role of an <a href="#">artifact</a> within a <a href="#">test configuration</a> that is required to perform a <a href="#">test case</a>.</p> <p>The stereotype «<a href="#">TestComponent</a>» specializes «<a href="#">TestConfigurationRole</a>» and declares that a certain element (i.e., either a Classifier or Property) is responsible for driving the execution of a <a href="#">test case</a>. The use of the stereotype «<a href="#">TestComponent</a>» on Classifier is optional but, if it is used, all Properties of that type must also have «<a href="#">TestComponent</a>» applied, if they are used in a <a href="#">test configuration</a>.</p>
Extension	<b>Classifier, Property</b>
Super Class	<a href="#">TestConfigurationRole</a>
Sub Class	<a href="#">DataProvider</a>
Associations	<pre>/configuration {subsets roleConfiguration} : TestComponentConfiguration [*]</pre> <p>Refers to the configurations that are defined for this «TestComponent». This set of configurations is derived from all Constraints with «TestComponentConfiguration» applied that are either owned rules (in case of «TestComponent» is applied on a Classifier) of the «TestComponent» or inversely referring to the «TestComponent» (in case of «TestComponentConfiguration» is applied on Constraint without having a context but using <i>Constraint.constrainedElement</i> to refer to the «TestComponent»).</p>
Change from UTP 1.2	Changed from UTP 1.2. In UTP 1.2., « <a href="#">TestComponent</a> » only extended Class.

### 8.4.2.3 TestComponentConfiguration

Description	<p><b>TestComponentConfiguration</b>: A set of configuration options offered by an <a href="#">artifact</a> in the role of a <a href="#">test component</a> chosen to meet the requirements of a particular <a href="#">test configuration</a>.</p> <p>The stereotype «<a href="#">TestComponentConfiguration</a>» specializes the abstract stereotype «<a href="#">RoleConfiguration</a>». The eventual set of configurations for a NamedElement with «<a href="#">TestComponent</a>» applied is derived from the union of all <a href="#">test component configuration</a>s declared for that NamedElement (i.e., either on <b>Classifier</b> or <b>Property</b> level).</p>
Extension	<b>Constraint</b>
Super Class	<a href="#">RoleConfiguration</a>
Associations	<pre>/testComponent {subsets role} : TestComponent [1..*]</pre> <p>Refers to the set of at least one test components that are configured by the given test component configuration. The resulting set is derived from both the Classifier stereotyped with «TestComponent» that is the context of the underlying Constraint and all test components regardless of whether Classifier or Property that are referenced by the underlying <i>Constraint.constrainedElement</i>.</p>
Constraints	<p>Ownership of «TestComponentConfiguration»</p> <p>Each «TestComponentConfiguration» shall refer to at least one «TestComponent», i.e., there is no «TestComponentConfiguration» that exists without referring to a «TestComponent».</p>
Change from UTP 1.2	« <a href="#">TestComponentConfiguration</a> » has been newly introduced into UTP 2.

#### 8.4.2.4 TestConfiguration

Description	<p><b>TestConfiguration:</b> A specification of the test item and test components as well as their interconnection and configuration data.</p> <p>The stereotype «<a href="#">TestConfiguration</a>» extends <b>StructuredClassifier</b> which effectively extends a variety of UML metaclasses such as Class, Collaboration, and Component, etc. The <a href="#">test configuration</a> then refers to the composite structure of the underlying <b>StructuredClassifier</b>. Every <a href="#">test configuration</a> must have at least one member stereotyped «<a href="#">TestItem</a>» which is connected to at least one member stereotyped with «<a href="#">TestComponent</a>».</p> <p>The <a href="#">test configurations</a> of any two distinct <a href="#">test procedures</a> that are intended to be executed together, as part of a potentially third <a href="#">test procedure</a>, and must have a compatible <a href="#">test configuration</a>. Compatibility of <a href="#">test configurations</a> is partially defined by UML and the substitution principle of Liskov, but also by means of the idea of EncapsulatedClassifiers. The attempt to invoke <a href="#">test procedures</a> together will most likely fail due to technical incompatibility.</p> <p>Test cases or <a href="#">test procedures</a> may come along with their own <a href="#">test configurations</a> expressed by means of their respective composite structures. In that case, the application of the «<a href="#">TestConfiguration</a>» stereotype will be done in addition to «<a href="#">TestCase</a>» or «<a href="#">TestProcedure</a>». In case of shared <a href="#">test configurations</a> it is recommended, though not required, to facilitate the UML concept of a «<a href="#">TestConfiguration</a>» stereotyped Collaboration. Collaborations are meant to be reused by other <b>StructuredClassifiers</b>, including Behaviors, by means of CollaborationUse and role bindings. Inheritance and redefinition, as defined by UML, are additional means to express shared and reusable <a href="#">test configurations</a>, as well.</p>
Extension	<b>StructuredClassifier</b>
Attributes	<p>ID : String [0..1]</p> <p>A unique identifier that unambiguously identifies the given <a href="#">test configuration</a>.</p>
Associations	<p>/part : TestConfigurationRole [*]</p> <p>Refers to the test configuration parts that are involved in this test configuration. They are derived from all members of the underlying StructuredClassifier that has a subclass of the abstract stereotype «TestConfigurationRole» applied.</p>
Constraints	<p>Minimal test configuration</p> <p>A StructuredClassifier with «TestConfiguration» applied must at least specify one part having «TestItem» applied.</p>
Change from UTP 1.2	« <a href="#">TestConfiguration</a> » has been newly introduced into UTP 2. It was conceptually represented by the composite structure of a «TestContext» in UTP 1.2.

#### 8.4.2.5 TestConfigurationRole

Description	<p>The abstract stereotype «<a href="#">TestConfigurationRole</a>» extends both Classifier and Property.</p> <p>The advantage of assigning the role to a certain part assumes in a <a href="#">test configuration</a> that the very same Type of this part (i.e., Class or Component) can be reused in different <a href="#">test configuration</a> with different roles. This entails that the application of a concrete subclass of «<a href="#">TestConfigurationRole</a>» on a Classifier is not required at all and limits reusability of this Classifier. If a concrete substereotype of «<a href="#">TestConfigurationRole</a>» is applied on a Classifier, any part of a <a href="#">test configuration</a> must have the very same concrete substereotype applied.</p>
Extension	<b>Classifier, Property</b>

Sub Class	<a href="#">TestComponent</a> , <a href="#">TestItem</a>
Associations	<pre>/roleConfiguration {read-only, union} : RoleConfiguration [*]</pre> <p>Refers to the role configuration that is defined for this test configuration role.</p>
Change from UTP 1.2	« <a href="#">TestConfigurationRole</a> » is newly introduced in UTP 2.

#### 8.4.2.6 TestItem

Description	<p><a href="#">TestItem</a>: A role of an <a href="#">artifact</a> that is the object of testing within a <a href="#">test configuration</a>.</p> <p>The stereotype «<a href="#">TestItem</a>» always indicates that a certain <a href="#">artifact</a> (i.e., either applied on Classifier or Property) specifies (parts of) the system under test. The use of the stereotype «<a href="#">TestItem</a>» on a Classifier is optional, but if it is used, all Properties of that type within a <a href="#">test configuration</a> must also have «<a href="#">TestItem</a>» applied, if they are used in a <a href="#">test configuration</a>.</p>
Extension	<a href="#">Classifier</a> , <a href="#">Property</a>
Super Class	<a href="#">TestConfigurationRole</a>
Associations	<pre>/configuration {subsets roleConfiguration} : TestItemConfiguration [*]</pre> <p>Refers to the configurations that are defined for this test item. This set of configurations is derived from all Constraints with «<a href="#">TestItemConfiguration</a>» applied that are either owned rules of the «<a href="#">TestItem</a>» (in case of «<a href="#">TestItem</a>» is applied on a Classifier) or that refer to the given test item using the underlying Constraint's <i>constrainedElement</i> attribute.</p>
Change from UTP 1.2	« <a href="#">TestItem</a> » has been newly introduced into UTP 2 and supersedes the «SUT» stereotype in UTP 1.

#### 8.4.2.7 TestItemConfiguration

Description	<p><a href="#">TestItemConfiguration</a>: A set of configuration options offered by an <a href="#">artifact</a> in the role of a <a href="#">test item</a> chosen to meet the requirements of a particular <a href="#">test configuration</a>.</p> <p>The stereotype «<a href="#">TestItemConfiguration</a>» specializes the abstract stereotype «<a href="#">RoleConfiguration</a>». The eventual set of configurations for a NamedElement with «<a href="#">TestItem</a>» applied is derived from the union of all <a href="#">test item configurations</a> declared for that NamedElement (i.e., either on <a href="#">Classifier</a> or <a href="#">Property</a> level).</p>
Extension	<a href="#">Constraint</a>
Super Class	<a href="#">RoleConfiguration</a>
Associations	<pre>/testItem {subsets role} : TestItem [1..*]</pre> <p>Refers to the set of at least one test items that are configured by the given configuration. The resulting set is derived from both the Classifier stereotyped with «<a href="#">TestItem</a>» that is the context of the underlying Constraint and all «<a href="#">TestItem</a>» elements, regardless whether Classifier or Property, that are referenced by the underlying <i>Constraint.constrainedElement</i>.</p>
Constraints	<p>Ownership of «<a href="#">TestItemConfiguration</a>»</p> <p>Each «<a href="#">TestItemConfiguration</a>» shall refer to at least one «<a href="#">TestItem</a>», i.e., there is no «<a href="#">TestItemConfiguration</a>» that exists without referring to a «<a href="#">TestItem</a>».</p>
Change from UTP 1.2	« <a href="#">TestItemConfiguration</a> » has been newly introduced into UTP 2.

## 8.5 Test Behavior

Test behavior is a collective term for concepts that can be executed as part of a [test set](#) or [test case](#). Since the behavioral descriptions of UML are orthogonal to each other to a certain extent, UTP introduces a set of

test execution-relevant stereotypes independently of the underlying UML Behaviors or its constituting parts. Integration with these Behaviors is done via partially multiple [extensions](#).

The concepts for test behaviors are separated into the following blocks:

- Concepts for test-specific [procedures](#) (see section [Test-specific Procedures](#))
- Concepts for [procedural element](#) (see section [Procedural Elements](#))
- Concepts for test-specific actions (see section [Test-specific Actions](#))

### 8.5.1 Test-specific Procedures

The fundamental executable concept in UTP is a [procedure](#). Any UML **Behavior** without «[TestCase](#)», «[TestExecutionSchedule](#)» or «[TestProcedure](#)» applied is considered as a [procedure](#). A [procedure](#) comprises [procedural elements](#) regardless whether the building blocks are called **InteractionFragment** (if the [procedure](#) is realized as **Interaction**) or Action (if the [procedure](#) is realized as Activity). For example, the [procedural element loop](#) is represented by the stereotype «[Loop](#)» and denotes a repeated execution of procedural elements that are contained in that loop. «[Loop](#)» extends the UML metaclasses CombinedFragment (integrating with **Interactions**) and the StructuredActivityNode [loop](#) (integrating with Activities). Furthermore, it adds some test-specific information such as the ability to provide [arbitration specifications](#), when the [loop](#) is part of a [test procedure](#).

Test-specific [procedures](#) are [procedures](#) that deliver a [verdict](#) (i.e., they can, or must in the case of a [test case](#), be arbitrated (see section Arbitration Specifications for further information about arbitration). This includes that its constituting procedural elements are arbitrated as well and provide their respective verdict to a test case arbitration specification, which potentially provides its test case verdict to a test set arbitration specification. UTP defines three different test-specific [procedures](#) for:

- [test procedure](#), represented by the stereotype «[TestProcedure](#)»
- [test case](#), represented by the stereotype «[TestCase](#)»
- [test execution schedule](#), represented by the stereotype «[TestExecutionSchedule](#)»

A [test procedure](#) is a reusable behavior that comprises [procedural elements](#) and runs on a [test configuration](#). A [test case](#) invokes one or more [test procedures](#) and assigns either of these roles: setup, main or teardown to the invoked [test procedure](#). A [test execution schedule](#) represents the invocation order of a [test set's test cases](#).

The allowed invocation scheme for test-specific [procedures](#) is as follows:

- [Test execution schedule](#) must only invoke other [test execution schedules](#), [test cases](#) or [procedures](#). The invocation of [test procedures](#) by a [test execution schedule](#) is not allowed.
- [Test case](#) must only invoke [test procedures](#) or [procedures](#), but must invoke at least one [test procedure](#) as its main part. The invocation of [test cases](#) or [test execution schedules](#) is not allowed.
- [Test procedure](#) must only invoke other [test procedures](#) or [procedures](#). The invocation of [test cases](#) or [test execution schedules](#) is not allowed.

The [test configuration](#) of the invoking [test case](#) or [test procedure](#) must be compatible with the [test configuration](#) of the invoked [test procedure](#). In the case of contained [test configurations](#) and inheritance thereof, compatibility is given by the substitution principle of Liskov. In the case of shared [test configurations](#) based on Collaboration, compatibility is defined by UML.

#### 8.5.1.1 Test Case Overview

The following diagram shows the abstract syntax of the test-specific [procedures](#).

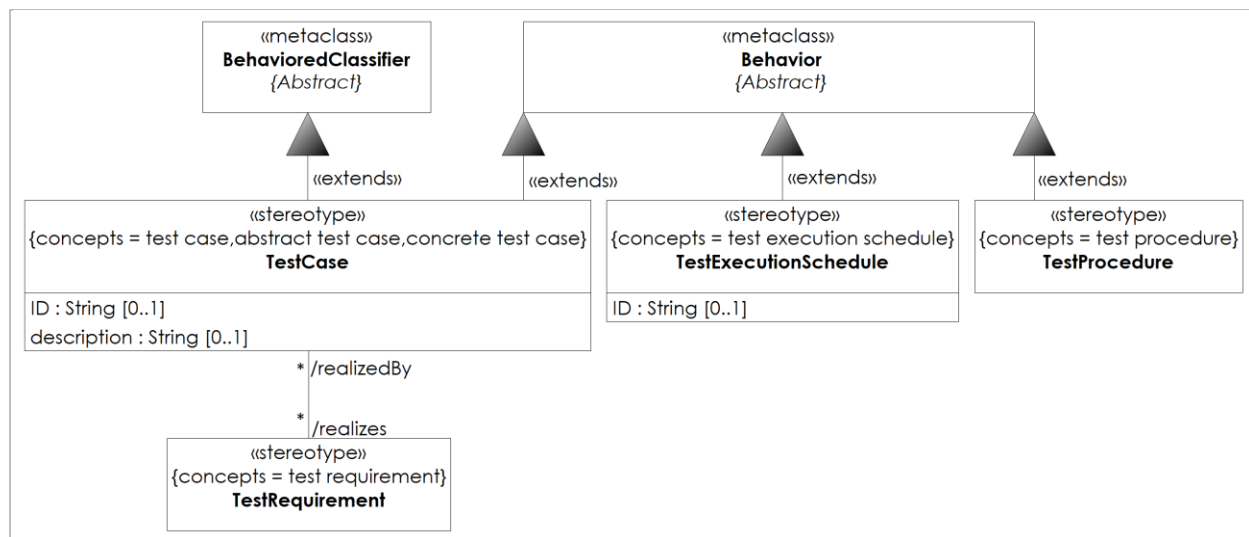


Figure 8.12 - Test Case Overview

## 8.5.1.2 Stereotype Specifications

### 8.5.1.2.1 TestProcedure


Description	<p><b>TestProcedure:</b> A <a href="#">procedure</a> that constrains the execution order of a number of <a href="#">test actions</a>.</p> <p>A <a href="#">test procedure</a> is a reusable Behavior that constitutes the building blocks for other <a href="#">test procedures</a> or <a href="#">test cases</a>. A <a href="#">test procedure</a> consists of <a href="#">procedural elements</a>, in particular <a href="#">test actions</a>.</p> <p>A <a href="#">test procedure</a> must always run on a <a href="#">test configuration</a> (i.e., its constituting <a href="#">procedural elements</a> are either executed by a <a href="#">test component</a> or a <a href="#">test item</a>). Since «TestProcedure» extends Behavior (as such both StructuredClassifier as well as BehavedClassifier), a <a href="#">test procedure</a> may provide its own dedicated <a href="#">test configuration</a> defined by its composite structures. In that case, compatibility with the <a href="#">test configuration</a> of any invoking test-specific <a href="#">procedure</a> (i.e., <a href="#">test procedure</a> or <a href="#">test case</a>) must be ensured.</p> <p>A <a href="#">test procedure</a> must only invoke other <a href="#">test procedures</a> or <a href="#">procedures</a> and must only be invoked by other <a href="#">test procedures</a> or <a href="#">test cases</a>. If invoked by a <a href="#">test case</a>, a <a href="#">test procedure</a> may assume either of these roles: main, setup or teardown. If a <a href="#">test procedure</a> invokes another <a href="#">test procedure</a> by means of «ProcedureInvocation» the attribute role of «ProcedureInvocation» must not be set. A <a href="#">test procedure</a> is not allowed to determine the role of other <a href="#">test procedures</a>, because this role can only be determined by <a href="#">test cases</a>. Implicitly, any <a href="#">test procedure</a> assigns their current role assigned by the invoking <a href="#">test case</a> to any other <a href="#">test procedure</a> they invoke. This transitive assignment will be recursively continued until no more <a href="#">test procedures</a> are available. This recursion ensures consistency for the invoking <a href="#">test case</a>.</p>
Extension	<b>Behavior</b>
Constraints	<p>Test procedure operates on test configuration</p> <p>A TestProcedure must always run on a (potentially implicit) TestConfiguration comprising at least one instance of a TestComponent connected to a TestItem</p> <p>Allowed invocation scheme</p> <p>A TestProcedure must only invoke other TestProcedures or procedures.</p>

	Use of «ProcedureInvocation» A TestProcedure must not make use of the role attribute of «ProcedureInvocation» when used as ProceduralElement of the given TestProcedure.
	Test case invokes one main procedure <a href="#">DRTP04</a> : It is necessary that each <a href="#">test case</a> invokes at least one <a href="#">test procedure</a> as a <a href="#">main procedure invocation</a> .
	Procedure sequentializes procedural element <a href="#">DRTP02</a> : It is necessary that each <a href="#">procedure</a> prescribes the execution order of at least one <a href="#">procedural element</a> .
	Test procedure sequentializes test action <a href="#">DRTP03</a> : It is necessary that each <a href="#">test procedure</a> prescribes the execution order of at least one <a href="#">test action</a> .
Change from UTP 1.2	«TestProcedure» has been newly introduced by UTP 2.

#### 8.5.1.2.2 TestCase

Description	<p><a href="#">TestCase</a>: A <a href="#">procedure</a> that includes a set of preconditions, inputs and expected results, developed to drive the examination of a <a href="#">test item</a> with respect to some <a href="#">test objectives</a>.</p> <p>«<a href="#">TestCase</a>» extends both <a href="#">BehavioredClassifier</a> and <a href="#">Behavior</a>. According to the conceptual model, a <a href="#">test case</a> must provide different functionality like defining pre-/<a href="#">postconditions</a>, being executable etc., and the UML allows different ways for implementing the <a href="#">test case</a> concept. In general, a <a href="#">test case</a> can be either defined as a standalone <a href="#">Behavior</a> stereotyped with «<a href="#">TestCase</a>» or as a compound construct consisting of a «<a href="#">TestCase</a>» <a href="#">BehavioredClassifier</a>, and a «<a href="#">TestCase</a>» <a href="#">Behavior</a> set as the classifierBehavior of the «<a href="#">TestCase</a>» <a href="#">BehavioredClassifier</a>. In the second <a href="#">alternative</a>, both the <a href="#">BehavioredClassifier</a> and its classifierBehavior are semantically treated as a single concept.</p> <p>A <a href="#">test case</a> describes the interplay of the <a href="#">test item</a> with its controlled environment, the so called test environment, consisting of <a href="#">test components</a>. A <a href="#">test case</a> has to operate on a <a href="#">test configuration</a>. The composite structure of a StructuredClassifier with «<a href="#">TestConfiguration</a>» applied determines the different roles the composite structures assume for that <a href="#">test case</a>. Test cases may define their own <a href="#">test configurations</a> as part of their dedicated composite structure (e.g. in case the stereotype «<a href="#">TestCase</a>» is applied on an instance of StructuredClassifier, or it may operate on a shared «<a href="#">TestConfiguration</a>» StructuredClassifier such as a Collaboration. If a «<a href="#">TestCase</a>» <a href="#">Behavior</a> invokes a «<a href="#">TestProcedure</a>» <a href="#">Behavior</a>, the invoked <a href="#">test procedure</a> has to operate on the same or a compatible <a href="#">test configuration</a>.</p> <p>The pre- and <a href="#">postconditions</a> of a <a href="#">test case</a> are always declared by the <a href="#">Behavior</a> with «<a href="#">TestCase</a>» applied by means of the underlying UML capability that each <a href="#">Behavior</a> may contain a number of Constraints as pre- and <a href="#">postconditions</a>. A <a href="#">test case</a> must be parameterizable. This feature is also determined by the <a href="#">Behavior</a> with «<a href="#">TestCase</a>» applied. Again, the underlying capability of a UML <a href="#">Behavior</a> is reused by UTP.</p> <p>A <a href="#">test case</a> may only invoke <a href="#">test procedures</a> as main, setup or teardown part or ordinary <a href="#">procedures</a>. A <a href="#">test case</a> must invoke at least one <a href="#">test procedure</a> as its main part. This can be either done explicitly using the stereotype «<a href="#">ProcedureInvocation</a>» or by using the underlying native UML elements for <a href="#">Behavior</a> invocation (e.g., CallBehaviorAction, InteractionUse, BehaviorExecutionSpecification etc.) If a native UML <a href="#">Behavior</a> invocation element is used and refers to a <a href="#">Behavior</a> with «<a href="#">TestProcedure</a>» applied, it is semantically equivalent with explicitly applying the stereotype «<a href="#">ProcedureInvocation</a>» on the UML <a href="#">Behavior</a> invocation element and setting the tagged value of role to main. Any <a href="#">procedural element</a> that is directly contained in <a href="#">Behavior</a> with «<a href="#">TestCase</a>» applied is considered semantically equivalent to an explicit <a href="#">Behavior</a> with «<a href="#">TestProcedure</a>» applied that contains the <a href="#">procedural element</a> and the use of «<a href="#">ProcedureInvocation</a>» within the «<a href="#">TestCase</a>» instead of the <a href="#">procedural elements</a>. This</p>
-------------	---



	<p>ensures flexibility and guarantees simplicity when defining <a href="#">test cases</a>.</p> <p>The semantics of the default arbitration specification of a test case is defined by «TestCaseArbitrationSpecification». The default arbitration specification is always active, unless an explicit «TestCaseArbitrationSpecification» is bound to the «TestCase».</p>
Graphical syntax	
Extension	<b><a href="#">Behavior</a>, <a href="#">BehavedClassifier</a></b>
Super Class	<a href="#">ArbitrationTarget</a>
Attributes	<p>ID : String [0..1]</p> <p>A unique identifier to unambiguously distinguish between any two <a href="#">test cases</a>. This is mainly intended to interface easier with management tools such as test management tools.</p> <p>description : String [0..1]</p> <p>Usually, a narrative description of the given <a href="#">test case</a>.</p> <p>/realizes : TestRequirement [*]</p> <p>The test requirements that are realized by the given test case.</p> <p>They are derived from the set of UML Realization dependencies that point from the base BehavedClassifier to UML Classes stereotyped by «<a href="#">TestRequirement</a>».</p> <p>testCaseAS : TestCaseArbitrationSpecification [0..1]</p> <p>Refers to the explicit static test case arbitration specification that overrides the implicit default test case arbitration specification.</p>
Constraints	<p>Each test case returns a verdict statement</p> <p>Any Behavior stereotyped as «<a href="#">TestCase</a>» returns a ValueSpecification typed by <a href="#">verdict</a> after arbitration had happened.</p> <p>Use of BehavedClassifier</p> <p>If «<a href="#">TestCase</a>» is applied to a BehavedClassifier that is not an instance of the metaclass Behavior, the 'classifierBehavior' of that BehavedClassifier shall be Behavior with «<a href="#">TestCase</a>» applied.</p> <p>Allowed invocation scheme</p> <p>A <a href="#">TestCase</a> must only invoke <a href="#">TestProcedure</a> or <a href="#">procedures</a>, but not other <a href="#">TestCases</a> or <a href="#">TestExecutionSchedule</a>.</p> <p>Owned UseCases not allowed</p> <p>A BehavedClassifier or Behavior with «<a href="#">TestCase</a>» applied must not own UseCases with «<a href="#">TestCase</a>» applied.</p> <p>Nested Classifier not allowed</p>



	A Behavior with «TestCase» applied must not nest any other Behavior that has «TestCase» applied.
Change from UTP 1.2	Changed from UTP 1.2. «TestCase» extended Behavior and Operation in UTP 1.2.

### 8.5.1.2.3 TestExecutionSchedule

Description	<p><b>TestExecutionSchedule:</b> A <a href="#">procedure</a> that constrains the execution order of a number of <a href="#">test cases</a>.</p> <p>A <a href="#">test execution schedule</a> is a <b>Behavior</b> with «<a href="#">TestExecutionSchedule</a>» applied that schedules the execution order of a number of <a href="#">TestCases</a>.</p> <p>A <a href="#">test execution schedule</a> can be either defined standalone or related to one or more <a href="#">test sets</a>. If a <a href="#">test execution schedule</a> is related to a <a href="#">test set</a>, the <a href="#">test execution schedule</a> is only allowed to schedule the execution of <a href="#">test cases</a> that belong to its related <a href="#">test set</a>. This holds true, even if many <a href="#">test sets</a> share the same <a href="#">test execution schedule</a>. However, it is possible, due to the semantics of <b>Behavior</b>, to specialize, invoke or redefine <a href="#">test execution schedules</a>. This enables the composition and decomposition of <a href="#">test execution schedules</a>, which, in turn, fosters reusability. A standalone <a href="#">test execution schedule</a> has the same semantics like defining a <a href="#">test set</a> that owns the <a href="#">test execution schedule</a> and assembles all the <a href="#">test cases</a> scheduled for execution by the standalone <a href="#">test execution schedule</a>. Standalone <a href="#">test execution schedules</a> may specialize or invoke non-standalone <a href="#">test execution schedules</a>. However, the semantics of the standalone <a href="#">test execution schedule</a> remains the same.</p> <p>A <a href="#">test execution schedule</a> may produce a <a href="#">test set verdict</a>, calculated by an implicit or explicit <a href="#">arbitration specification</a> for that <a href="#">test execution schedule</a>. The semantics of the default arbitration specification of a test execution schedule is defined by «TestSetArbitrationSpecification». The default arbitration specification is always active, unless an explicit «TestSetArbitrationSpecification» is bound to the «<a href="#">TestExecutionSchedule</a>».</p> <p>A <a href="#">test execution schedule</a> may invoke other <a href="#">test execution schedules</a>, <a href="#">test cases</a> or auxiliary <a href="#">procedures</a> (e.g., to retrieve required test <a href="#">data</a>), however, a <a href="#">test execution schedule</a> is not allowed to invoke a <a href="#">test procedure</a> directly (see «<a href="#">ProcedureInvocation</a>» for further information on the allowed invocation schemes). Invocation of <b>Behaviors</b> relies on the underlying UML concepts for invoking <b>Behaviors</b>. These are for Activities and StateMachines CallBehaviorAction, StartObjectBehaviorAction and StartClassifierBehaviorAction, and for Interactions InteractionUse. If such an invocation element is stereotyped with «<a href="#">ProcedureInvocation</a>», and part of a «<a href="#">TestExecutionSchedule</a>» <b>Behavior</b>, e.g., such as an Activity, the following <b>Behaviors</b> can be invoked:</p> <ul style="list-style-type: none"> <li>• <b>Behaviors</b> with «<a href="#">TestExecutionSchedule</a>» applied: Useful for decomposing and reusing <a href="#">test execution schedules</a>. If the user assigns a ProcedurePhaseKind to the invoked «<a href="#">TestExecutionSchedule</a>», it will not have an effect.</li> <li>• <b>Behaviors</b> with «<a href="#">TestCase</a>» applied: Useful for decomposing and reusing <a href="#">test cases</a>. If</li> </ul>
-------------	--

	<p>the user assigns a ProcedurePhaseKind to the invoked «<a href="#">TestCase</a>», it will not have an effect.</p> <ul style="list-style-type: none"> <li>• <b>Behaviors</b> without «<a href="#">TestExecutionSchedule</a>», «<a href="#">TestCase</a>» or «<a href="#">TestProcedure</a>» applied: Such a Behavior invoked by a «<a href="#">ProcedureInvocation</a>» is considered as auxiliary <b>Behavior</b> required to prepare the execution of succeeding «<a href="#">TestExecutionSchedule</a>», and thus, «<a href="#">TestCase</a>». The user may mark the invoked <b>Behavior</b> as setup or teardown activity by means of the role attribute.</li> </ul> <p>In the last case, a role might be assigned to an invoked <b>Behavior</b>. This role is either of setup or teardown. If the role main is assigned, it will not have an effect. <b>Behaviors</b> executed as setup or teardown <b>Behaviors</b> will not be arbitrated by a corresponding arbitration specification. The meaning of the ProcedurePhaseKind in the context of an <a href="#">test execution schedule</a> are as follows:</p> <ul style="list-style-type: none"> <li>• Setup: A means to declare that the executed <b>Behavior</b> is responsible to prepare the execution of succeeding arbitrated <a href="#">test cases</a> contained in that <a href="#">test execution schedule</a>. UTP does not prescribe which <a href="#">verdict</a> will be assigned in case something goes wrong while executing the setup phase of an arbitrated <a href="#">test execution schedule</a>.</li> </ul> <p>Teardown: A means to declare that the executed <b>Behavior</b> is responsible to clean-up after the arbitrated <a href="#">test cases</a> of this <a href="#">test execution schedule</a> have been executed. UTP does not prescribe which <a href="#">verdict</a> will be assigned in case something goes wrong while executing the teardown phase.</p>
Extension	<b>Behavior</b>
Super Class	<a href="#">ArbitrationTarget</a>
Attributes	<p>ID : String [0..1]</p> <p>A unique identifier to unambiguously distinguish between any two test execution schedules. This is mainly intended to interface easier with management tools such as test management tools.</p>
Associations	<p>testSetAS : TestSetArbitrationSpecification [0..1]</p> <p>Refers to the explicit static test set arbitration specification that overrides the implicit default test set arbitration specification. An explicit test set arbitration specification has only an effect, if the attribute <i>isArbitrated</i> is set to <i>true</i>.</p>
Constraints	<p>Allowed invocation scheme</p> <p>If a Behavior with «<a href="#">TestExecutionSchedule</a>» contains an Element with «<a href="#">ProcedureInvocation</a>» applied, the invoked Behavior shall have either none or one of the stereotypes «<a href="#">TestExecutionSchedule</a>» or «<a href="#">TestCase</a>» applied. The direct invocation of «<a href="#">TestProcedure</a>» Behaviors is not allowed from within a «<a href="#">TestExecutionSchedule</a>» Behavior.</p>
Change from UTP 1.2	« <a href="#">TestExecutionSchedule</a> » has been newly introduced by UTP 2. It was conceptually represented as the classifier behavior of a « <a href="#">TestContext</a> » in UTP 1.2.

## 8.5.2 Procedural Elements

Procedural elements constitute the building blocks of [procedures](#) and [test procedures](#). They can be realized by any building block of UML **Behavior** (e.g., **InteractionFragments** in case of Interactions, **Actions** in case of Activities and **Transitions**/Vertices in case of StateMachines). The stereotypes for [procedural elements](#) reflect the minimal language concepts that are deemed necessary for testers to specify test-specific [procedures](#). Each [procedural element](#) in a test-specific [procedure](#) has an effective [arbitration specification](#) assigned that delivers a [procedural element verdict](#) to the surrounding arbitration specification at runtime.

Since the UML **Behavior** building blocks outnumber the UTP [procedural elements](#), test-specific [procedures](#) may consist of more than just the few predefined [procedural elements](#). CombinedFragments of Interactions, for example,

offer more than just the four predefined [compound procedural elements](#) of UTP. Such a plain UML Behavior building block provides implicitly the predefined verdict instances *none* to the surrounding arbitration specification. This default semantics can be overridden by means of «[OpaqueProceduralElement](#)».

In general, UTP provides the following [procedural elements](#) out of the box:

- [procedural element](#) represented by the abstract stereotype «[ProceduralElement](#)»
- [atomic procedural element](#) represented by the abstract stereotype «[AtomicProceduralElements](#)»
- [compound procedural element](#) represented by the abstract stereotype «[CompoundProceduralElement](#)»
- opaque [procedural element](#) represented by the stereotype «[OpaqueProceduralElement](#)»

Specialized [compound procedural elements](#) comprises:

- loop represented by the stereotype «[Loop](#)»
- [sequence](#) represented by the stereotype «Sequence»
- parallel represented by the stereotype «[Parallel](#)»
- alternative represented by the stereotype «[Alternative](#)»
- negative represented by the stereotype «[Negative](#)»
- [procedure invocation](#) represented by the stereotype «[ProcedureInvocation](#)»

Specialized [atomic procedural elements](#) are described by the test-specific actions (see section [Test-specific Actions](#)).

The [procedural elements](#) have been introduced by UTP to offer a harmonized view on technically different UML behavioral building blocks.

#### 8.5.2.1 Procedural Elements Overview

The following diagram shows the abstract syntax of the core [procedural elements](#).

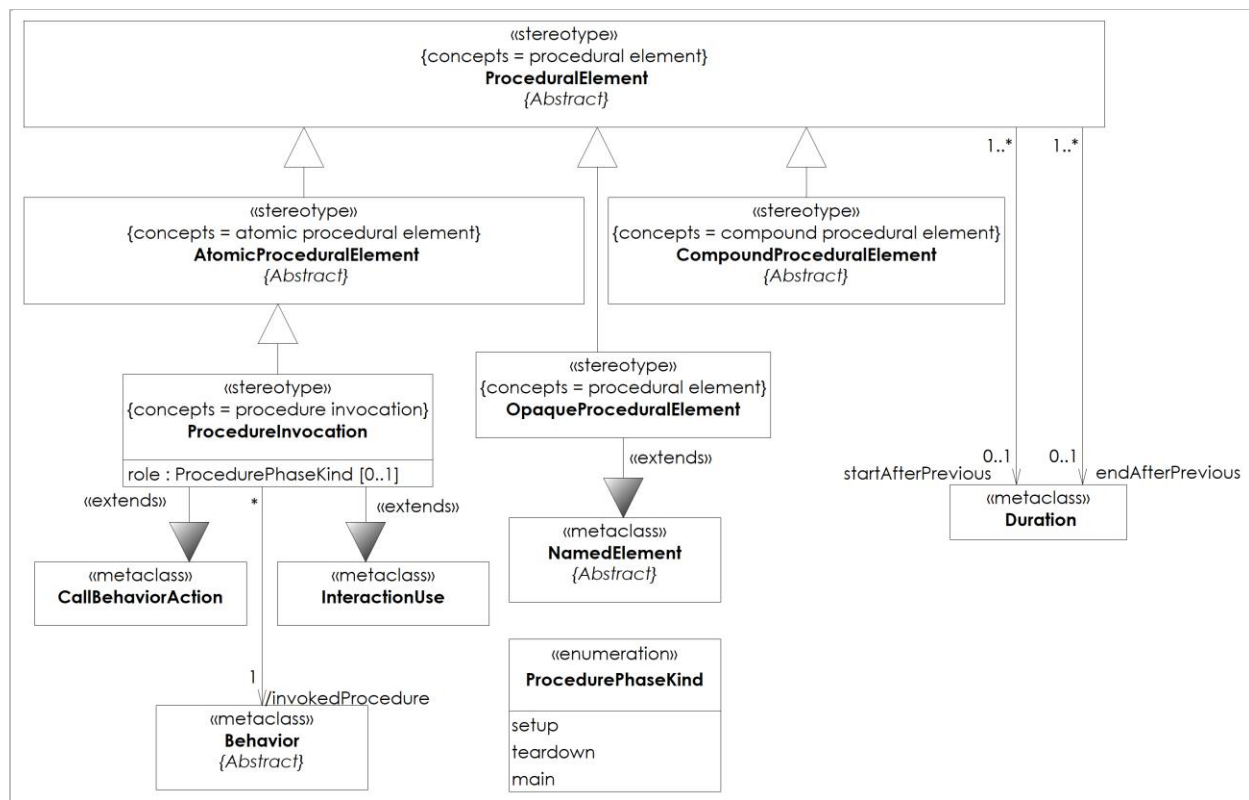


Figure 8.13 - Procedural Elements Overview

### 8.5.2.2 Compound Procedural Elements Overview

The following diagram shows the abstract syntax of the [compound procedural elements](#).

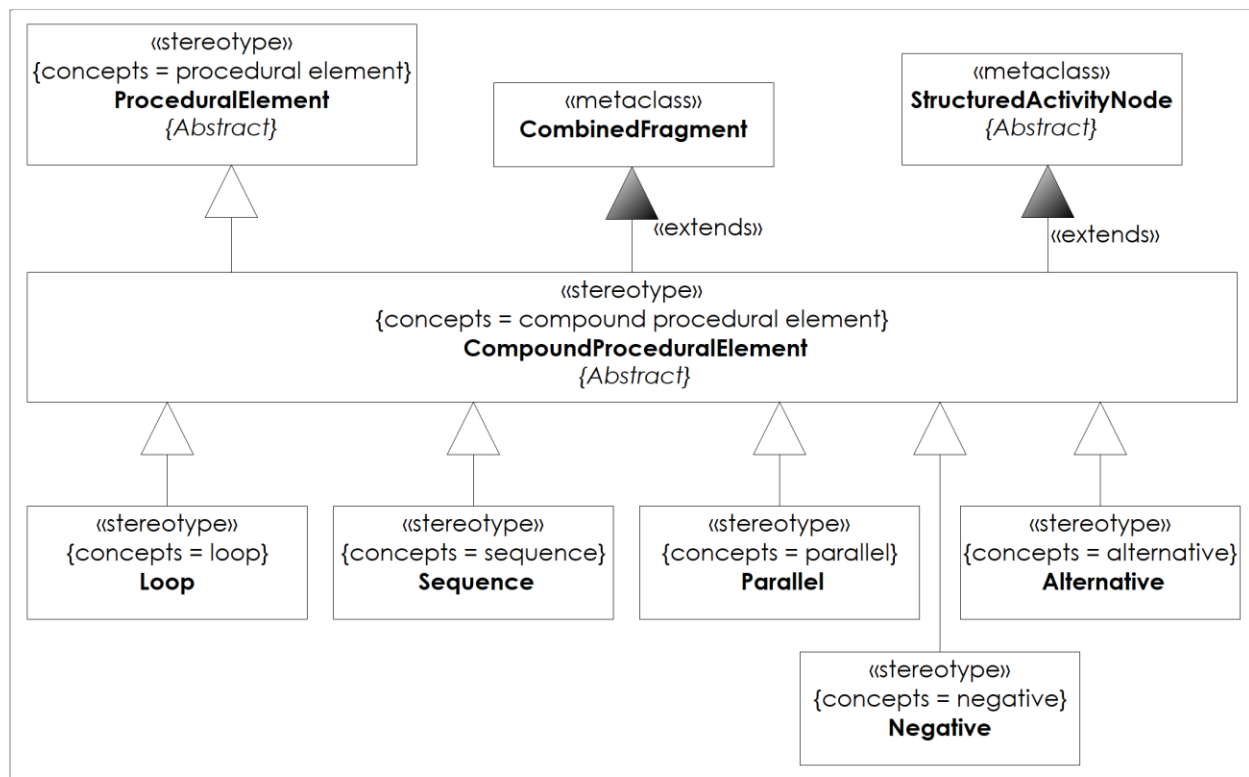


Figure 8.14 - Compound Procedural Elements Overview

### 8.5.2.3 Stereotype Specifications

#### 8.5.2.3.1 Alternative

Description	<p><a href="#">Alternative</a>: A <a href="#">compound procedural element</a> that executes only a subset of its contained <a href="#">procedural elements</a> based on the evaluation of a <a href="#">boolean expression</a>.</p> <p>If «<a href="#">Alternative</a>» is applied to <a href="#">CombinedFragment</a>, the underlying <a href="#">CombinedFragment</a> must have the <a href="#">InteractionOperatorKind alt</a> or <a href="#">opt</a> set.</p> <p>In an Activity, «<a href="#">Alternative</a>» must only be applied to <a href="#">ConditonationalNode</a>.</p>
Extension	<a href="#">CombinedFragment</a> , <a href="#">StructuredActivityNode</a>
Super Class	<a href="#">CompoundProceduralElement</a>
Associations	<p><code>arbitrationSpecification {redefines arbitrationSpecification} : AlternativeArbitrationSpecification [0..1]</code></p> <p>Refers to an alternative arbitration specification that overrides the default and implicit arbitration specification, if set. It redefines the Property <i>arbitrationSpecification</i> of <a href="#">CompoundProceduralElement</a>.</p>
Constraints	<p>Application in Interactions</p> <p>If «<a href="#">Alternative</a>» is applied to <a href="#">CombinedFragment</a>, the underlying <a href="#">CombinedFragment</a> must have the <a href="#">InteractionOperatorKind alt</a> or <a href="#">opt</a> set.</p> <p>Application in Activities</p> <p>In an Activity, «<a href="#">Alternative</a>» must only be applied to <a href="#">ConditonationalNode</a>.</p>
Change from UTP 1.2	« <a href="#">Alternative</a> » has been newly introduced by UTP 2.

### 8.5.2.3.2 AtomicProceduralElement

Description	<p><b>AtomicProceduralElement:</b> A <a href="#">procedural element</a> that cannot be further decomposed.</p> <p>«<a href="#">AtomicProceduralElement</a>» is an abstract stereotype that does not extend UML metaclass at all. This means that its substereotypes have to define suitable UML metaclass for <a href="#">extension</a>.</p> <p>Atomic <a href="#">procedural element</a>s resembles the semantics of UML Behavior building blocks that are not able to be further decomposed. Message and CallOperationAction are examples for concrete UML Behavior building block that adhere to the definition of <a href="#">atomic procedural element</a>. In contrast, CombinedFragment or LoopNode are examples for <a href="#">compound procedural elements</a> for they contain potentially further <a href="#">procedural elements</a>.</p>
Super Class	<a href="#">ProceduralElement</a>
Sub Class	<a href="#">CheckPropertyAction</a> , <a href="#">CreateLogEntryAction</a> , <a href="#">CreateStimulusAction</a> , <a href="#">ExpectResponseAction</a> , <a href="#">ProcedureInvocation</a> , <a href="#">SuggestVerdictAction</a>
Associations	<p>arbitrationSpecification {redefines arbitrationSpecification} : AtomicProceduralElementArbitrationSpecification [0..1]</p> <p>Refers to an atomic arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of procedural element.</p>
Change from UTP 1.2	«AtomicProceduralElement» has been newly introduced by UTP 2.

### 8.5.2.3.3 CompoundProceduralElement

Description	<p><b>CompoundProceduralElement:</b> A <a href="#">procedural element</a> that can be further decomposed.</p> <p>«<a href="#">CompoundProceduralElement</a>» is an abstract stereotype that extends CombinedFragment and StructuredActivityNode to interface with the UML Behaviors Interaction and Activity.</p> <p>A <a href="#">compound procedural element</a> resembles the semantics of UML Behavior building blocks that consist of other <a href="#">procedural element</a>. As such, it may obtain the <a href="#">verdicts</a> of its contained executed <a href="#">procedural elements</a> in order to calculate its own <a href="#">procedural element verdict</a>. The difference between an atomic <a href="#">procedural element verdict</a> and <a href="#">compound procedural element verdict</a> is that the latter is potentially composed out of multiple atomic <a href="#">procedural element verdicts</a>.</p>
Extension	<a href="#">CombinedFragment</a> , <a href="#">StructuredActivityNode</a>
Super Class	<a href="#">ProceduralElement</a>
Sub Class	<a href="#">Alternative</a> , <a href="#">Loop</a> , <a href="#">Negative</a> , <a href="#">Parallel</a> , <a href="#">Sequence</a>
Associations	<p>arbitrationSpecification {redefines arbitrationSpecification} : CompoundProceduralElementArbitrationSpecification [0..1]</p>
Change from UTP 1.2	«CompoundProceduralElement» has been newly introduced by UTP 2.

#### 8.5.2.3.4 Loop

Description	<p><b>Loop</b>: A <a href="#">compound procedural element</a> that repeats the execution of its contained <a href="#">procedural elements</a>.</p> <p>If «<b>Loop</b>» is applied to CombinedFragement, the underlying CombinedFragment must have the InteractionOperatorKind <a href="#">loop</a> set.</p> <p>In an Activity, «<b>Loop</b>» must only be applied to LoopNode.</p> <p>The nature of the <a href="#">loop</a> (i.e., counter-controlled <a href="#">loop</a>, conditional-controlled <a href="#">loop</a> or collection-controlled <a href="#">loop</a>) is determine by the configuration of the underlying UML element for expressing <a href="#">loops</a>.</p>
Extension	<b>CombinedFragment, StructuredActivityNode</b>
Super Class	<a href="#">CompoundProceduralElement</a>
Associations	<p>arbitrationSpecification {redefines arbitrationSpecification} : LoopArbitrationSpecification [0..1]</p> <p>Refers to a loop arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of CompoundProceduralElement.</p>
Constraints	<p>Application in Interactions</p> <p>If «<b>Loop</b>» is applied to <b>CombinedFragment</b>, the underlying <b>CombinedFragment</b> must have the InteractionOperatorKind <i>loop</i> set.</p> <p>Application in Activities</p> <p>In an Activity, «<b>Loop</b>» must only be applied to LoopNode.</p>
Change from UTP 1.2	«Loop» has been newly introduced by UTP 2.

#### 8.5.2.3.5 Negative

Description	<p><b>Negative</b>: A <a href="#">compound procedural element</a> that prohibits the execution of its contained <a href="#">procedural elements</a> in the specified structure.</p> <p>If «<b>Negative</b>» is applied to CombinedFragement, the underlying CombinedFragment must have the InteractionOperatorKind <i>neg</i> set.</p> <p>In an Activity, «<b>Negative</b>» must only be applied to StructuredActivityNode.</p>
Extension	<b>CombinedFragment, StructuredActivityNode</b>
Super Class	<a href="#">CompoundProceduralElement</a>
Associations	<p>arbitrationSpecification {redefines arbitrationSpecification} : NegativeArbitrationSpecification [0..1]</p>
Constraints	<p>Application in Interactions</p> <p>If «<b>Negative</b>» is applied to <b>CombinedFragment</b>, the underlying <b>CombinedFragment</b> must have the InteractionOperatorKind <i>neg</i> set.</p> <p>Application in Activities</p> <p>In an Activity, «<b>Negative</b>» must only be applied to StructuredActivityNode.</p>
Change from UTP 1.2	« <b>Negative</b> » has been newly introduced by UTP 2.

### 8.5.2.3.6 OpaqueProceduralElement

Description	« <a href="#">OpaqueProceduralElement</a> » adds the possibility to assign <a href="#">arbitration specifications</a> to UML Behavior building blocks that are not covered by UTP <a href="#">procedural elements</a> . Thus, it is a plain technical stereotype introduced for flexibility of UTP. Similar to the semantics of opaque elements in UML (i.e., OpaqueBehavior, OpaqueExpression, OpaqueAction), there is no additional semantics for « <a href="#">OpaqueProceduralElement</a> » given apart from the ability to assign <a href="#">arbitration specifications</a> to UML elements for which no dedicated <a href="#">procedural element</a> stereotype has been defined.
Extension	<a href="#">NamedElement</a>
Super Class	<a href="#">ProceduralElement</a>
Constraints	Only applicable to UML Behavior building blocks  « <a href="#">OpaqueProceduralElement</a> » must only be applied on instances of the UML metaclass <a href="#">Action</a> , <a href="#">InteractionFragment</a> , Vertex and <a href="#">Transition</a> .
Change from UTP 1.2	« <a href="#">OpaqueProceduralElement</a> » has been newly introduced by UTP 2.

### 8.5.2.3.7 Parallel

Description	<p><a href="#">Parallel</a>: A <a href="#">compound procedural element</a> that executes its contained <a href="#">procedural elements</a> in parallel to each other.</p> <p>If «<a href="#">Parallel</a>» is applied to CombinedFragement, the underlying CombinedFragment must have the InteractionOperatorKind <i>par</i> set.</p> <p>If used in Activities, the metaclass ConditionalNode is reused to describe parallel execution of <a href="#">procedural elements</a> (i.e., ExecutableNodes). The branches that must be executed in parallel are defined by the Clauses that are contained in a ConditionalNode with «<a href="#">Parallel</a>» applied. If such a ConditionalNode is activated and ready for execution, the evaluation of the Clauses by executing the test parts are executed as described by UML. In contrast to a plain ConditionalNode, where at most one Clause's body part will be executed, even if more than one Clause's test part eventually enabled the Clause, all enabled Clause's body parts are executed in parallel, if the ConditionalNode has «<a href="#">Parallel</a>» applied.</p>
Extension	<a href="#">CombinedFragment</a> , <a href="#">StructuredActivityNode</a>
Super Class	<a href="#">CompoundProceduralElement</a>
Associations	<pre>arbitrationSpecification {redefines arbitrationSpecification} :</pre> <pre>ParallelArbitrationSpecification [0..1]</pre> <p>Refers to a parallel arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of CompoundProceduralElement.</p>
Constraints	<p>Application in Interactions</p> <p>If «<a href="#">Parallel</a>» is applied to <a href="#">CombinedFragment</a>, the underlying <a href="#">CombinedFragment</a> must have the InteractionOperatorKind <i>par</i> set.</p>
Change from UTP 1.2	« <a href="#">Parallel</a> » has been newly introduced by UTP 2.



### 8.5.2.3.8 ProceduralElement

Description	<p><a href="#">ProceduralElement</a>: An instruction to do, to observe, and/or to decide.</p> <p>«<a href="#">ProceduralElement</a>» is an abstract stereotype that does not extend any UML metaclass. This means that its substereotypes have to define suitable UML metaclasses for <a href="#">extension</a>.</p> <p>A <a href="#">procedural element</a> is the lowest common denominator for the building blocks of the different UML Behaviors. If used as constituting part (possibly transitively) of a <a href="#">test case</a> execution, every <a href="#">procedural element</a> delivers a <a href="#">verdict</a> depending on both the execution of the respective <a href="#">procedural element</a> and the effective <a href="#">arbitration specification</a> of that <a href="#">procedural element</a>. Every procedural element has an effective arbitration specification assigned at evaluation time. This effective arbitration specification is either the default arbitration specification of the respective procedural element or an explicitly bound arbitration specification. If no explicit arbitration specification is bound to the procedural element, the default arbitration specification becomes the effective arbitration specification.</p> <p>A <a href="#">procedural element</a> adds the ability to specify the expected starting and end point of the execution of <a href="#">procedural element</a> related to a previously executed <a href="#">procedural element</a>, represented by the tag definitions <i>startAfterPrevious</i> and <i>endAfterPrevious</i>. These timing-related characteristics are represented by means of explicit tag definitions in addition to the existing simple time concepts of UML and time-related information potentially available by further UML profiles such as MARTE. UTP 2 does not prescribe which of these timing-related concepts should be used. As a recommendation, users should not mix different mechanisms to express timing-related information.</p>
Sub Class	<a href="#">AtomicProceduralElement</a> , <a href="#">CompoundProceduralElement</a> , <a href="#">OpaqueProceduralElement</a>
Super Class	<a href="#">ArbitrationTarget</a>
Associations	<p>arbitrationSpecification : ProceduralElementArbitrationSpecification [0..1]</p> <p>Refers to a procedural element arbitration specification that overrides the default and implicit arbitration specification for procedural elements.</p> <p>startAfterPrevious : Duration [0..1] Determines the Duration the execution of this ProceduralElement shall start at the earliest.</p> <p>endAfterPrevious : Duration [0..1] Determines the Duration the execution of this ProceduralElement shall end at the latest.</p>
Constraints	<p>Valid duration</p> <p><b>DRTP01</b>: It is necessary that the <a href="#">PE start duration</a> of a <a href="#">procedural element</a> is smaller than the <a href="#">PE end duration</a> of the same <a href="#">procedural element</a>.</p>
Change from UTP 1.2	«ProceduralElement» has been newly introduced by UTP 2.

### 8.5.2.3.9 ProcedureInvocation

Description	<p><b>ProcedureInvocation</b>: An atomic procedural element of a procedure that invokes another procedure and waits for its completion.</p> <p>«ProcedureInvocation» is a means to invoke procedures from within other procedures. Since the constituents of UML <b>Behavior</b>s are not based on an integrated metaclass, the concrete metaclasses for «<b>ProcedureInvocation</b>» depend on the <b>Behavior</b> kind in which the «<b>ProcedureInvocation</b>» is used. If it represents a building block of an Activity or StateMachine, «<b>ProcedureInvocation</b>» must only be applied on the metaclass CallBehaviorAction, StartObjectBehaviorAction or StartClassifierBehaviorAction. If it represents a building block of an Interaction, «<b>ProcedureInvocation</b>» must only be applied on the metaclass InteractionUse.</p> <p>The allowed invocation scheme for a «<b>ProcedureInvocation</b>» is as follows:</p> <ul style="list-style-type: none"> <li>• If it constitutes a <a href="#">procedural element</a> of a <a href="#">test execution schedule</a>, only <a href="#">test execution schedules</a>, <a href="#">test cases</a> or <a href="#">procedures</a> must be invoked.</li> <li>• If it constitutes a procedural element of a <a href="#">test case</a>, only <a href="#">test procedures</a> and <a href="#">procedures</a> must be invoked.</li> <li>• If it constitutes a procedural element of a <a href="#">test procedure</a>, only <a href="#">test procedure</a> or <a href="#">procedures</a> must be invoked.</li> </ul> <p>If <a href="#">procedure invocation</a> is part of a <a href="#">test case</a> it must assign a <a href="#">role</a> to the invoked <a href="#">test procedure</a>. This <a href="#">role</a> is either <i>main</i>, <i>setup</i> or <i>teardown</i>. The semantics of these <a href="#">roles</a> in UTP are:</p> <ul style="list-style-type: none"> <li>• <i>main</i>: A <a href="#">test procedure</a> that implements the reason why the invoking <a href="#">test case</a> has been designed, i.e., it contribute to the coverage of a <a href="#">test objective</a> or <a href="#">test requirement</a>. The main part of a <a href="#">test case</a> is relevant for calculating coverage and controlling the progress.</li> <li>• <i>setup</i>: A means to declare that the executed <a href="#">test procedure</a> is responsible to prepare the main part of a <a href="#">test case</a>.</li> <li>• <i>teardown</i>: A means to declare that the executed <a href="#">test procedure</a> is responsible to clean-up after the main part of a <a href="#">test case</a> has been executed.</li> </ul> <p>If <a href="#">procedure invocation</a> is part of a <a href="#">test execution schedule</a> it may assign a <a href="#">role</a> to an invoked <b>Behavior</b>. This <a href="#">role</a> is either of <i>setup</i> or <i>teardown</i>. The semantics of these <a href="#">roles</a> in UTP are:</p> <ul style="list-style-type: none"> <li>• <i>setup</i>: A means to declare that the executed <b>Behavior</b> is responsible to prepare the execution of arbitrated <a href="#">test cases</a> contained in that <a href="#">test case</a>.</li> <li>• <i>teardown</i>: A means to declare that the executed <b>Behavior</b> is responsible to clean-up after the arbitrated <a href="#">test cases</a> of this <a href="#">test execution schedule</a> have been executed.</li> </ul>
Extension	<b>CallBehaviorAction</b> , <b>InteractionUse</b>
Super Class	<a href="#">AtomicProceduralElement</a>
Attributes	<p><code>role : ProcedurePhaseKind [0..1]</code></p> <p>The role, the invoked procedure assumes within the invoking test-specific procedure.</p>
Associations	<p><code>arbitrationSpecification {redefines arbitrationSpecification} : ProcedureInvocationArbitrationSpecification [0..1]</code></p> <p>Refers to a procedure invocation arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of «CompoundProceduralElement».</p>

	<p>/invokedProcedure : Behavior</p> <p>The procedure that was invoked by that «ProcedureInvocation». If «ProcedureInvocation» is applied to CallBehaviorAction, it is derived from the property 'behavior' of the underlying CallBehaviorAction. If «ProcedureInvocation» is applied to InteractionUse, it is derived from the property 'refersTo' of the underlying InteractionUse.</p>
Constraints	<p>Role only in context of test cases relevant</p> <p>If «<a href="#">ProcedureInvocation</a>» is part of a «<a href="#">TestProcedure</a>» <b>Behavior</b>, the tag definition <a href="#">role</a> must be empty. If it is empty, it will be ignored.</p>
Change from UTP 1.2	«ProcedureInvocation» has been newly introduced by UTP 2.

#### 8.5.2.3.10 Sequence

Description	<p><b>Sequence:</b> A <a href="#">compound procedural element</a> that executes its contained <a href="#">procedural elements</a> sequentially.</p> <p>If «<a href="#">Sequence</a>» is applied to CombinedFragement, the underlying CombinedFragment must have the InteractionOperatorKind <i>strict</i> or <i>seq</i> applied.</p> <p>In an Activity, «<a href="#">Sequence</a>» must only be applied to SequenceNode.</p>
Extension	<b>CombinedFragment, StructuredActivityNode</b>
Super Class	<a href="#">CompoundProceduralElement</a>
Associations	<p>arbitrationSpecification {redefines arbitrationSpecification} : SequenceArbitrationSpecification [0..1]</p> <p>Refers to a SequenceArbitrationSpecification that overrides the default and implicit ArbitrationSpecification if set. It redefines the Property <i>arbitrationSpecification</i> of CompoundProceduralElement.</p>
Constraints	<p>Application in Interactions</p> <p>If applied on a <b>CombinedFragment</b>, the underlying <b>CombinedFragment</b> must have set InteractionOperatorKind::seq or InteractionOperatorKind::strict as the <i>interactionOperator</i>.</p>
Change from UTP 1.2	«Sequence» has been newly introduced by UTP 2.

#### 8.5.2.4 Enumeration Specifications

Name	Description	Enumeration literals
ProcedurePhaseKind	An enumeration of the three possible values a <a href="#">procedure</a> or <a href="#">test procedure</a> can assume.	<p>setup</p> <p>The invoked procedure or test procedure is considered as a preamble of the test case or a test execution schedule, intended to prepare the execution of test cases.</p>
		<p>teardown</p> <p>The invoked procedure or test procedure is considered as a postamble of the test case or a test execution schedule, intended to clean-up or finalize the execution of test cases.</p>
		<p>main</p> <p>The invoked test procedure is considered as the essential part of a test case's execution with respect to coverage.</p>

### 8.5.3 Test-specific Actions

UTP introduces dedicated test-specific actions that denote actions a tester, regardless whether this is an automated or human tester, can carry out in order to communicate with the [test item](#). In context of dynamic testing, communicating with a [test item](#) either means to stimulate the [test item](#) with a [create stimulus action](#) (implemented as stereotype «[CreateStimulusAction](#)») or observing and evaluating its actual [responses](#) with the expected ones (represented by the stereotypes «[ExpectResponseAction](#)», «[CheckPropertyAction](#)»).

Test-specific actions are specialized [procedural elements](#). As such, they contribute a dedicated [procedural element verdict](#) to the eventual calculation of a [test case](#) or [test set verdict](#). The test-specific actions can be categorized by the entity that contributes information to the calculation of the respective [procedural element verdict](#).

The [procedural element verdicts](#) of the following test-specific actions are calculated by taking into consideration the information provided by the [test component](#) or tester. These test-specific actions are henceforth called [test component controlled actions](#), because an erroneous execution of these [test actions](#) indicates a misbehavior of the [test component](#) (submitting the wrong [stimulus](#), performing a test-specific action too late/too early) or technical issues in the test environment (e.g., breakdown of connectivity etc.):

- Create [stimulus](#) action represented by the stereotype «[CreateStimulusAction](#)»
- Suggest [verdict](#) action represented by the stereotype «[SuggestVerdictAction](#)»
- Create log entry action represented by the stereotype «[CreateLogEntryAction](#)»

It is highly recommended that the [verdicts](#) calculated by these [test component controlled actions](#) should only result in the predefined [verdict](#) instances [pass](#) or [error](#).

The [verdict](#) of following test-specific actions is calculated by taken into consideration information received by the [test items](#). These test-specific actions are henceforth called [test item controlled actions](#), because the arbitration of these test-specific actions depend on the [responses](#) of the [test items](#) during execution and as such indicate deviations between the expected [response](#) and actual [response](#):

- Expect [response](#) action represented by the stereotype «[ExpectResponseAction](#)»
- Check property action represented by the stereotype «[CheckPropertyAction](#)»

It is highly recommended that the verdicts calculated by test component-controlled actions should only result in the predefined verdict instances pass or error.

#### 8.5.3.1 Test-specific actions Overview

The following diagram shows the abstract syntax of the [test action](#).

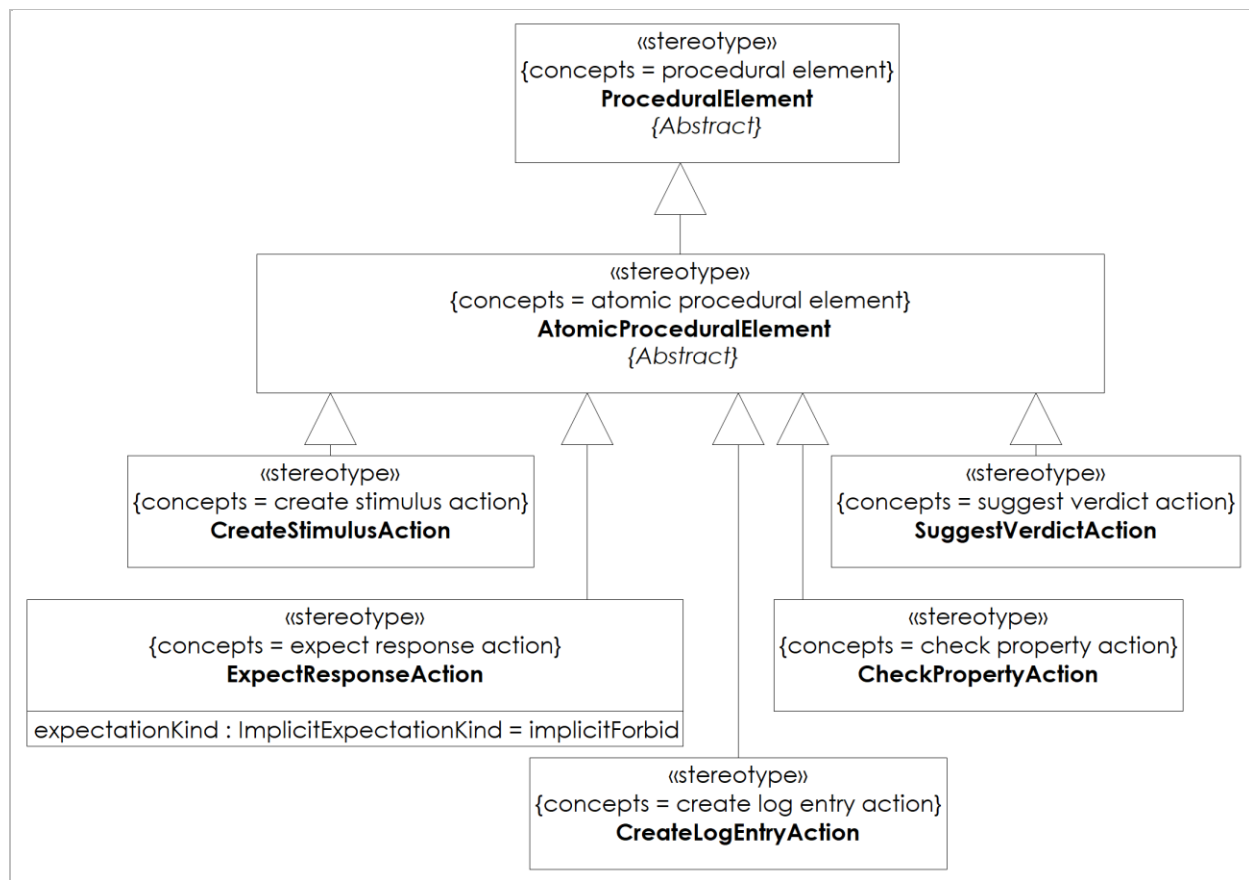


Figure 8.15 - Test-specific actions Overview

### 8.5.3.2 Tester Controlled Actions

The following diagram shows the details of the [test component](#) controlled [test actions](#).

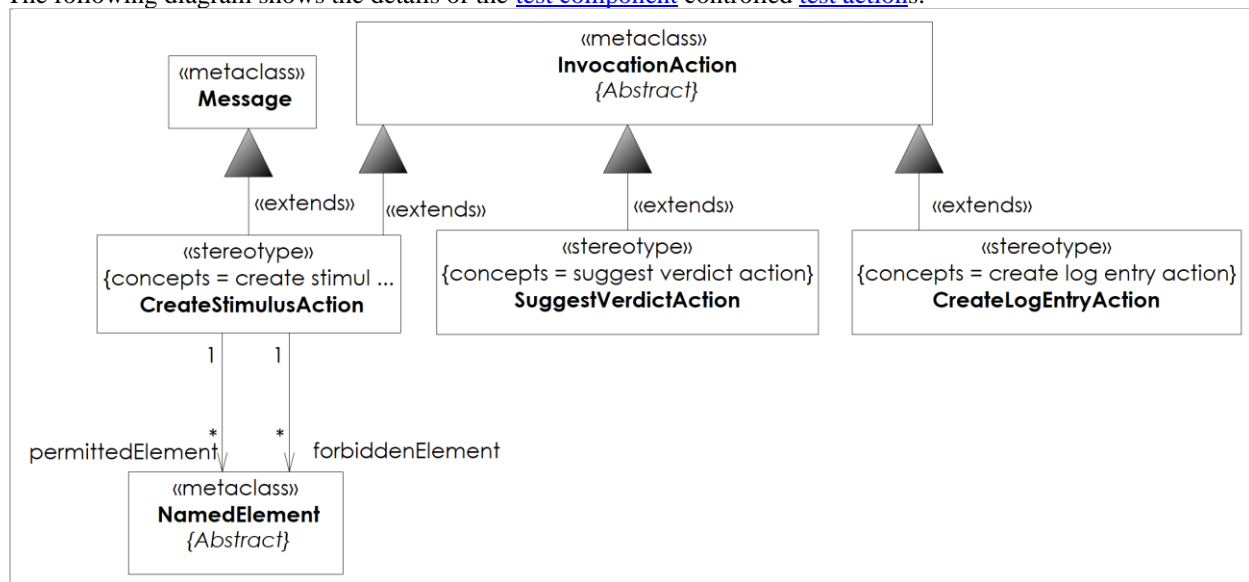


Figure 8.16 - Tester Controlled Actions

### 8.5.3.3 Test Item Controlled Actions

The following diagram shows the details of the [test item](#) controlled [test actions](#).

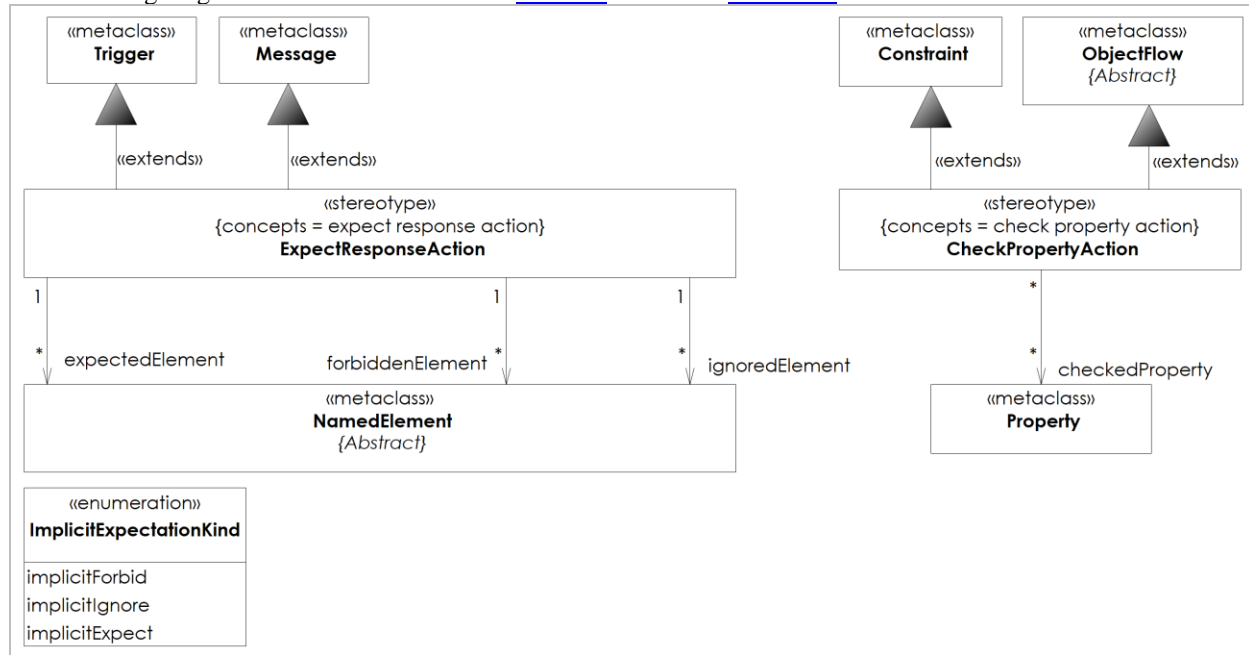
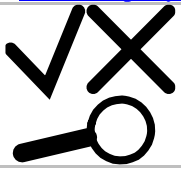


Figure 8.17 - Test Item Controlled Actions


## 8.5.3.4 Stereotype Specifications

### 8.5.3.4.1 CheckPropertyAction

Description	<p><a href="#">CheckPropertyAction</a>: A <a href="#">test action</a> that instructs the tester to check the conformance of a <a href="#">property</a> of the <a href="#">test item</a> and to set the <a href="#">procedural element verdict</a> according to the result of this check.</p> <p>The stereotype «<a href="#">CheckPropertyAction</a>» extends Constraint (for integration with Interaction's StateInvariant and StateMachines), and ObjectFlow (for integration with Activities) and enables the <a href="#">test component</a> to check certain properties of the <a href="#">test item</a> that cannot be checked via the publicly available or known APIs of the <a href="#">test item</a>. Thus, it is not defined how the <a href="#">test component</a> accesses the <a href="#">test item</a>'s <a href="#">property</a>.</p> <p>If used in Interactions, <a href="#">check property action</a> is used as Constraint of a StateInvariant that covers a <a href="#">test component</a>. Such a Constraint must be contained by StateInvariants. The specification of the StateInvariant's «<a href="#">CheckPropertyAction</a>» Constraint is intended to determine the Property of the <a href="#">test item</a> that must be checked and the value the Property has to match with. As specification of the «<a href="#">CheckPropertyAction</a>» Constraint, any kind of suitable ValueSpecification can be utilized. For example, the «<a href="#">CheckPropertyAction</a>» Constraint may specify location expressions with OCL or Alf for declaring access and expected values of the <a href="#">test item</a>'s Property.</p> <p>If used in StateMachines, <a href="#">check property action</a> is expressed as stateInvariant attribute of a State. Since the stateInvariant attribute is of type Constraint, the usage, application and semantics is similar to the <a href="#">check property action</a> used in Interactions (i.e., use of StateInvariant in Interactions).</p> <p>If used in Activities, <a href="#">check property action</a> is expressed as «<a href="#">CheckPropertyAction</a>» ObjectFlow that emanates from a ReadStructuralFeatureAction and is used to access a StructuralFeature of the <a href="#">test item</a>. The expected value of the checked Property is defined by the guard condition of the <a href="#">CheckPropertyAction</a> ObjectFlow.</p> <p>In addition, it is possible to point directly to the Property that will be checked by the <a href="#">check property action</a> by means of the tag definition checkedProperty. This information is helpful, if, for example, natural language is used to describe «<a href="#">CheckPropertyAction</a>» Constraint.</p> <p>The default <a href="#">arbitration specification</a> for the <a href="#">check property action</a> is described by «<a href="#">CheckPropertyArbitrationSpecification</a>».</p>
Graphical syntax	
Extension	<a href="#">Constraint</a> , <a href="#">ObjectFlow</a>
Super Class	<a href="#">AtomicProceduralElement</a>
Associations	<pre>arbitrationSpecification {redefines arbitrationSpecification} : CheckPropertyArbitrationSpecification [0..1]</pre> <p>Refers to a check property action arbitration specification that overrides the default and implicit arbitration specification, if set. It redefines the Property <i>arbitrationSpecification</i> of test action.</p>

	<p><code>checkedProperty : Property [*]</code></p> <p>Refers to set of Properties of a test item that is supposed to be checked by the check property action.</p>
Constraints	<p>Owner of Constraint</p> <p>If applied on a <b>Constraint</b>, the owner of this <b>Constraint</b> must only be a <b>State</b> (referring to the <b>Constraint</b> as <b>StateInvariant</b>) or <b>StateInvariant</b>.</p> <p>Owner of Property</p> <p>If '<code>checkedProperty</code>' is not empty, the referenced <b>Property</b> must belong to a <b>TestItem</b> participating in the current test-specific procedure.</p> <p>At least one property</p> <p><b>DRTA03</b>: It is necessary that a <a href="#">check property action</a> checks at least one <a href="#">property</a> of the <a href="#">test item</a> against the <a href="#">data</a>.</p>
Change from UTP 1.2	«CheckPropertyAction» has been newly introduced by UTP 2.

#### 8.5.3.4.2 CreateLogEntryAction

Description	<p><b>CreateLogEntryAction</b>: A <a href="#">test action</a> that instructs the tester to record the execution of a <a href="#">test action</a>, potentially including the outcome of that <a href="#">test action</a> in the <a href="#">test case log</a>.</p> <p>The stereotype «<a href="#">CreateLogEntryAction</a>» extends <a href="#">InvocationAction</a> which allows for using a variety of metaclasses for application. The <a href="#">create log entry action</a> is a <a href="#">test action</a> that instructs the tester or the test execution system to log certain information about the execution of a <a href="#">test case</a>. This information is henceforth called content to be logged. The content to be logged has to be provided as the argument InputPin of the underlying <a href="#">InvocationAction</a>. It is not specified how the variety of potentially logable contents is eventually be represented in the log. Test execution systems are responsible for eventually writing the content to be logged into the actual <a href="#">test log</a>.</p> <p>If used in an Interaction, the <a href="#">InvocationAction</a> that is stereotyped with «<a href="#">CreateLogEntryAction</a>» should be referenced from an <a href="#">ActionExecutionSpecification</a> that indirectly covers a Lifeline that represents a <a href="#">test component</a> role in the underlying <a href="#">test configuration</a>. Indirectly means that the corresponding start and end OccurrenceSpecification of the <a href="#">ActionExecutionSpecification</a> cover the <a href="#">test component</a> lifeline.</p> <p>If used in Activities or StateMachines, e.g., <a href="#">CallOperationAction</a> could be used to invoke a (not standardized, yet proprietary) logging interface operation. Another possibility is to use <a href="#">SendObjectAction</a> without specifying the target Pin which has the semantics to submit the information to be logged to the logging facility of the test execution system without needing a dedicated interface. However, during test execution the <a href="#">create log entry action</a> must be made executable and eventually carried out. This may include manually writing some information into a paper-based document.</p> <p>The default <a href="#">arbitration specification</a> for the <a href="#">create log entry action</a> is described by «<a href="#">CreateLogEntryArbitrationSpecification</a>».</p>
Graphical syntax	
Extension	<a href="#">InvocationAction</a>
Super Class	<a href="#">AtomicProceduralElement</a>



Associations	<pre>arbitrationSpecification {redefines arbitrationSpecification} :</pre> <pre>CreateLogEntryArbitrationSpecification [0..1]</pre> <p>Refers to a create log entry action arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of test action.</p>
Change from UTP 1.2	«CreateLogEntryAction» has been newly introduced by UTP 2.

#### 8.5.3.4.3 CreateStimulusAction

Description	<p><a href="#">CreateStimulusAction</a>: A <a href="#">test action</a> that instructs the tester to submit a <a href="#">stimulus</a> (potentially including <a href="#">data</a>) to the <a href="#">test item</a>.</p> <p>«<a href="#">CreateStimulusAction</a>» extends Message (for integration with Interaction) and InvocationAction (for integration with Activities and StateMachines).</p> <p>The <a href="#">create stimulus action</a> is performed by an instance of a <a href="#">test component</a> and represents a set of possible invocations of the <a href="#">test item</a>, potentially conveyed by a payload. Invocation means that either a BehavioralFeature of the <a href="#">test item</a> is invoked (e.g. using a Message or a SendSignalAction respectively CallOperationAction) or by simply sending a <a href="#">stimulus</a> to the <a href="#">test items</a> (e.g., SendObjectAction or BroadcastSignalAction).</p> <p>The set of stimuli to be sent is derived from the arguments of the underlying UML element and the elements specified by the tag definition permittedElement. This set is then reduced by the elements yield by forbiddenElement. If the set of stimuli is empty (i.e., neither the underlying UML element yields arguments nor the permittedElement tag definition yields an element), it is semantically equivalent to a situation where any possible and known by the invoking <a href="#">test component</a> stimuli at this point in time can be send to the <a href="#">test item</a>. This set of any possible and known stimuli is potentially reduced by the elements yield by forbiddenElement. In case the set of permitted elements and the set of forbidden elements are overlapping, the elements in the intersection belong to the set of forbidden elements. If both sets are empty, every known stimuli can be send to the <a href="#">test item</a>.</p> <p>The default <a href="#">arbitration specification</a> for the <a href="#">create stimulus action</a> is described by «<a href="#">CreateStimulusArbitrationSpecification</a>».</p>
Extension	<a href="#">InvocationAction</a> , <a href="#">Message</a>
Super Class	<a href="#">AtomicProceduralElement</a>
Associations	<pre>arbitrationSpecification {redefines arbitrationSpecification} :</pre> <pre>CreateStimulusArbitrationSpecification [0..1]</pre> <p>Refers to a create stimulus action arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of test action.</p> <pre>forbiddenElement : NamedElement [*]</pre> <p>A set of elements that are explicitly removed from the set of stimuli to be sent.</p> <pre>permittedElement : NamedElement [*]</pre> <p>Additional set of stimuli that contribute to the set of permitted stimuli.</p>
Constraints	<p>Type of forbidden elements</p> <p>The tag definition '<a href="#">forbiddenElement</a>' shall only contain instances of the following metaclasses: Message, Event, Signal, BehavioralFeature, Trigger, InstanceSpecification.</p>

	<p>Type of permitted elements</p> <p>The tag definition '<a href="#">permittedElement</a>' shall only contain instances of the following metaclasses: Message, Event, Signal, BehavioralFeature, Trigger, InstanceSpecification.</p>
	<p>At least one stimulus</p> <p><a href="#">DRTA01</a>: It is necessary that a <a href="#">create stimulus action</a> permits to send at least one <a href="#">stimulus</a>.</p>
Change from UTP 1.2	«CreateStimulusAction» has been newly introduced by UTP 2.


#### 8.5.3.4.4 ExpectResponseAction

Description	<p><a href="#">ExpectResponseAction</a>: A <a href="#">test action</a> that instructs the tester to check the occurrence of one or more particular <a href="#">responses</a> from the <a href="#">test item</a> within a given time window and to set the <a href="#">procedural element verdict</a> according to the result of this check.</p> <p>The stereotype «<a href="#">ExpectResponseAction</a>» extends Message (for integration with Interactions) and Trigger (for integration with StateMachines and Activities) and denotes the expectation of the <a href="#">test component</a> to receive an actual <a href="#">response</a>, potentially conveyed by some payload, from the <a href="#">test item</a> at a certain point in time during test execution.</p> <p>Actually received information from the <a href="#">test item</a> can be classified into one of the following three sets:</p> <ul style="list-style-type: none"> <li>• expected elements: The actually received element is expected by the <a href="#">test component</a>.</li> <li>• ignored elements: The actually received element may be received from the <a href="#">test item</a>, but if it is received, it will be ignored by the <a href="#">test component</a>.</li> <li>• forbidden elements: The actually received element is forbidden to be received from the <a href="#">test item</a>.</li> </ul> <p>The classification of received elements as member of one of the three sets helps calculating the <a href="#">verdict</a> by the arbitration specification of the executed expect response action. The classification itself does not prescribe which <a href="#">verdict</a> will be produced for the currently executed expect response action. It is the responsibility of the associated arbitration specification to derive a verdict from the received elements and their classification. For further details of the semantics of the default «ExpectResponseArbitrationSpecification», refer to the corresponding sub-section.</p> <p>Basically, only two sets are required to be explicitly stated, the third set is then derived from the complement set of the union of the other two sets. The decision, which set shall be derived by the complement set of the union of the other two sets is determined by the tag definition 'expectationKind'. In case of overlapping sets the following precedencies are given: forbidden elements &gt; ignored elements &gt; expected elements. The reason for this precedence is to reduce the possibility of 'false negative' results.</p> <p>In case of a Message extension, the expected response is defined by the Message's signature and its arguments, if any. If more than one response type is expected at the same point in time, the tag definition 'expectedElement' can be used to denote further expected responses in addition to the expected response denoted by the Message's argument. The eventual number of expected responses is the union of the Message with «ExpectResponseAction» applied, including its arguments, joined with the elements of the tag definition 'expectedElement'. If the signature of the Message is left empty, the expect response action accepts and consumes any kind of actual responses from the test item. In that case, the tag definition 'expectationKind' shall be set to 'implicitExcept' only. The effective set of expected elements is eventually determined by the complement set of the union of forbidden elements and ignored elements.</p> <p>In case of Trigger extension, the expected responses are the union of the MessageEvents obtained from the underlying Trigger and the expected responses yield by the expectedElement tag definition, if any. A Trigger with «ExpectResponseAction» that defines an AnyReceiveEvent accepts and consumes any kind of actual responses from the test item. In that case, the tag definition 'expectationKind' shall be set to 'implicitExcept' only. The effective set of expected elements is eventually determined by the complement set of the union of forbidden elements and ignored elements.</p> <p>The default <a href="#">arbitration specification</a> for the <a href="#">expect response action</a> is described by «<a href="#">ExpectResponseArbitrationSpecification</a>».</p>
-------------	--

Extension	<a href="#">Message, Trigger</a>
Super Class	<a href="#">AtomicProceduralElement</a>
Attributes	<p><code>expectationKind</code> : <code>ImplicitExpectationKind</code> [1] = <code>implicitForbid</code></p> <p>The expectation kind determines which of the three explicit sets in the context of an <code>ExpectResponseAction</code> is implicitly merged (union) with the complement set of the union of the other two sets. The following possibilities are:</p> <ul style="list-style-type: none"> <li>Forbidden elements are implicitly unified (<code>implicitForbid</code>): Any received element that does not belong to the set of expected or ignored elements will be unified with the explicit set of forbidden elements during test execution. This prevents (or reduces the likelihood of) 'false negatives'.</li> <li>Ignored elements are implicitly unified (<code>implicitIgnore</code>): Any received element that does not belong to the set of expected or forbidden elements will be unified with the explicit set of ignored elements during test execution. Care must be taken when going for this mechanism, since it is prone to 'false negative' results in case a forbidden element was forgotten to be explicitly defined in the corresponding set.</li> <li>Expected elements are implicitly unified (<code>implicitExpect</code>): Any received element that does not belong to the set of ignored or forbidden elements will be unified with the explicit set of expected elements during test execution. Care must be taken when going for this mechanism, since it is prone to 'false negative' results in case a forbidden element was forgotten to be explicitly defined in the corresponding set.</li> </ul>
Associations	<p><code>expectedElement</code> : <code>NamedElement</code> [*]</p> <p>A set of elements that are expected from the <a href="#">test item</a> during test execution. Depending on the <a href="#">expectationKind</a> for this «<a href="#">ExpectResponseAction</a>» this set might be implicitly joined with the complement set of union of the sets '<a href="#">forbiddenElement</a>' and '<a href="#">ignoredElement</a>'.</p> <hr/> <p><code>arbitrationSpecification</code> {redefines <code>arbitrationSpecification</code>} : <code>ExpectResponseArbitrationSpecification</code> [0..1]</p> <p>Refers to an expect response action arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of test action.</p> <hr/> <p><code>forbiddenElement</code> : <code>NamedElement</code> [*]</p> <p>A set of elements that are forbidden to be received from the <a href="#">test item</a> during test execution. Depending on the <a href="#">expectationKind</a> for this «<a href="#">ExpectResponseAction</a>» this set might be implicitly joined with the complement set of union of the sets '<a href="#">expectedElement</a>' and '<a href="#">ignoredElement</a>'.</p> <hr/> <p><code>ignoredElement</code> : <code>NamedElement</code> [*]</p> <p>A set of elements that are ignored when being received from the <a href="#">test item</a> during test execution. Depending on the <a href="#">expectationKind</a> for this «<a href="#">ExpectResponseAction</a>» this set might be implicitly joined with the complement set of union of the sets '<a href="#">expectedElement</a>' and '<a href="#">forbiddenElement</a>'.</p>
Constraints	<p>Type of elements for the explicit sets</p> <p>The tag definitions '<a href="#">forbiddenElement</a>', '<a href="#">expectedElement</a>' and '<a href="#">ignoredElement</a>' shall only contain instances of the following metaclasses: Message, Event, Signal, BehavioralFeature, Trigger, InstanceSpecification.</p> <hr/> <p>At least one response</p> <p><a href="#">DRTA02</a>: It is necessary that a <a href="#">expect response action</a> expects to receive at least one <a href="#">response</a>.</p> <hr/> <p>Enforced expectation kind 'implicitExcept'</p> <p>In the cases, when «<a href="#">ExpectResponseAction</a>» is applied to a Message in the context of an Interaction, and the Message's signature is left empty, or when «<a href="#">ExpectResponseAction</a>» is</p>

	applied to a Trigger that yields an AnyReceiveEvent, the ' <a href="#">expectationKind</a> ' of the « <a href="#">ExpectResponseAction</a> » shall be set to ' <a href="#">implicitExpect</a> '.
Change from UTP 1.2	« <a href="#">ExpectResponseAction</a> » has been newly introduced by UTP 2.

#### 8.5.3.4.5 SuggestVerdictAction

Description	<p><a href="#">SuggestVerdictAction</a>: A <a href="#">test action</a> that instructs the tester to suggest a particular <a href="#">procedural element verdict</a> to the <a href="#">arbitration specification</a> of the <a href="#">test case</a> for being taken into account in the final <a href="#">test case verdict</a>.</p> <p>Stereotype «<a href="#">SuggestVerdictAction</a>» extends <a href="#">InvocationAction</a> which allows for using a variety of metaclasses for application. However, there must be at least one argument <a href="#">InputPin</a> defined for the <a href="#">InvocationAction</a> of the predefined type <a href="#">verdict</a> or subclasses thereof.</p> <p>For example, a <a href="#">CallOperationAction</a> could be used to invoke a (not standardized, yet proprietary) arbiter-specific interface operation. Another possibility is to use <a href="#">SendObjectAction</a> without specifying the target Pin, which has the semantics of providing the <a href="#">Verdict</a> instance to the arbitrating facility of a test execution system without needing a dedicated Interface. However, during test execution the <a href="#">suggest verdict action</a> must be made executable. This may include manually writing the <a href="#">verdict</a> instance into a paper-based document.</p> <p>If used in an Interaction, the <a href="#">InvocationAction</a> that is stereotyped with «<a href="#">SuggestVerdictAction</a>» must be referenced from an <a href="#">ActionExecutionSpecification</a> that indirectly covers a <a href="#">Lifeline</a> that represents a <a href="#">test component</a> role in the underlying <a href="#">test configuration</a>. Indirectly means that the corresponding start and end OccurrenceSpecification of the <a href="#">ActionExecutionSpecification</a> cover the <a href="#">test component</a> lifeline.</p> <p>The default <a href="#">arbitration specification</a> for the <a href="#">suggest verdict action</a> is described by «<a href="#">SuggestVerdictArbitrationSpecification</a>».</p>
Graphical syntax	
Extension	<a href="#">InvocationAction</a>
Super Class	<a href="#">AtomicProceduralElement</a>
Associations	<pre>arbitrationSpecification {redefines arbitrationSpecification} : SuggestVerdictArbitrationSpecification [0..1]</pre> <p>Refers to a suggest verdict action arbitration specification that overrides the default and implicit arbitration specification if set. It redefines the Property <i>arbitrationSpecification</i> of test action.</p>
Constraints	<p>Type of Argument</p> <p>The type of the argument <a href="#">InputPin</a> must be the predefined <a href="#">verdict</a> type or a subtype thereof.</p>
Change from UTP 1.2	« <a href="#">SuggestVerdictAction</a> » has been newly introduced by UTP 2.

### 8.5.3.5 Enumeration Specifications

Name	Description	Enumeration literals
ImplicitExpectationKind	Determines which of the three received element sets in the context of an ExpectResponseAction is implicitly joined with the complement set of the union of the other two sets. The three sets of elements that are meaningful in the context of an « <a href="#">ExpectResponseAction</a> » are the expected elements, ignored element and forbidden elements. Two of these sets have to be stated explicitly in the context of an ExpectResponseAction, the third one is implicitly derived from the complement set of the union of the two explicit sets.	<a href="#">implicitForbid</a> Determines that the explicit set of forbidden elements is implicitly joined with the complement set of the union of the explicitly expected and ignored element sets.
		<a href="#">implicitIgnore</a> Determines that the explicit set of ignored elements is implicitly joined with by the complement set of the union of the explicitly expected and element sets.
		<a href="#">implicitExpect</a> Determines that the explicit set of expected elements is implicitly joined with the complement set of the union of the explicitly forbidden and ignored element sets.

## 8.6 Test Data

Testing is mainly about the exchange of [data](#) and the ability to compare actual [responses](#) and their payload received from the [test item](#) at test execution with the expected one stated in the [test case](#). Therefore, testers usually have to take at least two [data](#)-related concepts into account. First, the specification of [data](#), i.e., the known types and the [constraints](#) applied on these types for deriving [data](#) values that abide by these [constraints](#). Second, a flexible mechanism to specify [data](#) values and their allowed matching mechanisms for [test case](#) execution.

Data specification-related concepts are provided and further described by the concepts of the [Data Specifications](#) chapter.

Data value-related concepts are provided and further described by the concepts of the [Data Values](#) chapter.

### 8.6.1 Data Specifications

This section specifies the stereotypes to implement the [data specification](#) concepts introduced in section Test Data of the Conceptual Model.

#### 8.6.1.1 Data Specifications Overview

The diagram below shows abstract syntax of the [data specification](#) package.

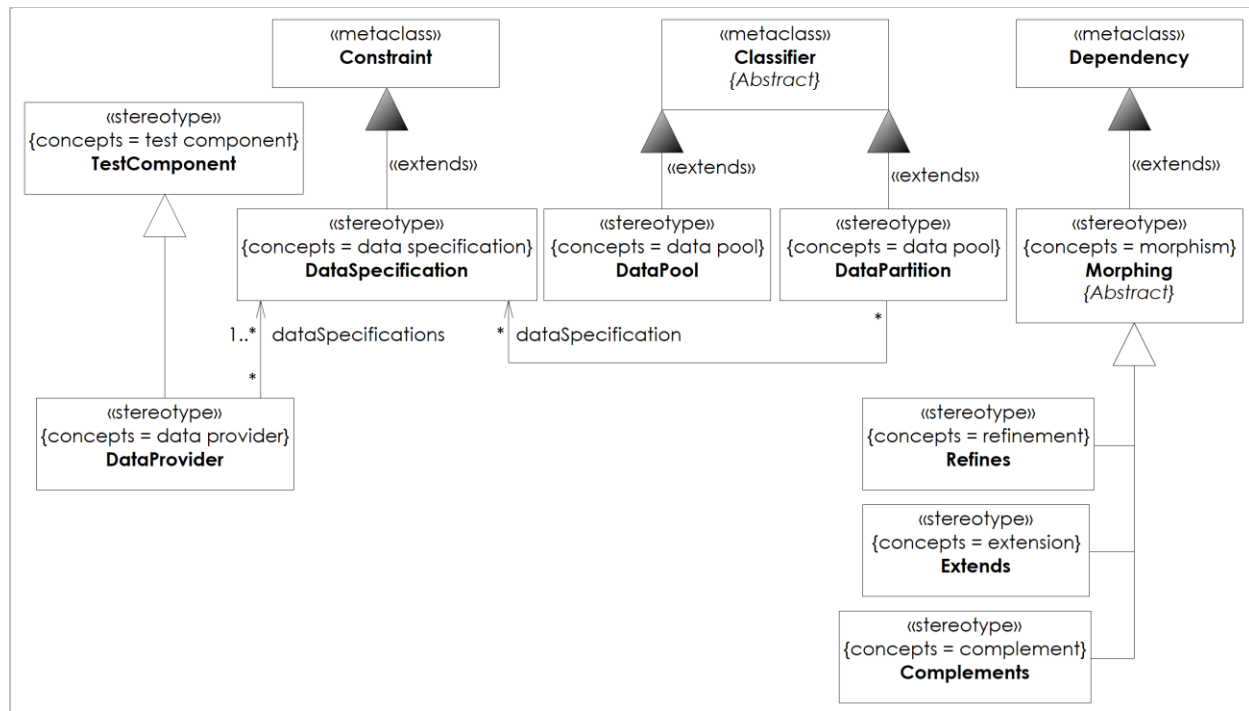


Figure 8.18 - Data Specifications Overview

## 8.6.1.2 Stereotype Specifications

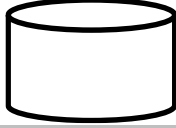
### 8.6.1.2.1 Complements

Description	<p><b>Complements:</b> A <a href="#">morphism</a> that inverts <a href="#">data</a> ) i.e., that replaces the <a href="#">data items</a> of a given set of <a href="#">data item</a>s by their opposites).</p> <p>The stereotype «<a href="#">Complements</a>» specializes the abstract stereotype «<a href="#">Morphing</a>» and logically negates the specification of the morphed <a href="#">data specification</a>s within the morphing <a href="#">data specification</a>. That means that <a href="#">complement morphism</a> result in a complementing <a href="#">data specification</a> that is the difference set of the complemented or morphed <a href="#">data specification</a>.</p>
Extension	<a href="#">Dependency</a>
Super Class	<a href="#">Morphing</a>
Change from UTP 1.2	«Complements» has been newly introduced by UTP 2.

### 8.6.1.2.2 DataPartition

Description	<p><b>DataPartition:</b> A role that some <a href="#">data</a> plays with respect to some other <a href="#">data</a> (usually being a subset of this other <a href="#">data</a>) with respect to some <a href="#">data specification</a>.</p> <p>The stereotype «<a href="#">DataPartition</a>» extends a UML Classifier and represents a set of <a href="#">data</a> that complies with one or more <a href="#">data specifications</a>.</p>
Extension	<a href="#">Classifier</a>
Associations	<code>dataSpecification : DataSpecification [*]</code>
Change from UTP 1.2	«DataPartition» has been newly introduced by UTP 2.

### 8.6.1.2.3 DataPool

Description	<p><b>DataPool</b>: Some <a href="#">data</a> that is an explicit or implicit composition of other <a href="#">data items</a>.</p> <p>The stereotype «<b>DataPool</b>» extends a UML Classifier and represents a set of physical <a href="#">data</a> without complying to any particular <a href="#">data specification</a>.</p>
Graphical syntax	
Extension	<b>Classifier</b>
Change from UTP 1.2	Changed from UTP 1.2. In UTP 1.2 « <b>DataPool</b> » extended both Classifier and Property.

### 8.6.1.2.4 DataProvider

Description	<p><b>DataProvider</b>: A <a href="#">test component</a> that is able to deliver (i.e., either select and/or generate) <a href="#">data</a> according to a <a href="#">data specification</a>.</p> <p>The stereotype «<b>DataProvider</b>» is a specialization of stereotype «<b>TestComponent</b>». Such a <a href="#">test component</a> is used to provide a <a href="#">data partition</a>, represented as a Classifier extended by the stereotype «<b>DataPartition</b>», by generating some new <a href="#">data</a> or by selecting some existing <a href="#">data</a> from another <a href="#">data partition</a> or a <a href="#">data pool</a> according to some <a href="#">data specifications</a> (represented as a Constraint extended by the stereotype «<b>DataSpecification</b>»).</p>
Extension	<b>Classifier, Property</b>
Super Class	<b>TestComponent</b>
Associations	<code>dataSpecifications : DataSpecification [1..*]</code>
Change from UTP 1.2	«DataProvider» has been newly introduced by UTP 2.

### 8.6.1.2.5 DataSpecification

Description	<p><b>DataSpecification</b>: A named <a href="#">boolean expression</a> composed of a <a href="#">data type</a> and a set of <a href="#">constraints</a> applicable to some <a href="#">data</a> in order to determine whether or not its <a href="#">data items</a> conform to this <a href="#">data specification</a>.</p> <p>The stereotype «<b>DataSpecification</b>» extends Constraint and is used to describe the <a href="#">constraints</a> within the context of one or more types, instances of those types have to comply with. <a href="#">DataSpecifications</a> are used to build and define <a href="#">DataPartitions</a>.</p> <p>Since «<b>DataSpecification</b>» is an <a href="#">extension</a> of Constraint the specification of the Constraint is defined by a ValueSpecification. This specification might be as simple as a LiteralString (e.g., natural language describing the <a href="#">constraint</a>) or as complex as a formal language statement (e.g., Alf or OCL). UTP does not prescribe the notation used for describing the specification of a «<b>DataSpecification</b>» Constraint.</p> <p>In case a Constraint with «<b>DataSpecification</b>» is directly contained in Classifier, it is considered semantically equivalent to «<b>DataSpecification</b>» Constraint defined outside of this Classifier and with a «<b>Refines</b>» Dependency established between the «<b>DataSpecification</b>» Constraint and the Classifier.</p>
Extension	<b>Constraint</b>
Constraints	<p>DataType in DataSpecification</p> <p><b>DRTD01</b>: It is necessary that each <a href="#">data specification</a> specifies at least one <a href="#">data type</a>.</p>
Change from UTP 1.2	«DataSpecification» has been newly introduced by UTP 2.



#### 8.6.1.2.6 Extends

Description	<p><b>Extends:</b> A <a href="#">morphism</a> that increases the amount of <a href="#">data</a> (i.e., that adds more <a href="#">data items</a> to a given set of <a href="#">data items</a>).</p> <p>The stereotype «<a href="#">Extends</a>» specialized the abstract stereotype «<a href="#">Morphing</a>» and logically OR-combines the specification of the morphed <a href="#">data specifications</a> within the morphing <a href="#">data specification</a>. That means that <a href="#">extension morphism</a> result in a <a href="#">data specification</a> that is more general than the extended or morphed <a href="#">data specifications</a>.</p>
Extension	<a href="#">Dependency</a>
Super Class	<a href="#">Morphing</a>
Change from UTP 1.2	«Extends» has been newly introduced by UTP 2.

#### 8.6.1.2.7 Morphing

Description	<p><b>Morphing:</b> A structure-preserving map from one mathematical structure to another.</p> <p>The abstract stereotype «<a href="#">Morphing</a>» extends <a href="#">Dependency</a> and is used to derive <a href="#">data specifications</a> from other <a href="#">data specifications</a>. This enables a high degree of reusability of existing <a href="#">data specifications</a>. «<a href="#">Morphing</a>» is intended to be subclassed and simply acts as a common superclass for shared semantics and constraints.</p> <p>A <a href="#">Dependency</a> stereotyped with a subclass of «<a href="#">Morphing</a>» always emanates from a <a href="#">Constraint</a> with «<a href="#">DataSpecification</a>» applied. It must point to a UML <a href="#">Classifier</a>, to a UML <a href="#">Package</a> containing some UML <a href="#">Classifiers</a>, or to a <a href="#">Constraint</a> with «<a href="#">DataSpecification</a>» applied. If it targets a «<a href="#">DataSpecification</a>» <a href="#">Constraint</a>, it morphs the definitions of that <a href="#">data specification</a> (called the morphed <a href="#">data specification</a>) into a new <a href="#">data specification</a> (called morphing <a href="#">data specification</a>). If it targets a <a href="#">Classifier</a> (or a set of <a href="#">Classifiers</a> contained in a <a href="#">Package</a>), all constraints applied on those <a href="#">Classifiers</a> or their attributes are considered as an implicit morphed <a href="#">data specification</a> attached to the <a href="#">Classifier</a> which is eventually morphed into a morphing <a href="#">data specification</a>.</p> <p>The exact effect of morphing a <a href="#">data specification</a> into another <a href="#">data specification</a> is defined by the concrete subclasses of the stereotype «<a href="#">Morphing</a>».</p>
Extension	<a href="#">Dependency</a>
Sub Class	<a href="#">Complements</a> , <a href="#">Extends</a> , <a href="#">Refines</a>
Constraints	<p>Clients of a «Morphing» Dependency</p> <p><a href="#">DRTD03</a>: As clients of a <a href="#">Dependency</a> stereotyped with a concrete substereotype of «<a href="#">Morphing</a>» only the following elements are allowed: <a href="#">Constraint</a> with «<a href="#">DataSpecification</a>» applied.</p> <p>Suppliers of a «Morphing» Dependency</p> <p><a href="#">DRTD04</a>: As suppliers of a <a href="#">Dependency</a> stereotyped with a concrete substereotype of «<a href="#">Morphing</a>» only the following elements are allowed: <a href="#">Constraint</a> with «<a href="#">DataSpecification</a>» applied, UML <a href="#">Classifier</a>, and UML <a href="#">Package</a>.</p>
Change from UTP 1.2	«Morphing» has been newly introduced by UTP 2.

#### 8.6.1.2.8 Refines

Description	<p><b>Refines:</b> A <a href="#">morphism</a> that decreases the amount of <a href="#">data</a> (i.e., that removes <a href="#">data items</a> from a given set of <a href="#">data items</a>).</p> <p>The stereotype «<a href="#">Refines</a>» specialized the abstract stereotype «<a href="#">Morphing</a>» and logically AND-combines the specification of the morphed <a href="#">data specifications</a> within the morphing <a href="#">data specification</a>. That means that <a href="#">refinement morphism</a> result in a <a href="#">data specification</a> that is more specific than the refined or morphed <a href="#">data specifications</a>.</p>
Extension	<a href="#">Dependency</a>
Super Class	<a href="#">Morphing</a>
Change from UTP 1.2	«Refines» has been newly introduced by UTP 2.

### 8.6.2 Data Values

The payload of an [expect response action](#) is also called expected [response](#) argument value as opposed to the actual [response](#) argument value. During [arbitration specification](#), usually a comparator evaluates whether the actual [response](#) matches with the expected ones in terms of event type and its payload. It is then the task of the [arbitration specification](#) to decide on the [verdict](#) that has to be assigned. In UTP [data](#) values are expressed by means of ValueSpecifications to specify both the payload for a [stimulus](#) and the payload of expected [responses](#). In case of an expected [response](#), the ValueSpecification does also implicitly define a matching mechanism used by a comparator during arbitration in order to evaluate whether the expected payload matches the actual payload.

The implicitly applied matching mechanism is determined by the ValueSpecification used to describe an expected payload argument in the context of an expected [response](#). The prescribed matching mechanisms semantics, inherently bound to ValueSpecifications, are defined by UTP as follows:

- ValueSpecification (abstract metaclass): In general, any native UML ValueSpecification infers an *equality matching mechanism*, i.e., the actual payload, also known as [response](#) argument value, must be exactly the same as the expected payload. Any deviation will result in a mismatch.
- LiteralInteger: Checks for equality of the expected and actual [response](#) Integer-typed argument value.
- LiteralString: Checks for equality of the expected and actual [response](#) String-typed argument value.
- LiteralReal: Checks for equality of the expected and actual [response](#) Real-typed argument value.
- LiteralBoolean: Checks for equality of the expected and actual [response](#) Boolean-typed argument value.
- LiteralUnlimitedNatural: Checks for equality of the expected and actual [response](#) Integer-typed argument value including infinity.
- LiteralNull: Checks for absence of an actual [response](#) argument value of any type.
- InstanceValue: Checks for equality of the expected and actual [response](#) complex [data type](#) instance argument value.

All these equality matching mechanisms are natively given by UML, whereas UTP adds just a few more ValueSpecifications that provide matching mechanisms currently not given by UML. These kinds of ValueSpecifications are sometimes called Wildcards (TTCN-3) or Facets (XML Schema):

- [AnyValue](#): Represents a set of all possible values for a given type and checks if actual [response](#) argument value is contained in this set. In case of optionality, the set of known values includes the absence of a value. This is implemented as stereotype «[AnyValue](#)».
- [RegularExpression](#): Represents a set of values for a given type described by a regular expression and checks if the actual [response](#) argument value belongs to that set. This is implemented as stereotype «[RegularExpression](#)».

#### 8.6.2.1 Data Value Extensions

The diagram below shows the abstract syntax of the ValueSpecification [extension](#)s introduced by UTP.

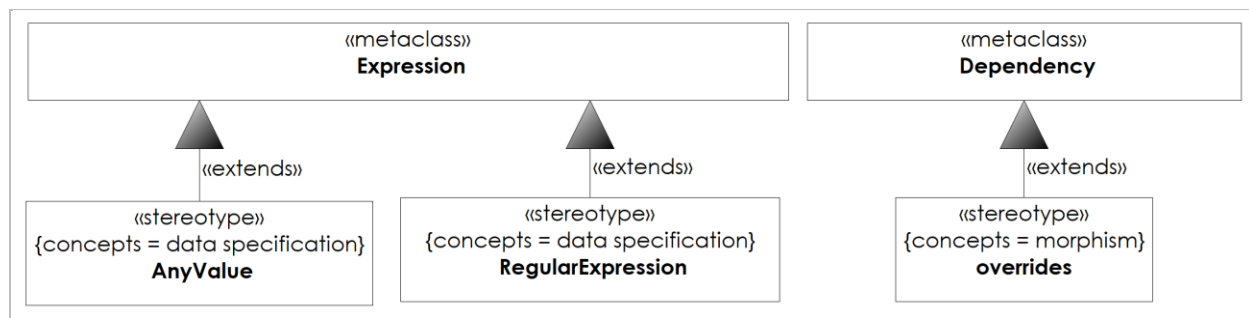


Figure 8.19 - Data Value Extensions

## 8.6.2.2 Stereotype Specifications

### 8.6.2.2.1 AnyValue

Description	The stereotype « <a href="#">AnyValue</a> » extends ValueSpecification and represents an implicit set of known values for a given type. The expected <a href="#">response</a> argument value matches with each actual <a href="#">response</a> argument value, as long as type-compliance is given. In case of optionality, the set of known values includes the absence of a value.
Extension	<a href="#">Expression</a>
Change from UTP 1.2	Changed and renamed from UTP 1.2. In UTP 1.2, « <a href="#">AnyValue</a> » was called «LiteralAny» and extended LiteralSpecification.

#### 8.6.2.2.2 overrides

Description	<p>Overrides is a relationship between at least two <b>InstanceSpecification</b>s, i.e., the modifying <b>InstanceSpecification</b> and the modified <b>InstanceSpecification</b>. Modifying <b>InstanceSpecification</b>s constitute the client elements of the underlying dependency, and consequently, modified <b>InstanceSpecification</b>s constitute the supplier elements of the underlying dependency.</p> <p>A modifying <b>InstanceSpecification</b> reuses all slot values of the modified <b>InstanceSpecification</b> in a way as if the slot values would have been copied into the modifying <b>InstanceSpecification</b> as its owned slots. Furthermore, the modifying <b>InstanceSpecification</b> is allowed to specify slots, which have not been declared by the modified <b>InstanceSpecification</b> at all. This enables user to gradually complete <b>InstanceSpecification</b>s and to reuse already or maybe partially defined <b>InstanceSpecification</b>s in order to create large sets of <a href="#">data</a> by avoiding redundancy.</p> <p>Additionally, a modifying <b>InstanceSpecification</b> is able to overwrite slots with new values. A slot is considered to be overwritten if a modifying <b>InstanceSpecification</b> defines an owned slot that refers to the very same defining feature as the owned slot of the modified <b>InstanceSpecification</b>, or to a feature that redefines, directly or transitively, the slot's defining feature. An overwriting slot's value list entirely replaces the value list of the slot that is overwritten.</p> <p>Modification requires type compatibility between the modifying and modified <b>InstanceSpecification</b>s. Type compatibility is given if a modifying <b>InstanceSpecification</b>'s classifier list is compatible with the modified <b>InstanceSpecification</b>'s classifier list. Two classifier lists are compatible if the modifying <b>InstanceSpecification</b>'s classifier list is a proper subset of the modified <b>InstanceSpecification</b>'s classifier list. A proper subset is considered to be given if each classifier of the modifying <b>InstanceSpecification</b>'s classifier list is type compatible with at least one classifier of the modified <b>InstanceSpecification</b> classifier list. Type compatibility between classifiers is defined in the UML specifications.</p> <p>Cyclic modifications are not allowed. A cyclic modification describes a situation in which a modifying <b>InstanceSpecification</b> establishes a modification to a modified <b>InstanceSpecification</b> and the latter one already modifies, directly or transitively, the modifying <b>InstanceSpecification</b>.</p>
Extension	<b>Dependency</b>
Constraints	<p>Restriction of client and supplier</p> <p>As client and supplier of the underlying <b>Dependency</b>, only <b>InstanceSpecification</b> are allowed.</p>
	<p>Cyclic modifications</p> <p>Cyclic override are not allowed. A cyclic override means that an overridden <b>InstanceSpecification</b> transitively overrides its overriding <b>InstanceSpecification</b>.</p>
Change from UTP 1.2	«overrides» was renamed by UTP 2. In UTP 1.2, it was named «modifies».

### 8.6.2.2.3 RegularExpression

Description	<p>The stereotype «<a href="#">RegularExpression</a>» extends Expression and represents an implicit set of values for a given type described by a regular expression. The expected <a href="#">response</a> argument value matches with each actual <a href="#">response</a> argument value if the actual one belongs to the set of values defined by the regular expression.</p> <p>A <a href="#">RegularExpression</a> can be used for test <a href="#">data</a> generation or to compare whether an actual <a href="#">response</a> matches with expected <a href="#">response</a>.</p> <p>The attribute <i>symbol</i> of the underlying Expression must contain the String that is evaluated as the regular expression. It might be omitted; in that case the <i>operands</i> of the underlying Expression must be used as abstract syntax tree for the regular expression.</p>
Extension	<a href="#">Expression</a>
Change from UTP 1.2	«RegularExpression» has been newly introduced by UTP 2.

## 8.7 Test Evaluation

The concepts for test evaluation are necessary to decide about the outcome of the dynamic test process activities. They implement in the specification of (proprietary) [arbitration specifications](#) on [test set](#), [test case](#) and [procedural element](#) level, as well as in the ability to incorporate the [test logs](#) produced during the execution of a test-specific [procedure](#) and its [procedural element](#) in a platform-independent, but user-specific way.

### 8.7.1 Arbitration Specifications

In dynamic testing, the term *Arbitration* describes the application of a certain rule set on the outcome of a test execution activity, usually captured as [test log](#) for comprehensibility, in order to derive the final [verdict](#) of an execution [test set](#) or [test case](#). Thus the arbitration of an executed [test set](#) or [test case](#) is the most important activity of the test evaluation activities with respect to requirements, [test requirement](#) or [test objective](#) coverage. Arbitration can both happen immediately during test execution (dynamic arbitration) and after test execution based on the captured [test logs](#) (post-execution arbitration). Due to whatever reason (organizational, technical etc.), one might be preferred over the other.

The UTP arbitration facility offers stereotypes for specifying proprietary [arbitration specifications](#) that vary from the default [arbitration specifications](#) in terms of their [verdict](#) calculation algorithm. Users can define user-specific [arbitration specifications](#) for [test sets](#), [test execution schedules](#), [test cases](#) and [procedural elements](#) by simply applying the stereotypes offered by the UTP arbitration facility to applicable metaclasses. The degree of formalism of a user-defined [arbitration specification](#) is left open. An [arbitration specification](#) might be represented by something as simple as an identifier (referring to an implementation), by natural language describing the arbitration rules, by any kind of UML Behavior or by something formal as executable specifications or mathematical definitions.

Arbitration specifications are usually implemented (or interpreted) by an arbiter component that belongs to the utilized test execution tool. UTP does not prescribe any implementation details of an arbiter component as part of a test execution tool, nor how or when information from [test sets](#), [test cases](#) and [procedural elements](#) are passed to an arbiter component.

It is left open if the arbitration activities are carried out automatically or by a human.

UTP introduces three different kinds of [verdicts](#) that can be produced:

- [procedural element verdicts](#): Verdicts produced by a [procedural element arbitration specification](#).
- [test case verdicts](#): Verdicts produced by a [test case arbitration specification](#).

- [test set verdicts](#): Verdicts produced by a [test set arbitration specification](#).

The fundamental [verdict](#) calculation and provisioning schema is as follows:

- [test set arbitration specification](#)s: they derive the [test set verdict](#) from the [test case verdicts](#) that have been executed as part of the [test set](#) (i.e., the [test case verdicts](#) are passed to the [arbitration specification](#) of the surrounding [test set](#)).
- [test case arbitration specification](#)s: they derive the [test case verdicts](#) from the [procedural element verdicts](#) (first and foremost the [test action verdicts](#)) that have been executed as part of the [test case](#) (i.e., the [procedural element verdicts](#) are assembled and passed on to the [test case arbitration specification](#)).
- [procedural element arbitration specifications](#): they derive [procedural element verdicts](#) from the information conveyed by the [procedural element](#), or in case of a [compound procedural element](#), the [procedural element verdicts](#) received from the [arbitration specifications](#) of the contained [procedural elements](#).

### 8.7.1.1 Test Procedure Arbitration Specifications

In dynamic testing, the term arbitration describes the process of calculating a [verdict](#) (also known as test result) for an executed [test case](#) based on so called pass/fail criteria. Arbitration is the most important activity of a dynamic test process with respect to calculating the coverage of requirements, [test requirements](#) or [test objectives](#). Arbitration includes the process of comparing the expected and actual behavior of a [test item](#) to detect any deviation from the expectation. Often, the comparison process is used as a synonym for arbitration, but this is not correct. Comparison is one part of arbitration that simply decides whether an expectation was met, or not. The outcome is usually a Boolean value true or false. This Boolean value is then passed to the pass/fail criteria that is responsible to derive a [verdict](#) (the test result) for a [test case](#). When the calculation is done automatically, the component that implements the pass/fail criteria is usually referred to as the arbiter. In UTP 2, the pass/fail criteria are formalized as [arbitration specifications](#).

In UTP 2, several [artifacts](#) may deliver a [verdict](#). Such an [artifact](#) is called an arbitration target. UTP 2 distinguished the following arbitration targets: [procedural element](#), [test case](#), [test execution schedule](#) and [test set](#). These arbitration targets define a hierarchy of arbitration scopes. Verdicts produced by arbitration targets on lower scopes are passed to higher scopes, more precisely, passed to the [arbitration specification](#) of higher scope arbitration targets. The arbitration scopes of UTP 2 are subsequently described in ascending hierarchy (i.e., starting with the lowest arbitration scope):

- Procedural element scope: Every [procedural element](#) is (potentially implicitly) associated with a [procedural element arbitration specification](#) that produces a corresponding [procedural element verdict](#). Procedural elements [verdicts](#) might be atomic, i.e., they cannot be further decomposed, in case the [verdicts](#) have been produced by an [atomic procedural element arbitration specification](#). It might also consist of other [procedural element verdicts](#) in case the [verdict](#) has been produced by a [compound procedural element verdict](#).
- Test case scope: Every [test case](#) is (potentially implicitly) associated with a [test case arbitration specification](#) that produces the corresponding [test case verdict](#) for that [test case](#). A [test case verdict](#) is usually calculated by evaluating the [procedural element verdicts](#) it received from its [procedural element](#) arbitration scope.
- Test execution schedule/[test set](#) scope: Every [test set](#) is (potentially implicitly) associated with a [test set arbitration specification](#) that produces the corresponding [test set verdict](#) for that [test set](#). A [test set verdict](#) is usually calculated by evaluating the [test case verdicts](#) it received from its [test case](#) arbitration scope.

For each specific arbitration target (e.g., a [test case](#), an [expect response action](#)) a corresponding [arbitration specification](#) exist (e.g. [test case arbitration specification](#), [expect response action arbitration specification](#)). This strong typing of [arbitration specifications](#) results in a high number of stereotypes, but prevents confusion, manual errors and ambiguities when binding [arbitration specifications](#) to arbitration targets. UTP 2 defines default [arbitration specifications](#) for every specific arbitration target. If the default [arbitration specification](#) is sufficient in a given context, the user does not have to care about defining specific [arbitration specifications](#) at all. However, in some situations it might be required to define proprietary [arbitration specifications](#), [verdicts](#) or [verdict](#) precedence rules. Users can define user-specific [arbitration specifications](#) for any arbitration target. The degree of formalism of a user-defined [arbitration specification](#) is left open. An [arbitration specification](#) might be represented by something as simple as an identifier (referring to an implementation), by natural language describing the arbitration rules, by any kind of UML Behavior or by something formal as executable specifications or mathematical definitions. The



semantics of the default [arbitration specification](#) is explained in the stereotype specification of «[ArbitrationSpecification](#)» and its subclasses.

Arbitration can both happen immediately during test execution (dynamic arbitration) and after test execution based on the captured [test logs](#) (post-execution arbitration). Due to whatever reason (organizational, technical etc.), one might be preferred to the other.

Arbitration specifications are usually implemented (or interpreted) by an arbiter that belongs to the utilized test execution tool. UTP 2 does not prescribe any implementation details of an arbiter, nor how or when information from [test sets](#), [test cases](#) and [procedural elements](#) are passed to an arbiter component.

It is left open if the arbitration process is carried out automatically or by a human being.

### Arbitration Specification Binding

In case it is needed, users can override the default [arbitration specifications](#) of arbitration targets with proprietary ones. The UTP 2 arbitration facility offers a binding mechanism to empower user to integrate their own [arbitration specification](#) into the arbitration facility. The binding mechanism was designed to provide a high degree of flexibility, i.e., the user can override [arbitration specifications](#) of single arbitration targets without the need to care about all the other arbitration targets.

A binding connects an arbitration target with a corresponding, matching [arbitration specification](#). The matching mechanism for arbitration targets and [arbitration specifications](#) are explained in the stereotype specifications of «[ArbitrationSpecificationBinding](#)» and the concrete sub-classes of «[ArbitrationSpecification](#)».

Due to the hierarchical nature of arbitration targets (e.g. an arbitration target «[TestCase](#)» may consist of arbitration targets «[ExpectResponseAction](#)») the binding of an [arbitration specification](#) to an arbitration target may also be defined on different hierarchical scopes. UTP 2 defines the following evaluation scopes where bindings can be defined for a given arbitration target:

- Global scope: The topmost scope; each binding on global scope applies to all matching arbitration targets on lower scopes if not overridden in a cascading manner.
- Test Set scope: The first scope with a concrete arbitration targets; bindings on that scope apply to all matching arbitration targets on lower scopes if not overridden in a cascading manner.
- Test Case scope: The scope of a single [test case](#); bindings on that scope apply only to the [procedural elements](#) of the given [test case](#), if not overridden in a cascading manner.
- Procedural Element scope: The lowest scope; a [procedural element](#) binding [overrides](#) any binding that was defined for that [procedural element](#) on any higher scope.

Bindings that are declared on lower scopes override bindings on higher scopes in a cascading manner. When an arbitration process is carried out eventually, the arbiter will first calculate the set of effective [arbitration specifications](#) for a given arbitration target and then execute the rules of that [arbitration specification](#) target to produce a [verdict](#). The cascading nature of the [arbitration specification](#) bindings allows for specifying multiple bindings for a given arbitration target on different binding scopes, but only one (i.e. the most downwards one) will be the effective [arbitration specification](#) for that arbitration target.

Let us assume we have a [test set](#) 'TS\_1' defined that consists of two [test cases](#) 'TC\_1' and 'TC\_2'. A [test set arbitration specification](#) binding exists that defines a target-free binding on [test set](#) scope that binds explicit [expect response arbitration specification](#) 'ERAS\_1' to the [test set](#) 'TS\_1'. Let's further assume that there is a sub-scope [test case arbitration specification](#) binding (i.e., 'TCASB\_1' with arbitration target 'TC\_1') defined for the [test set arbitration specification](#) binding. TCASB\_1 further defines that for all [expect response actions](#) of that evaluation scope, the [expect response arbitration specification](#) 'ERAS\_2' shall be bound. If the arbitration process for the [test set](#) 'TS\_1' is started, 'ERAS\_2' will be bound to all [expect response actions](#) (i.e. all [expect response actions](#) of [test case](#) 'TC\_1') in the scope of 'TCASB\_1', whereas 'ERAS\_1' will be used for all remaining [expect response actions](#) (i.e. of [test case](#) 'TC\_2').

### Arbitration Directives

Arbitration bindings simply declare that arbitration targets and [arbitration specifications](#) refer to each other. These relationships need to be put into effect at a certain point of time. It is therefore required to explicitly define which binding should be taken into effect. The UTP 2 arbitration facility leverages the UTP 2 test directive facility for that purpose. An arbitration directive collects a set of [arbitration specification](#) bindings and ties them to an input for which these bindings shall be put into effect. The input is usually a [test log](#) on which the arbitration process shall operate (i.e., post-execution arbitration) or a [test set](#), a [test execution schedule](#) or a set of [test cases](#) for which the bindings shall be put into effect (i.e., dynamic arbitration). The arbitration directive be the configuration of an arbiter

for [verdict](#) calculation.

### Deprecation of the former binding mechanism

Before UTP 2.2 each arbitration target (i.e., [test set](#), [test execution schedule](#), [test case](#) and [procedural element](#) as well as its subclasses) had a direct association to their matching [arbitration specifications](#). This association is for the sake of backward compatibility technically still present but marked as deprecated and should not be used anymore. The newly introduced cascading [arbitration specification](#) facility is more flexible and less intrusive than the former one.

#### 8.7.1.1.1 Arbitration Facility Overview

The following figure shows the foundations of the arbitration facility of UTP. It illustrates how arbitration specifications and arbitration targets relate to each other using the arbitration specification binding mechanism. An arbitration specification produce arbitration results. The most important, yet not the sole information conveyed by an arbitration result is the verdict. Due to the design of the stereotype «[ArbitrationResult](#)» it is easily possible to incorporate further, yet proprietary information into the «ArbitrationResult» using UML's native InstanceSpecification mechanism.

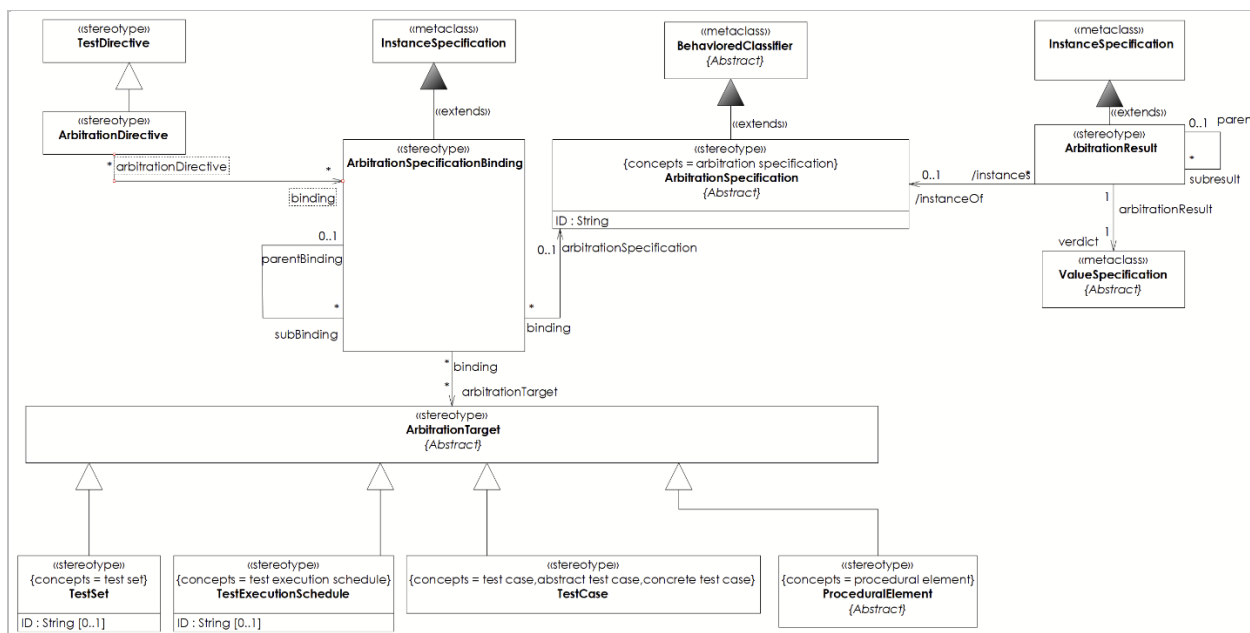


Figure 8.20 - Arbitration Specifications Overview

#### 8.7.1.1.2 Stereotype Specifications

##### 8.7.1.1.2.1 ArbitrationDirective

Description	An arbitration directive instructs a human tester or a machine to calculate <a href="#">verdicts</a> (represented by the stereotype « <a href="#">ArbitrationResult</a> »).
Super Class	<a href="#">TestDirective</a>
Associations	binding : ArbitrationSpecificationBinding [*] {unique} The set of arbitration specification bindings that are defined in the scope of the arbitration directive.
Change from UTP 1.2	«ArbitrationDirective» was newly introduced by UTP 2.2.




### 8.7.1.1.2.2 ArbitrationResult

Description	<p>An «<a href="#">ArbitrationResult</a>» stores information about the execution and the <a href="#">verdict</a> produced by an <a href="#">arbitration specification</a> that was put into effect by an arbiter . Arbitration results can be calculated for any arbitration target. The characteristics of an «<a href="#">ArbitrationResult</a>» is determined by the «<a href="#">ArbitrationSpecification</a>» of which the «<a href="#">ArbitrationResult</a>» represents an instance of.</p> <p>The most important information an <a href="#">arbitration specification</a> produces is the eventual <a href="#">verdict</a>. Other helpful, but not standardized information may include the timestamp of the execution, characteristics of the arbiter (e.g. the human being or implementation) that produced the result, the outcome of the comparison process of actual and expected value including deviation details in case of mismatches, etc. Additional information can be incorporated by using the ordinary underlying UML <a href="#">InstanceSpecification</a> mechanism.</p> <p>An «<a href="#">ArbitrationResult</a>» may point to the corresponding «<a href="#">TestLogElement</a>» that provides the actual information captured during test execution. The expected information is specified by the corresponding arbitration target linked by the <a href="#">arbitration specification</a> the arbitration result is an instance of. All information that were involved in calculating the <a href="#">verdict</a> are therefore accessible for analysis or understanding.</p> <p>«<a href="#">ArbitrationResult</a>»s may link with other «<a href="#">ArbitrationResult</a>»s. An arbitration result of a <a href="#">test set</a> is usually calculated by the arbitration result of the executed <a href="#">test cases</a>, which, in turn, are calculated by the arbitration result of the executed <a href="#">procedural elements</a>. The tag definitions 'subresults' and 'parent' of «<a href="#">ArbitrationResult</a>» enable keeping depending «<a href="#">ArbitrationResult</a>» closely connected to one another.</p>
Extension	<a href="#">InstanceSpecification</a>
Associations	<p><a href="#">verdict</a> : <a href="#">ValueSpecification</a></p> <p>The <a href="#">verdict</a> that was produced for a given <a href="#">test case</a>, <a href="#">test set</a> or <a href="#">procedural element</a> according to the respective bound <a href="#">arbitration specification</a> and the actual information captured in the corresponding <a href="#">test log</a>.</p> <p><a href="#">/instanceOf</a> : <a href="#">ArbitrationSpecification</a> [0..1]</p> <p>The <a href="#">arbitration specification</a> whose rules were used to produce the <a href="#">test set</a>. The <a href="#">arbitration specification</a> is derived from the underlying <a href="#">InstanceSpecification</a>'s set of <a href="#">Classifiers</a> with «<a href="#">ArbitrationSpecification</a>» applied or specializations thereof. There can be more than one <a href="#">Classifier</a> set for an «<a href="#">ArbitrationResult</a>» <a href="#">InstanceSpecification</a>, but only one of these <a href="#">Classifiers</a> are allowed to be stereotyped with «<a href="#">ArbitrationSpecification</a>» or a specialization thereof.</p> <p><a href="#">resultFor</a> : <a href="#">TestLogElement</a> [0..1]</p> <p>The corresponding test log element for which the given «<a href="#">ArbitrationResult</a>» captures the calculated <a href="#">test set</a> and any other relevant information.</p> <p><a href="#">subresult</a> : <a href="#">ArbitrationResult</a> [*]</p> <p>A set of linked «<a href="#">ArbitrationResult</a>»s that influenced the calculation of the current <a href="#">verdict</a>.</p> <p>In case of a compound <a href="#">procedural element</a>, it is possible (not mandatory, though) to link all the «<a href="#">ArbitrationResult</a>»s produced for the <a href="#">procedural element</a>s contained by the compound procedural element.</p> <p><a href="#">parent</a> : <a href="#">ArbitrationResult</a> [0..1]</p> <p>The superior «<a href="#">ArbitrationResult</a>» the current «<a href="#">ArbitrationResult</a>» has an impact on.</p>
Constraints	<p>Type of verdict <a href="#">ValueSpecification</a></p> <p>The type of the <a href="#">ValueSpecification</a> referenced by the tag definition <a href="#">verdict</a> must be</p>

	of type <a href="#">verdict</a> (or a subtype thereof) as defined in the UTP Types Library.
Change from UTP 1.2	« <a href="#">ArbitrationResult</a> » has been newly introduced by UTP 2.

#### 8.7.1.1.2.3 ArbitrationSpecification

Description	<p><a href="#">ArbitrationSpecification</a>: A set of rules that calculates the eventual <a href="#">verdict</a> of an executed <a href="#">test case</a>, <a href="#">test set</a> or <a href="#">procedural element</a>.</p> <p>The stereotype «<a href="#">ArbitrationSpecification</a>» extends <b>BehavioredClassifier</b> and is used to specify the decision process for <a href="#">verdicts</a>. It is an abstract stereotype that is specialized by stereotypes that deal with the <a href="#">verdicts</a> of <a href="#">test sets</a>, <a href="#">test cases</a>, and <a href="#">procedural elements</a> (i.e., <a href="#">test set verdicts</a>, <a href="#">test case verdicts</a>, and <a href="#">procedural element verdicts</a>).</p> <p>The concept of an <a href="#">arbitration specification</a> allows for specifying user-defined algorithms for the calculation of the <a href="#">verdict</a> based on the executed <a href="#">test cases</a> or the captured <a href="#">test case logs</a>.</p> <p>The semantics of the default <a href="#">arbitration specification</a> defines a default precedence of the predefined instances, which is: <a href="#">None</a> &lt; <a href="#">Pass</a> &lt; <a href="#">Inconclusive</a> &lt; <a href="#">Fail</a> &lt; <a href="#">Error</a>.</p> <p>That means that <a href="#">verdicts</a> with lower precedence can be overwritten with <a href="#">verdicts</a> of higher precedence, but not vice versa.</p> <p>Other default <a href="#">arbitration specifications</a> defined by UTP adhere by that precedence rule defined by «<a href="#">ArbitrationSpecification</a>» and <a href="#">complement</a> it with their specific semantics. User-defined <a href="#">arbitration specifications</a> may override that default semantics as well as the precedence of <a href="#">verdicts</a>.</p> <p>The result of an <a href="#">arbitration specification</a> is stored in an «<a href="#">ArbitrationResult</a>» that contains the eventual <a href="#">verdict</a> and links the «<a href="#">ArbitrationSpecification</a>» to the element it was applied to.</p>
Graphical syntax	
Extension	<b>BehavioredClassifier</b>
Sub Class	<a href="#">ProceduralElementArbitrationSpecification</a> , <a href="#">TestCaseArbitrationSpecification</a> , <a href="#">TestSetArbitrationSpecification</a>
Attributes	<p>ID : String [1]</p> <p>A unique identifier that unambiguously identifies the given <a href="#">arbitration specification</a>.</p>
Constraints	<p>Verdict of ArbitrationSpecification</p> <p><b>DRAS01</b>: It is necessary that an <a href="#">arbitration specification</a> determines exactly one <a href="#">verdict</a>.</p>
Change from UTP 1.2	« <a href="#">ArbitrationSpecification</a> » has been newly introduced into UTP 2.

#### 8.7.1.1.2.4 ArbitrationSpecificationBinding

Description	<p>«ArbitrationSpecificationBinding» binds <a href="#">arbitration specification</a>s to suitable arbitration targets.</p> <p>A bound <a href="#">arbitration specification</a> is responsible to calculate the arbitration results (i.e., the <a href="#">verdict</a>) for a given arbitration target. The given arbitration target defines the scope where the binding and potential sub-bindings become valid.</p> <p>Every <a href="#">arbitration specification</a> binding on the very same evaluation scope shall refer to unique arbitration targets. If not, it is ambiguous which binding shall become valid during the evaluation process. Let us assume there are two <a href="#">test set</a> bindings defined in the same binding scope and both refer to the <a href="#">test set</a> 'TS_1' as arbitration target but different <a href="#">test set arbitration specifications</a>, it is unclear which <a href="#">arbitration specification</a> should be utilized for the arbitration process.</p> <p>If there are multiple matching arbitration targets defined by the binding, the corresponding <a href="#">arbitration specification</a> is bound to all mentioned arbitration targets.</p> <p>The arbitration target might be omitted in a binding. These kinds of bindings are called target-free bindings. They apply not only to the explicitly mentioned arbitration targets, but to all matching arbitration targets in the current evaluation scope (and, if not overridden also in lower evaluation scopes). In that case, the bound <a href="#">arbitration specification</a> is applied to all matching arbitration targets within the given and lower evaluation scopes. Let us assume there is a <a href="#">test case arbitration specification</a> binding that binds the <a href="#">test case arbitration specification</a> 'TCAS_1' to an empty arbitration target. This empty arbitration target is evaluated at processing time to all possible <a href="#">test cases</a> that exist in the current evaluation scope. Let us further assume that there is another <a href="#">test case arbitration specification</a> binding that binds the <a href="#">test case arbitration specification</a> 'TCAS_2' to an explicit arbitration target, i.e., <a href="#">test case</a> 'TC_5'. If both bindings are defined in the same evaluation scope, for all <a href="#">test cases</a> but 'TC_5' the <a href="#">test case arbitration specification</a> 'TCAS_1' becomes valid. For 'TC_5' the <a href="#">test case arbitration specification</a> 'TCAS_2' becomes valid.</p> <p>There should only be one target-free binding of the same type defined on the same evaluation scope. Otherwise, it is ambiguous which binding shall become valid during evaluation process. Let us assume there are two target-free <a href="#">test case</a> bindings in the scope of the same <a href="#">test set</a> binding. Since both bindings would apply concurrently to all <a href="#">test cases</a>, it is not clear which binding should eventually be evaluated during the evaluation process.</p> <p>The bound <a href="#">arbitration specification</a> might also be omitted. In that case, the (implicitly defined) default <a href="#">arbitration specification</a> for the given arbitration target will be bound. This construct is helpful in situations where a binding on a higher evaluation scope is defined, but the default <a href="#">arbitration specification</a> shall be applied nonetheless for an arbitration</p>
-------------	--

	<p>target.</p> <p>Arbitration specification bindings have a cascading nature, i.e., sub-scope arbitration specification bindings override bindings on a higher evaluation scope. Target-free bindings are not allowed to define sub-bindings. The reason for this restriction is comprehensibility.</p> <p>Arbitration specification bindings must only bind matching arbitration targets and <a href="#">arbitration specifications</a>. The type of the binding is determined by the characteristics of the arbitration target and the <a href="#">arbitration specification</a>. For example, if an arbitration specification binding binds a <a href="#">test case arbitration specification</a> to a <a href="#">test case</a> (i.e., the arbitration target) the binding is called <i>test case arbitration specification binding</i>.</p>
Extension	<b>InstanceSpecification</b>
Associations	<p>subBinding : ArbitrationSpecificationBinding [*]{unique}</p> <p>The set of arbitration specification bindings on a lower evaluation scope that this binding specification binding refers to as sub-bindings.</p>
	<p>parentBinding : ArbitrationSpecificationBinding[*]{unique}</p> <p>The set of arbitration specification bindings on a higher evaluation scope that refer to this arbitration specification binding as sub-binding.</p>
	<p>arbitrationTarget : ArbitrationTarget [*] {unique}</p> <p>The set of arbitration targets to which referenced <a href="#">arbitration specification</a> will be bound to. If left empty, it means that the bound <a href="#">arbitration specification</a> will be bound implicitly to all matching arbitration targets within the scope of the binding.</p>
	<p>arbitrationSpecification : ArbitrationSpecification [0..1]</p> <p>The <a href="#">arbitration specification</a> that is eventually bound to the arbitration target. If it is left empty, the default <a href="#">arbitration specification</a> for the corresponding arbitration target is set implicitly.</p>
Constraints	Verdict of ArbitrationSpecification
	<b><u>DRAS01: It is necessary that an <a href="#">arbitration specification</a> determines exactly one verdict.</u></b>
	Target-free bindings shall not define sub-scope bindings
	If the arbitration target is omitted, no sub-bindings shall be defined.
	Uniqueness of arbitration targets on same evaluation scope
	Any two arbitration specification bindings on the very same evaluation scope shall refer to different arbitration targets.

	Uniqueness of target-free bindings on same arbitration scope  Any two target-free arbitration specification bindings on the very same evaluation scope shall be of different type.
Change from UTP 1.2	«ArbitrationSpecificationBinding» was newly introduced by UTP 2.2.

#### 8.7.1.1.2.5 ArbitrationTarget

Description	An arbitration target is any element that may produce a <a href="#">verdict</a> , i.e. an arbitration result. It is an abstract metaclass that simply marks certain elements as arbitration targets, i.e., test set, test case, test execution schedule and <a href="#">procedural element</a> as well as any of its subclasses.
Sub Class	<a href="#">ProceduralElement</a> , <a href="#">TestCase</a> , <a href="#">TestExecutionSchedule</a> , <a href="#">TestSet</a>
Change from UTP 1.2	« <a href="#">ArbitrationTarget</a> » was newly introduced by UTP 2.2.

#### 8.7.1.1.2.6 TestCaseArbitrationSpecification

Description	<p><a href="#">TestCaseArbitrationSpecification</a>: A set of rules that calculates the eventual <a href="#">verdict</a> of an executed <a href="#">test case</a>, test set or procedural element.</p> <p>A «<a href="#">TestCaseArbitrationSpecification</a>» specifies the rules for the eventual calculation of a <a href="#">test case verdict</a> based on the <a href="#">procedural element verdicts</a> that have been executed in the context of the corresponding <a href="#">test case</a>.</p> <p>The semantics of the default «TestCaseArbitrationSpecification» complements the semantics of «ArbitrationSpecification» by defining the rule that determines the assignment of test case verdicts. The rule of the default test case arbitration specification is as follows:</p> <ul style="list-style-type: none"> <li>• None: The verdict 'None' is assigned when the test case was not yet executed, or no other procedural element verdict was produced yet.</li> <li>• Pass: The verdict 'Pass' is assigned, if all procedural elements that participate in the arbitration process of that specific test case evaluate to 'Pass'.</li> <li>• Inconclusive: The verdict 'Inconclusive' is assigned, if at least one procedural element that participates in the arbitration process of that test case, evaluates to 'Inconclusive', while the remaining procedural elements evaluate to 'Pass' or 'None'.</li> <li>• Fail: The verdict 'Fail' is assigned, if at least one procedural element that participates in the arbitration process of that test case evaluates to 'Fail', while the remaining procedural elements evaluate to 'Inconclusive', 'Pass' or 'None'.</li> <li>• Error: The verdict 'Error' is assigned, if at least one procedural element that participates in the arbitration process of that test case evaluates to 'Error', or the arbitration process itself failed with a technical error.</li> </ul>
Extension	<a href="#">BehavioredClassifier</a>
Super Class	<a href="#">ArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.1.2.7 TestSetArbitrationSpecification

UMLTP22-10:	<p><a href="#">TestSetArbitrationSpecification</a>: A set of rules that calculates the eventual <a href="#">verdict</a> of an executed <a href="#">test case</a>, test set or procedural element.</p> <p>A «TestSetArbitrationSpecification» specifies the rules of how a test set verdict will be calculated based on the verdicts of the test cases that have been executed in the context of the corresponding test set. A <a href="#">test set arbitration specification</a> is used by both «<a href="#">TestSet</a>» and «<a href="#">TestExecutionSchedule</a>».</p> <p>The semantics of the default «TestSetArbitrationSpecification» complements the semantics of «ArbitrationSpecification» by defining the rule that determines the assignment of test set verdicts. The rule of the default test set arbitration specification is as follows:</p> <ul style="list-style-type: none"><li>• None: The verdict 'None' is assigned when the test set was not yet executed, i.e., any test case assembled or contained in the test set had produced a test case verdict yet.</li><li>• Pass: The verdict 'Pass' is assigned, if all executed test cases that participate in the arbitration process of that specific test set also evaluated to 'Pass'.</li><li>• Inconclusive: The verdict 'Inconclusive' is assigned, if at least one executed test case that participates in the arbitration process of that test set evaluates to 'Inconclusive', while the remaining test cases evaluate to 'Pass' or 'None'.</li><li>• Fail: The verdict 'Fail' is assigned, if at least one executed test case that participates in the arbitration process of that test set evaluates to 'Fail', while the remaining test cases evaluate to 'Inconclusive', 'Pass' or 'None'.</li><li>• Error: The verdict 'Error' is assigned, if at least one executed test case that participates in the arbitration process of that test set evaluates to 'Error', or the arbitration process itself failed with a technical error.</li></ul>
Extension	<a href="#">BehavioredClassifier</a>
Super Class	<a href="#">ArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2 Procedural Element Arbitration Specifications

The [procedural element arbitration specification](#) sections summarize the different type of [arbitration specifications](#) that can be used to define proprietary [procedural element arbitration specifications](#).

##### 8.7.1.2.1 Arbitration of AtomicProceduralElements

The diagram below shows the abstract syntax of [arbitration specification](#) elements for [atomic procedural elements](#).

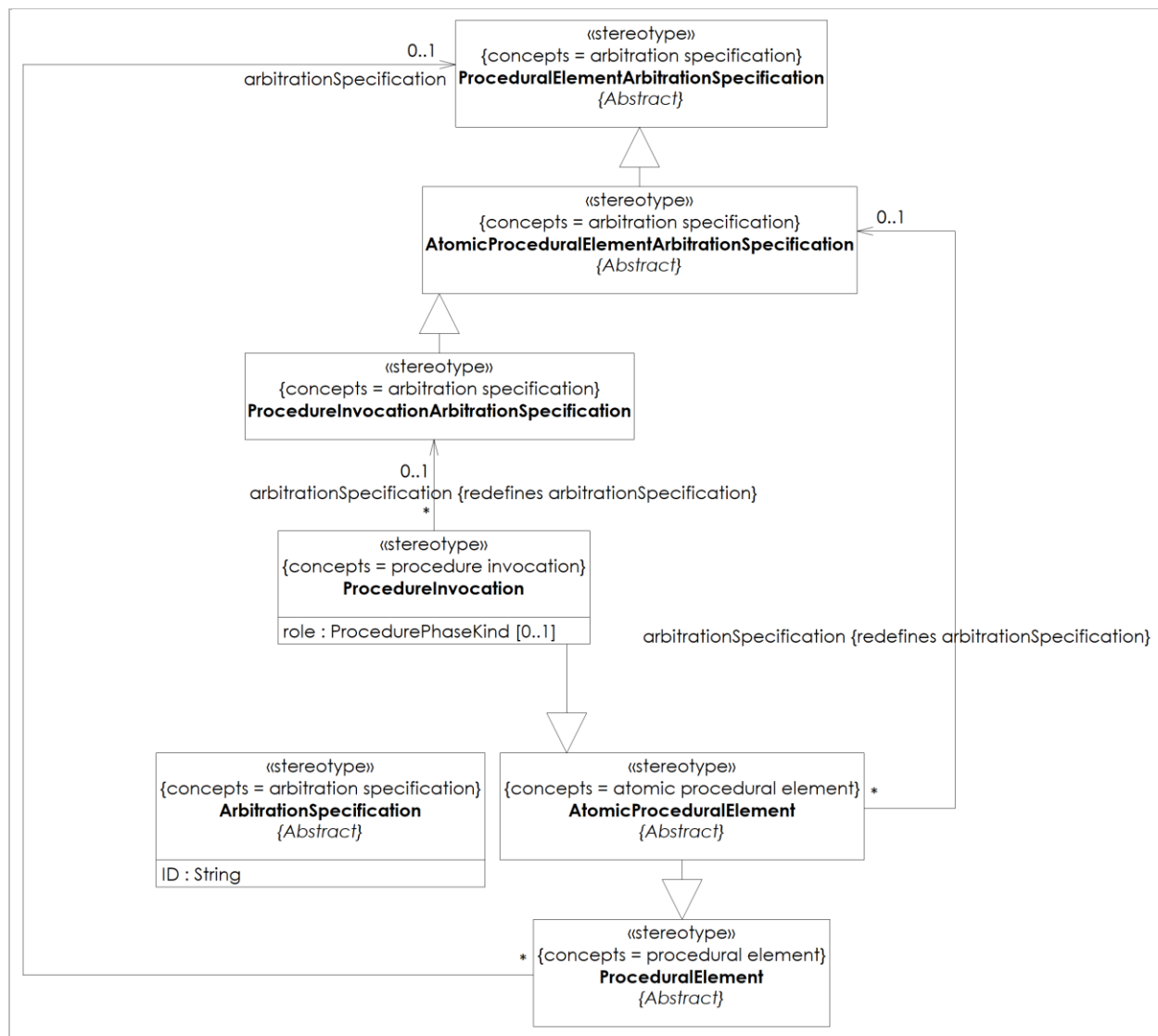


Figure 8.21 - Arbitration of AtomicProceduralElements

#### 8.7.1.2.2 Arbitration of CompoundProceduralElements

The diagram below shows the abstract syntax of [arbitration specification](#) elements for [compound procedural elements](#).

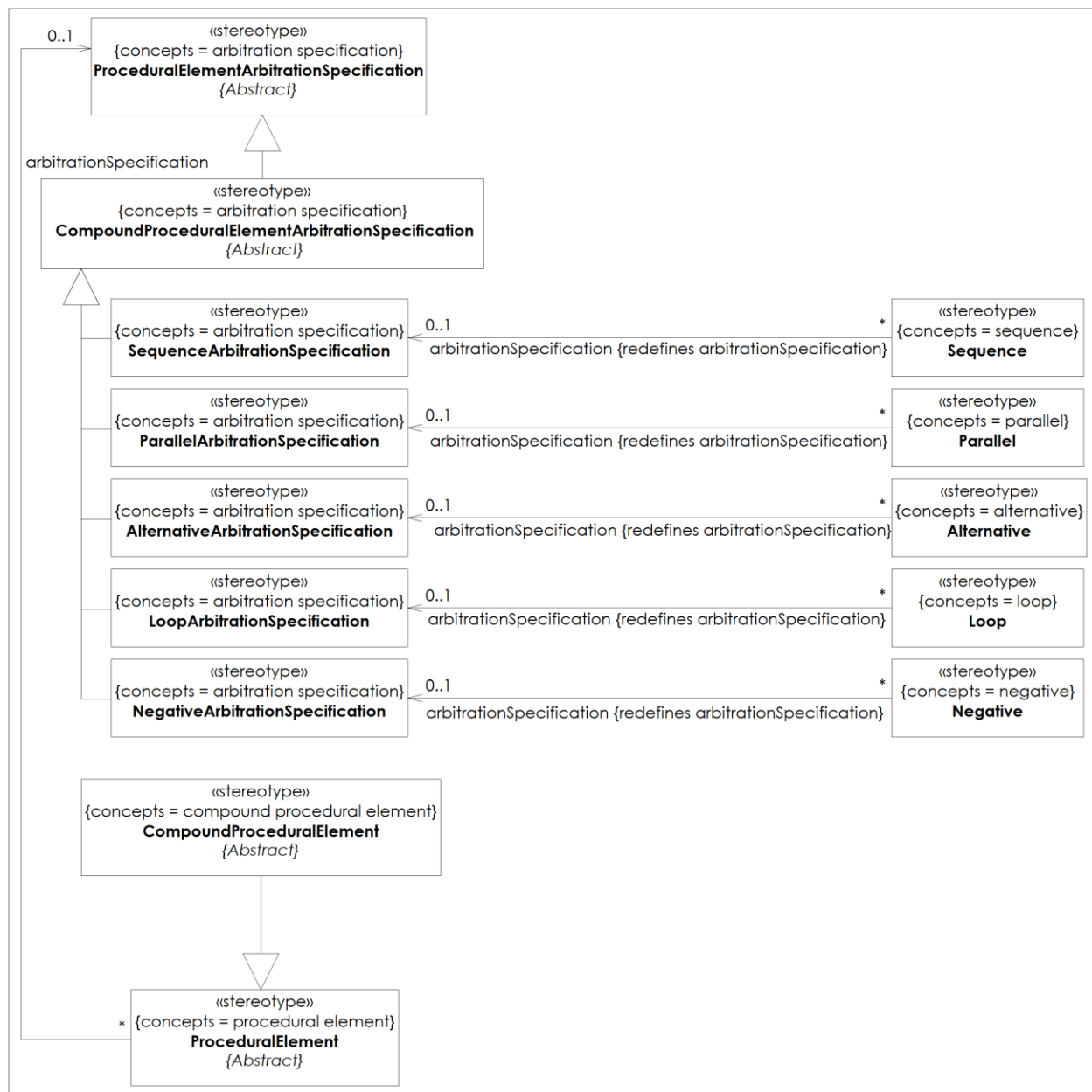


Figure 8.22 - Arbitration of CompoundProceduralElements

### 8.7.1.2.3 Stereotype Specifications

#### 8.7.1.2.3.1 AlternativeArbitrationSpecification

Description	<p>An «<a href="#">AlternativeArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a set of <a href="#">procedural elements</a> that are executed in mutually exclusive branches.</p> <p>«AlternativeArbitrationSpecification» adheres by the semantics of the default «CompoundProceduralElementArbitrationSpecification».</p>
Extension	<a href="#">BehavioredClassifier</a>
Super Class	<a href="#">CompoundProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.



#### 8.7.1.2.3.2 AtomicProceduralElementArbitrationSpecification

Description	<p>An «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a single <a href="#">atomic procedural element</a>.</p> <p>«AtomicProceduralElementArbitrationSpecification» adheres by the semantics of the default «ProceduralElementArbitrationSpecification».</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">ProceduralElementArbitrationSpecification</a>
Sub Class	<a href="#">CheckPropertyArbitrationSpecification</a> , <a href="#">CreateLogEntryArbitrationSpecification</a> , <a href="#">CreateStimulusArbitrationSpecification</a> , <a href="#">ExpectResponseArbitrationSpecification</a> , <a href="#">ProcedureInvocationArbitrationSpecification</a> , <a href="#">SuggestVerdictArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2.3.3 CompoundProceduralElementArbitrationSpecification

Description	<p>A «<a href="#">CompoundProceduralElementArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a set of <a href="#">procedural elements</a> that are executed together. The <a href="#">verdict</a> is derived from all or parts of the <a href="#">verdicts</a> calculated of their respective <a href="#">arbitration specifications</a>.</p> <p>The semantics of the default «CompoundProceduralElementArbitrationSpecification» refines the semantics of «ProceduralElementArbitrationSpecification» with respect to the following verdicts:</p> <ul style="list-style-type: none"><li>• Fail: The verdict 'Fail' is assigned, if any of the procedural elements, that were executed in the scope of the «CompoundProceduralElement», evaluates to 'Fail'.</li></ul>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">ProceduralElementArbitrationSpecification</a>
Sub Class	<a href="#">AlternativeArbitrationSpecification</a> , <a href="#">LoopArbitrationSpecification</a> , <a href="#">NegativeArbitrationSpecification</a> , <a href="#">ParallelArbitrationSpecification</a> , <a href="#">SequenceArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2.3.4 LoopArbitrationSpecification

Description	<p>A «<a href="#">LoopArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a set of <a href="#">procedural elements</a> that are sequentially executed in a <a href="#">loop</a>.</p> <p>«LoopArbitrationSpecification» adheres by the semantics of the default «CompoundProceduralElementSpecification». In addition, the maximal and minimal loop counters are part of the arbitration process for loops. With respect to verdict calculation, the following semantics is predefined for the default «LoopArbitrationSpecification»:</p> <ul style="list-style-type: none"><li>• Minimal number of loops violated: Verdict 'Error' is assigned.</li><li>• Maximal number of loops violated: Verdict 'Error' is assigned.</li></ul>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">CompoundProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2.3.5 NegativeArbitrationSpecification

Description	<p>A «<a href="#">NegativeArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for set of <a href="#">procedural elements</a> that are forbidden to be executed in this <a href="#">sequence</a>.</p> <p>«NegativeArbitrationSpecification» adheres by the semantics of the default «CompoundProceduralElementArbitrationSpecification», but refines it with an inversion of the verdicts 'Pass' and 'Fail'. In cases where a 'Fail' would be produced, a verdict 'Pass' shall be assigned. In cases where a 'Pass' would be produced, a verdict 'Fail' shall be assigned.</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">CompoundProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2.3.6 ParallelArbitrationSpecification

Description	<p>A «<a href="#">ParallelArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a set of <a href="#">procedural elements</a> that were executed in <a href="#">parallel</a>.</p> <p>«ParallelArbitrationSpecification» adheres by the semantics of the default «CompoundProceduralElementArbitrationSpecification».</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">CompoundProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

### 8.7.1.2.3.7 ProceduralElementArbitrationSpecification

Description	<p>A «<a href="#">ProceduralElementArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a single or a set of <a href="#">procedural elements</a>.</p> <p>A <a href="#">procedural element arbitration specification</a> incorporates sequence information about when and how long the execution of a corresponding <a href="#">procedural element</a> happened, because <a href="#">procedural elements</a> define an execution window in which their execution shall happen. This execution window is either defined by means of ordering (i.e., after the execution of a previous <a href="#">procedural element</a>, or after the start of a <a href="#">test case</a> execution) or by means of time. When using a time-based execution window, it is possible to specify the earliest and latest point in time when the execution of the <a href="#">procedural element</a> as well as the maximum duration the execution of the <a href="#">procedural element</a> may have. UTP does not prescribe how to specify time-based execution windows. Using UML Simple Time might be one solution, the time concepts of MARTE another one. If no time execution windows are defined, the ordering execution window is implicitly set, i.e., the execution of a <a href="#">procedural element</a> shall happen after the execution of its previous <a href="#">procedural element</a> has finished.</p> <p>Specific <a href="#">procedural element arbitration specifications</a> (e.g., <a href="#">expect response action arbitration specification</a>) incorporate the Boolean statement whether expected <a href="#">data</a> values, that belong to the corresponding <a href="#">procedural element</a>, match with the actual <a href="#">data</a> values that were used during execution of the corresponding <a href="#">procedural element</a>. Those <a href="#">data</a> values of interest comprise <a href="#">actual parameters</a> in case of a <a href="#">procedure invocation</a>, actual payload of a creat stimulus action or <a href="#">expect response action</a> or the actual value obtained from a checked <a href="#">property</a> in case of a <a href="#">check property action</a>. In UTP, the matching semantics of <a href="#">data</a> values are defined by the semantics of ValueSpecifications and the UTP-specific (normative and non-normative) <a href="#">data value extensions</a>.</p> <p>The semantics of the default «<a href="#">ProceduralElementArbitrationSpecification</a>» complements the semantics of «<a href="#">ArbitrationSpecification</a>» by defining the general rule that determines the assignment of <a href="#">verdicts</a>. All other sub-classes of «<a href="#">ProceduralElementArbitrationSpecification</a>» either adhere by, complement or refine that semantics. The semantics of the default <a href="#">procedural element arbitration specification</a> is as follows:</p> <ul style="list-style-type: none"> <li>• None: The verdict 'None' is assigned when the procedural element was not yet executed.</li> <li>• Pass: The verdict 'Pass' is assigned, when the expected execution of the procedural element matches with the actual execution of the procedural element, including sequence information and potentially data value comparison.</li> <li>• Inconclusive: The verdict 'Inconclusive' is never assigned by default arbitration specifications.</li> <li>• Fail: The verdict 'Fail' can only be assigned by the following arbitration specifications: compound procedural element arbitration specification, expect response arbitration specification, suggest verdict arbitration specification, and check property arbitration specification. The default semantics of these specific arbitration specifications will be described by these respective stereotypes.</li> <li>• Error: The verdict 'Error' is assigned, if the execution of a procedural element was not correctly performed (by a human or a test execution tool).</li> </ul>
Extension	<a href="#">BehavioredClassifier</a>
Super Class	<a href="#">ArbitrationSpecification</a>

Sub Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a> , <a href="#">CompoundProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2.3.8 ProcedureInvocationArbitrationSpecification

Description	<p>A «<a href="#">ProcedureInvocationArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for an executed <a href="#">procedure invocation</a>.</p> <p>«ProcedureInvocationArbitrationSpecification» complements the semantics of the default «ProceduralElementArbitrationSpecification»:</p> <p>Procedure invocations may pass actual parameter values to the invoked procedure. If there is a mismatch between the expected actual parameter values, prescribed by a «ProcedureInvocation», and the actual execution of the «ProcedureInvocation», the verdict 'Error' shall be assigned.</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.2.3.9 SequenceArbitrationSpecification

Description	<p>A «<a href="#">SequenceArbitrationSpecification</a>» calculates a <a href="#">verdict</a> for a <a href="#">sequence</a> of executed <a href="#">procedural elements</a>.</p> <p>«SequenceArbitrationSpecification» adheres by the semantics of the default «CompoundProceduralElementArbitrationSpecification».</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">CompoundProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

### 8.7.1.3 Test-specific Action Arbitration Specifications

The [test action arbitration specification](#) sections summarize the different types of [arbitration specifications](#) that can be used to define proprietary [arbitration specifications](#) for prescribing [test action](#).

#### 8.7.1.3.1 Arbitration of Test-specific Actions

The diagram below shows the abstract syntax of the [arbitration specifications](#) for dedicated [test actions](#).

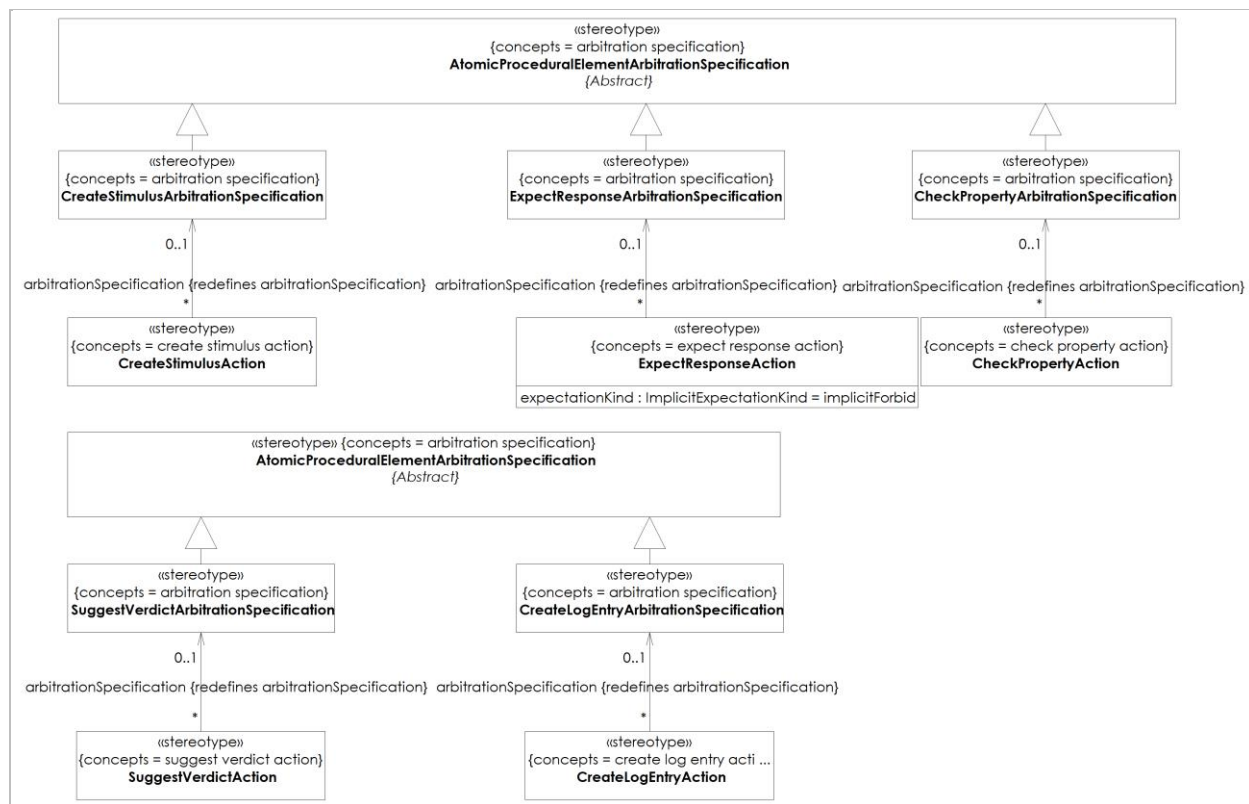


Figure 8.23 - Arbitration of Test-specific Actions

### 8.7.1.3.2 Stereotype Specifications

#### 8.7.1.3.2.1 CreateStimulusArbitrationSpecification

Description	<p>An «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» that specifies the <a href="#">verdict</a> calculation rule for a <a href="#">create stimulus action</a>.</p> <p>«<a href="#">CreateStimulusArbitrationSpecification</a>» complements the semantics of the default «<a href="#">AtomicProceduralElementArbitrationSpecification</a>»:</p> <p>The semantics of the default «<a href="#">CreateStimulusArbitrationSpecification</a>» shall include an evaluation of permitted and forbidden elements. If an element was sent to the test item that was declared as <a href="#">forbiddenElement</a>, the <a href="#">verdict</a> 'error' shall be assigned. If an element was sent to the test item that was declared as <a href="#">permittedElement</a> (including potential arguments of the «<a href="#">CreateStimulusAction</a>») and the expected data values of that element does not match with the actual data values of the actually sent element, the <a href="#">verdict</a> 'error' shall be assigned.</p>
Extension	<a href="#">BehavoredClassifier</a>
Super Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.3.2.2 ExpectResponseArbitrationSpecification

Description	<p>An «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» that specifies the <a href="#">verdict</a> calculation rule for an <a href="#">expect response action</a>.</p> <p>«<a href="#">ExpectResponseArbitrationSpecification</a>» complements the semantics of the default «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» with respect to sequence information and data value matching:</p> <p>If the expected execution time window of an «<a href="#">ExpectResponseAction</a>» does not match with the actual execution time point, the <a href="#">verdict 'fail'</a> shall be assigned. If the actual ordering of the execution of an «<a href="#">ExpectResponseAction</a>» does not match with the expected ordering, the <a href="#">verdict 'error'</a> shall be assigned.</p> <p>If the actual data values, that convey the «<a href="#">ExpectResponseAction</a>» as its payload, obtained from the test item do not match with the expected payload data values, the <a href="#">verdict 'fail'</a> shall be assigned.</p> <p>The semantics of the default «<a href="#">ExpectResponseArbitrationSpecification</a>» includes an evaluation of the ignored, forbidden and expected elements declaration. If a received element is declared as <a href="#">forbiddenElement</a>, the <a href="#">verdict 'fail'</a> shall be assigned. If a received element is declared as <a href="#">ignoredElement</a>, it shall be discarded and not contribute to the «<a href="#">ExpectResponseArbitrationSpecification</a>» for further evaluation. If a received element is declared as expected element, the <a href="#">verdict 'pass'</a> shall be assigned.</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.3.2.3 CheckPropertyArbitrationSpecification

Description	<p>An «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» that specifies the <a href="#">verdict</a> calculation rule for a <a href="#">check property action</a>.</p> <p>«<a href="#">CheckPropertyArbitrationSpecification</a>» adheres by the semantics of the default «<a href="#">AtomicProceduralElementArbitrationSpecification</a>».</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.3.2.4 SuggestVerdictArbitrationSpecification

Description	<p>An «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» that specifies the <a href="#">verdict</a> calculation rule for a <a href="#">suggest verdict action</a>.</p> <p>«<a href="#">SuggestVerdictArbitrationSpecification</a>» complements the semantics of the default «<a href="#">AtomicProceduralElementArbitrationSpecification</a>» with respect to the provision of the suggested verdict to the «<a href="#">TestCaseArbitrationSpecification</a>»:</p> <p>In case, the «<a href="#">SuggestVerdictArbitrationSpecification</a>» evaluates to a 'pass', the suggested verdict is passed to the «<a href="#">TestCaseArbitrationSpecification</a>». It will be discarded, if the «<a href="#">SuggestVerdictArbitrationSpecification</a>» evaluates to 'error'.</p>
Extension	<b>BehavoredClassifier</b>
Super Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.1.3.2.5 CreateLogEntryArbitrationSpecification

Description	An « <a href="#">AtomicProceduralElementArbitrationSpecification</a> » specification that specifies the <a href="#">verdict</a> calculation rule for a <a href="#">create log entry action</a> .  «CreateLogEntryArbitrationSpecification» adheres by the semantics of the default «AtomicProceduralElementArbitrationSpecification».
Extension	<a href="#">BehavioredClassifier</a>
Super Class	<a href="#">AtomicProceduralElementArbitrationSpecification</a>
Change from UTP 1.2	Newly introduced by UTP 2.

### 8.7.2 Test Logging

The UTP test logging facility allows incorporating details about the execution of test-specific procedures, such as test execution schedules and test cases, but also of procedural elements. The UTP test logging facility differs between two kinds of test log information:

- Test log header
- Test log details

The test log header represents the at least required information to comprehend or trace the status of the test execution such as also coverage of test objectives or test requirements. The test log details further refine the test log with the details of relevant events (i.e., execution of procedural elements) that happened at runtime. The information the test log details yield are in particular important for analyses of the test execution such comparison, verdict calculation, failure inspection or root cause analysis.

The UTP test logging facility builds upon UML's InstanceSpecification and classification mechanism (henceforth called classifier-instance relationship). Every test log element is represented by an InstanceSpecification with an inherent set of structural information. These inherently provided structural information are the executing entity, the execution start and the duration. The classifier-instance-based representation of test logs grants high flexibility to the user. It enables the definition of additional, user-defined structural information of arbitrary complexity to every test log element.

Logging of behavioral constituents (i.e., Actions or OccurrenceSpecifications) is not intended by UML but relevant for testing, though. UTP integrates the behavioral constituents of the underlying UML and the classifier-instance-based test log model by means of dedicated test log entries and their structural information. Every test log entry captures the details of a corresponding procedural element that was executed in the course of the execution of a test-specific procedure.

UTP defines dedicated test log entry structures for logging of procedure invocations, create stimulus actions and expect response actions. These test log entry structures specify the at least required structural information of those procedural elements such as formal parameters and invocation targets. The corresponding test log entries build upon this structural information and yield the corresponding actual parameter values captured in the course of the execution of such a procedural element.

#### 8.7.2.1 Test Logging Overview

The following diagram shows the abstract syntax of the basic concepts of the test logging facility.

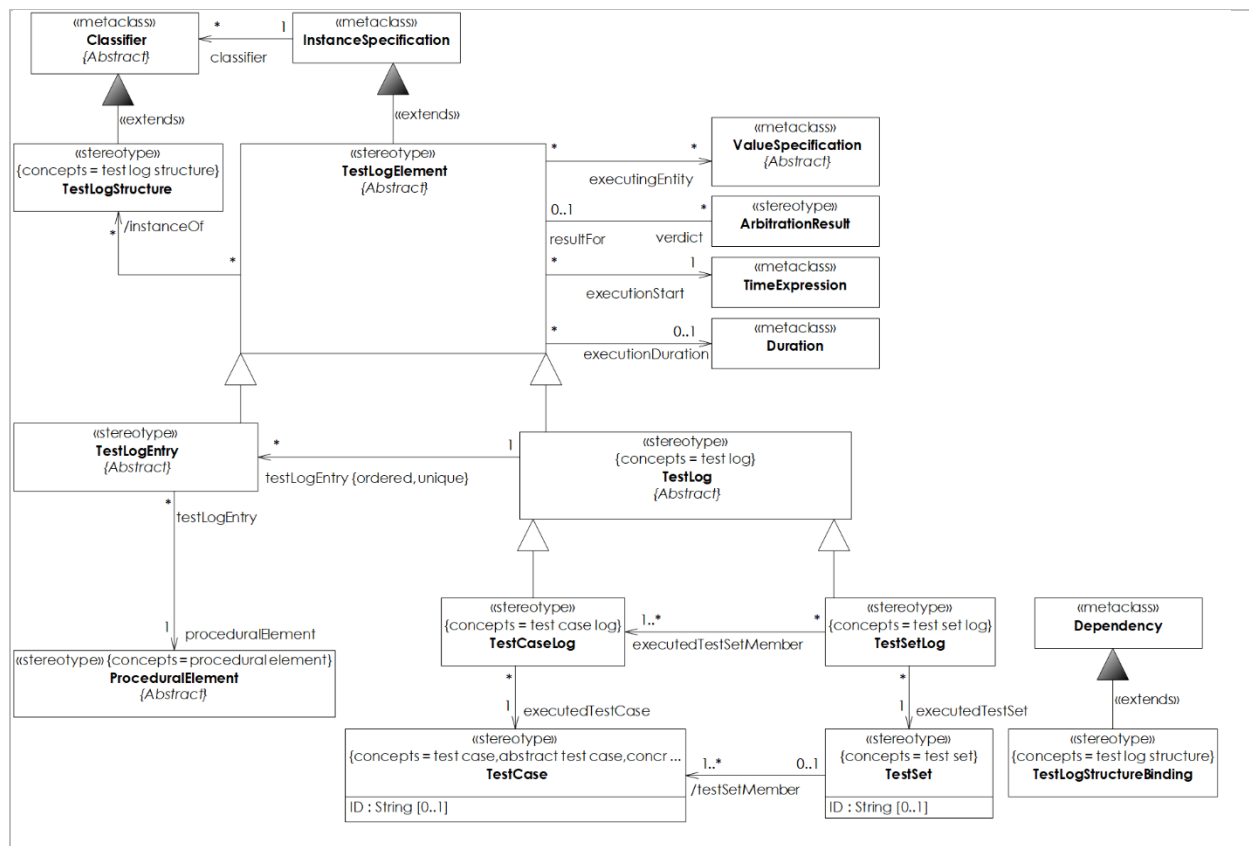


Figure 8.24 - Test Logging Overview

### 8.7.2.2 Test Log Entries Overview

The following diagram shows the abstract syntax of the basics of test log entries.



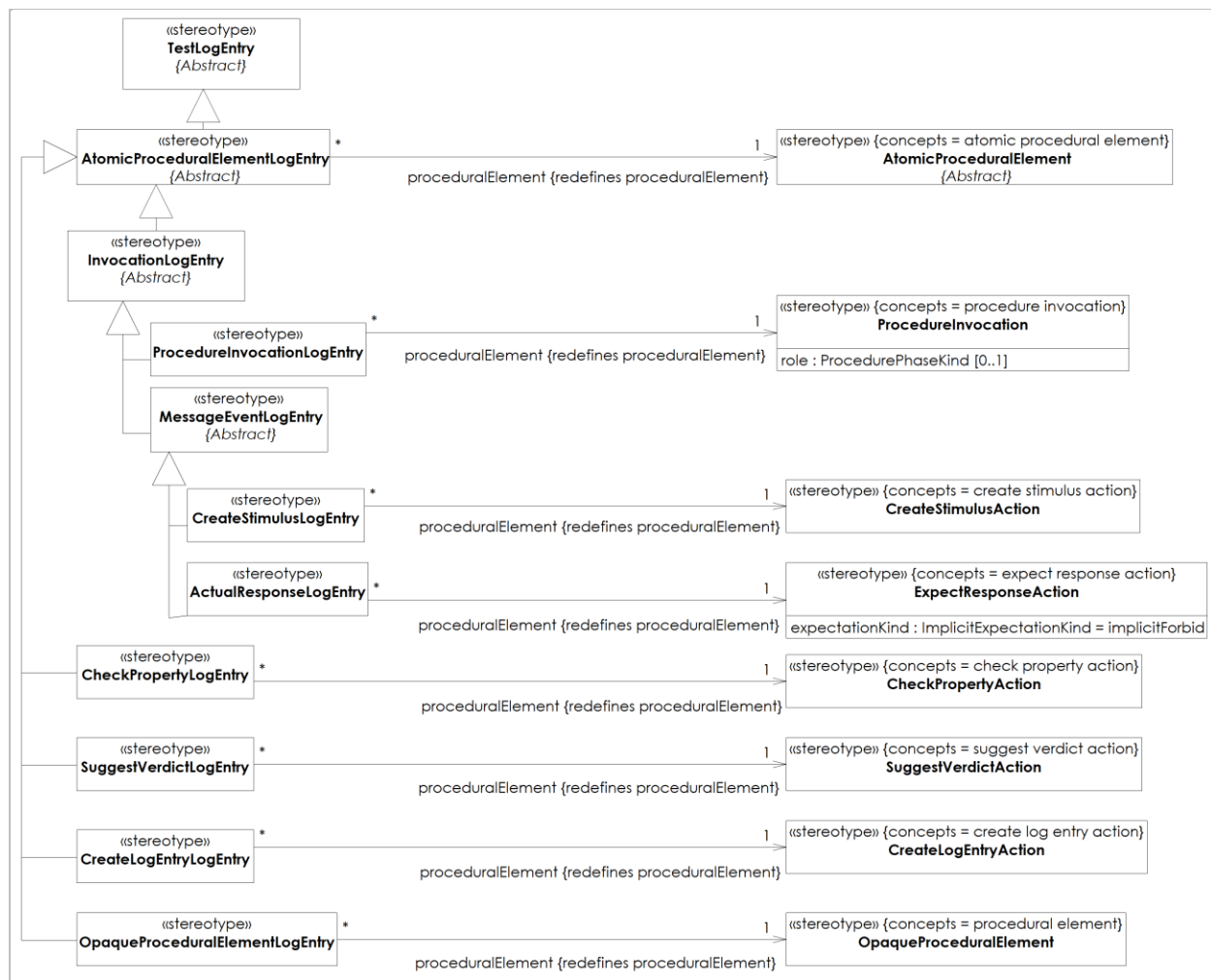


Figure 8.25 - Test Log Entries Overview

### 8.7.2.3 Test Log Entries Details

The following diagram shows the abstract syntax of the details of test log entries.

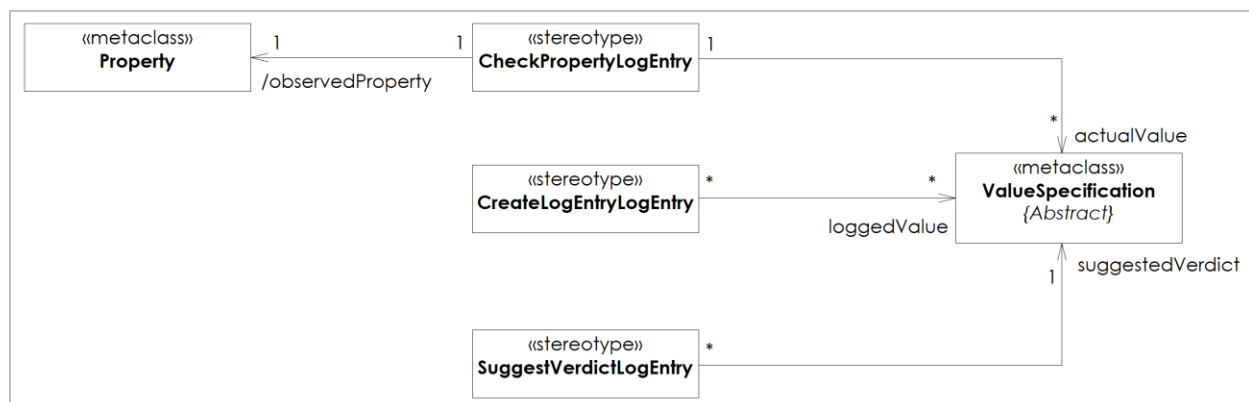


Figure 8.26 - Test Log Entries Details

### 8.7.2.4 Invocation Test Log Entry Details

The following diagram shows the abstract syntax of test log entries that capture details of invocations.

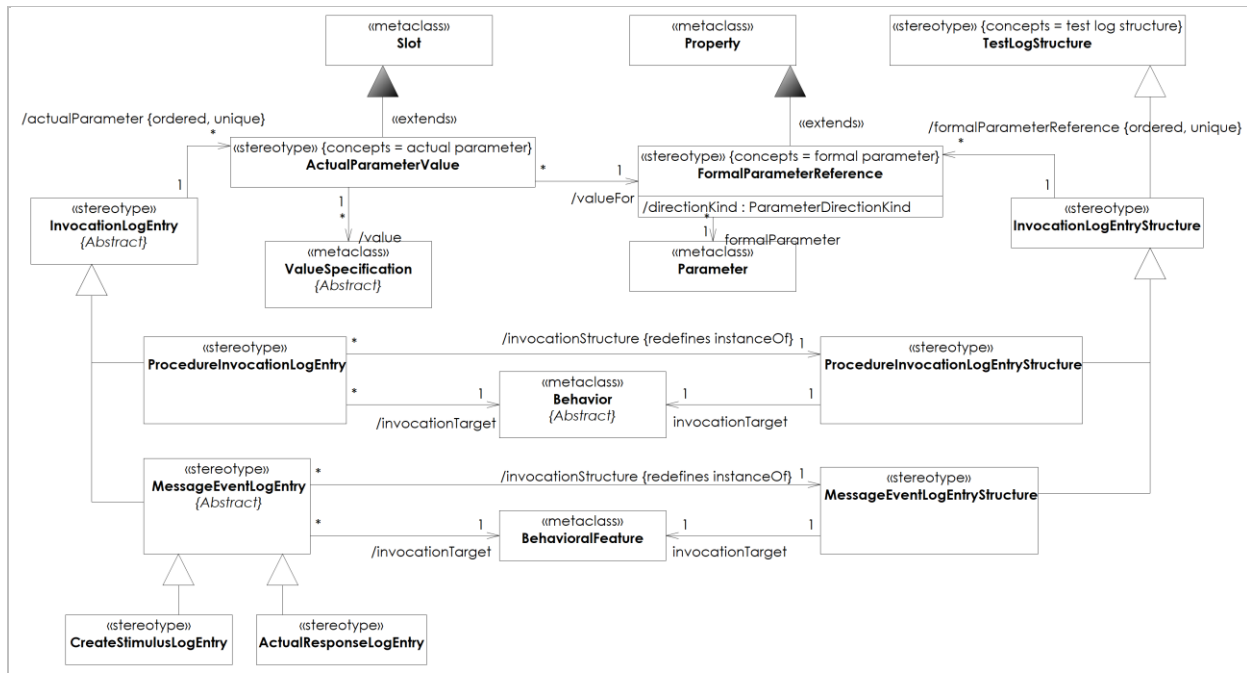


Figure 8.27 - Invocation Test Log Entry Details

### 8.7.2.5 Stereotype Specifications

#### 8.7.2.5.1 TestLogElement

Description	<p>A test log element represents a single building block in the realm of test logging. For each test log element there is an corresponding executable element that has been carried out. These corresponding executable elements can either be whole test sets or test cases or one or more procedural elements.</p> <p>A test log element provides a corresponding executable element with information about the starting point in time of the execution, the duration of the execution of the logged element and an entity (i.e., a machine or a human) that executed the element. These default structural information are common for each concrete test log element. Further structural information may be added by sub-stereotypes or by dedicated structural extension using the stereotype «TestLogStructure».</p> <p>Every test log element can be related with arbitration results (i.e., verdicts) that were calculated for that test log element, the corresponding executable element and the arbitration specification that ties both elements together for verdict calculation. Section ArbitrationSpecification provides further details about the verdict calculation process.</p> <p>The stereotype «TestLogElement» extends InstanceSpecification. User-defined structural information can be added by using the underling UML classification mechanism. The set of additional test log element structural information are determined by all Classifier of the underlying InstanceSpecification that have «TestLogStructure» applied.</p>
Extension	<a href="#">InstanceSpecification</a>
Sub Class	<a href="#">TestLog</a> , <a href="#">TestLogEntry</a>

Associations	/instanceOf : TestLogStructure [*] The set of additional structural information associated with that test log element. It is derived from the all Classifier with «TestLogStructure» applied that classify the underlying InstanceSpecification.
	executingEntity : ValueSpecification [*] Information about the executing entity or entities (i.e., either humans or machines) that were in charge of carrying out the element that corresponds to the test log element.
	verdict : ArbitrationResult [*] The verdicts that were calculated for the test log element.
	executionDuration : Duration [0..1] Denotes how long the execution of the corresponding executable element lasted.
	executionStart : TimeExpression Denotes the point in time when the execution of the corresponding executable element began.
Constraints	Restriction of extendable metaclasses «TestLogElement» shall not be applied to EnumerationLiteral.
Change from UTP 1.2	«TestLogElement» was newly introduced by UTP 2.1.

#### 8.7.2.5.2 TestLog

Description	<p><b>TestLog:</b> A <a href="#">test log</a> is the instance of a <a href="#">test log structure</a> that captures relevant information from the execution of a <a href="#">test case</a> or <a href="#">test set</a>. The least required information to be logged is defined by the <a href="#">test log structure</a> of the <a href="#">test log</a>.</p> <p>A <a href="#">test log</a> captures information on the execution of a <a href="#">test case</a> or <a href="#">test set</a> that actually happened according to the specification required by its <a href="#">test log structure</a>. Each <a href="#">test log</a> is, at least, an instance of the implicitly defined default <a href="#">test log structure</a>. This is reflected by its tag definitions that comprise the required log information. If further information is not required for capturing by an <a href="#">executing entity</a>, a <a href="#">test log</a> may not refer to an explicit <a href="#">test log structure</a> (i.e., the <b>Classifier</b> of the underlying <b>InstanceSpecification</b> remains empty).</p> <p>In addition to the information given by the implicit default <a href="#">test log structure</a>, users may set an explicitly defined a <a href="#">test log structure</a> of arbitrary complex internal structures. In that case, the underlying <b>InstanceSpecification</b> may capture the additional information by relying on the native UML <b>InstanceSpecification</b> mechanism, namely Slots.</p> <p>Structural information on <a href="#">test log</a> level are sufficient to comprehend the status of testing or coverage of <a href="#">test objectives</a> and <a href="#">test requirements</a>. This minimal log information are referred to as the <i>test log header</i>. Header information only contain high-level information about the status of a test run, not about the details of the run. Details of the test run are captured by means of test log entries. As opposed to the <a href="#">test log</a> header, detailed logging on <a href="#">procedural element</a> level is referred to as <i>test log details</i>. Test log details capture detailed information about the executed <a href="#">sequence of procedural element</a> represented by test log entries. Test log details provide a deeper insight into the test execution process and leverage the analysis of test runs (e.g., what is the reason for a failing <a href="#">test case</a>). Recording test log details is an optional, but powerful feature of a <a href="#">test log</a>.</p>
Extension	<b>InstanceSpecification</b>
Super Class	<a href="#">TestLogElement</a>
Sub Class	<a href="#">TestCaseLog</a> , <a href="#">TestSetLog</a>

	<pre>testLogEntry {ordered, unique} : TestLogEntry [*]</pre> <p>The sequence of test log entries that captures the details of the test execution. This sequence is referred to as <i>test log details</i>.</p>
Change from UTP 1.2	<p>Changed from UTP 1.2. In UTP 1.2 «TestLog» was used to capture the execution of a test case or a test set (called test content in UTP 1.2). In UTP 2, two dedicated concepts have been newly introduced therefore (i.e., «TestCaseLog» and «TestSetLog»).</p>

#### 8.7.2.5.3 TestSetLog

Description	<p>A <a href="#">test set log</a> captures the least required information on the execution of a <a href="#">test set</a> by an <a href="#">executing entity</a>. The least required information is defined by the corresponding (potentially implicit) <a href="#">test log structure</a> of the <a href="#">test set log</a>.</p> <p>A <a href="#">test set log</a> consists mainly of the logs of the executed <a href="#">test cases</a> that are members of the <a href="#">test set</a>. Since not all <a href="#">test cases</a> of a <a href="#">test set</a> must necessarily be executed by an <a href="#">executing entity</a>, a <a href="#">test set log</a> may only refer to the <a href="#">test case logs</a> of a subset of the test set's <a href="#">test cases</a>.</p>
Extension	<b>InstanceSpecification</b>
Super Class	<a href="#">TestLog</a>
Associations	<pre>executedTestSetMember : TestCaseLog [1..*]</pre> <p>Refers to the test cases that are the members of the test set log's corresponding test set and whose execution were captured as a result of the execution of the test set.</p> <pre>executedTestSet : TestSet</pre> <p>Refers to the test set whose execution was captured by means of the given test case log.</p>
Constraints	<p>Executed test cases and definition of test set members must be consistent</p> <p>A «<a href="#">TestSetLog</a>» must only refer to «<a href="#">TestCaseLog</a>»s of «<a href="#">TestCase</a>»s that are members of the executed «<a href="#">TestSet</a>».</p>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.2.5.4 TestCaseLog

Description	<p><a href="#">TestCaseLog</a>: A <a href="#">test log</a> that captures relevant information on the execution of a <a href="#">test case</a>.</p> <p>A <a href="#">test case log</a> captures the least relevant information on the execution of a <a href="#">test case</a> by an <a href="#">executing entity</a>. The at least required information is defined by the corresponding and potentially implicit <a href="#">test log structure</a> of the <a href="#">test case log</a>.</p>
Extension	<b>InstanceSpecification</b>
Super Class	<a href="#">TestLog</a>
	<pre>executedTestCase : TestCase</pre> <p>Refers to the TestCase whose execution was captured by means of the given TestCaseLog.</p>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.2.5.5 TestLogStructure

Description	<p>A <a href="#">test log structure</a> enables the specification of user-defined structures that must be logged by an <a href="#">executing entity</a>, such as human tester or a test execution tool, during the execution of test suites, <a href="#">test cases</a> or <a href="#">test execution schedules</a>. This information is also called the least required log information, because executing entities are not restricted to capturing only information mentioned in the <a href="#">test log structure</a>. A <a href="#">test log</a> structure may describe both the required information for the header part as well as the body part of a <a href="#">test log</a>.</p> <p>There is an implicit default (undefined) <a href="#">test log structure</a> available in UTP that every user-defined <a href="#">test log structure</a> complies with. The default <a href="#">test log structure</a> represents the least required log information for the header part. This information comprises:</p> <ul style="list-style-type: none"> <li>• One or more of an <a href="#">executing entity</a>.</li> <li>• A point in time where the execution of the <a href="#">test case</a>, test suite or <a href="#">test execution schedule</a> began.</li> <li>• The <a href="#">duration</a> the execution of the <a href="#">test case</a>, test suite or test schedule lasted.</li> <li>• The final <a href="#">verdict</a> that was calculated by the corresponding <a href="#">arbitration specification</a>.</li> </ul> <p>Those pieces of information of the default (implicit) <a href="#">test log structure</a> are represented as tag definitions of the stereotype <a href="#">test log</a> solely because they are eventually instantiated when a <a href="#">test log</a> is created.</p>
Extension	<b>Classifier</b>
Sub Class	<a href="#">InvocationLogEntryStructure</a>
Constraints	<p>Restriction of extendable metaclasses</p> <p>«TestLogStructure» shall only be applied to instances of their metaclass Datatype or Class.</p> <p>Specialization of TestLogStructure Classifier</p> <p>Classifiers with «<a href="#">TestLogStructure</a>» applied must only extend <b>Classifier</b> with «<a href="#">TestLogStructure</a>» applied.</p> <p>Internal structure of TestLogStructure Classifier</p> <p><b>Classifiers</b> with «<a href="#">TestLogStructure</a>» applied must only own Properties.</p> <p>CollaborationUse not allowed</p> <p>A «<a href="#">TestLogStructure</a>» <b>Classifier</b> must not participate in Collaborations.</p>
Change from UTP 1.2	Newly introduced by UTP 2.

#### 8.7.2.5.6 TestLogEntry

Description	<p>A test log entry represents an actual instance of an executed procedural element. While the referenced procedural elements denotes what is expected – either from the test item or from the test component (including human tester) – a test log entry denotes the actual instance of that procedural element, captured during runtime. Test log entry inherits all default structural information from test log element.</p>
Extension	<b>InstanceSpecification</b>
Super Class	<a href="#">TestLogElement</a>
Sub Class	<a href="#">AtomicProceduralElementLogEntry</a>
	<p><code>proceduralElement : ProceduralElement</code></p> <p>Refers to the expected procedural element that was actually carried out by an executing entity at runtime.</p>
Change from UTP 1.2	Changed from UTP 1.2. In UTP 1.2, «TestLogEntry» extended OccurrenceSpecification.

#### 8.7.2.5.7 AtomicProceduralElementLogEntry

Description	Atomic procedural element log entry captures details of the execution of an atomic procedural element.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">TestLogEntry</a>
Sub Class	<a href="#">CheckPropertyLogEntry</a> , <a href="#">CreateLogEntryLogEntry</a> , <a href="#">InvocationLogEntry</a> , <a href="#">OpaqueProceduralElementLogEntry</a> , <a href="#">SuggestVerdictLogEntry</a>
Associations	proceduralElement {redefines proceduralElement} : AtomicProceduralElement  Refers to the atomic procedural element that was carried out by an executing entity at runtime.
Change from UTP 1.2	«AtomicProceduralElementLogEntry» was newly introduced by UTP 2.1

#### 8.7.2.5.8 InvocationLogEntryStructure

Description	Invocation log entry structure refines test log structure for expressing log entries of any kind of invocations that happened at runtime. It specifies the at least required structural information for logging the invocation of a procedure or the exchange of a message, i.e., for one the formal parameters the invocation target offers and the actual target of the invocation.
Extension	<a href="#">Classifier</a>
Super Class	<a href="#">TestLogStructure</a>
Sub Class	<a href="#">MessageEventLogEntryStructure</a> , <a href="#">ProcedureInvocationLogEntryStructure</a>
Associations	/formalParameterReference {ordered, unique} : FormalParameterReference [*]  The ordered set of formal parameters offered by the invocation target. The derivation algorithm in the context of an «InvocationLogEntryStructure» is defined as follows: <ul style="list-style-type: none"> <li>• Iterate over the <i>ownedParameter</i> of the invocation target.</li> <li>• For <i>ownedParameter</i> p, look for any <i>ownedProperty</i> with «FormalParameterReference» of the underlying Classifier that refers as <i>formalParameter</i> to Parameter p.</li> <li>• Add the found «FormalParameterReference» to the ordered set of <i>formalParameterReference</i>.</li> </ul>
Change from UTP 1.2	«InvocationLogEntryStructure» was newly introduced by UTP 2.1

#### 8.7.2.5.9 FormalParameterReference

Description	In the classifier-instance-based representation of test logs in UTP 2, formal parameter of invocation targets are defined as Properties and actual parameter are defined as values of a Slots that refers to the corresponding Property. «FormalParameterReference» conveniently binds a Parameter of a Behavior or BehavioralFeature to the corresponding Property the underlying Classifier of any concrete «InvocationLogEntryStructure». In combination with «ActualParameterValue» both the formal and actual parameter are tightly integrated with each other. This integration simplifies the processing of parameters for they can be directly accessed via the abstracting stereotypes without considering the type of invocation target.
Extension	<a href="#">Property</a>
Attributes	directionKind : ParameterDirectionKind [1]  The direction kind of the formal parameter. It is derived from the direction kind of

	the owned parameter of the invocation target.
	<code>formalParameter : Parameter [*]</code>
	The owned parameter of the invocation target.
Change from UTP 1.2	«FormalParameterReference» was newly introduced by UTP 2.1.

#### 8.7.2.5.10 InvocationLogEntry

Description	Invocation log entry captures details about the execution of procedure invocations or message exchange that actually happened at runtime. In UTP 2, expected message exchange is represented by the test actions create stimulus action and expect response action. Both procedure invocation and message exchange can be parameterized. The actual values that convey a procedure invocation or message exchange are referred to as actual parameter.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">AtomicProceduralElementLogEntry</a>
Sub Class	<a href="#">MessageEventLogEntry</a> , <a href="#">ProcedureInvocationLogEntry</a>
Associations	<code>/actualParameter {ordered, unique} : ActualParameterValue [*]</code>  Refers to the ordered set of actual parameter values for the captured invocation. The order of actual parameter values is derived from the ordered set of formal parameter values specified by the corresponding invocation log entry structure of the given invocation log entry.
Change from UTP 1.2	«InvocationLogEntry» was newly introduced by UTP 2.1

#### 8.7.2.5.11 ActualParameterValue

Description	In the classifier-instance-based representation of test logs in UTP 2, actual parameter of invocation targets are defined as Slot values and formal parameters are defined as Properties of Classifiers to which these Slots refer. «ActualParameterValue» abstracts from the different kinds of UML representations of actual parameter values to simplify processing of that information. In Activities, actual parameter value are denoted as InputPins contained by an InvocationAction or OutputPins of an AcceptEventAction, in Interactions as ValueSpecifications of a Message. In combination with «FormalParameterReference» both the formal and actual parameter are tightly integrated with each other. This integration simplifies the processing of parameters for they can be directly accessed via the abstracting stereotypes without considering the invocation target.
Extension	<a href="#">Slot</a>
Associations	<code>/valueFor : FormalParameterReference</code>  Relates this <a href="#">actual parameter</a> value to its <a href="#">formal parameter</a> of the corresponding invocation log entry structure. It is derived from the association end 'definingFeature' of the underlying Slot. <code>/value : ValueSpecification [*]</code>  The actually submitted or received payload while invoking a <a href="#">procedure</a> or exchanging a message. It is derived from the association end <i>value</i> of the underlying Slot.
Change from UTP 1.2	«ActualParameterValue» was newly introduced by UTP 2.1.

#### 8.7.2.5.12 ProcedureInvocationLogEntryStructure

Description	Procedure invocation log entry structure provides the at least required structural information for logging the execution of procedure invocations.
Extension	<a href="#">Classifier</a>

Super Class	<a href="#">InvocationLogEntryStructure</a>
	<p>invocationTarget : Behavior</p> <p>Refers to the procedure whose invocations can be logged with details of the given procedure invocation log entry structure.</p>
Change from UTP 1.2	«ProcedureInvocationLogEntryStructure» was newly introduced by UTP 2.1

#### 8.7.2.5.13 ProcedureInvocationLogEntry

Description	A procedure invocation log entry yields the details about the execution of a procedure invocation.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">InvocationLogEntry</a>
Associations	<p>proceduralElement {redefines proceduralElement} : ProcedureInvocation</p> <p>Refers to the procedure invocation that was carried out by an executing entity at runtime.</p> <p>/invocationStructure {redefines instanceOf} : ProcedureInvocationLogEntryStructure</p> <p>Refers to the invoked procedure, i.e., Behavior. It is derived from the invocation target of the corresponding invocation structure.</p> <p>/invocationTarget : Behavior</p> <p>Refers to the structural information for the given invocation log entry. It is derived from the sequences of Classifier of the underlying InstanceSpecification with «ProcedureInvocationLogEntryStructure» applied.</p>
Change from UTP 1.2	«ProcedureInvocationLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.14 MessageEventLogEntryStructure

Description	Message event log entry structure provides the at least required structural information for logging the exchange of message, i.e., either the submission of a stimulus or the reception of an actual response.
Extension	<a href="#">Classifier</a>
Super Class	<a href="#">InvocationLogEntryStructure</a>
	<p>invocationTarget : BehavioralFeature</p> <p>Refers to the BehavioralFeature whose invocation details can be logged with the given message event log entry structure.</p>
Change from UTP 1.2	«MessageEventLogEntryStructure» was newly introduced by UTP 2.1.

#### 8.7.2.5.15 MessageEventLogEntry

Description	A message event log entry captures details about any message exchange that happened between the test item and a test component. Message exchange can happen when a stimulus is submitted, or an actual response was received by a test component. Sending a stimulus and receiving a response represent important events in the course of test execution with respect to verdict calculation.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">InvocationLogEntry</a>
Sub Class	<a href="#">ActualResponseLogEntry</a> , <a href="#">CreateStimulusLogEntry</a>
Associations	<p>/invocationTarget : BehavioralFeature</p> <p>Refers to the invoked or received Operation or Reception. It is derived from the invocation target of the corresponding invocation structure.</p>



	<pre>/invocationStructure {redefines instanceOf} : MessageEventLogEntryStructure</pre> <p>Refers to the structural information for the given invocation log entry. It is derived from the sequences of Classifier of the underlying InstanceSpecification with «MessageEventLogEntryStructure» applied.</p>
Change from UTP 1.2	«MessageEventLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.16 CreateStimulusLogEntry

Description	A create stimulus log entry yields details about the submission of a stimulus at runtime. It represents an instance of a corresponding create stimulus action contained in a test case.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">MessageEventLogEntry</a>
Associations	<pre>proceduralElement {redefines proceduralElement} : CreateStimulusAction</pre> <p>Refers to the create stimulus action that was carried out by an executing entity at runtime.</p>
Change from UTP 1.2	«CreateStimulusLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.17 ActualResponseLogEntry

Description	An actual response log entry yields details about the reception of test item's response at runtime. It represents an instance of a corresponding expect response action contained in a test case.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">MessageEventLogEntry</a>
Associations	<pre>proceduralElement {redefines proceduralElement} : ExpectResponseAction</pre> <p>Refers to the expect response action that was carried out by an executing entity at runtime.</p>
Change from UTP 1.2	«ActualResponseLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.18 CheckPropertyLogEntry

Description	A check property log entry yields the details about the execution of a check property action.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">AtomicProceduralElementLogEntry</a>
Associations	<pre>proceduralElement {redefines proceduralElement} : CheckPropertyAction</pre> <p>Refers to the check property action that was carried out by an executing entity at runtime.</p> <pre>/observedProperty : Property</pre> <p>Refers to the Property whose value was checked. Usually, this is the Property of the corresponding check property action.</p> <pre>actualValue : ValueSpecification [*]</pre> <p>The actual value or values of the observed Property.</p>
Change from UTP 1.2	«CheckPropertyLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.19 SuggestVerdictLogEntry

Description	A suggest verdict log entry yields the details about the execution of a suggest verdict action.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">AtomicProceduralElementLogEntry</a>
Associations	<pre>proceduralElement {redefines proceduralElement} : SuggestVerdictAction</pre> <p>Refers to the suggest verdict action that was carried out by an executing entity at runtime.</p> <pre>suggestedVerdict : ValueSpecification</pre> <p>The actual verdict that was suggested by the executing entity.</p>
Change from UTP 1.2	«SuggestVerdictLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.20 CreateLogEntryLogEntry

Description	A create log entry log entry yields the details about the execution of a create log entry action.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">AtomicProceduralElementLogEntry</a>
Associations	<pre>proceduralElement {redefines proceduralElement} : CreateLogEntryAction</pre> <p>Refers to the create log entry action that was carried out by an executing entity at runtime.</p> <pre>loggedValue : ValueSpecification [*]</pre> <p>Refers to the values that were actually logged.</p>
Change from UTP 1.2	«CreateLogEntryLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.21 OpaqueProceduralElementLogEntry

Description	An opaque procedural element log entry yields the details about the execution of an opaque procedural element.
Extension	<a href="#">InstanceSpecification</a>
Super Class	<a href="#">AtomicProceduralElementLogEntry</a>
Associations	<pre>proceduralElement {redefines proceduralElement} : OpaqueProceduralElement</pre> <p>Refers to the opaque procedural element that was carried out by an executing entity at runtime.</p>
Change from UTP 1.2	«OpaqueProceduralElementLogEntry» was newly introduced by UTP 2.1.

#### 8.7.2.5.22 TestLogStructureBinding

Description	<p>A <a href="#">test log structure</a> binding is responsible to explicitly bind <a href="#">test log structures</a> to <a href="#">test cases</a> or <a href="#">test sets</a>.</p> <p>It is possible to reuse the very same <a href="#">test log structure</a> at different locations. Since there are different possibilities how to model this, UTP suggests three methods to achieve multiple binding of <a href="#">test log structures</a>:</p> <ul style="list-style-type: none"><li>• Single <b>Dependency</b>/many suppliers method: This method binds many <a href="#">test cases</a> or <a href="#">test sets</a> as suppliers of the «<a href="#">TestLogStructureBinding</a>» <b>Dependency</b> to a single «<a href="#">TestLogStructure</a>» <b>Classifier</b> client.</li><li>• Multiple Dependencies/single suppliers method: This method binds a single <a href="#">test case</a> or <a href="#">test set</a> as supplier of the «<a href="#">TestLogStructureBinding</a>» <b>Dependency</b> to a single «<a href="#">TestCase</a>» <b>BehavioredClassifier</b> client.</li><li>• Combined method: This method combines the first two methods.</li></ul> <p>The sum of all bound test log structures for a <a href="#">test case</a> or <a href="#">test set</a> is calculated by merging all suppliers of all visible «<a href="#">TestLogStructureBinding</a>» Dependencies in a certain logical or technical scope. Visibility of <a href="#">test log structure</a> binding is not defined by this specification. Moreover, this specification neither prescribes how <a href="#">test log structure</a> bindings are finally put into effect by an <a href="#">executing entity</a> nor how to select them for later use by an <a href="#">executing entity</a>. Since <b>Dependency</b> is a PackageableElement, a possible method could be to use the UML deployment capabilities in order to implement the desired «<a href="#">TestLogStructureBinding</a>» <b>Dependency</b> to putting it into effect in the test execution system.</p>
Extension	<b>Dependency</b>
Constraints	<p>Specification of Dependency client</p> <p>A <b>Dependency</b> with «<a href="#">TestLogStructureBinding</a>» must have exactly one client containing a <b>Classifier</b> with «<a href="#">TestLogStructure</a>» applied.</p> <p>Specification of Dependency supplier</p> <p>A <b>Dependency</b> with «<a href="#">TestLogStructureBinding</a>» must have at least one but an unlimited number of suppliers containing a <b>BehavioredClassifier</b> with «<a href="#">TestCase</a>» applied.</p>
Change from UTP 1.2	Newly introduced by UTP 2.

## 8.8 Test Directives

The UTP 2 test directive facility builds the foundation for the specification of test-related activities. A test directive assembles a set of test techniques that shall be executed either manually or automatically. A test technique instructs a human or machine what to do, i.e., how to carry out the represented activity in detail. A test directive provides the assembled test techniques with all necessary information to carry out the corresponding test-related activity. Therefore, a test directive refers to a set of input elements that are accessible by the related test techniques. Usually, a test directive generates some output by processing the output.

Both test directive and test technique are intended to be sub-classed to specify concrete test-related activities. For example, the test design facility introduced in section 8.3.2 Test Design builds upon the test directive facility by specializing both test directive and test technique.

Additional structural information required to both the test directive and test technique shall be provided via the corresponding stereotypes <<TestDirectiveStructure>> and <<TestTechniqueStructure>>.

## 8.8.1 Test Directive Facility

The diagram below shows the abstract syntax of the test directive facility.

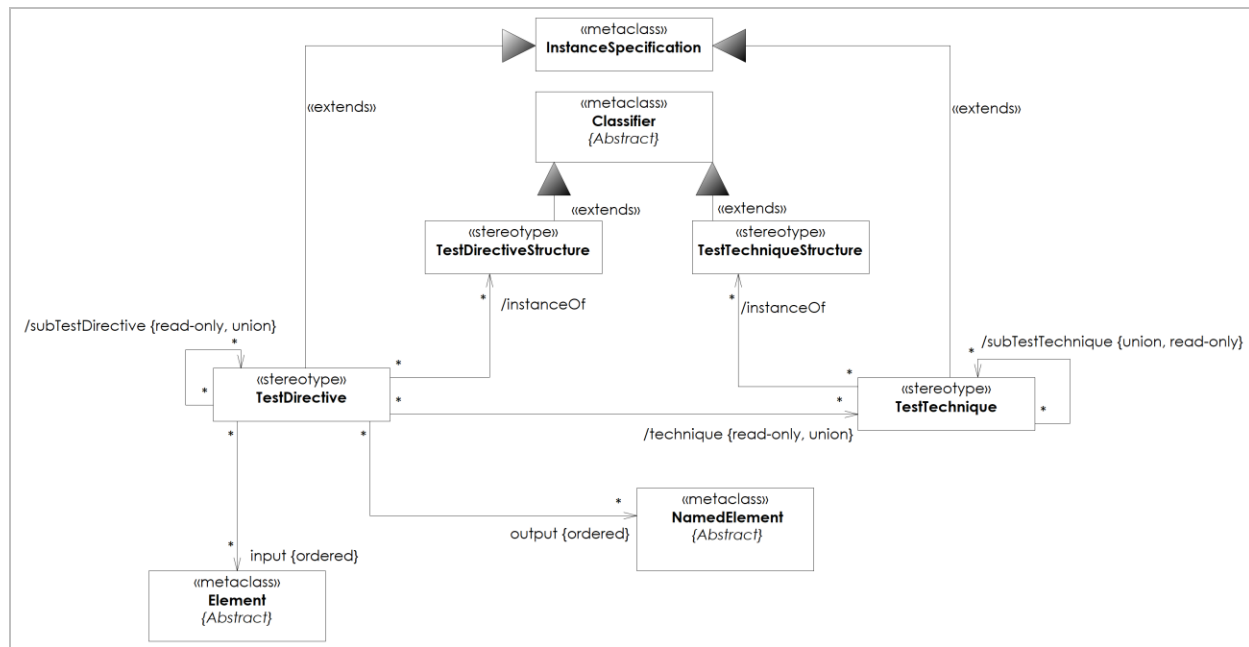


Figure 8.28 - Test Directive Facility

## 8.8.2 Stereotype Specifications

### 8.8.2.1 TestDirective

Description	<p>A test directive specifies a certain test-related activity that may consist of specific tasks and instructs a human or machine to carry out these tasks. Tasks are specified by test techniques. A test directive assembles and governs test techniques and provides them with all relevant information to be carried out.</p> <p>A test directive operates on a certain input set of NamedElements represented by the association end <i>input</i>. The input elements are visible to the test directive and transitively visible to and accessible by the assembled test techniques. The test techniques operate on the input elements to produce the output elements while processing the test directive. Output elements are represented by the association end <i>output</i>.</p> <p>A test directive may provide sub-directives by means of the association end <i>subDirective</i>. Providing a nested test directive enables testers to refine the test-related activity for certain input elements.</p>
Extension	<a href="#">InstanceSpecification</a>
Sub Class	<a href="#">TestDesignDirective</a>
Associations	<p>/instanceOf : TestDirectiveStructure [*]</p> <p>Refers to the test directive structures of which the given test directive is an instance of. The set of test directive structures is derived from all Classifiers with «TestDirectiveStructure» applied that are referred to as classifiers by the underlying InstanceSpecification.</p> <p>/technique {read-only, union} : TestTechnique [*]</p> <p>The set of test techniques that are assembled and governed by the test directive.</p>

	<code>/subTestDirective {read-only, union} : TestDirective [*]</code> Refers to potentially nested test directives that shall be carried out along with the nesting test directive.
	<code>input {ordered} : Element [*]</code> Refers to the sequence of NamedElements on which the test directive operates on.
	<code>output {ordered} : NamedElement [*]</code> Refers to the sequence of NamedElements which are generated while carrying out the test directive.
Change from UTP 1.2	«TestDirective» has been newly introduced by UTP 2.1.

### 8.8.2.2 TestDirectiveStructure

Description	The stereotype «TestDirectiveStructure» enables the definition of user-defined or context-specific additional information that augments a test directive. A Classifier with «TestDirectiveStructure» applied might be of arbitrary complexity. It enables the provision of information that is relevant in a certain context.
Extension	<a href="#">Classifier</a>
Sub Class	<a href="#">TestDesignDirectiveStructure</a>
Change from UTP 1.2	«TestDirectiveStructure» has been newly introduced by UTP 2.1.

### 8.8.2.3 TestTechnique

Description	<p>A test technique is the specification of a test-related task used to carry out test-related tasks manually or automatically. Test techniques are assembled and governed by test directives. Information visible to the assembling test directive are transitively visible and accessible to the assembled test technique.</p> <p>A test technique may define sub-techniques. Providing a sub test technique enables testers to refine the given test techniques with respect to certain elements contained in the test directive input and also to enrich existing (potentially pre-defined) test techniques with user-defined respectively context-specific information.</p>
Extension	<a href="#">InstanceSpecification</a>
Sub Class	<a href="#">TestDesignTechnique</a>
Associations	<code>/instanceOf : TestTechniqueStructure [*]</code> Refers to the test technique structures of which the given test directive is an instance of. The set of test technique structures is derived from all Classifiers with «TestTechniqueStructure» applied that are referred to as classifiers by the underlying InstanceSpecification. <code>/subTestTechnique {union, read-only} : TestTechnique [*]</code> The set of nested test techniques that augment the given test technique.
Change from UTP 1.2	«TestTechnique» has been newly introduced by UTP 2.1.

### 8.8.2.4 TestTechniqueStructure

Description	The stereotype «TestTechniqueStructure» enables the definition of user-defined or context-specific additional information that augments a test technique. A Classifier with «TestTechniqueStructure» applied might be of arbitrary complexity. It enables the provision of information that is relevant in a certain context.
Extension	<a href="#">Classifier</a>
Sub Class	<a href="#">TestDesignTechniqueStructure</a>
Change from UTP 1.2	«TestTechniqueStructure» has been newly introduced by UTP 2.1.



## 9 Model Libraries

This section describes a set of type libraries relevant to UTP.

### 9.1 UTP Types Library

#### 9.1.1 Predefined types

The following diagram shows the predefined types provided by UTP 2.

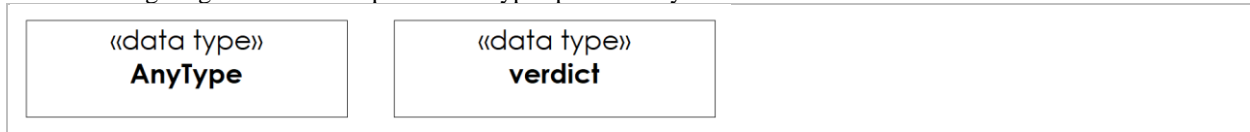


Figure 9.1 - Predefined types

Name	Description
AnyType	The pre-defined type <a href="#">AnyType</a> is the least common ancestor of any type known in the context of a certain test type system. As a result, StructuralFeatures typed with <a href="#">AnyType</a> can be assigned any value, regardless whether primitive or complex.
verdict	The pre-defined type <a href="#">verdict</a> represents the basis for the <a href="#">verdict</a> -related mechanisms and user-specific extensions thereof. Tester may subclass the <a href="#">verdict</a> type in order to define specialized <a href="#">verdict</a> types.

#### 9.1.2 Predefined verdict instances

The [verdict](#) instances predefined by UTP 2 are [none](#), [pass](#), [inconclusive](#), [fail](#) and [error](#). Test modellers can make use of those predefined verdicts out of the box to avoid redundancy.

There is a predefined (default) precedence rule for these [verdicts](#), with ascending precedence from left to right: none < pass < inconclusive < fail < error. That means that setting a verdict is a one-way street. It is not permitted to re-assign a verdict with lower precedence to a test set, test case or procedural element, whereas the other way round, verdicts with higher precedence may override verdicts with lower precedence at any point in time during verdict calculation process. The default verdict precedence reflects the default arbitration specification semantics. This semantics can be modified or even completely overridden by user-defined arbitration specifications. If any additional user-defined verdict types are introduced (e.g., complex verdict types and user-defined instances thereof), it is left open how precedence of those user-defined verdicts and the default verdicts integrate with each other.

Even though the predefined verdict instances are expressed using InstanceSpecifications, it is not forbidden to use other representation formats such as LiteralString, Expression or even OpaqueExpression to express user-defined verdict instances in a UTP-based test model.



Figure 9.2 - Predefined verdict instances

Name	Description
error	The predefined verdict 'error' indicates a result of a test set, test case or procedural element, where a non-test item related problem occurred. This might be a technical problem in the test environment (e.g., breakdown of a network connection that is required for executing the test case), a malfunction of a component in the test environment or an incorrectly executed test procedure, test case or test set. 'Error' differs from a 'fail' in a sense that the test item did not cause the deviation between the expected and the actual

Name	Description
	responses.
fail	The predefined <a href="#">verdict 'fail'</a> indicates a result of a <a href="#">test set</a> , <a href="#">test case</a> or <a href="#">procedural element</a> , where the <a href="#">test item</a> does not react as expected.
inconclusive	<p>The predefined <a href="#">verdict 'inconclusive'</a> indicates that a situation where it is not possible to determine whether the <a href="#">test item</a> behaved as expected or not. It is, however, not predefined when the <a href="#">verdict 'inconclusive'</a> shall be assigned. This depends on the rules of the applied <a href="#">arbitration specification</a>. The default <a href="#">arbitration specifications</a> do not utilize this <a href="#">verdict</a> instance.</p> <p>The concept was obtained from [ISO/IEC 9646-1] where it says: "Test <a href="#">verdict</a> given when the observed test outcome is such that neither a pass nor a fail <a href="#">verdict</a> can be given".</p>
none	The predefined <a href="#">verdict 'none'</a> indicates that a situation where either a <a href="#">test set</a> , <a href="#">test case</a> or <a href="#">procedural element</a> has not yet been executed, or <a href="#">verdict</a> calculation has not yet taken place (e.g., in post-execution comparison).
pass	The predefined <a href="#">verdict 'pass'</a> indicates a result of a <a href="#">test set</a> , <a href="#">test case</a> or <a href="#">procedural element</a> , where both the tester but in particular the <a href="#">test item</a> behaved, respectively responded as expected.

## 9.2 UTP Auxiliary Library

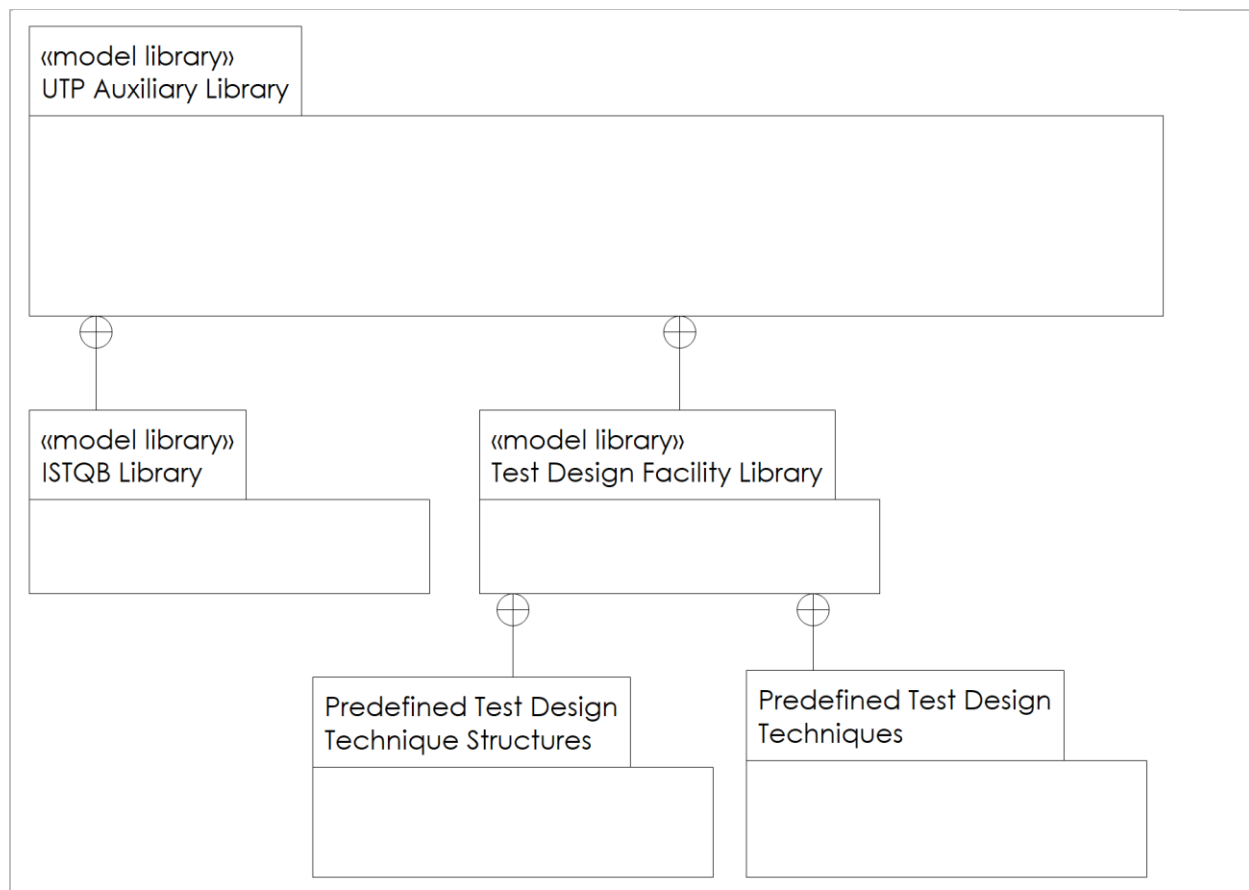
### 9.2.1 UTP Auxiliary Library

The UTP auxiliary library collects well-established and commonly accepted information whose use is optional. The purpose of the auxiliary library is to provide users with a set of useful and predefined types and values to foster reusability across modeling tools and approaches. For example, the ISO 25010 quality model is supposed to be used by multiple organizational units within the test process. Instead of building proprietary and potentially technically conflicting representations of the very same quality model, users may reuse the ISO 25010 [\[ISO25010\]](#) quality model that comes along with UTP itself. Of course, such types and values are often tailored to specific needs (e.g., Robustness testing is a frequently used testing type which is actually given in ISO 9216 or ISO 25010), but still needs to be specified. However, the existence of the UTP auxiliary model does not prevent such an approach.

#### 9.2.1.1 The UTP auxiliary library

Overview of the UTP auxiliary library.





**Figure 9.3 - The UTP auxiliary library**

### 9.2.1.2 ISTQB Library

The ISTQB library offers concepts that can be used to organize some aspects of the test process, if required. In particular, the ISTQB library offers a commonly used set of [test levels](#) and [test set purposes](#).

#### 9.2.1.2.1 Overview of the ISTQB library

The following diagram shows the predefined test process library provided by UTP to be used for the specification of [test contexts](#) and [test sets](#).

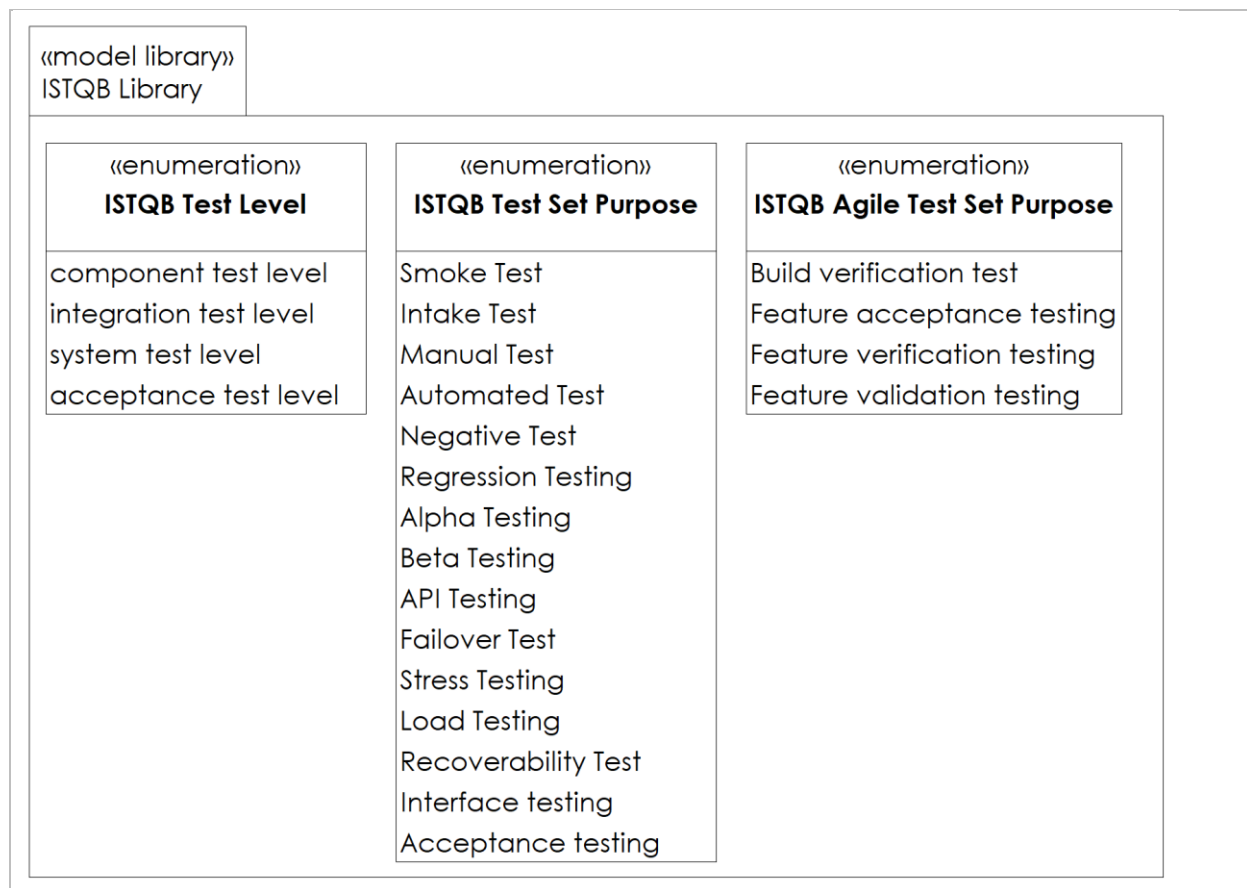


Figure 9.4 - Overview of the ISTQB library

Name	Description	Enumeration literals
ISTQB Agile Test Set Purpose		Build verification test "A set of automated tests which validates the integrity of each new build and verifies its key/core functionality, stability and testability. It is an industry practice when a high frequency of build releases occurs (e.g., Agile projects) and it is run on every new build before the build is released for further testing." [ISTQB]
		Feature acceptance testing Acceptance testing of a feature, often broken down into <a href="#">Feature verification testing</a> and <a href="#">Feature validation testing</a> .
		Feature verification testing Usually carried out automatically may be done by developers or testers and involves testing against the user story's acceptance criteria.
		Feature validation testing Usually carried out manually and can involve developers, testers, and business stakeholders working collaboratively to determine whether the feature is fit for use, to improve visibility of the progress made, and

Name	Description	Enumeration literals
		to receive real feedback from the business stakeholders.
ISTQB Test Level	A common set of test levels. A test level is considered as a set of testing activities related to the outermost boundaries of the test items.	component test level A test designed to provide information about the quality of the component.
		integration test level A test designed to provide information about the direct interface between two integrated components, for example in the form of a parameter list.
		system test level A test designed to assess the quality of the complete system after integration.
		acceptance test level A test designed to demonstrate to the customer the acceptability of the final system in terms of their specified requirements.
ISTQB Test Set Purpose	A set of reasons why <a href="#">test sets</a> might have been assembled.	Smoke Test "A subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertaining that the most crucial functions of a program work, but not bothering with finer details." <a href="#">[ISTQB]</a>
		Intake Test "A special instance of a smoke test to decide if the component or system is ready for detailed and further testing. An intake test is typically carried out at the start of the test execution phase." <a href="#">[ISTQB]</a>
		Manual Test A test set whose test cases will be executed manually.
		Automated Test A test set whose test cases will be executed automatically.
		Negative Test "Tests aimed at showing that a component or system does not work." <a href="#">[ISTQB]</a>
		Regression Testing "Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made." <a href="#">[ISTQB]</a>
		Alpha Testing "Simulated or actual operational testing by potential customers/users or an independent test team at the software developers' site, but outside the development organization. Alpha testing is employed for off-the-shelf software as a form of internal acceptance testing." <a href="#">[ISTQB]</a>

Name	Description	Enumeration literals
		<p><b>Beta Testing</b></p> <p>"Operational testing by potential and/or existing customers/users at an external site not otherwise involved with the developers, to determine whether or not a component of system satisfies the user needs and fits within the business processes. Note: Beta testing is often employed as a form of external acceptance testing in order to acquire feedback from the market." <a href="#">[ISTQB]</a></p>
		<p><b>API Testing</b></p> <p>"Testing the code which enables communication between different processes, programs and/or systems. API testing often involves negative testing, e.g., to validate the robustness of error handling." <a href="#">[ISTQB]</a></p>
		<p><b>Failover Test</b></p> <p>"Testing by simulating failure modes or actually causing failures in a controlled environment. Following a failure, the failover mechanism is tested to ensure that data is not lost or corrupted and that any agreed service levels are maintained (e.g., function availability or response times)." <a href="#">[ISTQB]</a></p>
		<p><b>Stress Testing</b></p> <p>"A type of performance testing conducted to evaluate a system or component at or beyond the limits of its anticipated or specified workloads, or with reduced availability of resources such as access to memory or servers. [After IEEE 610]" <a href="#">[ISTQB]</a></p>
		<p><b>Load Testing</b></p> <p>"A type of performance testing conducted to evaluate the behavior of a component or system with increasing load, e.g. number of parallel users and/or numbers of transactions to determine what load can be handled by the component or system." <a href="#">[ISTQB]</a></p>
		<p><b>Recoverability Test</b></p> <p>"The process of testing to determine the recoverability of a software product." <a href="#">[ISTQB]</a></p>
		<p><b>Interface testing</b></p> <p>"An integration test type that is concerned with testing the interfaces between components or systems." <a href="#">[ISTQB]</a></p>
		<p><b>Acceptance testing</b></p> <p>"Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to</p>

Name	Description	Enumeration literals
		accept the system." <a href="#">[ISTQB]</a>

### 9.2.1.3 Test Design Facility Library

The test design facility library provides a set of [test design techniques](#) as well as some default test design technique structures that can be used out of the box for the specification of the test design activities. Since these [test design techniques](#) are by definition not dependent upon the [test design input](#) element, they are called context-free [test design techniques](#).

#### 9.2.1.3.1 The UTP test design facility library

The following diagram shows the predefined [test design techniques](#) provided by UTP 2 to be used for the specification of test directives.

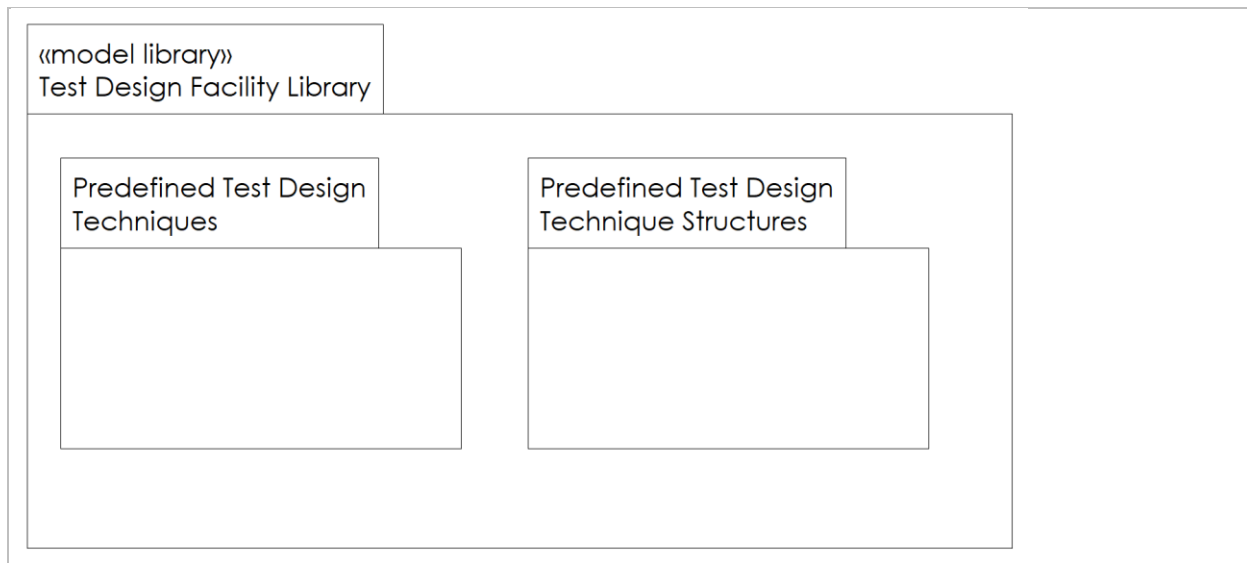


Figure 9.5 - The UTP test design facility library

#### 9.2.1.3.2 Predefined Test Design Techniques

UTP offers a set of context-free [test design techniques](#), meaning that these [test design techniques](#) do not require any further information from the [test design input](#) of the assembling [test design directive](#). They can be immediately used by the generic [test design directive](#) or any other predefined or specialized [test design technique](#) or test design directive.

##### 9.2.1.3.2.1 Predefined context-free test design techniques

The following diagram depicts the predefined and ready-to-use [test design technique](#) provided by UTP 2.



Figure 9.6 - Predefined context-free test design techniques

Name	Description
AllCombinations	A predefined instance of the CombinatorialTesting TestDesignTechnique ready for being assembled by TestDesignDirectives. The semantics is that all possible combinations of input parameters must be covered by the resulting test cases.
AllRepresentatives	A predefined instance of the <a href="#">EquivalenceClassPartitioning TestDesignTechnique</a> ready for being assembled by <a href="#">TestDesignDirectives</a> . All representatives of the equivalence classes must be selected.
AllStates	The predefined instance of the StateCoverage TestDesignTechnique ready for being assembled by TestDesignDirectives. The default semantics is that all States of the corresponding State Machine(s) must be covered by the resulting test cases.
AllTransitions	The predefined instance of the TransitionCoverage TestDesignTechnique ready for being assembled by TestDesignDirectives. The default semantics is that all Transitions of the corresponding State Machine(s) must be covered by

Name	Description
	the resulting test cases.
DefaultCBT	The predefined instance of the ChecklistBasedTesting TestDesignTechnique ready for being assembled by TestDesignDirectives.
DefaultCET	The predefined instance of the CauseEffectAnalysis <a href="#">TestDesignTechnique</a> ready for being assembled by <a href="#">TestDesignDirectives</a> .
DefaultCTM	The predefined instance of the ClassificationTreeMethod TestDesignTechnique ready for being assembled by TestDesignDirectives.
DefaultDTT	The predefined instance of the DecisionTableTesting TestDesignTechnique ready for being assembled by TestDesignDirectives.
DefaultEG	The predefined instance of the ErrorGuessing TestDesignTechnique ready for being assembled by TestDesignDirectives.
DefaultET	The predefined instance of the ExploratoryTesting TestDesignTechnique ready for being assembled by TestDesignDirectives.
DefaultPT	The predefined instance of the PairwiseTesting TestDesignTechnique ready for being assembled by TestDesignDirectives.
DefaultTPT	The predefined instance of the TransitionPairTesting TestDesignTechnique ready for being assembled by TestDesignDirectives. The default semantics is that at least all pairs of subsequent Transitions must be covered by the resulting test cases.
OneBoundaryValue	The predefined instance of the <a href="#">BoundaryValueAnalysis TestDesignTechnique</a> ready for being assembled by <a href="#">TestDesignDirectives</a> . The default semantics is that a single value at the boundaries of the equivalence class must be selected.
OneRepresentative	A predefined instance of the EquivalenceClassPartitioning TestDesignTechnique ready for being assembled by TestDesignDirectives. Exactly one representative of each equivalence class must be selected.

### 9.2.1.3.3 Predefined Test Design Technique Structures

The predefined test design technique structures offer some structural information to enrich test design techniques, if required.

#### 9.2.1.3.3.1 Overview of the predefined test design technique structures

The following diagram depicts the predefined and ready-to-use [test design technique](#) structures provided by UTP. They can be used to build proprietary generic [test design techniques](#) or to augment the predefined [test design techniques](#).

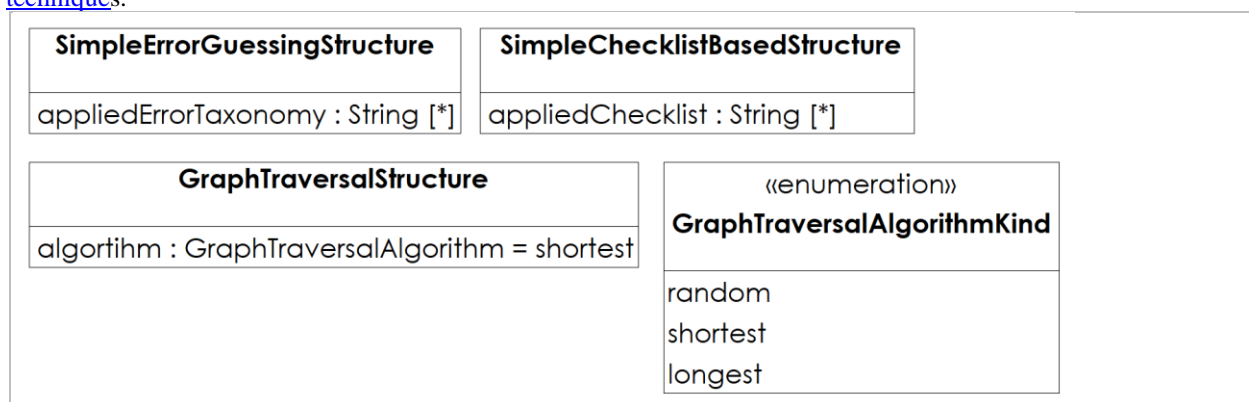


Figure 9.7 - Overview of the predefined test design technique structures

Name	Description
GraphTraversalStructure	A <a href="#">test design technique</a> structure that enables testers to specify the traversal algorithm a test designing entity must apply.
SimpleChecklistBasedStructure	A checklist-based test design technique that enables test engineers to refer to some checklists that should be used for test design.
SimpleErrorGuessingStructure	An error guessing <a href="#">test design technique</a> that enables test engineers to refer to some error taxonomies that should be used for test design.

Name	Description	Enumeration literals
GraphTraversalAlgorithmKind	A set of graph traversal strategies.	random A test designing entity must take a random walk through the graph in order to achieve a certain coverage criterion of the test design input element.
		shortest A test designing entity must take the shortest path possible in order to achieve a certain coverage criterion of the test design input element.
		longest A test designing entity must take the longest path possible to achieve a certain coverage criterion of the test design input element.



## Annex A (Informative): Examples

This section illustrates some concepts of the UML Testing Profile by means of different examples. These examples were provided by different companies reflecting different approaches to MBT, different interpretations of MBT with UTP and finally different methodologies for applying UTP. It underlines the flexibility and open-endedness of UTP.

### A.1 Croissants Example

#### A.1.1 The Test Item

This example illustrates some of the major concepts of UTP 2 on the "not so serious" [test item](#) (French) "Croissants". This is a particularly interesting example since the [test item](#) is not a software system (at least not in the classical sense ;-), but a rather common physical system (i.e., croissants).



Figure A.1 - The Croissants Example

#### A.1.1.1 Given Requirements on the Test Item

Id	Type	Description	Req. on
RQ-0001	functional	Each croissant shall have a chocolate core	Croissant
RQ-0002	functional	Each croissant shall have a consistency of greater than 3	Croissant
RQ-0003	functional	Each croissant shall be considered as "good tasting" by more than 80% of ordinary people	Croissant

## A.1.2 Test Requirements

The following diagram shows the hierarchy of test objectives as well as the constraints on this test series expressed as test requirements.

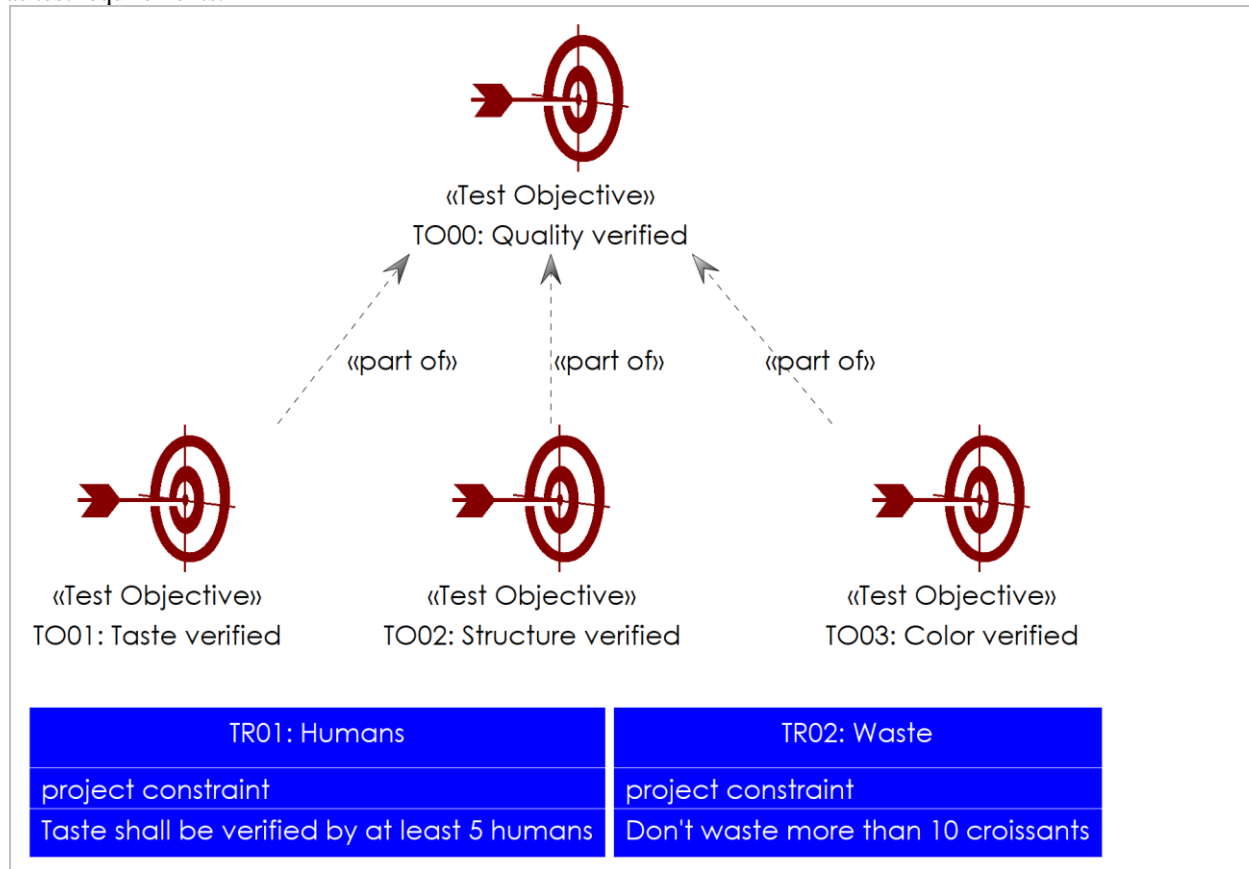


Figure A.2 - Test Objectives

### A.1.2.1 Given Test Objectives

Name	Description	Priority
TO00: Quality verified	The high quality of the croissants we enjoy during our working meetings is ensured.	n/a
TO01: Taste verified	The quality of the flavor of the croissants we enjoy during our working meetings is ensured.	high
TO02: Structure verified	The physical composition of the croissants we enjoy during our working meetings is ensured.	medium
TO03: Color verified	The tasteful look of the croissants we enjoy during our working meetings is ensured.	high

### A.1.2.2 Given Requirements

TR01: Humans	
Description	Taste shall be verified by at least 5 humans
Requirement type	project constraint
Requirement kind	Quality

TR02: Waste	
Description	Don't waste more than 10 croissants
Requirement type	project constraint
Requirement kind	Resource Consumption

### A.1.3 Test Design

The following diagram shows the applied test design strategy as well as the test directives derived from that test design strategy.

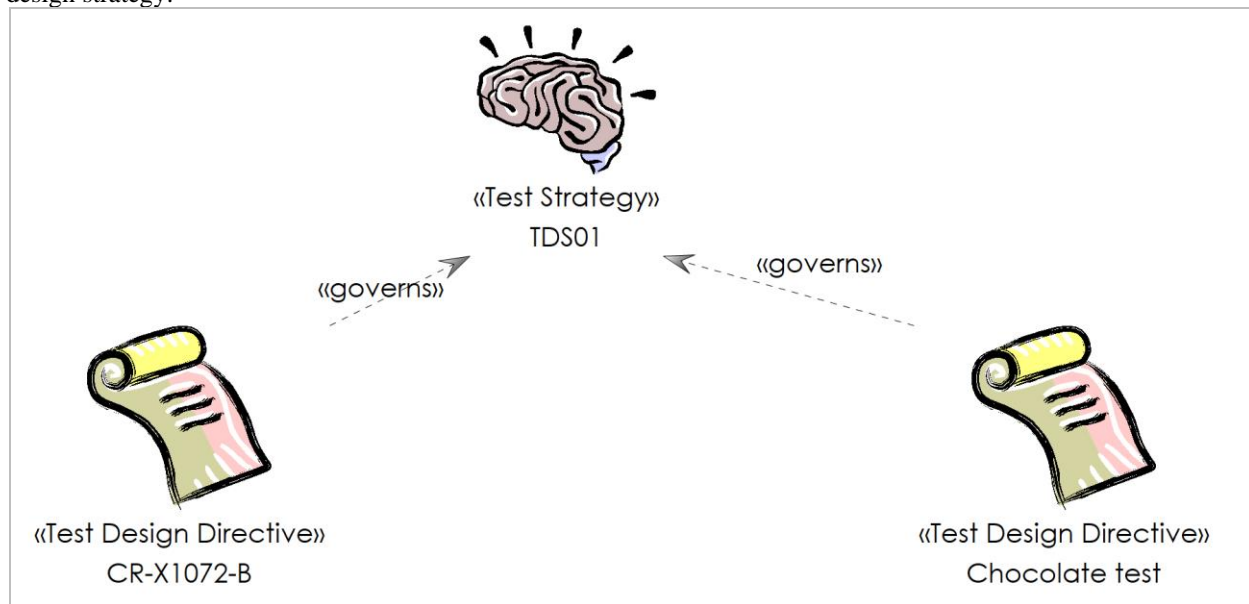


Figure A.3 - Test Strategy

#### A.1.3.1 Test Design Strategies shown on "Test Strategy"

TDS01	
Description	At least 5 members of the UTP 2 will take a bite of a croissant.

#### A.1.3.2 Test Directives shown on "Test Strategy"

Chocolate test	
Description	Keep every piece of chocolate at least 10 seconds on your tongue.
Applies to	Chocolate Portion
Requires capability	Gustaoceptionary Proficiency

CR-X1072-B	
Description	Apply Croissant-Standard CR-X1072-B to test them.
Applies to	Croissant
Requires capability	Knowledge of CR-X1072-B

### A.1.4 Test Configuration

The figure below shows the Test Configuration of the Croissants abstracted as a UML class diagram.

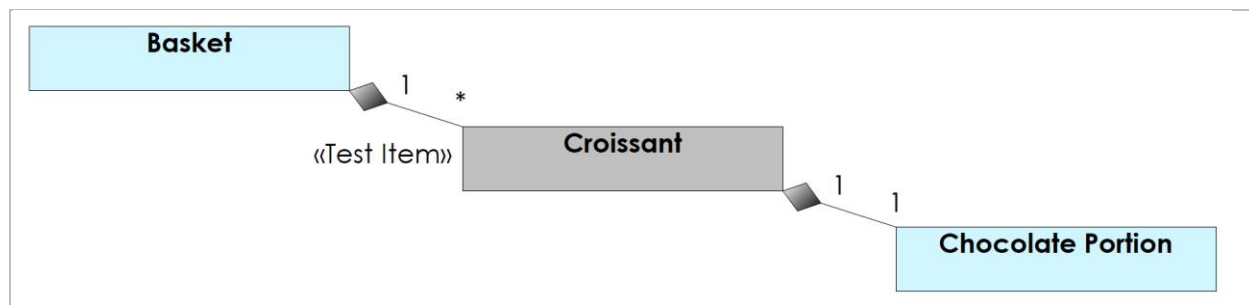


Figure A.4 - Objects

Based on this description, the following figure shows the [concrete test configuration](#) instantiated as a composite structure diagram.

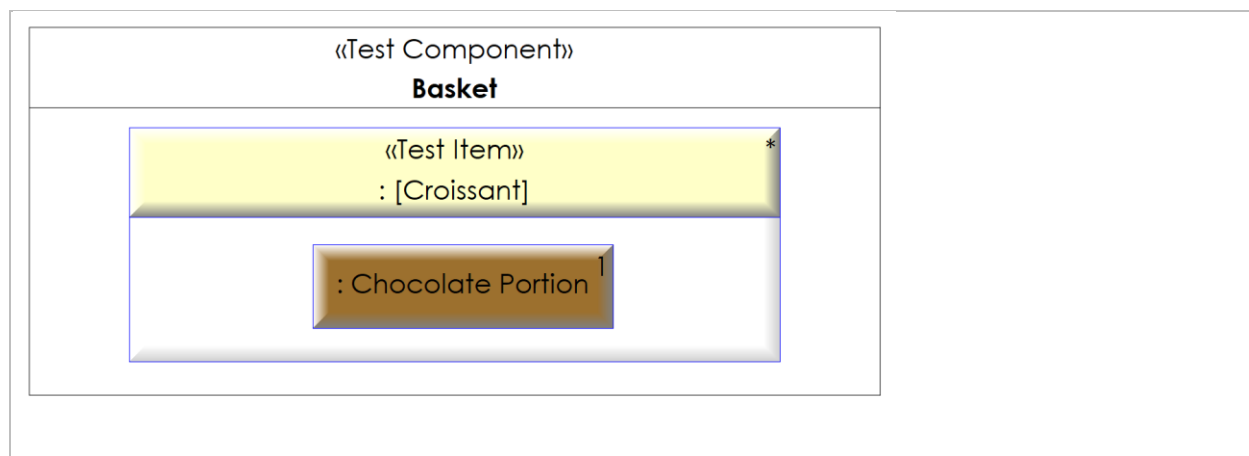


Figure A.5 - Test Configuration

## A.1.5 Test Cases

The [test cases](#) (particularly the [test procedures](#)) in this [test set](#) are not specified fully and formally but rather in a structured informal way. This is to show that [test cases](#) in UTPs don't always have to be fully formalized.

### A.1.5.1 Test Set "Manual croissants test"

The following diagram shows the [Test Set "Manual croissants test"](#) containing the relevant test cases and how they relate to the stated test objectives. Further, the test requirements constraining this test set also are shown.

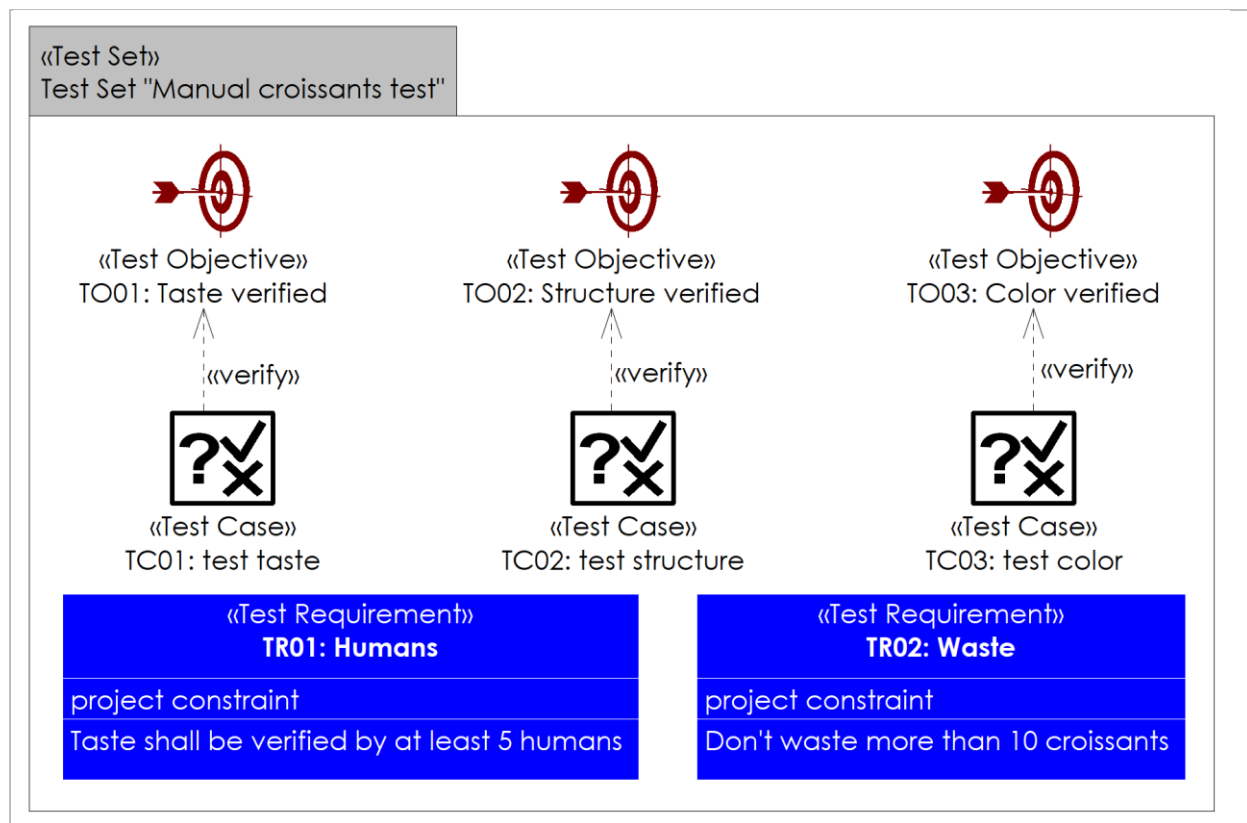


Figure A.6 - Test Map

#### Test Cases shown on "Test Map"

TC01: test taste	
Test objectives	<a href="#">TO01: Taste verified</a>
Priority	high
Precondition	<ul style="list-style-type: none"> <li>There must be a Croissant available</li> </ul>
Test procedure	Apply the following steps: <ul style="list-style-type: none"> <li>Break the Croissant in its middle</li> <li>Check whether there is chocolate in it</li> <li>Bite into the Croissant</li> <li>Evaluate its taste</li> <li>Eat the remains or throw them into the waste basket</li> </ul>
Postcondition	<ul style="list-style-type: none"> <li>The Croissant is eaten</li> </ul>
Verifies	<a href="#">TO01: Taste verified</a>
Estimated effort	10 seconds
Is abstract	FALSE

TC02: test structure	
Test objectives	<a href="#">TO02: Structure verified</a>
Priority	low
Precondition	<ul style="list-style-type: none"> <li>There must be a Croissant available</li> <li>The Croissant must not be broken</li> </ul>
Test procedure	Apply the following steps: <ul style="list-style-type: none"> <li>Press the Croissant with two fingers</li> <li>Check the elasticity of the Croissant</li> </ul>

	<ul style="list-style-type: none"> <li>• Bend the Croissant until it breaks</li> <li>• Check the breaking angle</li> <li>• Eat the remains or throw them into the waste basket</li> </ul>
Postcondition	• The Croissant is broken
Verifies	<a href="#">TO02: Structure verified</a>
Estimated effort	20 seconds
Is abstract	FALSE

TC03: test color	
Test objectives	<a href="#">TO03: Color verified</a>
Priority	medium
Precondition	• There must be a Croissant available
Test procedure	Apply the following steps: <ul style="list-style-type: none"> <li>• Look at the Croissant</li> <li>• Evaluate its color</li> </ul>
Postcondition	• There is still a Croissant available
Verifies	<a href="#">TO03: Color verified</a>
Estimated effort	5 seconds
Is abstract	FALSE

## A.2 LoginServer Example

The LoginServer example represents a simplified version of a real case study taken from the EU FP7 research project REMICS. It was optimized for the initial submission section to demonstrate the core concepts of UTP 2 that are stable enough and unlikely to be substantially changed in the revised submissions. The LoginServer offers functionality to log into a system (in the mentioned REMICS project, the login functionality was integrated into a Cloud-based system for managing travel excursions). In this example section, the following capabilities of UTP 2 are demonstrated:

- Defining the structure of a test plan using [test contexts](#) as well as [test level](#) and [test types](#).
- Specification of [test requirements](#) as a result of the test analysis activities.
- Modeling of the logical interfaces of the [test item](#) (also known as [test item](#) - [test item](#)) optimized for deriving logical [test cases](#).
- Modeling of the [test type](#) system and [data specifications](#) required for deriving appropriate [data](#).
- Specification of structural aspects of the test environment, in particular the required [test components](#), [test configuration](#) and connection between the test environment and the [test item](#).
- Modeling of logical [test cases](#) using sequence diagrams (i.e., Interactions).
- Informal and rough description of a mapping from UTP 2 [test cases](#) expressed as sequence diagrams (i.e. Interactions) to semantically equivalent TTCN-3 test scripts.

This example demonstrates the Test Model-only approach to model-based testing. There are no further (e.g., design or requirements) models available for reuse. In addition, the methodology follows the so called [test requirement/requirements analysis](#), since the test design activities are guided by [test requirements](#) which, in turn, are derived from the test basis. Both the applied MBT approach and the test approach (which is called test practice in ISO 29119) of the LoginServer example are just a single interpretation of how UTP 2 could be used and embedded into a methodology. The described test process and its distinct phases (e.g., test planning, test analysis, etc.) are inspired by the ISTQB fundamental test process.

## A.2.1 Requirements Specification

The following table shows a simplified excerpt of the requirements for the LoginServer example. These few requirements suffice to demonstrate most of the core concepts of UTP 2.

**Table A.9.1 - LoginServer Requirements**

<b>Id</b>	<b>Name</b>	<b>Description</b>
F1	User login	The user shall be able to log into the system using a valid ID/password combination.
F2	Failed user login	The system shall reject the login request and answer with an appropriate error message, if the user tries to log into the system with a known ID but invalid password.
F3	Unknown user login	The system shall reject the login request and answer with an appropriate error message, if an unknown user (i.e., a non-registered ID) requests a login.
F4	User banishing	The system shall banish an ID and answer with an appropriate message, if a user tries to log into system three times in a row with an invalid ID/password combination.
F5	Mail address modification	A user who is logged into the system shall be able to update his mail address. A valid mail address complies to the following regular expression: [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}
F6	User logout	A user who is logged in shall be able to log out from the system.
F7	Login response time	The system shall respond to login request within 5 seconds.

## A.2.2 Test Planning

In the test planning phase, the test manager usually starts specifying the test plan. This means that the resources for testing are estimated, requested and allocated. Furthermore, the test process is broken down into so called test sub-processes, each strives to fulfil the [test objectives](#) of this test sub-process. These test sub-processes are called [test context](#) in UTP 2.

Based on the knowledge about the system to be tested (also known as [test item](#) or [test item](#)), the test manager decides on the number of test sub-processes, their objectives and the strategies used to fulfil those [test objectives](#). The diagram below shows the corresponding structure of the test specification for the LoginServer [test item](#).



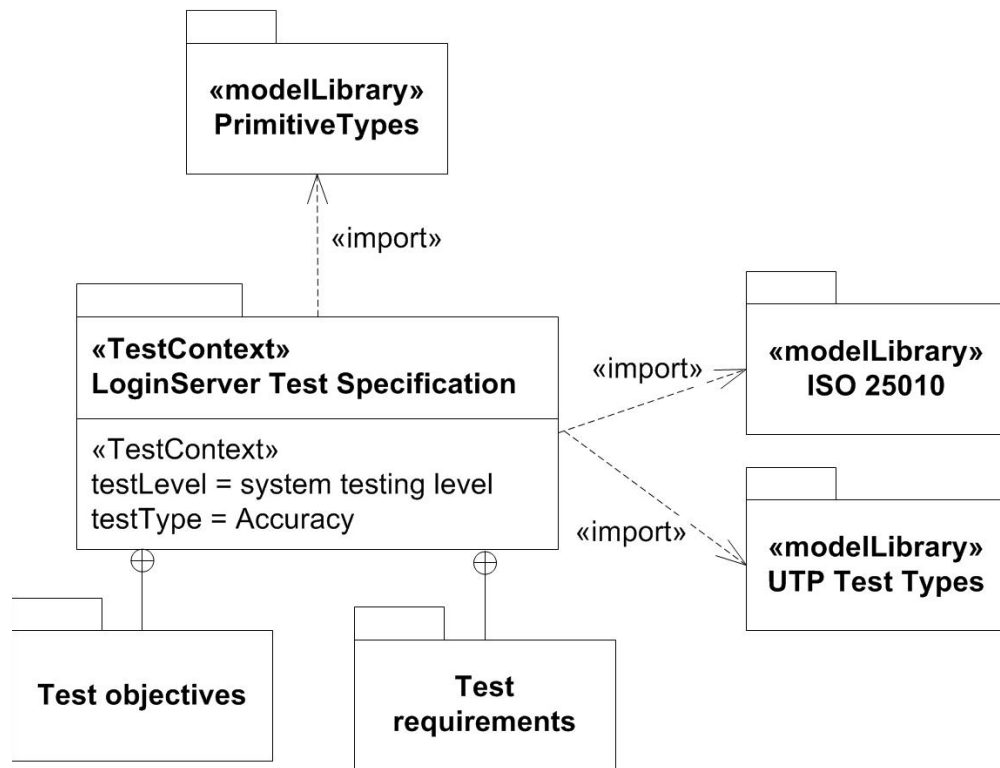


Figure A.7 - The LoginServer Test Context

Due to the simplicity of the LoginServer, the entire test plan only consists of a single [test context](#). In more sophisticated test processes, the test plan is usually sub-structured into multiple test (sub-)plans, so called master and level test plans. The [test context](#) copes with this need, since it allows for sub-structure [test contexts](#). The diagram above also demonstrate the use of two model libraries provided by the UTP Auxiliary library in order to specify the [test level](#) and [test type](#) that are addressed by the given [test context](#). In this example, the [test context](#) LoginServer Test Specification targets functional system testing. Each [test case](#) accessible to the [test context](#) is considered to be designed for the mentioned [test level](#) and [test type](#). This enables tester to apply the very same [test case](#) to different [test types](#) and [test levels](#) (if needed). For example, it is a good practice to reuse functional [test cases](#) with different [data](#) sets or a different, yet compatible [test configuration](#) for security or performance testing.

The LoginServer Test Specification contains two ordinary packages for storing the [test objectives](#) and [test requirements](#). Whereas the specification of [test objectives](#) is not shown in this example, the derivation of [test requirements](#) as one of the most important outcomes of the test analysis phase will be shown in the next section.

## A.2.3 Test Analysis

The activities in the test analysis phase are, first and foremost, dedicated to analyzing the test basis in order to comprehend both the test item and what is expected from the test item. Test basis is an abstract concept that comprises any information that helps deriving test cases for a certain test item with respect to the test objectives of the given test sub-process (i.e., test context). The requirements specification usually represents an important part of the test basis for functional system testing.

### A.2.3.1 Derivation and Modeling of Test Requirements

In UTP, [test requirements](#) specify which features of a requirement should be verified by corresponding [test cases](#). [test requirements](#) are an important means to establish traceability between [test cases](#) and the test basis, in particular the requirements. The degree of detail of [test requirements](#) varies between test processes and depends on different aspects like the applied test methodology, details of the test basis, sufficient time available to actually specify, review and validate those [test requirements](#) etc.



For the given example, only a subset of all possible [test requirements](#) is derived from the functional system requirements. For later submission, this specification will provide a more elaborated and complete example.

**Table 9.2 - Test Requirements**

Id	Description	Covers	Comments
TR-F1-1	Ensure that a user successfully logs into the system, if the login request is performed using a valid ID and corresponding password.	User login	No information about response of the definition of valid ID yet. Req. change request submitted (RCR-ID: 0015)
TR-F1-2	Ensure that the system responses with an error message “Invalid ID” if an invalid ID was provided with the login request.	User login	Invalid ID behavior discussed with system architect. An according req. change request was submitted (RCR-ID: 0016)
TR-F5-1	Ensure that the system responses with a message “Mail address updated” if the modification request was successful. This requires a valid mail address. Valid mail addresses shall comply with the following regular expression: [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}	Mail address modification	No information about response of the system available yet. Req. change request submitted (CR-ID: 0064). The current expected result is very likely to change in future.
TR-F5-2	Ensure that the system issues an error message “Invalid Format” if the mail address the user submitted for modification does not comply with the regular expression given in F5.	Mail address modification	No information about response of the system available yet. Req. change request submitted (CR-ID: 0065). The current expected result is very likely to change in future.
TR-F5-3	Ensure that the system rejects the modification request if the user is not logged into the system with the error message “Login required”.	Mail address modification	No information about response of the system available yet. Req. change request submitted (CR-ID: 0065). The current expected result is very likely to change in future.
TR-F6-1	Ensure that a user, who is currently logged into the system and requests a logout from the system, is actually logged out. The system shall respond with a message “User logged out”	User logout	
TR-F6-2	Ensure that the system responds with an error message “Logout requires to be logged in” if a user who is not logged into the system sends a logout request.	User logout	
TR-F7-1	Ensure that the system responds to login requests within 5 seconds.	Login response time	

The diagram below depicts the content of the corresponding [test requirement](#) package. To keep the diagram clean, only unique identifier of the [test requirements](#) are shown. In this methodology, [test requirements](#) do not have a name, so the name is automatically (by virtue of a UTP 2 tool) kept in synch with the unique identifier. Unfortunately and deliberately for this example, the targeted requirements are not available as model [artifacts](#), but stored somewhere else (e.g., a dedicated requirements management tool like DOORS or even Excel). Traceability from [test requirements](#) to requirements (i.e., from the test specification to the test basis) by means of UTP 2 can at most be established informally.

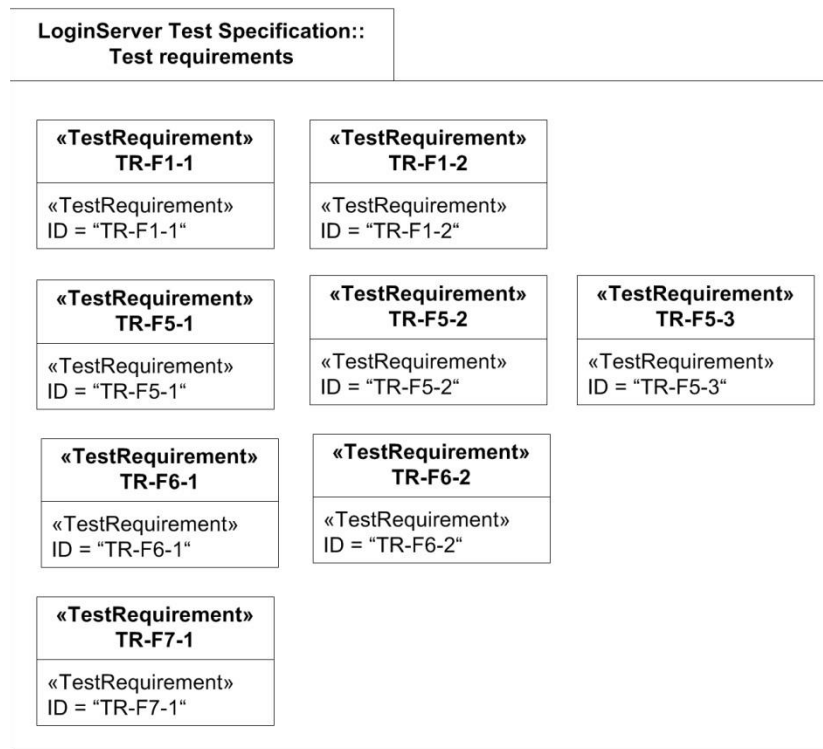


Figure A.8 - Test Requirements

#### A.2.3.2 Modeling the Type System and Logical Interfaces

Since the test model is designed in a standalone manner, it is in the responsibilities of the test analysts to identify and specify the means for interacting with the [test item](#). [test requirements](#) usually provide further information for the design of the logical interfaces of the [test item](#) and the [test type](#) system used for information exchange. For example, the phrase “a user ... logs into the system if the login request is performed using a valid ID and corresponding [Password](#).” indicates that has to be an operation that allows providing an ID and a [Password](#) for a login request. Of course, the same holds true, of course, for the specification of constraints on data in order to build [data specifications](#). The [test requirements](#) TR-F1-1 and TR-F5-1 are examples in which constraints on data are specified. These data constraints could be exploited for data-based test design strategies like equivalence class partitioning or boundary value analysis. Whatever [test design technique](#) will be applied, UTP 2 offers the required capabilities to capture such data constraints and explicitly specify [data specification](#) as means of equivalence classes or even classification trees.

The diagram below shows the logical interface operations and [test type](#) systems derived from the [test requirements](#) TR-F1-1, TR-F2-1, TR-F6-1 and TR-F6-2.

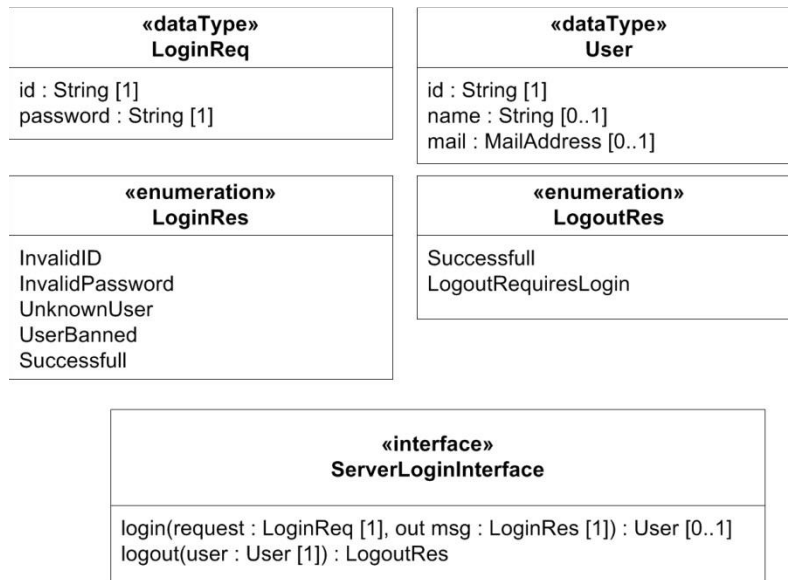


Figure A.9 - Logical Interface of LoginServer (1)

The diagram below depicts the logical interface operations and [test type](#) systems derived from the [test requirements](#) TR-F5-1, TR-F52, and TR-F5-3.

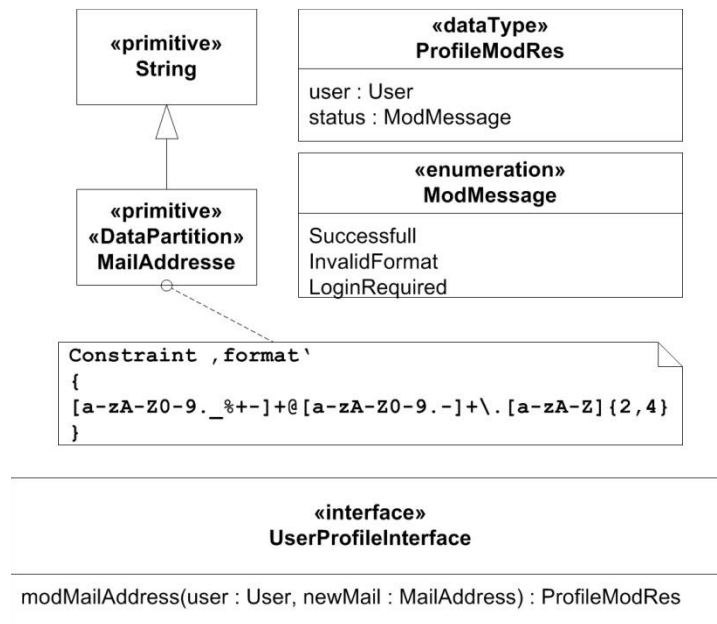


Figure A.10 - Logical Interface of LoginServer (2)

### A.2.3.3 Modeling Test Data

The [data specification](#) MailAddress specialized the primitive type String (provided by the UML PrimitiveTypes package imported by the surrounding [test context](#)) and restricts the values for this type according to requirement F5 and [test requirements](#) TR-F5-1. The actual specification of the Constraint 'format' is represented by a LiteralString (this cannot be inferred by the means of the diagram). The diagram below shows the corresponding object diagram of the relevant parts of the diagram above.

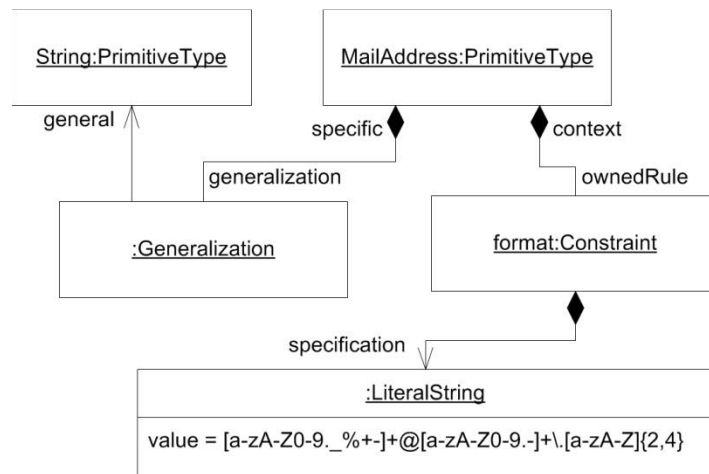


Figure A.11 - Object Diagram specifying data

Both names and representation of derived [artifacts](#) are just examples how UTP 2 could be applied to support test analysis and depend on the respective methodology.

## A.2.4 Test Design

The main target of the test design activity is to derive test cases by following either systematic test design techniques or in an ad-hoc manner. However, performed, the test design activity is responsible for:

- Deriving according to test data based on the test type system
- Deriving the test architecture and test configuration including the communication channels between the test components and the test item
- Designing test cases based on the findings of the test analysis activities
- Link test cases to test objectives and/or test requirements

### A.2.4.1 Test Architecture and Test Configuration

The test architecture comprises of the [test item](#) and the corresponding [test components](#) required driving the execution of [test cases](#) against the [test item](#). The diagram below depicts the specification of two components within the LoginServer Test Specification. The decision made to go for two distinct interfaces for the LoginServer instead of a single interface results in a bigger modeling efforts, since an interface component (see BasicPortConfiguration) is required in order to offer multi-offering Ports. This diagram does not make use of any UTP 2 stereotypes but relies completely on the class modeling capabilities of UML. The Port ~basicPort of type Client is a conjugated Port typed by BasicPortConfiguration.

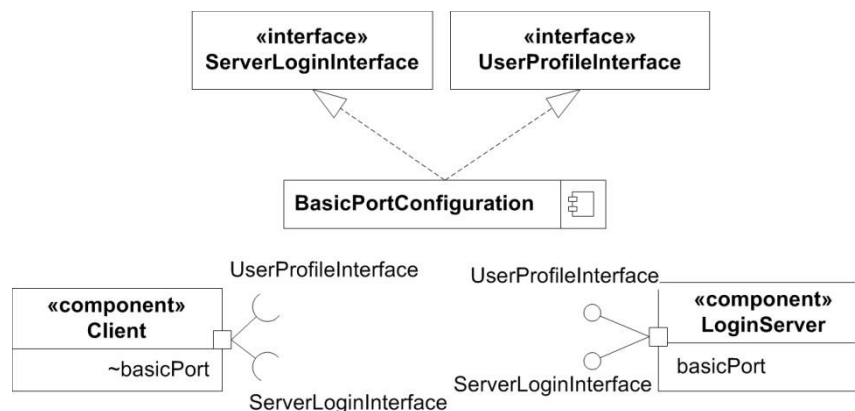
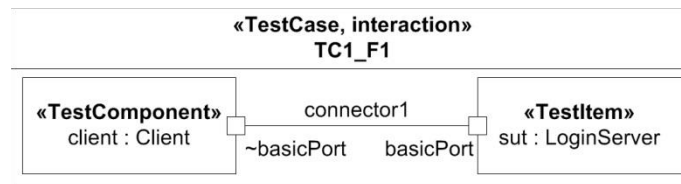


Figure A.12 - LoginServer Component Specification

The role each of those components will play in the given [test context](#) is not prescribed. Binding of roles for types is accomplished by the [test configuration](#). The [test configuration](#) also describes the communication channels over which information exchange among the [test component](#)(s) and the [test item](#) will be established later. UTP 2 allows for at least two ways to specify the [test configuration](#):

- **Shared [test configuration](#):** The shared [test configuration](#) mechanism enables the test analyst to bind [test cases](#) to a previously defined [test configuration](#). By doing so, the [test configuration](#) might be reused by different [test cases](#). One means to shared [test configuration](#) is by utilizing Collaborations. This is not shown in this example.
- **Isolated [test configuration](#):** In contrast to shared [test configuration](#), the isolated [test configuration](#) builds the [test configuration](#) every time from scratch. This option is only possible, if «[TestCase](#)» is applied on (a subclass of) Behavior directly. Since Behavior is a StructuredClassifier it is possible to directly make use of the stereotypes «[TestItem](#)» and «[TestComponent](#)» within the composite structure of the respective Behavior. However, this prevents the advantages of reuse.

The diagram below denotes the very simple [test configuration](#) contained in the [test case](#) TC1\_F1. The [test case](#) could be seen as a [test case](#) declaration which can be created and fostered very early in the test process. The [test configuration](#) comprises two parts, one being stereotyped as «[TestComponent](#)» and the other stereotyped as «[TestItem](#)», whose compatible Ports are connected by Connector c1. The Connector is an important means for specifying over which communication channel the information exchange between [test component](#)(s) and [test items](#) are supposed to take place during the execution of the [test case](#).



**Figure A.13 - LoginServer Test Configuration**

UTP 2 does not prescribe nor emphasize which variant to be used. Often, this depends on the applied test modeling methodology, the applied tooling, and the acceptance of the test analysts. For example, if generative approaches to test design are applied, then it might not be important to reuse [test configurations](#) throughout several [test cases](#) for the [test configurations](#) would be automatically derived from the boundary descriptions of the «[TestItem](#)».

#### A.2.4.2 Specification of Complex Test Data

The [test type](#) system specifies which data types are supposed to be exchanged within [test cases](#) among the [test components](#) and the [test item](#). For the actual specification of [test cases](#), values or instances for the [test type](#) systems need to be defined. This is in particular necessary for complex data types (e.g., DataType, Class, Signal etc.). The diagram below shows the InstanceSpecifications for the data types LoginReq and User required for the realization of [test cases](#).

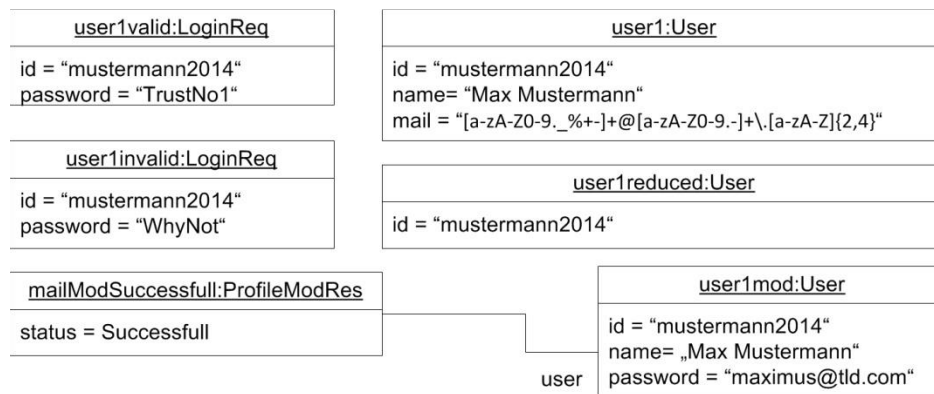


Figure A.14 - Test Data Specification

The interesting aspect in the [data](#) specification is the difference in dealing with the mail address attribute in the User-type InstanceSpecification. In the first case (user1), the Slot value is set to the regular expression, which was taken over from the type definition of MailAddress. It will later on be used to define expected results of the login operation. The semantics of such a concept is that as long as the actual response for a user's mail address complies with the stated regular expression, the actual response matches the [expect response action](#) and will not cause the [test case](#) to [Fail](#).

The InstanceSpecification user1reduced omits all slots that are not required for a user object. This will later on be used for the modification of a user's mail address. In the last case (user1mod) a concrete and very precise mail address was stated for the very same user. This InstanceSpecification is used as part of the profile modification response (i.e., data type ProfileModRes) after an update of the mail address was requested. This is necessary, since it is important to see that the modification of was actually successful. All other [data](#) values are defined directly within the [test cases](#) as ordinary ValueSpecifications.

#### A.2.4.3 Test Requirements Realization

The actual design of [test cases](#) is the most important part of the test design phase. According to the applied methodology for the given example, [test requirements](#) are supposed to be realized by [test cases](#), and thus, [test case](#) transitively verify or falsify the requirements that are covered by [test requirements](#). The assignment of [test requirements](#) to [test cases](#) is part of the test design phase and results in our case in the following (partially shown) assignments (see diagram below).

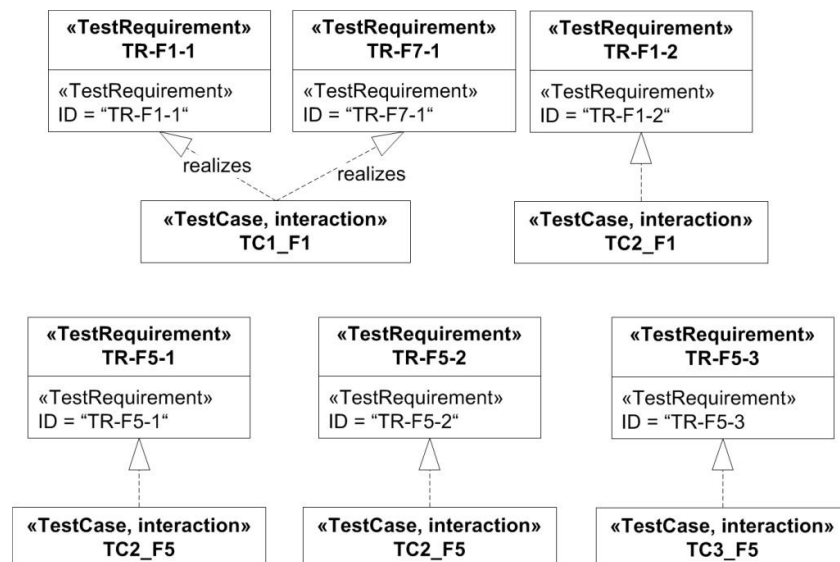


Figure A.15 – Realization of Test Requirements

The respective [test configuration](#) for each [test case](#) is not shown in the diagram for the sake of comprehensibility, but is present nevertheless for each [test case](#) and identical to the [test configuration](#) shown above.

#### A.2.4.4 Design of Test Case Procedures

Based on both the specification of the [test requirements](#) what to test and the precise specification of the [test configuration](#) in order to realize how to test what has to be tested, the [test case procedure](#) can be derived. As already shown, in this example sequence diagrams (i.e., Interactions) are going to be used as a [test procedure](#). The semantics of these [test case](#) interactions is that any deviation from the described interactions and message arguments will cause the [test case](#) to [Fail](#). However, if the actual response matches the expected ones during test execution, the [test case](#) will [Pass](#).

The two diagrams below“show the [test procedures](#) of two [test cases](#) for the [test requirements](#) TR-F1-1, TR-F1-2 and TR-F7-1. This specification deliberately neglected the parameterization of [test cases](#) due to an unresolved issue filed against UML Interactions.

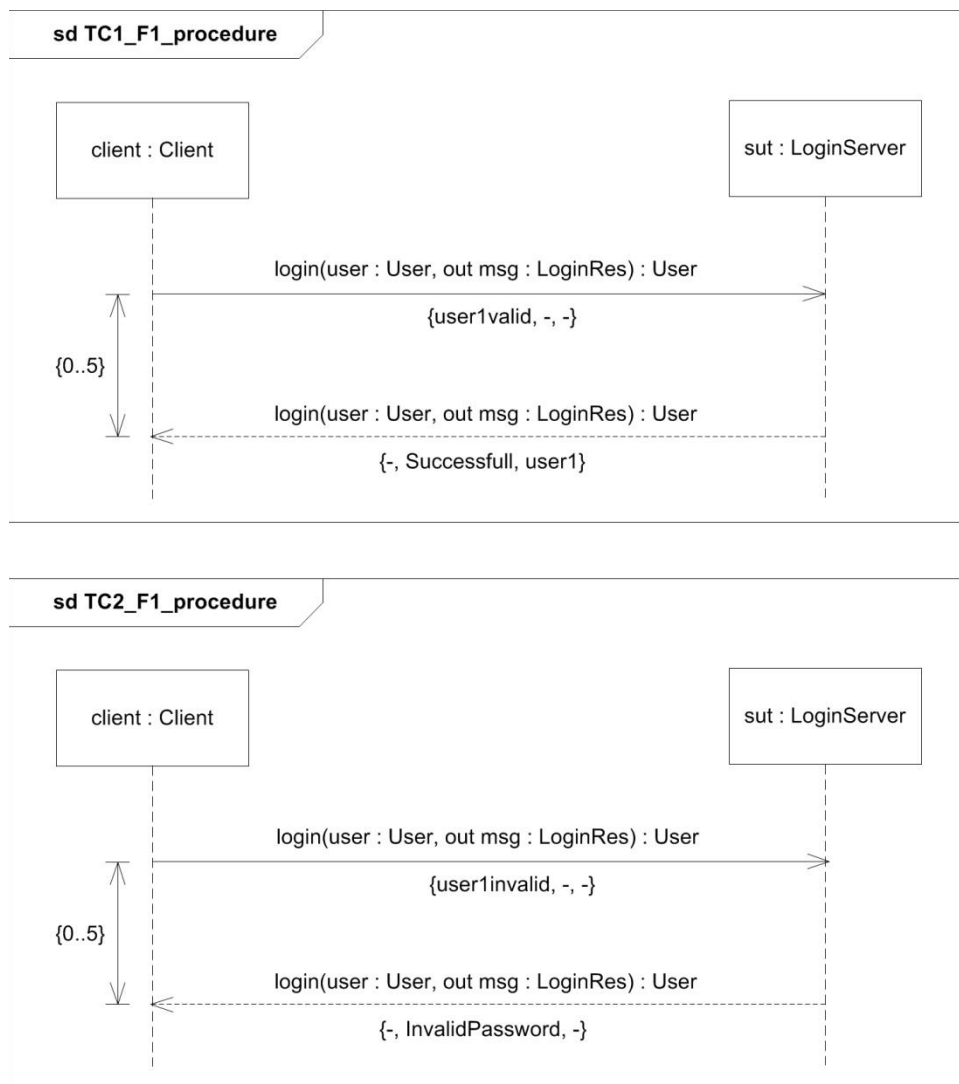


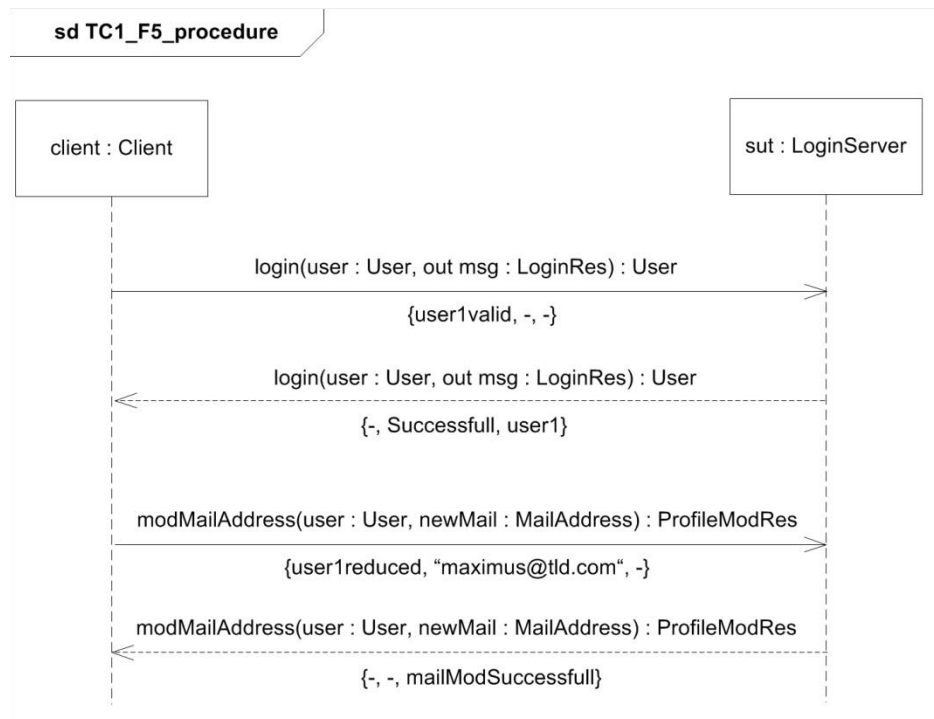
Figure A.16 – Two Test Procedures

The DurationConstraints ensure that any response to the login request that is recognized later than 5 time units (in this case seconds) after the actual request will violate the DurationConstraint, and thus, cause the [test case](#) to [Fail](#).



The message arguments used in these [test cases](#) are represented by InstanceValues that have the same name as the InstanceSpecifications they refer to. Successful and InvalidPassword are EnumerationLiterals of the Enumeration LoginRes. The messages are sent via the Connector connector1 of the corresponding [test configuration](#). This enables a precise definition of the Ports that should be used for sending stimuli and receiving [expect response actions](#).

The diagram below depicts a [test case](#) for the successful modification of a logged in user's mail address. It reuses (actually reimplements for no explicit reuse – by means of InteractionUse of the [test procedure](#) of [test case](#) TC1\_F1) the behavioral description for a successful user login request. The is usually called the preamble of the [test case](#) (although the current version of UTP 2 has no means to explicitly denote parts of the behavioral description as preamble or postamble; this is intended for revised submission).



**Figure A.17 – Successful Test Case**

The only technical deviation from the previously shown [test cases](#) is that the mailModAddress request message uses a LiteralString with value “maximus@tld.com” as message argument. Otherwise, no further peculiarities need to be discussed.

Note: The use of arguments of a message represented in curly brackets below the message arrow is not UML-compliant but was chosen for the sake of clarity.

## A.2.5 Mapping to TTCN-3

The Testing and Test Control Notation version 3 (TTCN-3) standardized by the European Telecommunications Standardization Institute (ETSI) prescribes a dedicated test language and test system framework that abide by the keyword-driven testing principle. Since its final adoption it has been heavily used within the telecommunications and automotive domain, but is in general, like UTP, independent of any domain. As TTCN-3 similarly to OMG standards is not restricted to certain methodology, the following described mapping represents just one possible way to translate UTP 2 [test cases](#) to TTCN-3. For example, it is restricted to Interactions for [test case procedures](#), whereas in principle each of the UML behavior kinds could be used for specifying [test procedures](#).



### A.2.5.1 Mapping the Test Type System

TTCN-3 comes along with a fine-grained and powerful type system that resembles the one provided by UML, which was taken over by UTP. The following snippet shows the corresponding TTCN-3 code for the LoginServer test type system starting with primitive types, over enumerations to complex types.

```
type charstring MailAddress
  (pattern "[a-zA-Z0-9._%+-\\]+@[a-zA-Z0-9.- \\]+\\.\\[a-zA-Z\\]\\{2,4\\}");
type enumerated LoginRes
  {InvalidID, InvalidPassword, UnknownUser, UserBanned, I};
type enumerated LogoutRes
  {I, LogoutRequiresLogin};
type enumerated ModMessage
  {I, InvalidFormat, LoginRequired};
type record LoginReq
{
  charstring id,
  charstring password
}
type record User
{
  charstring id,
  charstring name optional,
  charstring mail optional
}
type record ProfileModRes
{
  User user,
  ModMessage status
}
```

### A.2.5.2 Mapping Interface Descriptions

In TTCN-3, interface operations are represented by so called signature types. A signature is a type that can be instantiated and resembles the invocation of an operation. The concept of an Interface as grouping namespace for Operations has no correspondent concept in TTCN-3. In case of ambiguous signature names (i.e., two Operation with the same name contained in different Interfaces) the qualified name of the Operation could be used as name of the signature since TTCN-3 does not offer type overloading. The mapping presented in this example utilizes the TTCN-3 group concept to logically cluster the signature types according to their containing UTP Interfaces; however, one has to be aware of the fact that a TTCN-3 group has no further semantics than to group elements. A TTCN-3 group is neither comparable to a UML Package nor any other Namespace for it does not have scoping semantics. The suggested mapping of the LoginServer interface descriptions is shown in the following snippet:

```
group ServerLoginInterface
{
  signature login (LoginReq request, out LoginRes msg) return User;
  signature logout (User user) return LogoutRes;
}
group UserProfileInterface
{
  signature modMailAddress (User user, MailAddress newMail) return ProfileModRes;
}
```

### A.2.5.3 Mapping the Test Architecture

TTCN-3 relies on a component- and port-based architecture. That fits quite well with the offered concepts by UML, and thus, UTP. The following snippet demonstrates the mapping of the LoginServer test architecture to TTCN-3:

```
type port BasicPortConfiguration procedure
{
  inout login, logout, modMailInterface;
}
type component LoginSever
{
  port BasicPortConfiguration basicPort;
```

```

}
type component Client
{
    port BasicPortConfiguration basicPortConjugated;
}

```

#### A.2.5.4 Mapping the Test Data Specification

Data values utilized in message exchanges are called templates in TTCN-3. A template resembles an InstanceSpecification or dedicated ValueSpecification in UTP (actually UML). Templates can be either defined explicitly outside of a [test case](#) (called global templates), and thus, being reused by multiple [test cases](#), or directly within in a message (called inline). At first this specification is going to show the mapping of global templates:

```

template LoginReq userlvalid() :=
{
    id := "mustermann2014",
    password := "TustNo1"
};

template LoginReq userlinvalid() :=
{
    id := "mustermann2014",
    password := "WhyNot"
};

template User userl() :=
{
    id := "mustermann2014",
    name := "Max Mustermann",
    mail := (pattern "[a-zA-Z0-9._%+-\]+@[a-zA-Z0-9.- \]+\.[a-zA-Z]{2,4}")
};

template User userlreduced() :=
{
    id := "mustermann2014",
    name := omit,
    mail := omit
};

template User userlmod() :=
{
    id := "mustermann2014",
    name := "Max Mustermann",
    mail := "maximus@tld.com"
};

template ProfileModRes mailModSuccessfull() :=
{
    user := userlmod,
    status := Successful
};

```

#### A.2.5.5 Mapping Test Cases and Test Configuration

In TTCN-3 a [test configuration](#) is inherently bound to a [test case](#), whereas in UTP a [test configuration](#) could be potentially shared across multiple [test cases](#) (even though this feature is not shown in the given example). The following snippet shows the mapping of the [test case](#) TC1\_F1:

```

//determines the roles for Client and LoginSever
//runs on declares Client as TestComponent
//system declares LoginServer as TestItem
testcase TC1_F1() runs on Client system LoginServer
{
    //establishes the Connector connector1
    map(self:basicPortConjugated, system:basicPort);

    //invokes the login operation by sending an instance of the
    //signature type login and starts an implicit timer with the
    //duration of 5 seconds
    basicPortConjugated.call(login:{userlvalid,-}, 5000.0)
}

```

```

//continually checks whether the expected response is received
//by the test system
[]basicPortConjugated.getreply(login:{-,I}
                                value user1)
{
    //indicates that the test case has passed
    setverdict(pass);
}
//continually checks whether any other response is received
[]basicPortConjugated.getreply
{
    //indicates that the test case has failed due to mismatch
    //between actual and expected response
    setverdict(fail)p;
}
//continually checks whether the implicit timer expired
[]basicPortConjugated.catch(timeout)
{
    //indicates that the test case has failed due to timout
    setverdict(fail);
}
}
}

```

## A.3 Videoconferencing Example

This example is inspired from the case study about a Videoconferencing System (VS) that is reported in [1] with the aim of demonstrating the application of UTPV.2. This example illustrates some of the major concepts of UTP 2 on the software of the VS such as [test item](#), [test item configuration](#), and [test component configuration](#) on the three key features of the VS. One focuses on the establishing the videoconference, the second one related to sending presentations in addition to the videoconference, and third one focuses on modeling behavior of VS in the presence of packet loss.

The rest of this section is organized as follows. Section [Given Requirements on the Test Item](#) lists the key requirements that are focused for modelling in this section, Section [Modeling the Structure of the System](#) demonstrates how this specification models structure of the VS using the UML class diagrams with UTP, Section [Modeling the Behavior of the System](#) demonstrates how this specification modeled the three key requirements as UML State Machines and UTP, Section [The TRUST Test Generator](#) shows our test generator that generates executable [test cases](#) from UML Class Diagrams and UML State Machines, and Section [Mapping to Code](#) shows an example of mapping from the models to code.

### A.3.1 Given Requirements on the Test Item

In this section, this specification will demonstrate modelling the four key functionalities of a VS that must be tested. These functionalities are listed in the table below:

**Table A.9.3 – Videoconferencing Requirements**

Id	Type	Description
R-0001	functional	A VS should be able to connect to maximum n number of VSs at the same time.
R-0002	functional	A VS should be able to start presentation even it is not in the videoconference. In this case, the presentation will be only shown to the VS itself.
R-0003	functional	A VS should be able to start presentation when it is in a videoconference. In this case, the presentation will be transmitted to all the connected VSs (referred as end points).
R-0004	non-functional	A VS should be able to handle packet loss. If the VS cannot handle packet loss of greater than x% for t minutes, it disconnects the current active call.

### A.3.2 Modeling the Structure of the System

In this section, this specification models the structure of VS that is modeled as a UML class diagram. A VS can establish calls with 1 to \* number of endpoints, i.e., other VSs. The VS is stereotyped as «[TestItem](#)» and «[TestDesignInput](#)» to label the system being tested, whereas other endpoints (i.e., Endpoint) is stereotyped as «[TestComponent](#)». The VS has five attributes, NumberOfParticipants, MaximumParticipants, Presentation, H323, and packetLoss representing the current number of endpoints in a videoconference, maximum number of calls supported by the VS, if the VS is in presentation or not, if H323 mode is on or not, and percentage of packet loss it is facing. The packetLoss attribute is of type NFP\_Percentage from the MARTE profile. The VS class has five operations to support dialing to an endpoint (connectCall()), disconnecting a participant from a videoconference (disconnectCall()), starting presentation (presentationOn()), stopping presentation (presentationOff()), and disconnecting all the participants in a call (disconnectAll()). In addition, this specification defines a constraint in OCL on VS to model configuration for testing:

```
context VS inv:  
self.H323
```

This constraint demonstrates that the VS must be configured to support a videoconference with h323 conferencing protocol. The constraint is stereotyped as «[TestItemConfiguration](#)» to signify that the constraint is a configuration for VS and is handled accordingly by test generator. In addition, «[TestItem](#)» has an attribute [configuration {subsets roleConfiguration}](#), which is linked to this OCL constraint with «[TestItemConfiguration](#)» (not shown in the figure).

A similar constraint for Endpoint is also specified in the figure below and is stereotyped as [«TestComponentConfiguration»](#).

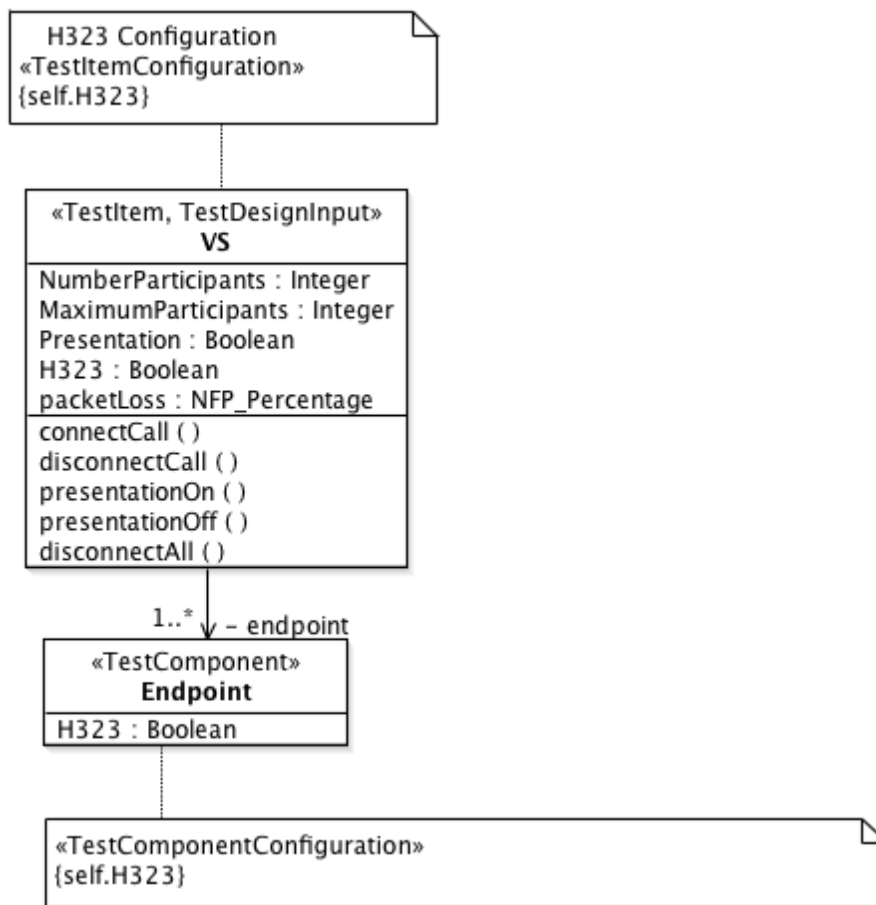


Figure A.18 – UML Class Diagram

### A.3.3 Modeling the Behavior of the System

The figure below shows the behavior of the VS modeled as a UML state machine stereotyped as [«TestDesignInput»](#) to instruct test generator that the state machine should be used for generation of test cases. In our context it is important to stereotype a state machine that must be used for generation of test cases since not all the state machines are used for generation of test cases. The state machine has three regions: 1) The first region models first requirement for testing, i.e., establishing videoconference, 2) The second region models the second two requirements related to presenting while in a videoconference and presenting without a conference, and 3) The third region models the fourth requirement.

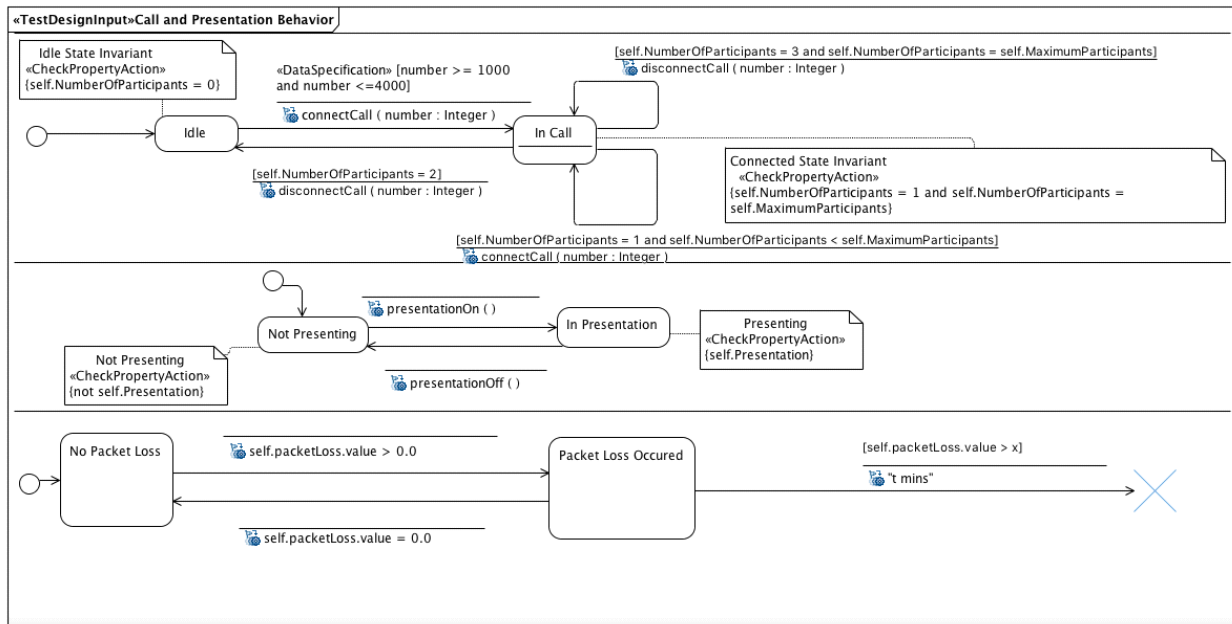


Figure A.19 – UML State Machine Diagram

In the first region, this specification models the behavior of a VS related to establishing a videoconference. The first region has two states, i.e., Idle and In Call demonstrating that the VS is Idle state, and the VS is in a videoconference respectively. Each state has a state invariant defined as an OCL constraint based on the attributes defined in the VS class diagram. For example, the Idle state has the following state invariant specified as an OCL constraint:

```
context VS inv:
self.NumberOfParticipants = 0
```

The state invariant is stereotyped as «[CheckPropertyAction](#)» to instruct the test generator to use the constraint to generate code that compares the actual state of VS at the runtime (e.g., value of NumberOfParticipants in this example) with the one specified as [CheckPropertyAction](#). If the state matches then it means everything is fine, however, if the state doesn't match it means there is a fault. The attributes of «[CheckPropertyAction](#)» are shown below in the figure. For example, the [checkedProperty](#) attribute is linked to the NumberOfParticipants in the VS class (only shown as Entries:1). The value of expected is set to true meaning that the expected evaluation value of this constraint is true.

Property	Value
▼ <a href="#">CheckPropertyAction</a>	
checkedProperty	Entries: 1
expected	true
UTP::CheckPropertyAction::arbitrationSpecification	null

Figure A.20 - Attribute values of «[CheckPropertyAction](#)»

Transitions in the state machine are modeled with Call Events corresponding to the operations defined in the VS class. For example, from the Idle state, the transition with connectCall() trigger will lead to InCall if the call is established successfully. In addition, some of the transitions have guard conditions with the stereotype «[DataSpecification](#)». Recall that [DataSpecification](#) is "A named boolean expression composed of a data type and a set of constraints applicable to some data in order to determine whether or not its data items are conformant to this data specification" as defined in the conceptual model. A [DataSpecification](#) (e.g., guard condition in this example) signifies that the transition from the Idle state to the In Call state with a guard condition number>=100 and number

$\leq 4000$ , (i.e., an OCL constraint) can only be triggered by calling the `connectCall(number:Integer)` Call Event with a number between the range of values specified by the guard condition. In our context, this guard condition, i.e., an OCL constraint is used by the test generator to generate valid values within the range to trigger a transition, for example, the `connectCall()` operation in this case.

The second region of the state machine models the behavior of VS related to starting the presentation in parallel to the videoconference. The region has two states (i.e., Not Presenting and In Presentation) showing the states that the VS is not presenting and presenting respectively. As with the first region, each state has a state invariant modeled as an OCL constraint. Similarly, the third region models the behavior of VS in presence of packet loss.

### A.3.4 The TRUST Test Generator

The figure below shows a very high-level architecture of test case generator. The full details of the test generator can be found in [3]. At a high level, the test generator called as TRUST takes UML State Machines and UML Class Diagrams with stereotypes from UTP as input and generates executable test cases based on various coverage criteria such as All State coverage and All Transition coverage (e.g., `ts:TestStrategy` with [«StateTransitionTechnique»](#)) [3]. According to [ISTQB] [StateTransitionTechnique](#) is "A black box [test design technique](#) in which [Test Cases](#) are designed to execute valid and invalid state transitions". In addition, TRUST has a built-in algorithm that flattens the state machines with hierarchy and concurrency before generating test cases. The details of the algorithm can also be found in [3]. TRUST also invokes a test data generation tool called EsOCL that takes input an OCL constraint (specified in class diagrams and state machines) and provides a set of data that satisfy the constraint based on a test data generation strategy (e.g., `td:TestDataGenerationStrategy` with the [«BoundaryValueAnalysis»](#) stereotype). According to [ISTQB], [BoundaryValueAnalysis](#) is "A black box [test design technique](#) in which [Test Cases](#) are designed based on boundary values". The details of EsOCL can be found in [4].

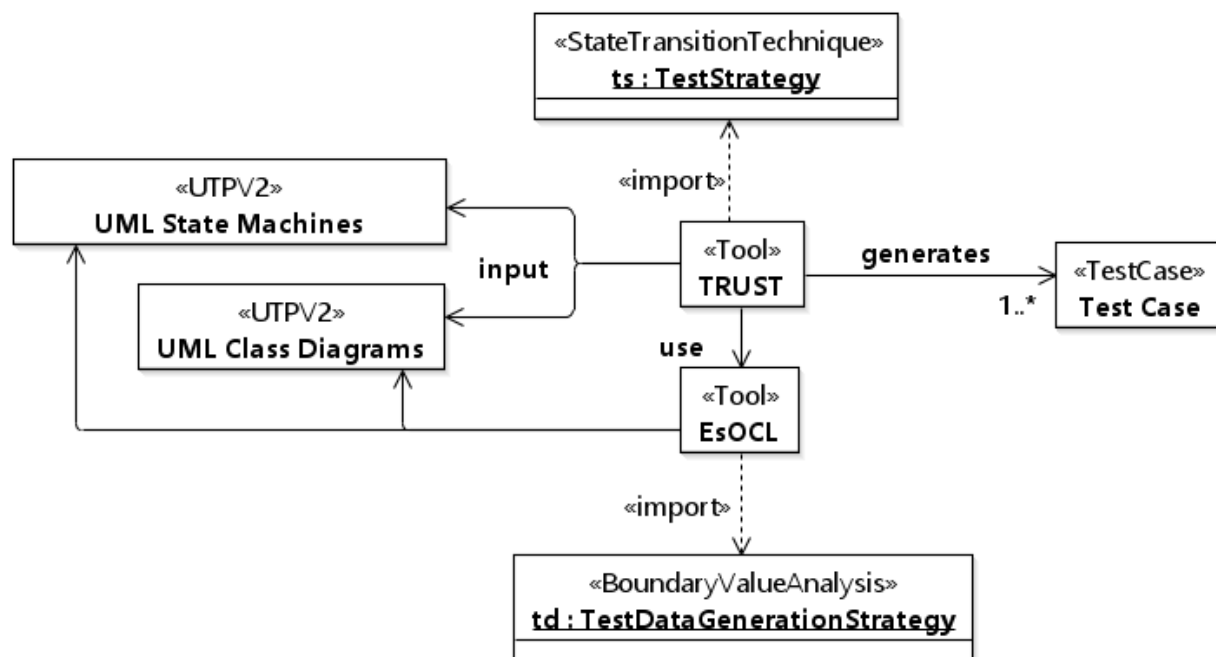


Figure A.21 - Test Generator

The figure below shows a high-level architecture of our Test Driver. The test driver takes input a test case and executes it on the VS that communicates with the  $n$  number of endpoints. The test driver also sends commands to configure endpoints based on test configurations specified in the test case. In our current example, the test driver executes only test cases on one VS; however, in reality it can execute test cases on multiple VSs in a videoconference. During the execution, test driver invokes an OCL Evaluator called DresdenOCL ([www.dresden-ocl.org/](http://www.dresden-ocl.org/)) to evaluate OCL constraints that were stereotyped as [«CheckPropertyAction»](#) against the actual state of the

VS that eventually determines the success or failure of the execution of test cases.

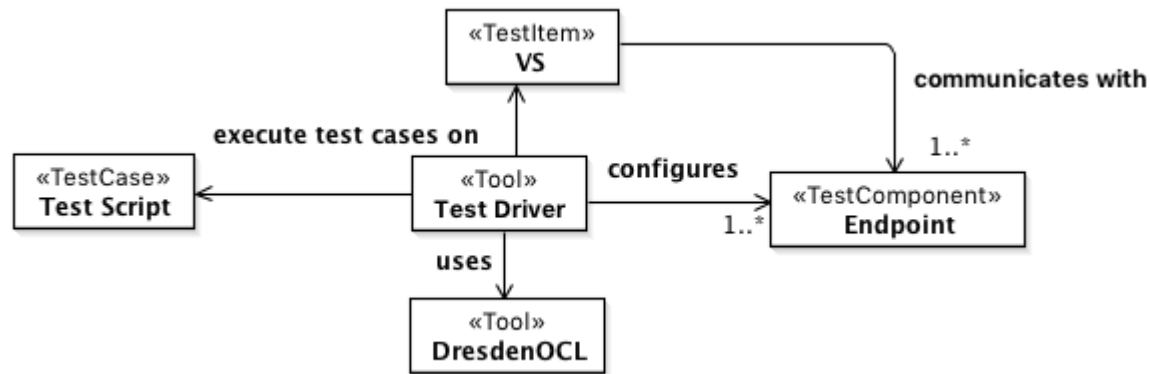


Figure A.22 - Test Driver

### A.3.5 Mapping to Code

Below, this specification shows a sample code corresponding to [test item configuration](#) and [test component configuration](#). Line 1 and Line 2 reserves VS (A) and Endpoint (B) for the execution of [test cases](#), whereas Line 3 enables H323 mode on [test item](#) based on the constraints with stereotype in «[TestItemConfiguration](#)».

```

Line 1: self.A=test.api.initialize('a')
Line 2: self.B=test.api.initialize('b')
Line 3: self.A.H323 = true

```

Below, this specification shows the code corresponding to the start and stop presentation behavior and also the code that checks state of the VS. Line 1 executes presentation start command on the VS and Line 2 checks whether the VS is in correct state by checking the value for the Presentation attribute of the VS, which should be equal to true.

```

Line 1: Execute.Command("Command.Presentation.Start()", self.A)
Line 2: self.assertFalse(self.A.Presentation == true)

```

### A.3.6 References

- [1] Ali, Shaukat, Lionel Claude Briand, and Hadi Hemmati. "Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems." *Software and Systems Modeling* 11 (2012): 633-670.
- [2] Ali, Shaukat, Lionel Claude Briand, Andrea Arcuri, and Suneth Walawege. *An Industrial Application of Robustness Testing Using Aspect-Oriented Modeling, UML/MARTE, and Search Algorithms* In *ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (Models 2011)*, Edited by Jon Whittle, Tony Clark and Thomas Kühne. : ACM/IEEE, 2011.
- [3] Ali, Shaukat, Hadi Hemmati, Nina Elisabeth Holt, Erik Arisholm, and Lionel Briand. *Model Transformations As a Strategy to Automate Model-Based Testing - a Tool and Industrial Case Studies*. Simula Research Laboratory, 2010.
- [4] Ali, Shaukat, Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel Claude Briand. "Generating Test Data From OCL Constraints With Search Techniques." *IEEE Transactions on Software Engineering* 39 (2013).



## A.4 Subsea Production System Example

### A.4.1 Description of Case Study

A subsea production system is a cyber-physical system that produces oil and gas from subsea. Typically, such subsea production systems are highly configurable in the sense that their hardware topologies and software parameters can be configured based on requirements customer such as the size of a subsea field and its natural environment (e.g., depth of sea). A subsea production system is composed of two sets of systems: topside and subsea systems. Umbilical connections (e.g., cables or hoses which supply air, power, electrical power, fiber optics to subsea equipment) are established to connect topside and subsea. Commands (e.g., opening valves) are sent by operators via topside systems to subsea systems, which control different kinds of subsea actuators (e.g., choke and valve) and monitor various sensors (e.g., pressure and temperature).

Please note that the case study is designed to demonstrate that the UTP 2 stereotypes can be used for developing domain specific language based MBT methodologies such as RTCM [3].

### A.4.2 Functionality to Test

To demonstrate the application of UTP 2 to this case study, this specification specifies one of the key functionalities of Subsea Electronic Module (SEM), which has configurable software deployed to control subsea instruments. This functionality OpenValve is specified using the Restricted Use Case Modeling methodology (RUCM) [1][2] and the RUCM Editor, as shown in the figure below. Notice that the use case model (i.e., UCModel) is indicated as a [TestRequirement](#) using <<TestRequirement>>, which is a UTP 2 stereotype.

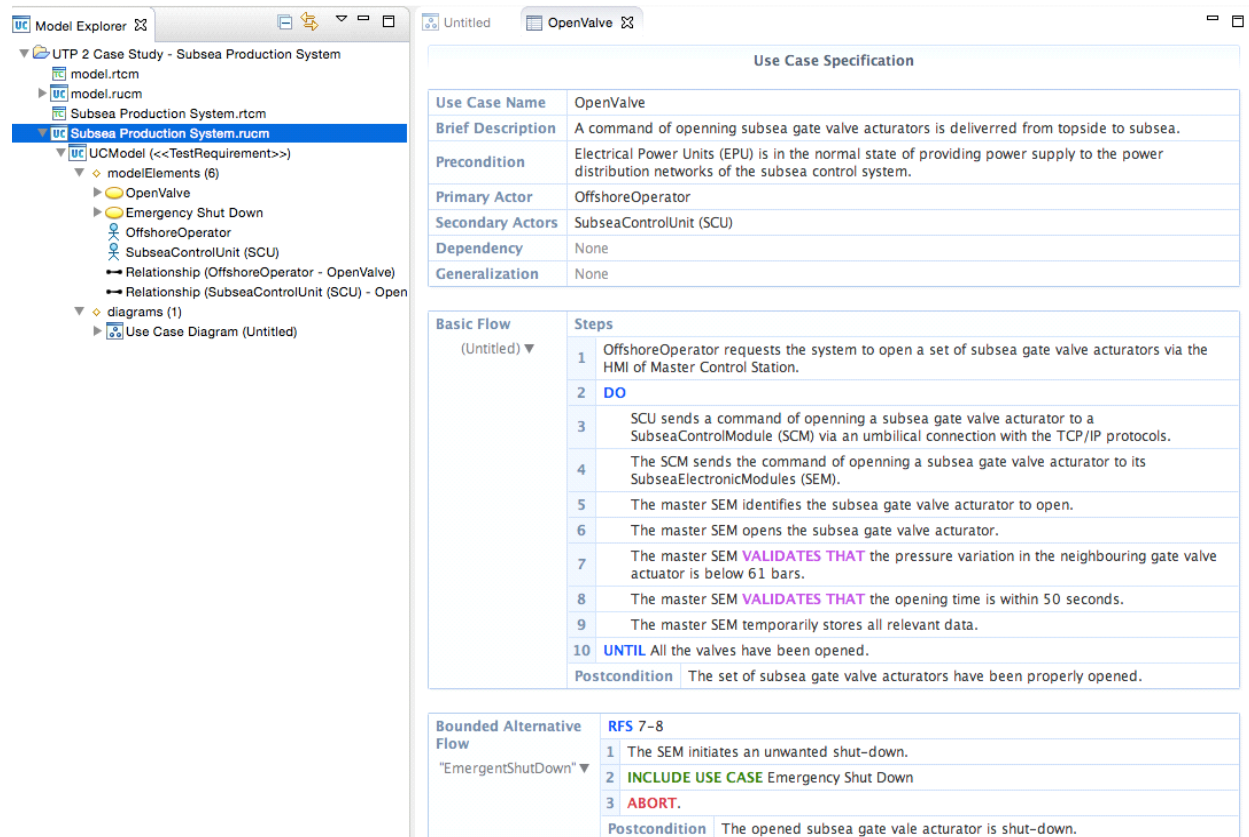


Figure A.23 - Use Case OpenValve (Specified in RUCM)

### A.4.3 Test Design Inputs

To test the *OpenValve* functionality presented in the figure above, this specification defines four test design inputs, as shown in the figure below. Notice that this specification aims to test the functionality of *OpenValve* of SEM using a simulator that is particularly designed for testing SEM.

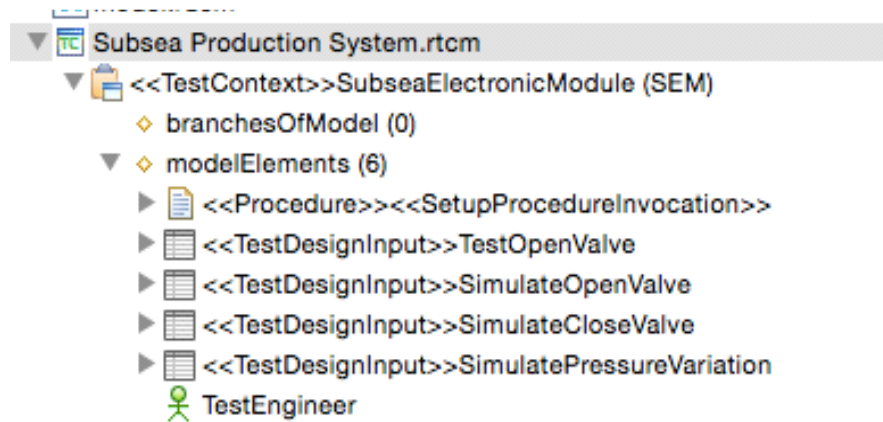


Figure A.24 - The Four TestDesignInput and one procedure

The test objective of the test context SubseaElectronicModule (SEM) is defined as the description of the test context: “<<[TestObjective](#)>> The goal of these tests is for system testing of the functionalities of <<[TestItem](#)>> SEM.”

In the figure below, this specification presents the test design input of *TestOpenValve*, which is specified/modeled using the Restricted Test Case Specification methodology (RTCM) [3]. Notice that the test case specification is annotated with UTP 2 stereotypes using stereotype notations. For example, steps 3, 4 and 10 of the basic flow (i.e., <<[Sequence](#)>>Pass) are annotated as <<[ExpectResponseAction](#)>>. Step 1 is annotated with <<[CreateStimulusAction](#)>> and steps 2, 6, 8 and 9 are annotated with <<[ProcedureInvocation](#)>> as these four steps invoke other test case specifications with keywords INCLUDE TC SPEC. Steps with keyword VERIFIES THAT are annotated with either <<[ExpectResponseAction](#)>> or <<[CheckPropertyAction](#)>>. TestSetup is annotated with <<[TestConfiguration](#)>> and can be reused across test case specifications.

Test Case Specification		
Name	<<TestDesignInput>>TestOpenValve	
Brief Description	This test case specification tests <<TestRequirement>><<UseCase>>OpenValve.	
Precondition (Test Data Specification)	None	
Tester	None	
Dependency	INCLUDE TC SPEC <<TestDesignInput>>SimulateOpenValve, INCLUDE TC SPEC <<TestDesignInput>>SimulateCloseValve, INCLUDE TC SPEC <<TestDesignInput>>SimulatePressureVariation	

Test Setup ▼	Name	<<TestConfiguration>>Test Setup
	Description	TestEngineer makes sure that <<TestItem>>SEM and <<TestComponent>>Simulator are connected and powered on.

Basic Flow (Test Setup) "<<Sequence>>setup"▼	Steps	
	1	<<CreateStimulusAction>>TestEngineer turns on the power of SEM.
	2	TestEngineer turns on the power of Simulator.
	3	<<CreateLogEntryAction>>TestEngineer records the initial state of SEM.
	4	<<CreateLogEntryAction>>TestEngineer records the initial state of Simulator.
Postcondition (Test Oracle)		SEM is turned on. Simulator is turned on.

Basic Flow (Test Sequence) "<<Sequence>>pass"▼	Steps	
	1	<<CreateStimulusAction>>TestEngineer uses Simulator to simulate the OpenValve command.
	2	<<ProcedureInvocation>>INCLUDE TC SPEC <<TestDesignInput>>SimulateOpenValve. ⚠
	3	<<ExpectResponseAction>>TestEngineer VERIFIES THAT the OpenValve signal received by Simulator is correct.
	4	<<CheckPropertyAction>>TestEngineer VERIFIES THAT the opening time is within the allowed threshold.
	5	<<CreateStimulusAction>>TestEngineer uses Simulator to simulate the CloseValve command.
	6	<<ProcedureInvocation>>INCLUDE TC SPEC <<TestDesignInput>>SimulateCloseValve ⚠
	7	<<Parallel>><<CreateStimulusAction>>TestEngineer uses Simulator to simulate the OpenValve command MEANWHILE <<CreateStimulusAction>>TestEngineer uses Simulator to simulate the pressure variation in the neighbouring gate valve actuator is above 61 bars.
	8	<<ProcedureInvocation>>INCLUDE TC SPEC <<TestDesignInput>>SimulateOpenValve ⚠
	9	<<ProcedureInvocation>>INCLUDE TC SPEC <<TestDesignInput>>SimulatePressureVariation ⚠
	10	<<CheckPropertyAction>>TestEngineer VERIFIES THAT SEM initiates emergent shut-down.
Postcondition (Test Oracle)		<<TestSetArbitrationSpecification>>The test is passed.

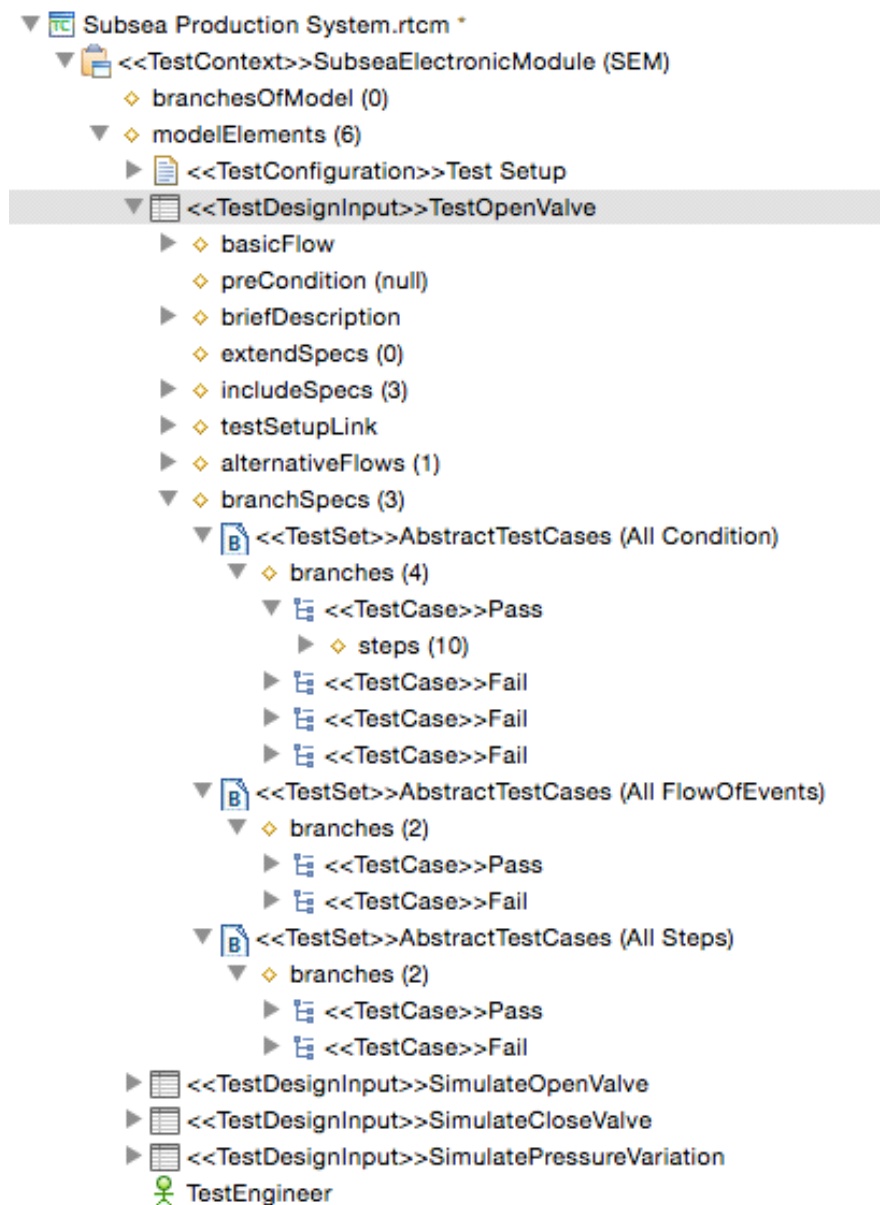
  

Bounded Alt. Flow (Test Sequence) "<<Alternative>>fail"▼	RFS <<Sequence>>pass 3-4,10	
	1	<<CreateLogEntryAction>>TestEngineer reports a failure.
	2	ABORT
Postcondition (Test Oracle)		<<TestSetArbitrationSpecification>>The test is failed.

Figure A.25 - test design input TestOpenValve

#### A.4.4 Generation of Test Sets and Abstract Test Cases

By taking the test design inputs as the input, the test generator of RTCM [3] automatically generates abstract test cases, as shown in the figure below. Based on different coverage criteria, from the [test design input](#) of *TestOpenValve*, the generator can generate three test sets, which contain various numbers of abstract [test cases](#).



**Figure A.26 - Generated test sets**

The automated generation is possible due to the fact that use case specifications in RUCM and test case specifications in RTCM can all be formalized as instances of the UCMeta [2] and TCMeta [3][4] metamodels respectively. Paths can then be automatically generated from formalized specifications/paths by following various coverage strategies (e.g., *All Sentence Coverage* and *All FlowOfEvents Coverage*).

One example of the abstract [test cases](#) generated from the [test design input](#) of *TestOpenValve* is provided in the figure below for reference. The step marked with the red color means the step failed. The step marked with the Green color means the step passes.

Steps Of Branch ▼	1	Simulator simulates the OpenValve command.
	2	Simulator sends the simulated OpenValve command to SEM.
	3	TestEngineer <b>VALIDATES THAT</b> the OpenValve command is properly simulated.
	4	TestEngineer <b>VALIDATES THAT</b> the OpenValve command is properly sent by Simulator to SEM.
	5	TestEngineer reports a failure.
	6	<b>ABORT</b>

Figure A.27 - An Example of a generated abstract test case XE "abstract test case"

#### A.4.5 References

- [1] Tao Yue, Lionel Briand, and Yvan Labiche, "Facilitating the Transition from Use Case Models to Analysis Models: Approach and Experiments", in Transactions on Software Engineering and Methodology (TOSEM), Volume 22, Issue 1, 2013.
- [2] Tao Yue, Lionel Briand, and Yvan Labiche. "Toucan: an Automated Framework to Derive UML Analysis Models From Use Case Models.", in ACM Transactions on Software Engineering and Methodology (TOSEM) 24, no. 3 (2015).
- [3] Tao Yue, Shaukat Ali, and Man Zhang. Applying A Restricted Natural Language Based Test Case Generation Approach in An Industrial Context, in International Symposium on Software Testing and Analysis (ISSTA)., 2015.
- [4] Man Zhang, Tao Yue, Shaukat Ali, Huihui Zhang and Ji Wu. "A Systematic Approach to Automatically Derive Test Cases From Use Cases Specified in Restricted Natural Lan-guages", 8th System Analysis and Modelling Conference (SAM), 2014

## A.5 ATM Example

### A.5.1 General

This annex contains the Banking example introduced in the earlier version of UTP [UTP1.2]. The following model has been updated for the current UTP standard. It shows how to utilize UTP, version 2, to specify test models for unit level tests, component level tests and system tests.

The given example is motivated by an interbank exchange scenario in which a customer with an EU Bank account wishes to deposit money into that account from an Automated Teller Machine (ATM) in the United States. The figure below provides an overview of the architecture of the system. The ATM used by this customer interconnects to the EU Bank, through the SWIFT Network<sup>1</sup>, which plays the role of a gateway between the logical networks of the US Bank and the EU Bank.

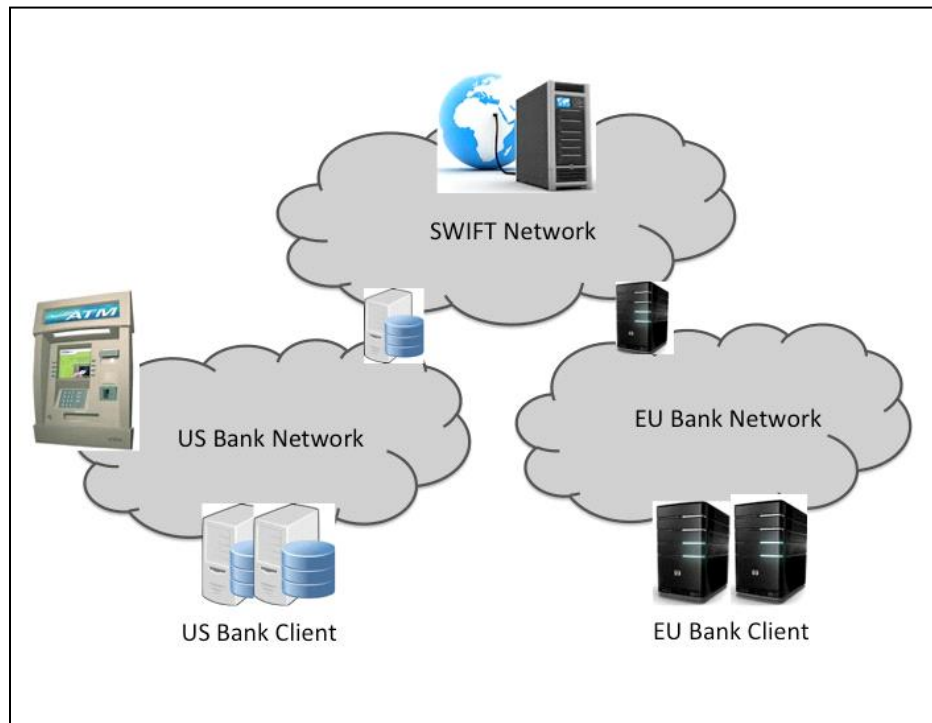


Figure A.28 - Overview on the InterBank Exchange Network (IBEN)

The figure below shows the UML system model<sup>2</sup> of the InterBank Exchange Network. In the model, five UML packages called *ATM*, *Bank*, *SWIFTNetwork*, *HWControl* and *Money* are provided. The dashed arrows between the packages show their import dependencies.

The following sub-sections demonstrate the use of UTP 2 for:

- unit test modeling on *Money* classes (Subsection 2)
- integration test modeling of the components *ATM*, *HWControl* and *Bank* (Subsection 3)
- system test modeling of IBEN system (Subsection 4)

<sup>1</sup> SWIFT = Society for Worldwide Interbank Financial Telecommunication

<sup>2</sup> The diagrams of this example are modelled in Papyrus.

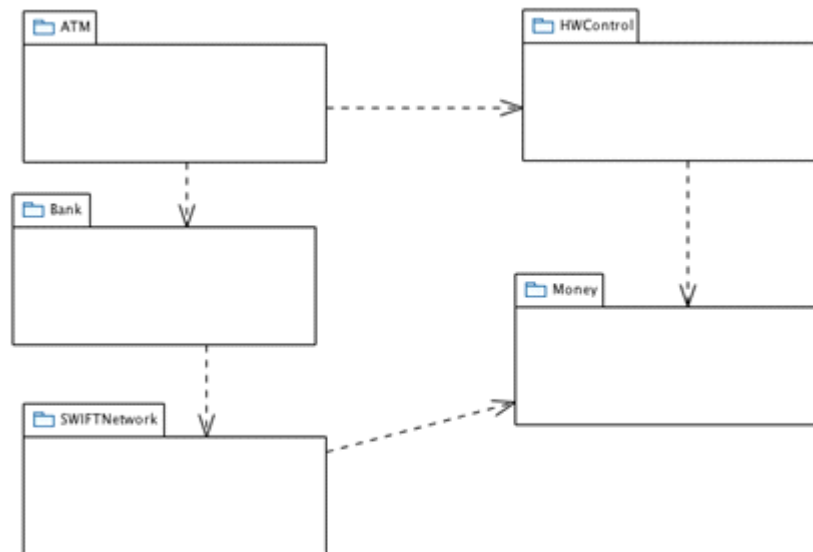


Figure A.29 - Packages of the InterBank Exchange Network (IBEN) System Model

### A.5.2 Unit Test Example

This sub-section illustrates the use of UTP version 2 in order to define unit [test level test cases](#). It reuses and extends the *Money* and *MoneyBag* classes provided as examples of the well-known JUnit test framework ([JUnit\_web], [JUnit\_Example]).

Before starting modeling tests, [the test item](#) is first explained. The figure below shows the package *Money* (blue color) which will be tested.

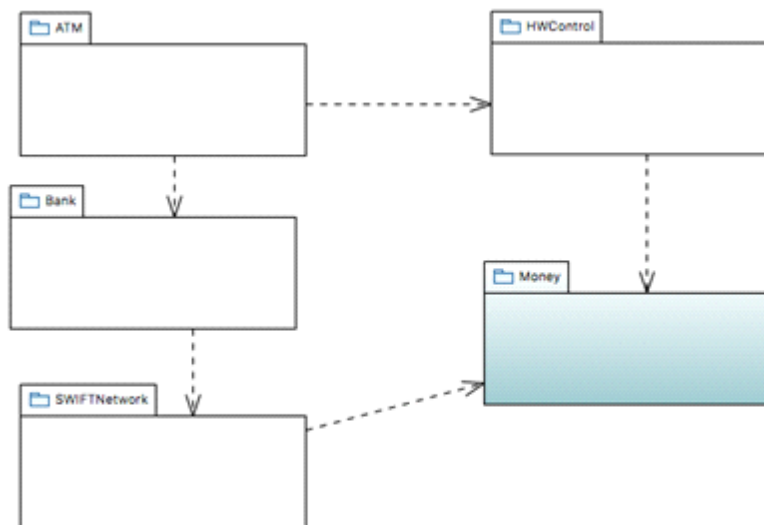


Figure A.30 - Package Money with Test Items for Unit Test of IBEN

The figure below shows the classes defined in the package *Money*<sup>3</sup>. It shows an interface class called *IMoney*, which is realized by the class *Money*, and class *MoneyBag*.

<sup>3</sup> Even though the naming of the package *Money* and of the class *Money* may lead to misunderstanding, the definition of the example provided by [www.junit.org](http://www.junit.org) is still used



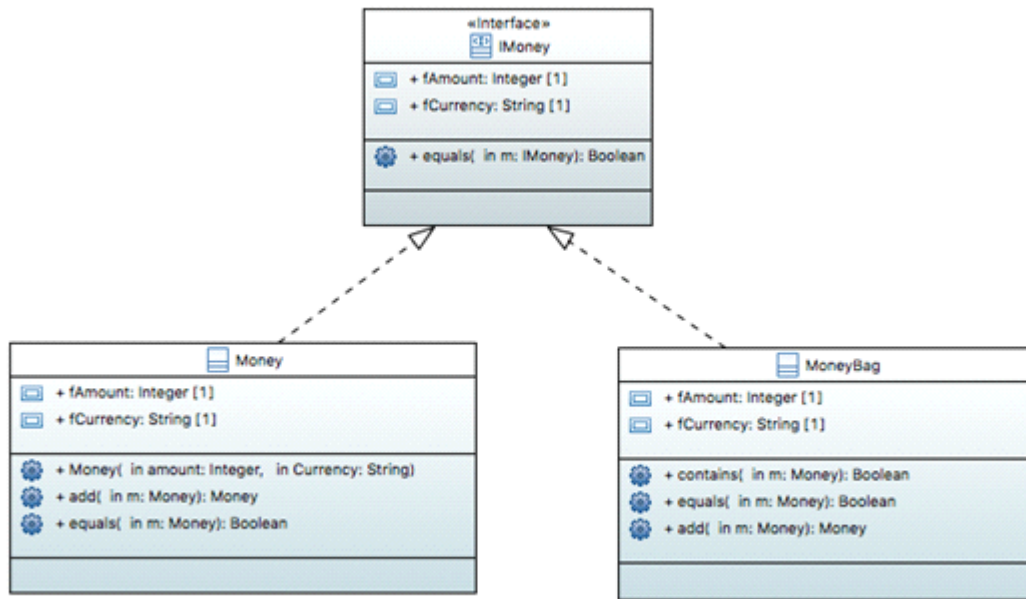


Figure A.31 - Classes in Package Money in IBEN Model

The ATM uses these classes in order to count the bills entered by a user when making a deposit in cash. Two [test requirements](#) are defined:

- Verify that the Money class is appropriately counting the bills added by the user, when bills from the same currency are entered.
- Verify that the Money and MoneyBag classes are appropriately recognizing the bills added by the user when bills from different currencies are entered.

The figure below shows the [test configuration](#) between the [test component](#) named *unitTestComponent* and the [test items](#) called *myMoney1* and *myMoney2* of class *Money* and *myMoneyBag* of class *MoneyBag*. The [test configuration](#) is modeled as UML Collaboration in order to be able to apply as CollaborationUse to the [test cases](#).

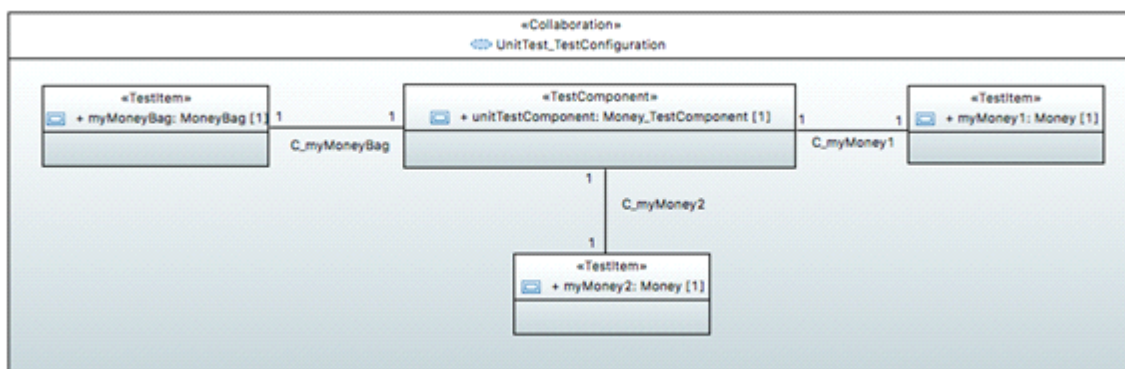
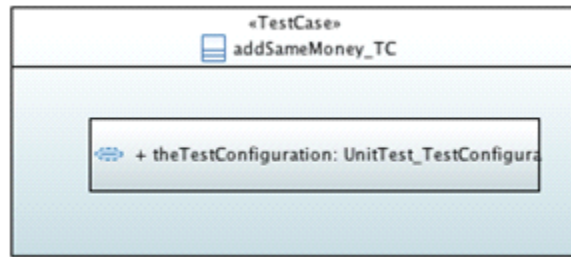


Figure A.32 - Unit Test Configuration

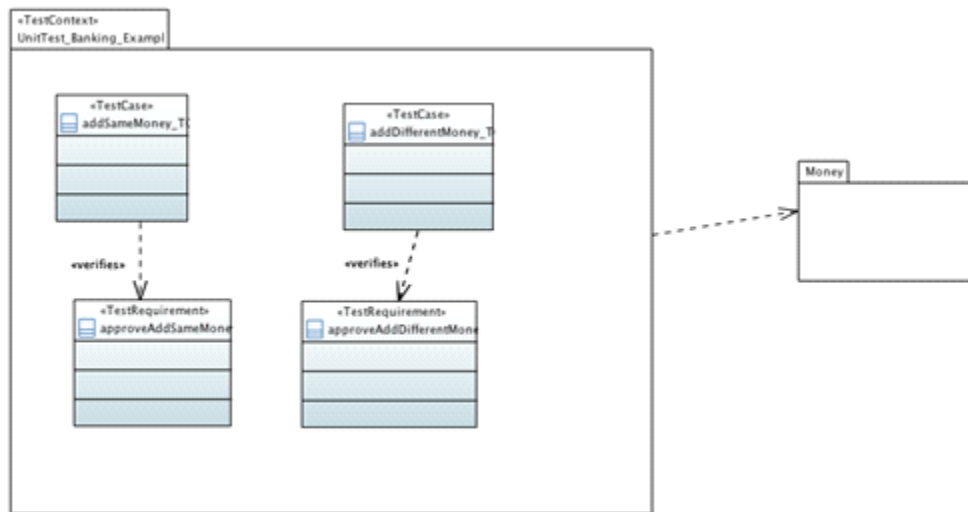
The figure below shows the application of the unit [test configuration](#) to the [test case](#) *addSameMoney\_TC*. By using the UML CollaborationUse the binding between the [test configuration](#) and the [test case](#) is guaranteed.





**Figure A.33 - Use of Test Configuration for Test Case AddSameMoney\_TC**

The figure below shows the [test context](#) of the unit test *UnitTest\_Banking\_Example*. Class *Money* is the item to be tested. It is defined in package *Money* which is imported from the system model. The package must be imported in order to get access during test execution. The [test requirements](#) *approveAddSameMoney* and *approveAddDifferentMoney* should approve that the addition of two money objects returns an object of class *Money* with the correct amount and currency. In the former requirement, money of the same currency will be added. In the latter, money of different currencies are to be added. The [test cases](#) called *addSameMoney* and *addDifferentMoney* verify the test [test requirements](#).



**Figure A.34 - Test Context for the Unit Test**

The figure below specifies the behavior of the [test case](#) called *addSameMoney* verifying the [test requirement](#) *approveAddSameMoney*. In this test scenario, two objects of class *Money* are created, namely *myMoney1* with 20 USD and *myMoney2* with 50 USD. Afterward, *myMoney2* is added to *myMoney1*. The result is sent back to the [test component](#) for approval.

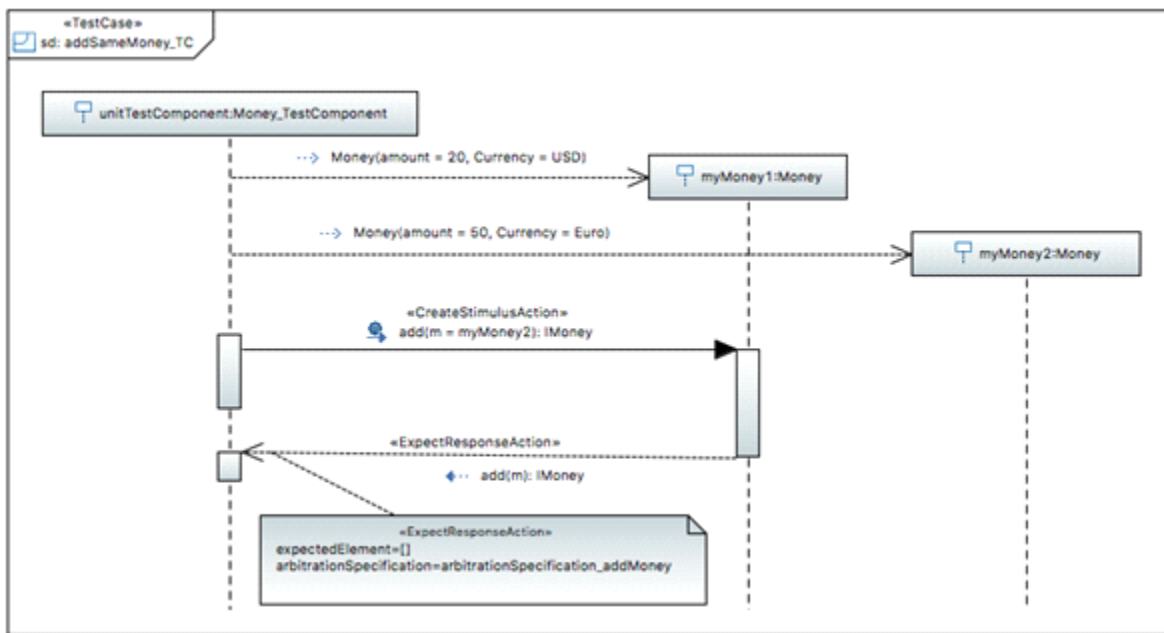


Figure A.35 - Test case addSameMoney\_TC

The correctness of the response is checked in either the default [arbitration specification](#)<sup>4</sup>, or as in this case, by the user-defined [arbitration specification](#) called *arbitrationSpecification\_addMoney*. Finally, the figure shows that in case the result of add() is 70 USD, the [arbitration specification](#) sets the test [verdict](#) to [Pass](#), otherwise to [Fail](#).

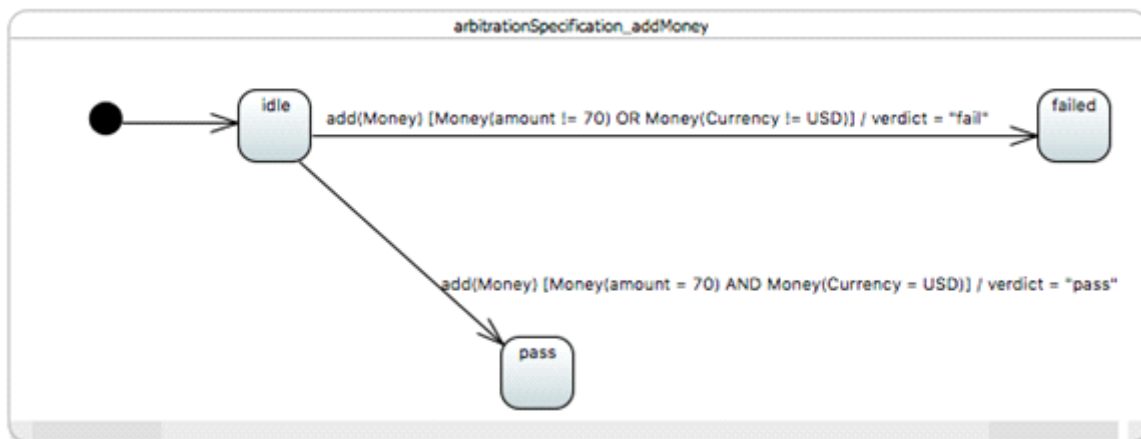


Figure A.36 - User-Defined Arbitration Specification

The second [test requirement](#) *approveAddDifferentMoney* is verified by [test case](#) *addDifferentMoney* (see figure below). For this [test case](#), a third [test item](#) of class *MoneyBag* is needed in order to be able to distinguish money of different currencies. This [test case](#) uses the default [arbitration specifications](#) that should be provided by the tool vendor.

<sup>4</sup> The default arbitration is provided by the tool vendor.

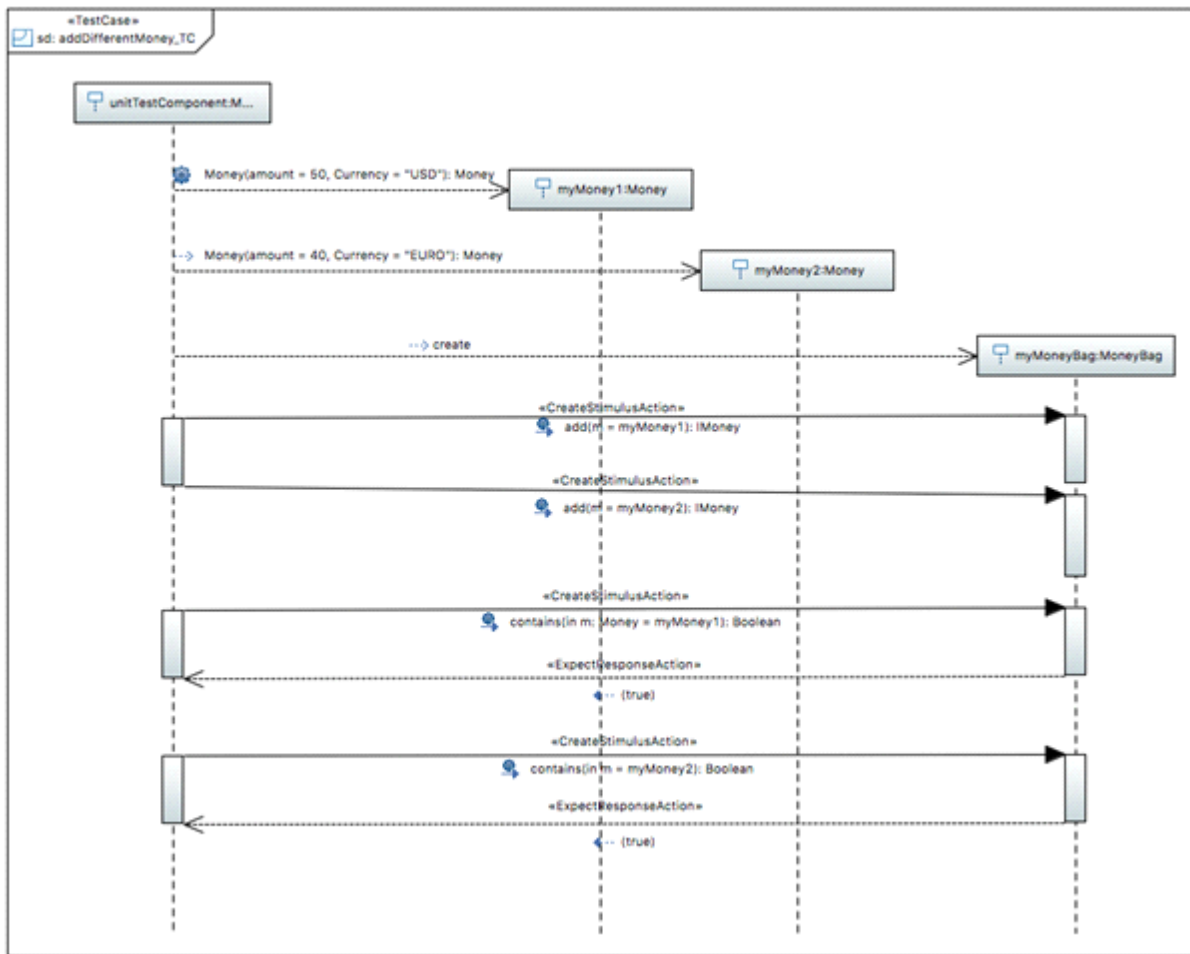


Figure A.37 - Test Case AddDifferentMoney

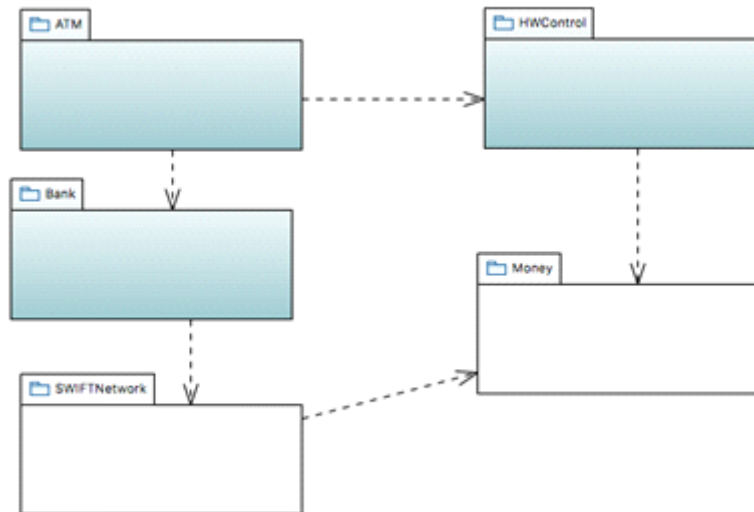
### A.5.3 Integration Testing Example

This section illustrates how UTP 2 can be used for specifying tests at integration [test level](#). The main focus of integration testing is the communication of the [test item](#) and its [test components](#).

The [test requirements](#) are to verify the logic of the ATM machine when a user initiates a money deposit transaction to an account in another part of the world. Thus, the [test requirements](#) include:

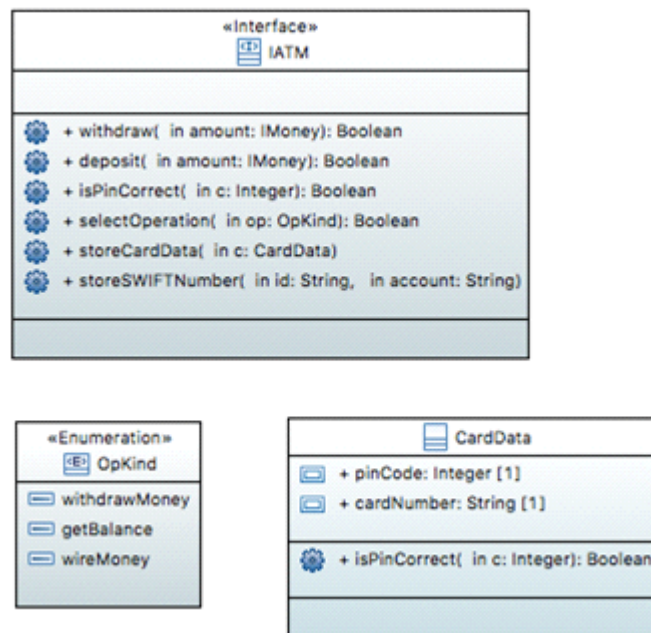
- The hardware terminal (*HWControl*) provides user's card and user's pin-code. The ATM shall authorize this card and its pin-code.
- After a successful authorization of user's data, money shall be deposited into the bank. The ATM shall assure a correct transaction communication with the *Bank*.

Since the logic of ATM itself is being tested, the rest of the IBEN (i.e. *HWControl*, *Bank*, and *SWIFTNetwork*) shall be emulated. The figure below shows the [test items](#) of blue color.



**Figure A.38 - Test Items for Integration Test of IBEN**

The logic of the *ATM* is specified in the figure below. It imports both the *HWControl* and the *Bank* packages where only the interfaces to the hardware and the bank are needed. Component *ATM* controls the logic of ATM and is the [test item](#) for our integration test. It provides the *IATM* interface for the control logic and communicates with the hardware and the bank via interface. Since the hardware and the bank are emulated in the test, only the interface classes of the *HWControl* and *Bank* packages are needed (see the following three figures).



**Figure A.39 - Classes and Interface in Package ATM**

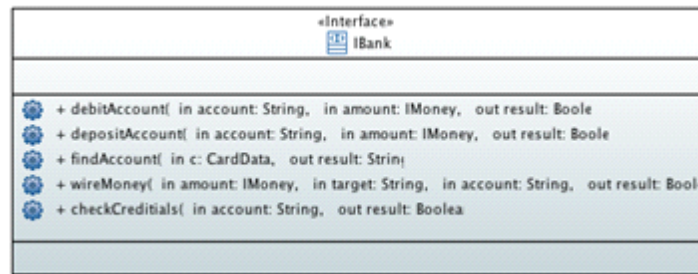


Figure A.40 - Interface Class in Package Bank

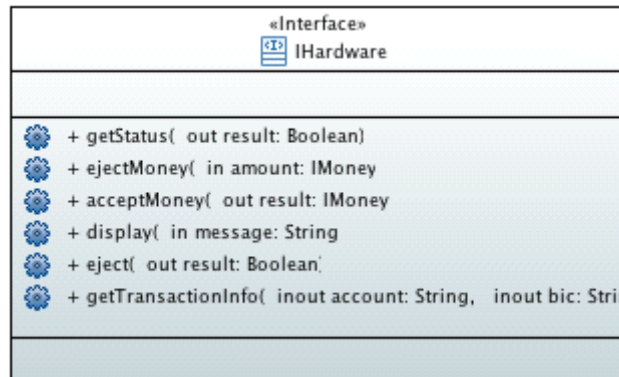


Figure A.41 - Interface Class in Package HWControl

The figure below shows the [test configuration](#) of the test. It specifies the relationship between the [test item](#), the emulated [test component](#)s for the hardware and bank (*hw* and *be*), and a card data management component (*card*).

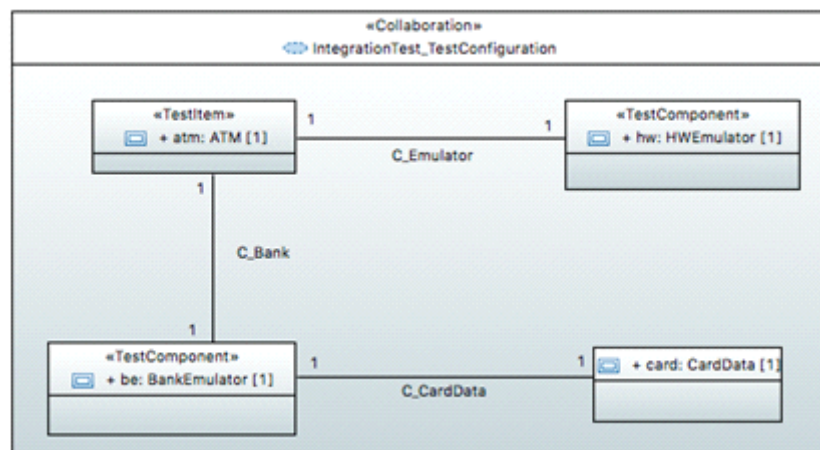


Figure A.42 - Integration Test Configuration

The figure below shows the binding of the [test configuration](#) to [test case](#) *invalidPIN\_TCI*.

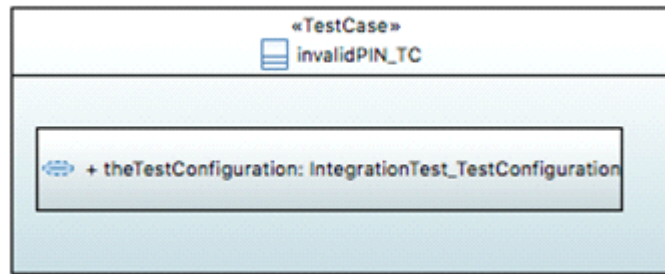


Figure A.43 - Binding of Test Configuration to Test Case invalidPIN\_TC

The ATM integration test package (see figure below) shows the model elements necessary to specify integration tests. It imports the ATM package of the system model in order to get access to the elements to be tested. The package contains two [test components](#): *BankEmulator* and *HWEulator* and three testcases: *validWiring*, *invalidPIN*, and *authorizeCard*. The [test components](#) *BankEmulator* and *HWEulator* realize the interfaces of the *HWControl* and *Bank* packages and serve as emulators in order to communicate with the *ATM*.

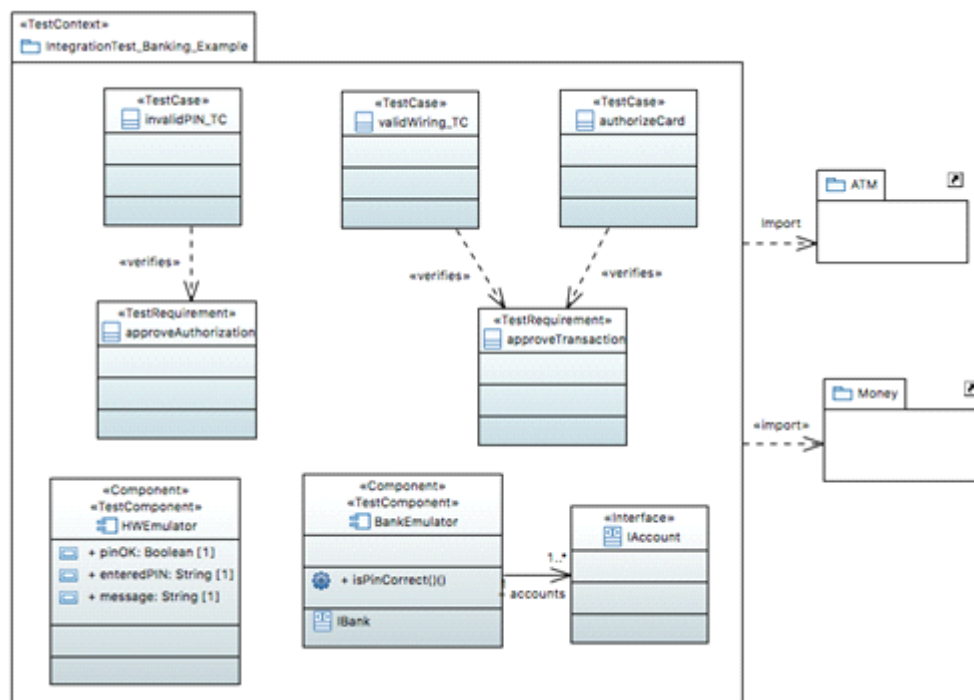


Figure A.44 - Test Context for Integration Test

The following section only concentrates on the modeling of the [test case](#) *invalidPIN*, which approves the requirement of a correct authorization mentioned on earlier. The objective of this test is:

- Verify that if a valid card is inserted, and an invalid pin-code is entered, the user is prompted to re-enter the pin-code.

Behaviors of a [test case](#) can be specified using any UML behavior Diagrams (e.g. Interaction Diagram, State Machine, Sequence Diagram etc.). In this case, UML Sequence Diagram has been chosen (see figure below).

The signals between the [test components](#) are all stereotyped by UTP 2 actions (e.g. <<CreateStimulus-Action>>). By doing so, the default [arbitration specifications](#) are activated and it is assured that unexpected behavior is caught within the [arbitration specifications](#). In parallel, the setting of [test case verdicts](#) is also done in the [arbitration](#)

[specification](#). The response time of *isPinCorrect* should last no more than 3 seconds, otherwise the [arbitration specification](#) <<ExpectResponseAction>> will be carried out.

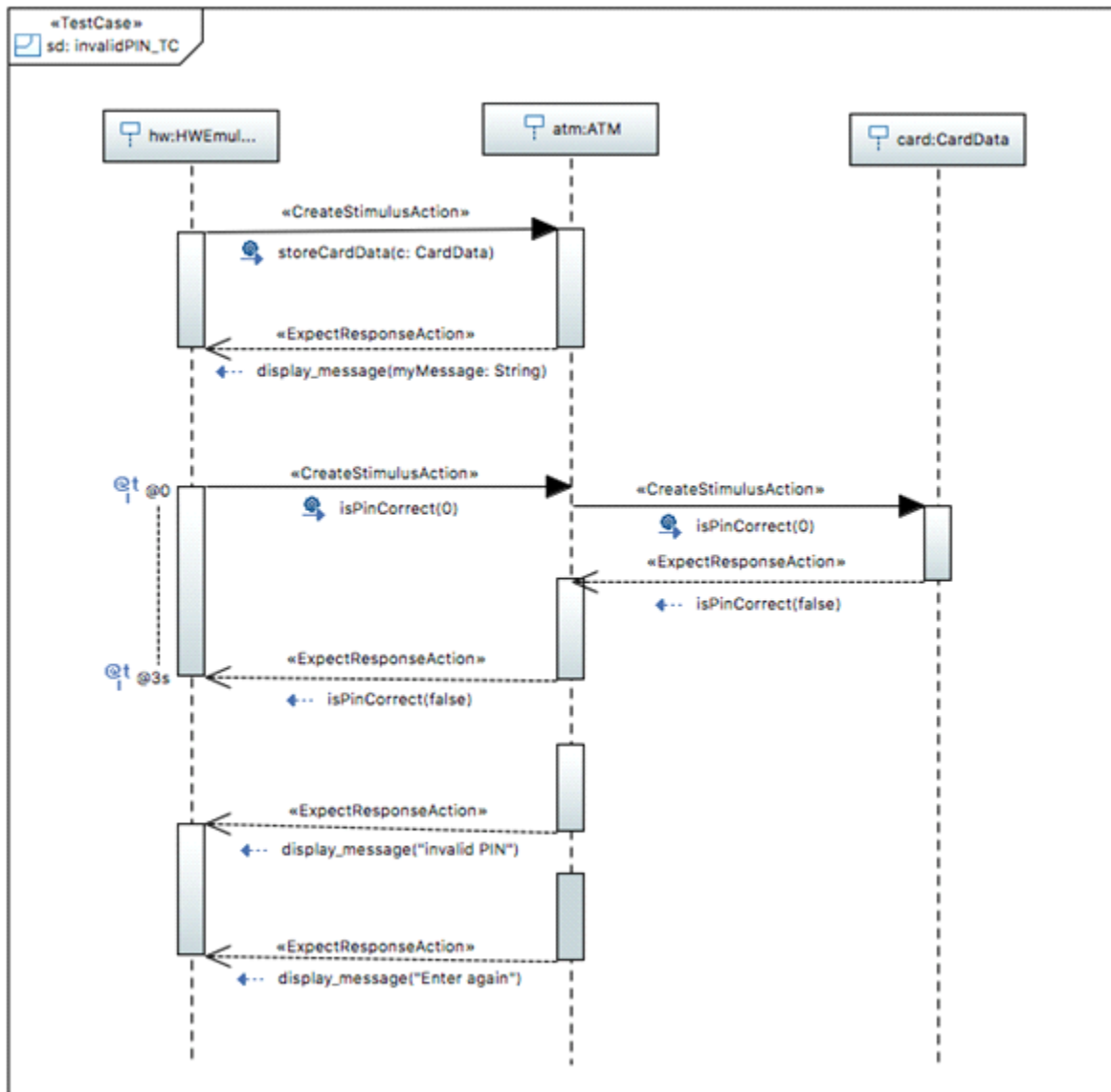


Figure A.45 - Test Case invalidPIN\_TC

In many cases, there's a need to specify the detailed behavior of individual [test components](#) (e.g., for test generation purposes). Therefore, state machines provide good means. The figure below shows an excerpt of test behavior for the *HWEmulator* [test component](#) which corresponds to [test case invalidPIN\\_TC](#). The validation action <<ExpectResponseAction>> evaluates the test result and sets the [test case verdict](#).

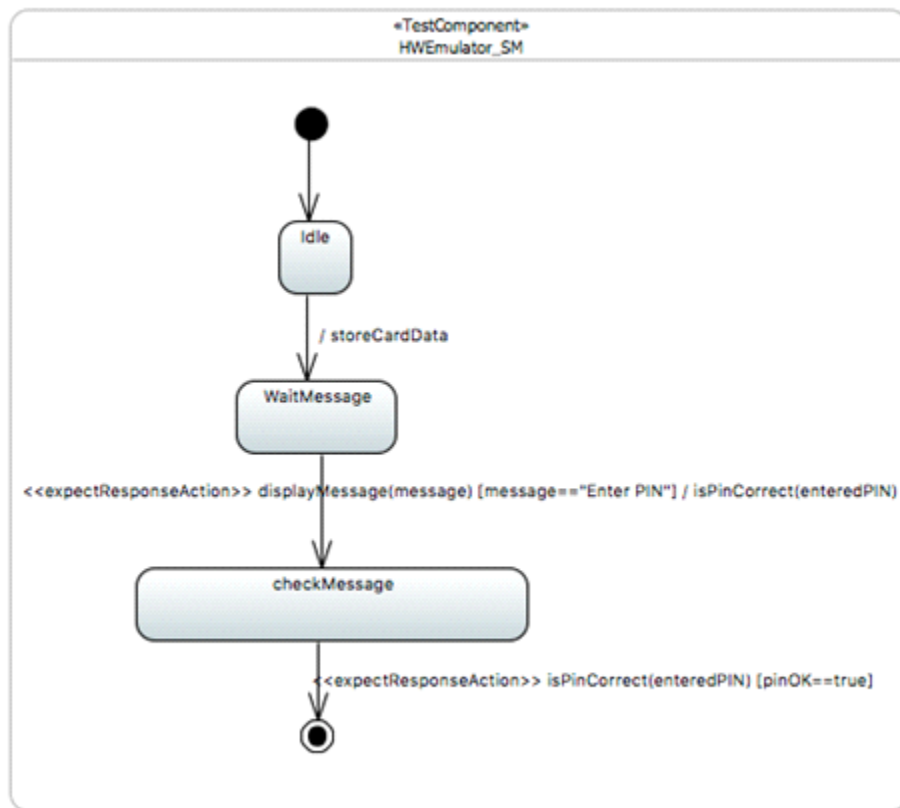


Figure A.46 - Statemachine for the Hardware Emulator

#### A.5.4 System Test Example

This chapter shows the UTP2 model for system level tests. The test model shows an interbank exchange scenario where a customer with an EU bank account deposits money into his/her account from an ATM in the United States.

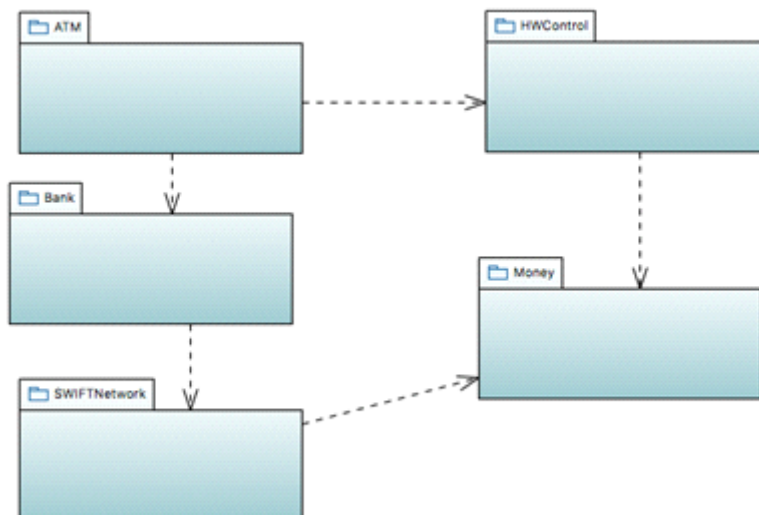
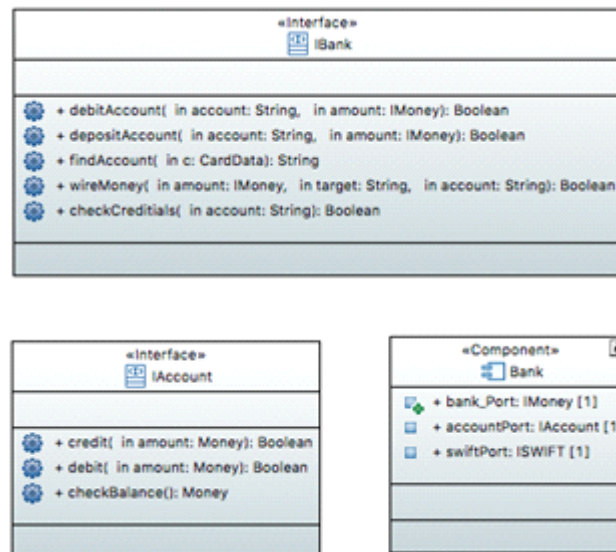


Figure A.47 - Packages with Test Items for System Test of IBEN

In order to perform the system testing of IBEN, all the five packages in the system model are needed. The packages

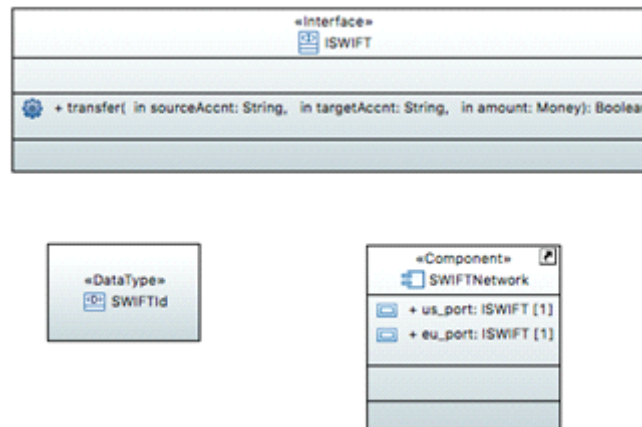


*ATM*, *Money*, and *HWControl* are known from the previous examples. The figure below illustrates the contents of the *Bank* package. The *IBank* interface provides methods to find, credit, and debit accounts. It checks credentials and wires money from one account to another. The *IAccount* interface also provides operations to credit and debit accounts, in addition to checking the balance of an account.



**Figure A.48 - Classes and Components in Bank Package**

The figure below shows the content of the *SWIFTNetwork* package. The *ISWIFT* interface provides an operation to transfer a given amount from a source account to a target account. Since system testing is a black-box test strategy, only the communication between the interfaces is of interest.



**Figure A.49 - Classes and Components in the SWIFTNetwork Package**

For the system testing, the following [test requirements](#) are defined:

1. EU and US initiated transactions must behave correctly.
2. Money can be transferred from an US account to an EU account, and vice-versa.
3. An invalid transfer should be identified and canceled.
4. The system should handle up to 1000000 transactions in parallel without system failure.

The figure below shows the system [test context](#). The [test items](#) are the SWIFTNetwork, the US and EU Banks, and the ATM systems. Three [test cases](#) called *runUSTrxn*, *runEUTrxn* and *loadTest* are specified in this [test context](#). The [test cases](#) *runUSTrxn* or *runEUTrxn* approve that a transaction that is initiated from the US ATM will be transferred

to the EU Bank, or vice versa. The [test case](#) *loadTest* verifies a non-functional [test requirement](#). It shall approve that IBEN behaves correctly even by high transaction requests. Two additional [test components](#) called *TransactionController* and *LoadManager* provide the capability to execute and verify that the money is transferred correctly.

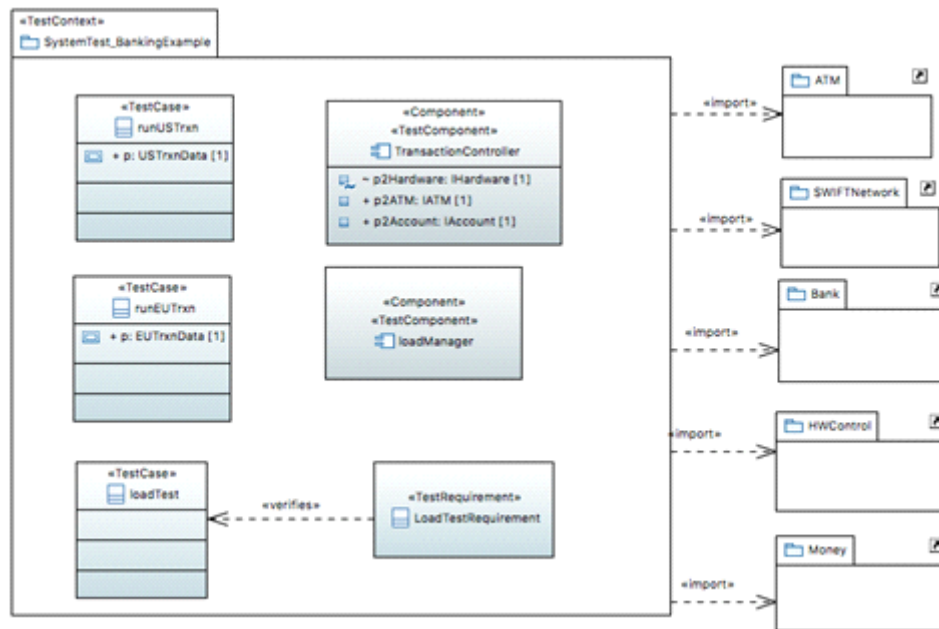


Figure A.50 - System Test Context

The [test configuration](#) is illustrated in the figure below. The *TransactionController* drives both ATMs on the European and US sides and is used to represent the accounts for both the US and EU banks. The *LoadManager* provides and controls the workload of the load test. It has access to the test data in the *SystemTestDataPool*.

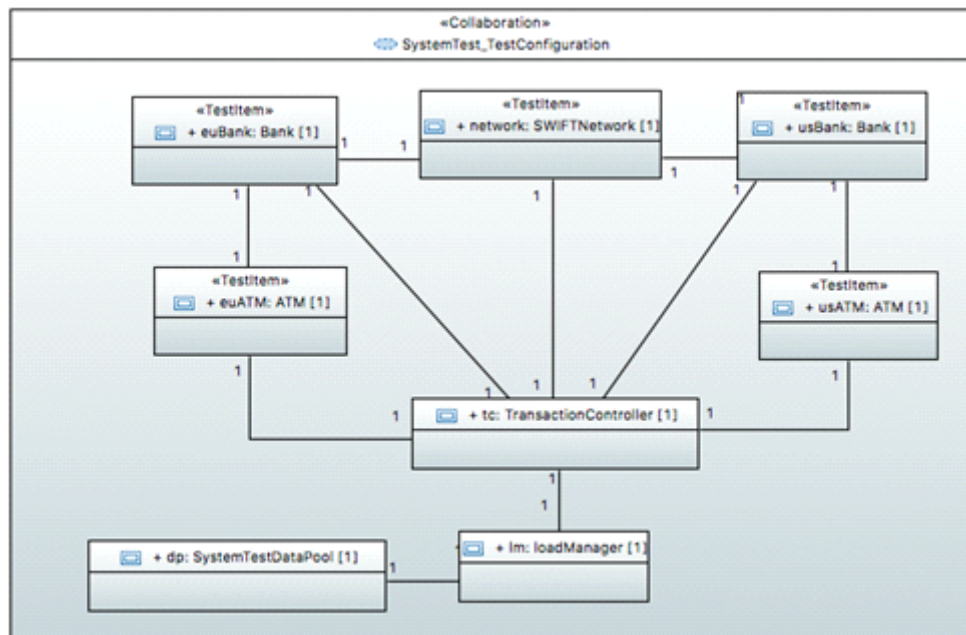
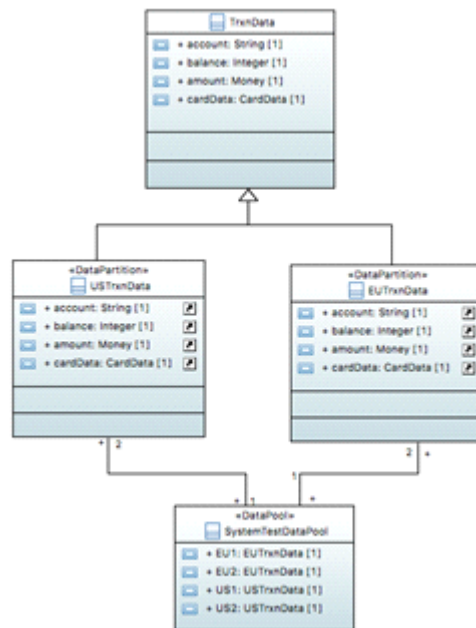


Figure A.51 - System Test Configuration

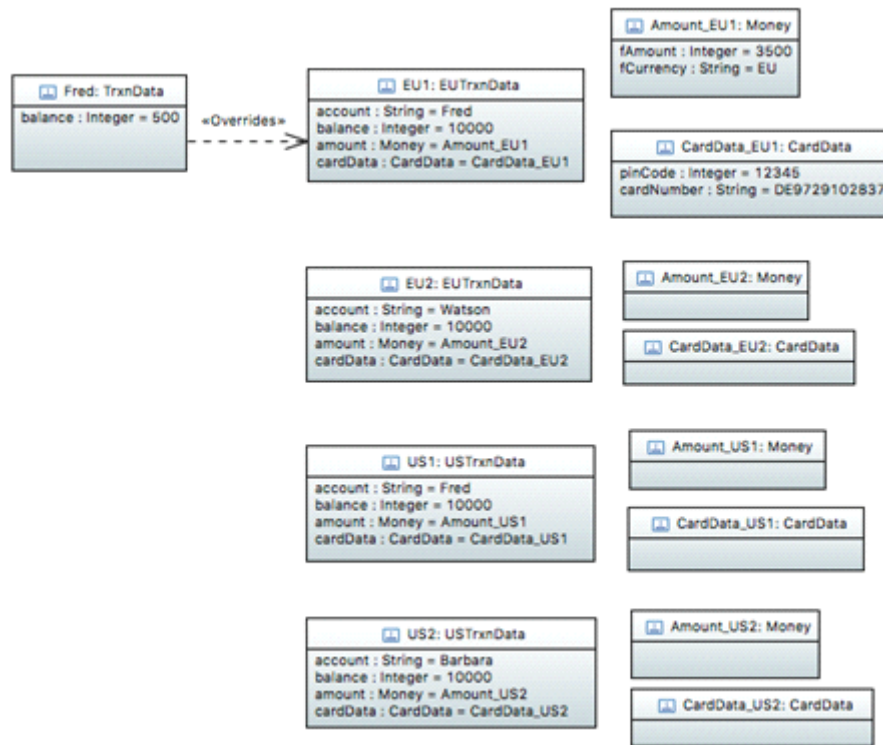
The figure below shows data used for the system test. *TrxnData* defines the transaction data.



**Figure A.52 - Test Data and its Variations**

The [data pool](#) *SystemTestDataPool* contains instances of *TrxnData* called *EU1*, *EU2*, *US1* and *US2* (see figure below). Two [data partitions](#) are defined in order to distinguish the EU transactions from the US transactions. These [data partitions](#) are chosen from the [data pool](#) and have two data samples each. Data instance *EU1* is shown in the diagram explicitly by all its attribute values<sup>5</sup>. Another data instance called *Fred* defines a modification of *EU1*, where 500 override the balance of 10000.

<sup>5</sup> This diagram only shows the data values of EU1. Those of EU2, US1 and US2 are equivalently defined.



**Figure A.53 - Data Instances and its Modification**

The figure below illustrates the behavior of [test case](#) *loadTest* which shall verify the [test requirement](#) 4 listed above. This [test case](#) shall approve that minimum 100 and maximum 1000000 transactions can be successfully handled in parallel. The *LoadArbitrationSpecification* will assure that whenever a transaction fails, the whole test will fail.

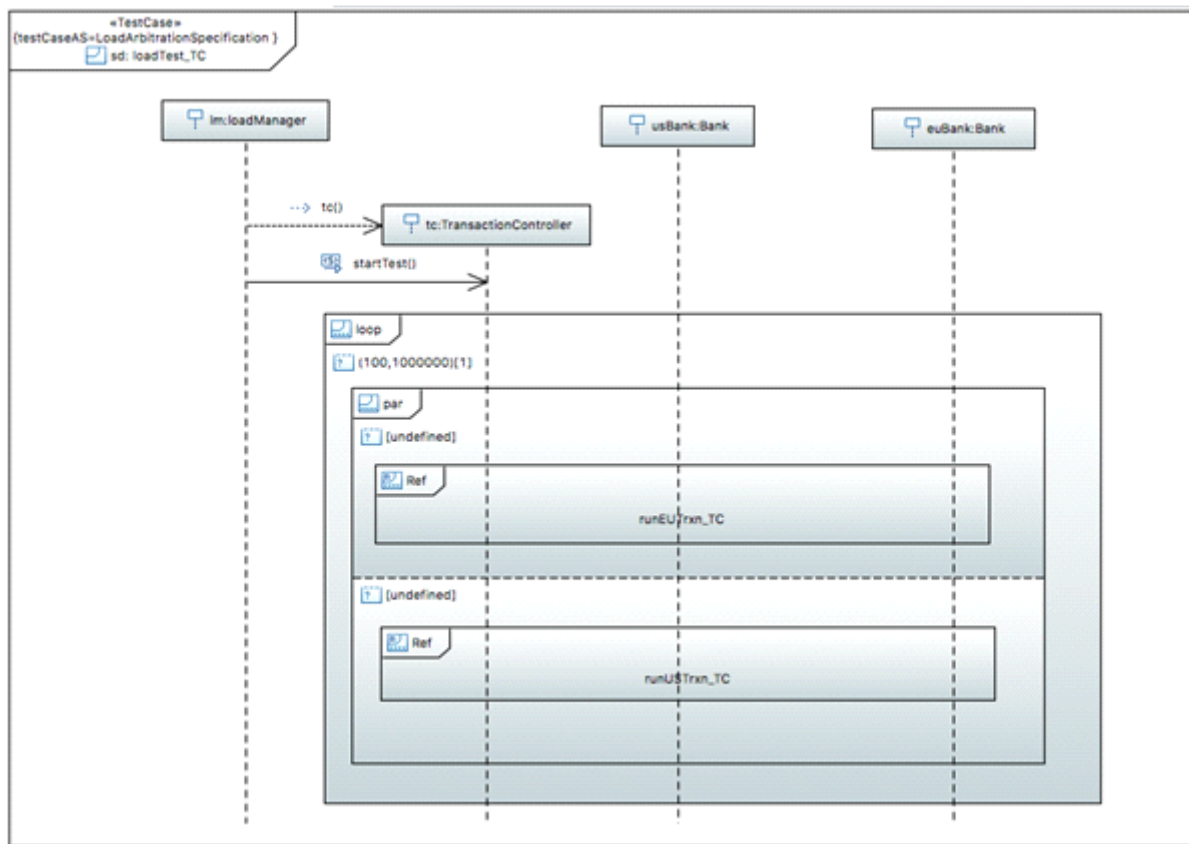


Figure A.54 - Test Case loadTest

### A.5.5 References

[UTP1.2] Object Management Group: "UML Testing Profile, version 1.2", OMG Document Number: formal/2013-04-03

[JUnit\_Example] <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>

[JUnit\_web] [www.junit.org](http://www.junit.org)

This page intentionally left blank.

# Annex B (Informative): Mappings

## B.1 Mapping between UTP 1 and UTP 2

The following table summarizes the changes on stereotypes of UTP 2 compared with UTP 1.2:

Name	Change from UTP 1.2
<a href="#">ActualParameterValue</a>	«ActualParameterValue» was newly introduced by UTP 2.1.
<a href="#">ActualResponseLogEntry</a>	«ActualResponseLogEntry» was newly introduced by UTP 2.1.
<a href="#">Alternative</a>	« <a href="#">Alternative</a> » has been newly introduced by UTP 2.
<a href="#">AlternativeArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">AnyValue</a>	Changed and renamed from UTP 1.2. In UTP 1.2, «AnyValue» was called «LiteralAny» and extended LiteralSpecification.
<a href="#">ArbitrationDirective</a>	«ArbitrationDirective» was newly introduced by UTP 2.2.
<a href="#">ArbitrationResult</a>	« <a href="#">ArbitrationResult</a> » has been newly introduced by UTP 2.
<a href="#">ArbitrationSpecification</a>	« <a href="#">ArbitrationSpecification</a> » has been newly introduced into UTP 2.
<a href="#">ArbitrationSpecificationBinding</a>	«ArbitrationSpecificationBinding» was newly introduced by UTP 2.2.
<a href="#">ArbitrationTarget</a>	« <a href="#">ArbitrationTarget</a> » was newly introduced by UTP 2.2.
<a href="#">AtomicProceduralElement</a>	«AtomicProceduralElement» has been newly introduced by UTP 2.
<a href="#">AtomicProceduralElementArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">AtomicProceduralElementLogEntry</a>	«AtomicProceduralElementLogEntry» was newly introduced by UTP 2.1
<a href="#">BoundaryValueAnalysis</a>	«BoundaryValueAnalysis» has been newly introduced by UTP 2.
<a href="#">CauseEffectAnalysis</a>	«CauseEffectAnalysis» has been newly introduced by UTP 2.
<a href="#">ChecklistBasedTesting</a>	«ChecklistBasedTesting» has been newly introduced by UTP 2.
<a href="#">CheckPropertyAction</a>	«CheckPropertyAction» has been newly introduced by UTP 2.
<a href="#">CheckPropertyArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">CheckPropertyLogEntry</a>	«CheckPropertyLogEntry» was newly introduced by UTP 2.1.
<a href="#">ChoiceOfValues</a>	«ChoiceOfValues» has been newly introduced by UTP 2.
<a href="#">ClassificationTreeMethod</a>	«ClassificationTreeMethod» has been newly introduced by UTP 2.
<a href="#">CollectionExpression</a>	
<a href="#">CombinatorialTesting</a>	«CombinatorialTesting» has been newly introduced by UTP 2.
<a href="#">ComplementedValue</a>	«ComplementedValue» has been newly introduced by UTP 2.
<a href="#">Complements</a>	«Complements» has been newly introduced by UTP 2.
<a href="#">CompoundProceduralElement</a>	«CompoundProceduralElement» has been newly introduced by UTP 2.
<a href="#">CompoundProceduralElementArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">CreateLogEntryAction</a>	«CreateLogEntryAction» has been newly introduced by UTP 2.
<a href="#">CreateLogEntryArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">CreateLogEntryLogEntry</a>	«CreateLogEntryLogEntry» was newly introduced by UTP 2.1.
<a href="#">CreateStimulusAction</a>	«CreateStimulusAction» has been newly introduced by UTP 2.
<a href="#">CreateStimulusArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">CreateStimulusLogEntry</a>	«CreateStimulusLogEntry» was newly introduced by UTP 2.1.
<a href="#">DataPartition</a>	«DataPartition» has been newly introduced by UTP 2.
<a href="#">DataPool</a>	Changed from UTP 1.2. In UTP 1.2 « <a href="#">DataPool</a> » extended both Classifier and Property.

Name	Change from UTP 1.2
<a href="#">DataProvider</a>	«DataProvider» has been newly introduced by UTP 2.
<a href="#">DataSpecification</a>	«DataSpecification» has been newly introduced by UTP 2.
<a href="#">DecisionTableTesting</a>	«DecisionTableTesting» has been newly introduced by UTP 2.
<a href="#">EquivalenceClassPartitioning</a>	«EquivalenceClassPartitioning» has been newly introduced by UTP 2.
<a href="#">ErrorGuessing</a>	«ErrorGuessing» has been newly introduced by UTP 2.
<a href="#">ExpectResponseAction</a>	« <a href="#">ExpectResponseAction</a> » has been newly introduced by UTP 2.
<a href="#">ExpectResponseArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">ExperienceBasedTechnique</a>	«ExperienceBasedTechnique» has been newly introduced by UTP 2.
<a href="#">ExploratoryTesting</a>	«ExploratoryTesting» has been newly introduced by UTP 2.
<a href="#">Extends</a>	«Extends» has been newly introduced by UTP 2.
<a href="#">FormalParameterReference</a>	«FormalParameterReferenece» was newly introduced by UTP 2.1.
<a href="#">GenericTestDesignDirective</a>	«GenericTestDesignDirective» has been newly introduced by UTP 2.
<a href="#">GenericTestDesignTechnique</a>	«GenericTestDesignTechnique» has been newly introduced by UTP 2.
<a href="#">InvocationLogEntry</a>	«InvocationLogEntry» was newly introduced by UTP 2.1
<a href="#">InvocationLogEntryStructure</a>	«InvocationLogEntryStructure» was newly introduced by UTP 2.1
<a href="#">Loop</a>	«Loop» has been newly introduced by UTP 2.
<a href="#">LoopArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">MatchingCollectionExpression</a>	«CollectionExpression» has been newly introduced by UTP 2.
<a href="#">MessageEventLogEntry</a>	«MessageEventLogEntry» was newly introduced by UTP 2.1.
<a href="#">MessageEventLogEntryStructure</a>	«MessageEventLogEntryStructure» was newly introduced by UTP 2.1.
<a href="#">Morphing</a>	«Morphing» has been newly introduced by UTP 2.
<a href="#">Negative</a>	« <a href="#">Negative</a> » has been newly introduced by UTP 2.
<a href="#">NegativeArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">NSwitchCoverage</a>	«NSwitchCoverage» has been newly introduced by UTP 2.
<a href="#">OpaqueProceduralElement</a>	« <a href="#">OpaqueProceduralElement</a> » has been newly introduced by UTP 2.
<a href="#">OpaqueProceduralElementLogEntry</a>	«OpaqueProceduralElementLogEntry» was newly introduced by UTP 2.1.
<a href="#">overrides</a>	«overrides» was renamed by UTP 2. In UTP 1.2, it was named «modifies».
<a href="#">PairwiseTesting</a>	«PairwiseTesting» has been newly introduced by UTP 2.
<a href="#">Parallel</a>	« <a href="#">Parallel</a> » has been newly introduced by UTP 2.
<a href="#">ParallelArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">ProceduralElement</a>	«ProceduralElement» has been newly introduced by UTP 2.
<a href="#">ProceduralElementArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">ProcedureInvocation</a>	«ProcedureInvocation» has been newly introduced by UTP 2.
<a href="#">ProcedureInvocationArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">ProcedureInvocationLogEntry</a>	«ProcedureInvocationLogEntry» was newly introduced by UTP 2.1.
<a href="#">ProcedureInvocationLogEntryStructure</a>	«ProcedureInvocationLogEntryStructure» was newly introduced by UTP 2.1
<a href="#">RangeValue</a>	«RangeValue» has been newly introduced by UTP 2.
<a href="#">Refines</a>	«Refines» has been newly introduced by UTP 2.
<a href="#">RegularExpression</a>	«RegularExpression» has been newly introduced by UTP 2.
<a href="#">RoleConfiguration</a>	« <a href="#">RoleConfiguration</a> » is newly introduced in UTP 2.
<a href="#">Sequence</a>	«Sequence» has been newly introduced by UTP 2.
<a href="#">SequenceArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">StateCoverage</a>	«StateCoverage» has been newly introduced by UTP 2.



Name	Change from UTP 1.2
<a href="#">StateTransitionTechnique</a>	«StateTransitionTechnique» has been newly introduced by UTP 2.
<a href="#">SuggestVerdictAction</a>	«SuggestVerdictAction» has been newly introduced by UTP 2.
<a href="#">SuggestVerdictArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">SuggestVerdictLogEntry</a>	«SuggestVerdictLogEntry» was newly introduced by UTP 2.1.
<a href="#">TestCase</a>	Changed from UTP 1.2. «TestCase» extended Behavior and Operation in UTP 1.2.
<a href="#">TestCaseArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">TestCaseLog</a>	Newly introduced by UTP 2.
<a href="#">TestComponent</a>	Changed from UTP 1.2. In UTP 1.2., « <a href="#">TestComponent</a> » only extended Class.
<a href="#">TestComponentConfiguration</a>	« <a href="#">TestComponentConfiguration</a> » has been newly introduced into UTP 2.
<a href="#">TestConfiguration</a>	« <a href="#">TestConfiguration</a> » has been newly introduced into UTP 2. It was conceptually represented by the composite structure of a «TestContext» in UTP 1.2.
<a href="#">TestConfigurationRole</a>	« <a href="#">TestConfigurationRole</a> » is newly introduced in UTP 2.
<a href="#">TestContext</a>	Changed from UTP 1.2. In UTP 1.2 «TestContext» extended StructuredClassifier and BehavioredClassifier as well as incorporated the concepts TestSet, TestExecutionSchedule and TestConfiguration into a single concept.
<a href="#">TestDesignDirective</a>	«TestDesignDirective» has been newly introduced by UTP 2.
<a href="#">TestDesignDirectiveStructure</a>	«TestDesignDirectiveStructure» has been newly introduced by UTP 2.
<a href="#">TestDesignInput</a>	«TestDesignInput» has been newly introduced by UTP 2.
<a href="#">TestDesignTechnique</a>	«TestDesignTechnique» has been newly introduced by UTP 2.
<a href="#">TestDesignTechniqueStructure</a>	«TestDesignTechniqueStructure» has been newly introduced by UTP 2.
<a href="#">TestDirective</a>	«TestDirective» has been newly introduced by UTP 2.1.
<a href="#">TestDirectiveStructure</a>	«TestDirectiveStructure» has been newly introduced by UTP 2.1.
<a href="#">TestExecutionSchedule</a>	« <a href="#">TestExecutionSchedule</a> » has been newly introduced by UTP 2. It was conceptually represented as the classifier behavior of a «TestContext» in UTP 1.2.
<a href="#">TestItem</a>	« <a href="#">TestItem</a> » has been newly introduced into UTP 2 and supersedes the «SUT» stereotype in UTP 1.
<a href="#">TestItemConfiguration</a>	« <a href="#">TestItemConfiguration</a> » has been newly introduced into UTP 2.
<a href="#">TestLog</a>	Changed from UTP 1.2. In UTP 1.2 «TestLog» was used to capture the execution of a test case or a test set (called test content in UTP 1.2). In UTP 2, two dedicated concepts have been newly introduced therefore (i.e., «TestCaseLog» and «TestSetLog»).
<a href="#">TestLogElement</a>	«TestLogElement» was newly introduced by UTP 2.1.
<a href="#">TestLogEntry</a>	Changed from UTP 1.2. In UTP 1.2, «TestLogEntry» extended OccurrenceSpecification.
<a href="#">TestLogStructure</a>	Newly introduced by UTP 2.
<a href="#">TestLogStructureBinding</a>	Newly introduced by UTP 2.
<a href="#">TestObjective</a>	Changed from UTP 1.2. In UTP 1.2, «TestObjective» was called «TestObjectiveSpecification».
<a href="#">TestProcedure</a>	«TestProcedure» has been newly introduced by UTP 2.
<a href="#">TestRequirement</a>	« <a href="#">TestRequirement</a> » has been newly introduced into UTP 2.
<a href="#">TestSet</a>	« <a href="#">TestSet</a> » has been newly introduced by UTP 2. It was part of the TestContext in UTP 1.2.
<a href="#">TestSetArbitrationSpecification</a>	Newly introduced by UTP 2.
<a href="#">TestSetLog</a>	Newly introduced by UTP 2.
<a href="#">TestTechnique</a>	«TestTechnique» has been newly introduced by UTP 2.1.
<a href="#">TestTechniqueStructure</a>	«TestTechniqueStructure» has been newly introduced by UTP 2.1.

Name	Change from UTP 1.2
<a href="#">TransitionCoverage</a>	«TransitionCoverage» has been newly introduced by UTP 2.
<a href="#">TransitionPairCoverage</a>	«TransitionPairCoverage» has been newly introduced by UTP 2.
<a href="#">UseCaseTesting</a>	«UseCaseTesting» has been newly introduced by UTP 2.
<a href="#">verifies</a>	«verifies» has been newly introduced into UTP 2. In UTP 1.2 the «verify» stereotype from SysML was recommended.

The three primitive data types including Timepoint, Duration, and Timezone are also removed from UTP 2.

The following stereotypes are also removed from UTP 2: «GetTimeZoneAction», «SetTimeZoneAction», «DataSelector», «CodingRule», «LiteralAnyOrNull», and «TestLogEntry».

# Annex C (Informative): Value Specification Extensions

## C.1 Profile Summary

The following table gives a brief summary on the stereotypes introduced by the UML Testing Profile 2 (listed in the second column of the table). The first column specifies the mapping to the conceptual model shown in the previous section and the third column specifies the UML 2.5 metaclasses that are extended by the stereotypes.

Stereotype	UML 2.5 Metaclasses	Concepts
ChoiceOfValues	<a href="#">Expression</a>	<a href="#">data</a>
CollectionExpression	<a href="#">Expression</a>	<a href="#">data</a>
ComplementedValue	<a href="#">ValueSpecification</a>	<a href="#">data</a>
MatchingCollectionExpression	<a href="#">Expression</a>	<ul style="list-style-type: none"><li>• <a href="#">data</a></li><li>• <a href="#">data specification</a></li></ul>
RangeValue	<a href="#">Expression</a>	<a href="#">data specification</a>

## C.2 Non-normative data value extensions

In addition to the normative [ValueSpecification extensions](#) of UTP, for sake of simplicity, UTP provides also some more [extensions](#) as part of this non-normative annex. These kinds of ValueSpecifications are:

- **Complemented:** Represents a set of expected [response](#) argument values for a known type described by a complemented set of values described the underlying ValueSpecification and checks if actual [response](#) argument value belongs to that set.
- **[RangeValue](#):** Represents a set of ordered expected [response](#) argument values for a known type described by its upper and lower boundaries. The Actual [response](#) argument value matches with each expected one if the actual one belongs to the set defined by its boundaries.
- **[ChoiceOfValues](#):** Represents a set of expected [response](#) argument values for a known type described by an enumeration of values. The actual [response](#) argument value matches with expected one if the actual one belongs to the set defined by the enumeration.
- **[MatchingCollectionExpression](#):** Represents a set of expected [response](#) argument collection values for a known type described by the members of the expected collection and the matching kind operator. The actual [response](#) argument collection value match with the expected ones if the actual one belongs to the set of collections values defined by members and the collection matching kind.
- **[CollectionExpression](#):** Represents a collection value used for defining argument collection values for stimuli or expected [response](#) values. If used as expected [response](#) argument collection value the actual [response](#) argument collection value matches with the expected one if their respective members match with each other. In case ordering is important, the members should also occur in the exact same order.

Implementations of the profile are free to decide how to incorporate and offer the non-normative [extensions](#) to the users.

### C.1.1 Overview of non-normative ValueSpecification Extensions

The diagram below shows some additional, non-normative [extensions](#) to the UML ValueSpecifications metamodel. These UTP ValueSpecification [extensions](#) are deemed helpful for testers in order to be express [data](#) values used to specify the payload for stimuli and expected [responses](#). It is treated as non-normative [extension](#) nonetheless, because all the given [extensions](#) could also be expressed by means of the OCL, which is considered as integral part of UML. However, OCL imposes additional knowledge on the test engineers which may result in a reduced acceptance by the industrial testing community. Therefore, this non-normative [extension](#) to the UTP provides dedicated concepts as special ValueSpecifications which can be immediately used by the testers without knowing anything about OCL at all. All these extended ValueSpecifications have been taken over from [TTCN-3] where they have been proven beneficial for the design of executable [test cases](#) in the industry since many years.

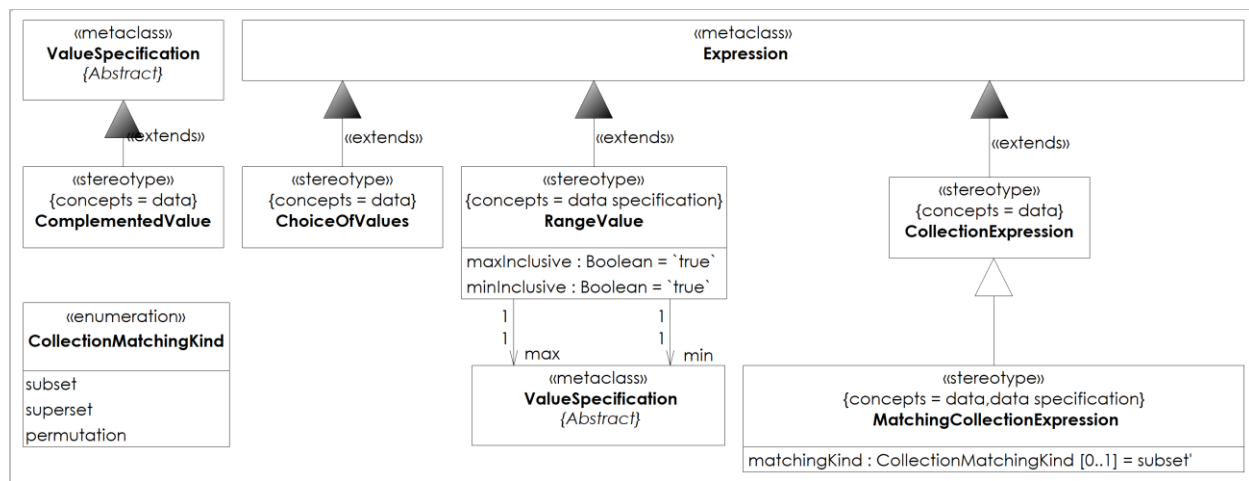


Figure 9.8 - Overview of non-normative ValueSpecification Extensions

## C.2.2 Stereotype Specifications

### C.2.2.1 ChoiceOfValues

Description	<p><a href="#">ChoiceOfValues</a> represents an enumeration of possible values defined for the payload of an expected <a href="#">response</a>, out of which at least one entry must match with the payload of the actual <a href="#">response</a>.</p> <p>If a choice of possible values is used in a check <a href="#">response data</a> action, then the enumerated values denote several possible check <a href="#">response data</a> actions out of which one possible value must match with the actually received <a href="#">response data</a>.</p> <p>The list of possible values is expressed as the list of ValueSpecifications composed by the underlying Expression's operand attribute. As defined above, any available ValueSpecification can be enumerated as choice of possible values.</p> <p>As a recommendation, <a href="#">ChoiceOfValues</a> must either be only in check <a href="#">response data</a> actions in <a href="#">test case</a> or for test generation. It is highly recommended to not use ChoiceOfValues as payload for <a href="#">create stimulus action</a> for it may negatively affect the repeatability of <a href="#">test case</a> executions.</p>
Extension	<a href="#">Expression</a>
Change from UTP 1.2	«ChoiceOfValues» has been newly introduced by UTP 2.

### C.2.2.2 CollectionExpression

Description	<p>A <a href="#">CollectionExpression</a> enables the modelling of collections based on the ValueSpecification metaclass Expression. Using collections values is essential when specifying stimuli and expected <a href="#">responses</a> of a <a href="#">test case</a>. By means of the stereotype «<a href="#">CollectionExpression</a>» it is possible to describe inline values for a given ConnectableElement (i.e., Property or Parameter) and use those collections values as payload for a <a href="#">stimulus</a> or an expected <a href="#">response</a> as required. The kind (i.e., order and uniqueness) of the <a href="#">CollectionExpression</a> is prescribed by the related MultiplicityElement (i.e., Property or Parameter) of this <a href="#">CollectionExpression</a>.</p> <p>«<a href="#">CollectionExpression</a>» might be used as payload for both <a href="#">stimulus</a> and expected <a href="#">responses</a>. If it represents the payload of an expected <a href="#">response</a>, the payload of the actual <a href="#">responses</a> must match with the expected <a href="#">CollectionExpression</a> with respect to both, items listed in the collection and their respective index in the actual payload collection, if the corresponding ConnectableElement (i.e., Property or Parameter) is ordered. Any deviation is supposed to result in a mismatch.</p>
Extension	<a href="#">Expression</a>
Sub Class	<a href="#">MatchingCollectionExpression</a>

### C.2.2.3 ComplementedValue

Description	A <a href="#">ComplementedValue</a> specifies a set of values that are not contained in the set specified by the genuine ValueSpecification.
Extension	<a href="#">ValueSpecification</a>
Change from UTP 1.2	«ComplementedValue» has been newly introduced by UTP 2.

### C.2.2.4 MatchingCollectionExpression

Description	<p>A <a href="#">MatchingCollectionExpression</a> is a <a href="#">CollectionExpression</a> that enables the tester to define matching criteria when used as the payload of an expected <a href="#">response</a>. Thus, it is not allowed to use a <a href="#">MatchingCollectionExpression</a> as payload for a <a href="#">stimulus</a>, but only as payload for expected <a href="#">responses</a>.</p> <p>The CollectionMatchingKind attribute of the <a href="#">CollectionExpression</a> determines the matching mechanism that must be applied on the actual payload when received in order to calculate a match or mismatch of actual and expected <a href="#">responses</a>. These matching kinds are the following:</p> <ul style="list-style-type: none"><li>• subset (default)</li><li>• superset</li><li>• permutation</li></ul> <p>If the corresponding MultiplicityElement (i.e., <b>Property</b> or Parameter) has is ordered (i.e., isOrdered = true), the collection items in the payload of the actual <a href="#">response</a> have to occur in the exact same order as the elements in the expected <a href="#">response</a>. Whether nested <a href="#">CollectionExpressions</a> are considered to be flattened for the comparison of expected and actual <a href="#">responses</a> is not defined in UTP 2.</p>
Extension	<a href="#">Expression</a>
Super Class	<a href="#">CollectionExpression</a>
Attributes	matchingKind : CollectionMatchingKind [0..1] = subset'
Constraints	<p>Must be used as payload for an expected responses</p> <p>A <a href="#">MatchingCollectionExpression</a> must only specify the payload of an expected response.</p> <p>Use of permutation matching kind</p> <p>The <a href="#">matchingKind</a> permutation must only be applied if the corresponding</p>

	ConnectableElement (i.e., <a href="#">Property</a> or Parameter) of the expected response has set <i>isOrdered</i> to <i>false</i> .
Change from UTP 1.2	«CollectionExpression» has been newly introduced by UTP 2.

### C.2.2.5 RangeValue

Description	<p>A <a href="#">RangeValue</a> represents a range between two naturally ordered boundaries, the upper and the lower bound. A <a href="#">RangeValue</a> can be used as wildcard value (i.e. qualified) instead of a concrete value (i.e. quantified). Conceptually, a range represents an enumeration of the values between the min and max values; however, it does not represent a set or collection of values. In that sense, <a href="#">RangeValue</a> is semantically equivalent to a ChoiceOfValue: ValueSpecification would explicitly enumerate all value between the min and max boundary. The eventual min value must always be less or equal than the eventual max value. In case that the min and max evaluate to the very same value, the range spans only a single value.</p> <p>If <a href="#">minInclusive</a> is set to true, the lower boundary (represented by the min value) is included in the range, otherwise it is excluded. Default is true (i.e., the min value is included). If <a href="#">maxInclusive</a> is set to true, the upper boundary (represented by the max value) is included in the range, otherwise it is excluded. Default is true, i.e., the max value is included. For example, if the min value evaluates to 10 and <a href="#">minInclusive</a> is set to false, the actual lowerBoundary is 11.</p> <p>If a <a href="#">RangeValue</a> is used in combination with an Integer- or Real-typed element, the lower and upper bounds describes the lowest and highest number of that numeric instance. If a <a href="#">RangeValue</a> used in combination with a String-typed element (or subclasses thereof), the lower and upper bounds determine the minimal and maximal length of that String's instance. Users are allowed to define other proprietary natural orderings (e.g., complex types and re-use <a href="#">RangeValue</a> to denote upper and lower boundaries for these types). The semantics how the ordering is defined; however, is out of scope of the <a href="#">RangeValue</a> concept.</p> <p>If applied to an expected <a href="#">response</a>, a <a href="#">RangeValue</a> matches with the actual received value from the <a href="#">test item</a>, and if the actual value is within the boundaries of the expected <a href="#">RangeValue</a>.</p>
Extension	<a href="#">Expression</a>
Attributes	maxInclusive : Boolean [1] = `true` minInclusive : Boolean [1] = `true`
Associations	min : ValueSpecification max : ValueSpecification
Constraints	Operands shall be empty  The attribute <i>operand</i> of the underlying <a href="#">Expression</a> must be empty.
Change from UTP 1.2	«RangeValue» has been newly introduced by UTP 2.

### C.2.3 Enumeration Specifications

Name	Description	Enumeration literals
CollectionMatching Kind	The CollectionMatchingKind lists different possibilities how a collections that specifies an expected <a href="#">response</a> shall be compared with an actual <a href="#">response</a> 's collection.	subset  The subsets matching kind indicates that all the elements in the expected response must be contained in the actual response, but there can be more elements in the actual response. The expected response is a real subset of the actual response.

Name	Description	Enumeration literals
		<p data-bbox="837 182 927 214">superset</p> <p data-bbox="837 220 1419 405">The superset matching kind indicates that the elements in expected response represent those values that might be contained in the actual response, but there can be possible less elements contained in the payload of the actual response. The expected response is a real superset of the actual response.</p> <p data-bbox="837 441 971 472">permutation</p> <p data-bbox="837 478 1430 663">The permutation matching kind indicates that all the elements of the expected response must be contained in the actual response, but in any arbitrary order. Permutation can only be applied if the corresponding MultiplicityElement (i.e., Property or Parameter) is unordered (i.e., isOrdered = false).</p>

## Annex D: Deprecated Elements

Some stereotypes and properties thereof became deprecated while the specification developed further. The following table lists all the deprecated elements, which are kept in the normative part of the specification to ensure backward compatibility. It is recommended to not use any deprecated concepts anymore for they are no longer maintained, nor ensured to endure over the next revisions.

Concept	Deprecated since	Replaced by
ProceduralElement::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
AtomicProceduralElement::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
Loop::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
Sequence::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
Parallel::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
Alternative::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
Negative::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
CreateStimulusAction::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
ExpectResponseAction::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
SuggestVerdictAction::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
CheckPropertyAction::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
CreateTestLogEntryAction::arbitrationSpecification	UTP 2.2	ArbitrationSpecificationBinding
TestCase::testCaseAS	UTP 2.2	ArbitrationSpecificationBinding
TestSet::testSetAS	UTP 2.2	ArbitrationSpecificationBinding
TestExecutionSchedule::testSetAS	UTP 2.2	ArbitrationSpecificationBinding



# Annex E: Index

- (  
(Informative) Conceptual Model [STUB], 13  
/  
/instanceOf, **112**  
/instances, **113**  
/realizedBy, **57**  
/realizes, **82**  
/testCase, **54**  
/utilizedBy, **82**  
[  
[BMM], **9**, 56  
[DD], **9**  
[ES20187301], **9**, 19  
[ES202951], **9**, 21  
[ES20311901], **9**, 19  
[ES20311902], **9**, 19  
[ES20311903], **9**, 19  
[ES20311904], **9**, 19  
[FUML], **10**  
[HWT2012], **10**, 21  
[IEC61508], **10**, 18  
[ISO1087-1], **10**, 24  
[ISO25010], **10**, 141  
[ISO29119], **10**, 18, 19, 20, 21, 52, 61, 62, 63, 64, 65,  
66, 68, 73  
[ISO9126], **10**  
[ISTQB], **10**, 19, 21, 44, 52, 64, 65, 66, 67, 68, 72,  
73, 144, 145, 172  
[MDA], **10**  
[MDAa], **10**  
[MDAb], **10**  
[MDAd], **10**  
[MOF], **9**  
[OCL], **9**  
[OSLC], **10**  
[SBVR], **10**, 24  
[SEP2014a], **10**, 39  
[SysML], **10**, 25, 52, 57, 59  
[TCM2008], **11**, 21  
[TestIF], **11**  
[UL2007], **11**, 21  
[UML], 3, 5, 6, **9**, 40, 41  
[UPL2012], **11**, 21  
[UTP], **11**  
[WikiCT], **11**, 39  
[WikiM], 6, **11**, 42  
[XMI], **9**  
{  
{read-only, union, subsets subTestDirective}  
subDirective, **70**  
{read-only, union, subsets subTestTechnique}  
subTechnique, **71**  
{read-only, union, subsets technique} capability, **70**  
{subsets capability} appliedTestDesignTechnique, **67**  
{subsets subDirective} genericSubDirective, **67**  
**A**  
a, 31, 33, 40, 90  
abstract test case, **5**, 21, 31, 32, 50  
abstract test configuration, **5**, 29, 30  
acceptance test level, **144**  
Acceptance testing, **145**  
Action, 84, 89  
actual data pool, **5**, 39, **40**, 41  
actual parameter, **5**, 32, **33**, 49, 120, 132  
actualParameter {ordered, unique}, **132**  
ActualParameterValue, **49**, **132**, 195  
ActualResponseLogEntry, **49**, 134, 195  
actualValue, **135**  
Additional Information, 13  
against, 37, 97  
AllCombinations, **147**  
Allowed invocation scheme, **79**, **82**, **84**  
AllRepresentatives, **147**  
AllStates, **147**  
AllTransitions, **147**  
Alpha Testing, **144**  
alternative, **5**, **33**, 34, 49, 81  
Alternative, **49**, 85, **86**, 87, 195  
AlternativeArbitrationSpecification, **49**, **117**, 118,  
195  
AnyType, **140**  
AnyValue, **49**, 107, **108**, 195  
API Testing, **145**  
Application in Activities, **86**, **88**, **89**, **92**  
Application in Interactions, **86**, **88**, **89**, **92**  
Arbitration & Verdict Overview, **42**  
Arbitration of AtomicProceduralElements, **115**, 116  
Arbitration of CompoundProceduralElements, **116**,  
117  
Arbitration of Test-specific Actions, **121**, 122  
arbitration specification, **5**, 6, 7, 19, 20, 24, 25, 34,  
35, 38, 42, 43, 49, 50, 51, 78, 83, 84, 89, 90, 96,  
97, 98, 100, 102, 107, 110, 111, 113, 115, 116,  
118, 120, 121, 130, 141, 183, 187, 188  
Arbitration Specifications, **42**, **110**  
Arbitration Specifications Overview, **111**  
ArbitrationResult, **49**, 111, **112**, 113, 195  
arbitrationSpecification, **55**, **90**  
ArbitrationSpecification, **49**, **113**, 114, 115, 120, 195  
arbitrationSpecification {redefines  
arbitrationSpecification}, **86**, **87**, **88**, **89**, **91**, **92**,  
**96**, **98**, **101**, **102**  
artifact, 1, 2, **5**, 7, 8, 18, 21, 27, 28, **30**, 44, 51, 52, 69,  
71, 75, 77, 158, 161  
at least one, 29, 31, 33, 37, 40, 80, 97, 99, 102, 106

At least one property, **97**  
 At least one response, **102**  
 At least one stimulus, **99**  
 at most one, 25, 31, 43, 80, 82, 84  
 ATM Example, **179**  
 atomic procedural element, **5**, 7, 32, **34**, 35, 36, 38, 49, 84, 85, 87, 115, 118  
 AtomicProceduralElement, **49**, 84, **87**, 90, 91, 96, 97, 98, 101, 102, 195  
 AtomicProceduralElementArbitrationSpecification, **49**, **118**, 120, 121, 122, 123, 124, 195  
 AtomicProceduralElementLogEntry, **49**, 130, **131**, 132, 134, 135, 195  
 Automated Test, **144**  
**B**  
 Behavior, 50, 51, 52, 58, 78, 79, 81, 82, 83, 84, 91, 92  
 BehavioedClassifier, 49, 50, 51, 81, 82, 113, 114, 115, 117, 118, 119, 120, 121, 122, 123, 124, 136  
 Beta Testing, **145**  
 boolean expression, 5, 6, 31, **32**, 33, 41, 86, 105  
 BoundaryValueAnalysis, **49**, **64**, 66, 148, 172, 195  
 Build verification test, **143**  
**C**  
 CallBehaviorAction, 50, 91  
 captures, 46  
 captures execution of, 46  
 CauseEffectAnalysis, **49**, **64**, 71, 195  
 Certifier, **16**  
 check property action, **5**, 37, 38, 49, 96, 97, 120, 123  
 check traceability, 16, **17**  
 checkedProperty, **97**, 171  
 ChecklistBasedTesting, **49**, **64**, 66, 195  
 CheckPropertyAction, **49**, 87, 93, **96**, 171, 172, 175, 195  
 CheckPropertyArbitrationSpecification, **49**, 96, 118, **123**, 195  
 CheckPropertyLogEntry, **49**, 131, **134**, 195  
 checks, 37, 97  
 Chocolate Portion, 152  
 Chocolate test, **152**  
 ChoiceOfValues, 195, **199**, **200**  
 Class, 51, 52, 56, 57  
 ClassificationTreeMethod, **49**, **65**, 71, 195  
 Classifier, 49, 50, 51, 70, 72, 75, 76, 77, 104, 105, 106, 128, 130, 131, 133, 136, 138, 139  
 Clients of a «Morphing» Dependency, **106**  
 CollaborationUse not allowed, **130**  
 CollectionExpression, 195, **199**, **201**  
 CollectionMatchingKind, **202**  
 CombinatorialTesting, **49**, **65**, 68, 71, 195  
 CombinedFragment, 49, 50, 86, 87, 88, 89, 92  
 complement, **5**, 39, **40**, 42, 45, 49, 104, 113  
 ComplementedValue, 195, **199**, **201**  
 Complements, **49**, **104**, 106, 195  
 component test level, **144**

compound procedural element, 5, 6, 7, 32, 33, 34, 35, 49, 84, 85, 86, 87, 88, 89, 92, 111, 116  
 Compound Procedural Elements Overview, **85**, 86  
 CompoundProceduralElement, **49**, 84, 86, **87**, 88, 89, 90, 92, 195  
 CompoundProceduralElementArbitrationSpecification, **49**, 117, **118**, 119, 120, 121, 195  
 Conceptual Model, 5, 53, 104  
 concrete test case, **5**, 21, 31, **32**, 50  
 concrete test configuration, **5**, 29, **30**, 153  
 configuration {subsets roleConfiguration}, **75**, **77**, 170  
 Conformance, 13  
 constraint, 3, **5**, 39, 40, 41, 52, 59, 103, 105  
 Constraint, 49, 50, 51, 74, 75, 77, 96, 97, 105, 106  
 create log entry action, **5**, **37**, 38, 49, 97, 124  
 create stimulus action, **5**, 37, **38**, 49, 93, 98, 99, 122, 200  
 CreateLogEntryAction, **49**, 87, 93, **97**, 195  
 CreateLogEntryArbitrationSpecification, **49**, 97, 118, **124**, 195  
 CreateLogEntryLogEntry, **49**, 131, **135**, 195  
 CreateStimulusAction, **49**, 87, 93, **98**, 122, 175, 195  
 CreateStimulusArbitrationSpecification, **49**, 98, 118, **122**, 195  
 CreateStimulusLogEntry, **49**, 134, 195  
 Croissant, 150, 152, 154, 155  
 Croissants, 153  
 Croissants Example, **150**  
 CR-X1072-B, **152**  
 Cyclic modifications, **109**  
**D**  
 data, 2, 4, 5, 6, 7, 8, 18, 21, 24, 27, 28, 30, 31, 37, 38, 39, 40, **41**, 42, 61, 69, 71, 83, 97, 98, 103, 104, 105, 106, 107, 109, 110, 120, 155, 157, 163, 199, 200  
 data item, 5, 6, 7, 39, 40, 41, 42, 104, 105, 106, 107  
 data partition, **5**, 39, **41**, 105, 192  
 data pool, 5, 24, 39, 40, 41, 49, 105, 192  
 data provider, **5**, 30, 39, 40, **41**, 49  
 data specification, 5, 21, 39, 40, 41, 42, 49, 50, 103, 104, 105, 106, 107, 155, 159, 160, 199  
 Data Specifications, 103  
 Data Specifications Overview, **104**  
 data structure, 41  
 data type, 5, **6**, 39, 40, 41, 105, 106, 107  
 Data Value Extensions, **107**, 108  
 Data Values, 103, **107**  
 DataPartition, **49**, **104**, 105, 195  
 DataPool, **49**, **105**, 195  
 dataProvider, **70**  
 DataProvider, **49**, 75, **105**, 195  
 dataSpecification, **104**  
 DataSpecification, **49**, 105, 106, 171, 195  
 dataSpecifications, **105**  
 DataType in DataSpecification, **106**

DecisionTableTesting, **49, 65, 71, 195**

DefaultCBT, **148**

DefaultCET, **148**

DefaultCTM, **148**

DefaultDTT, **148**

DefaultEG, **148**

DefaultET, **148**

DefaultPT, **148**

DefaultTPT, **148**

Dependency, **49, 50, 51, 59, 104, 106, 107, 109, 136**

Derivation and Modeling of Test Requirements, **157**

description, **57, 59, 82**

Description of Case Study, **174**

design acceptance tests, **16, 17**

design integration tests, **16, 17**

Design of Test Case Procedures, **163**

design system tests, **16, 17**

design test cases, **16, 17**

design test cases for a data-intensive system, **16, 17**

design test cases for a system that includes humans,  
**16, 17**

design test cases for a system with time-critical  
behavior, **16, 17**

design test data, **16, 17**

design test specifications, **16, 17**

design unit tests, **16, 17**

determine test coverage, **16, 17**

determines, **42, 43, 113**

directionKind, **131**

DRAS01, **42, 113**

DRAS02, **43**

DRTA01, **25, 37, 99**

DRTA02, **25, 37, 102**

DRTA03, **25, 37, 43, 97**

DRTC01, **31**

DRTC02, **31, 84**

DRTC03, **31, 82**

DRTC04, **31, 80**

DRTC05, **31, 84**

DRTC06, **31, 82**

DRTC07, **31, 80**

DRTC08, **31**

DRTC09, **43**

DRTD01, **40, 106**

DRTD02, **40**

DRTD03, **40, 106**

DRTD04, **40, 106**

DRTD05, **40**

DRTL01, **46**

DRTL02, **46**

DRTP01, **33, 90**

DRTP02, **33, 80**

DRTP03, **33, 80**

DRTP04, **33, 80**

DRTR01, **29**

DRTR02, **29**

duration, **6, 8, 32, 34, 35, 44, 46, 130**

## **E**

each, **25, 29, 31, 33, 40, 43, 46, 80, 82, 84, 106**

Each test case returns a verdict statement, **82**

emanates from, **40**

endAfterPrevious, **90**

Enforced expectation kind 'implicitExcept', **102**

EquivalenceClassPartitioning, **49, 64, 66, 71, 147, 196**

error, **93, 122, 123, 140**

Error, **6, 43, 44, 66, 113**

ErrorGuessing, **49, 66, 196**

evaluate test results, **16, 17**

exactly one, **40, 42, 43, 46, 113**

execute test cases, **16, 18**

Executed test cases and definition of test set members  
must be consistent, **129**

executedTestCase, **129**

executedTestSet, **129**

executedTestSetMember, **129**

executing entity, **6, 44, 46, 58, 128, 129, 130, 136**

executingEntity, **128**

executionDuration, **128**

executionStart, **128**

expect response action, **6, 37, 38, 49, 100, 102, 107, 120, 123, 163, 164**

expectationKind, **101, 102**

expectedElement, **101, 102**

ExpectResponseAction, **49, 87, 93, 100, 101, 102, 103, 123, 175, 196**

ExpectResponseArbitrationSpecification, **49, 100, 118, 123, 196**

expects to receive, **37, 102**

ExperienceBasedTechnique, **50, 64, 66, 71, 196**

ExploratoryTesting, **50, 66, 196**

Expression, **49, 50, 108, 110, 199, 200, 201, 202**

Extends, **50, 106, 196**

extension, **6, 39, 42, 50, 52, 58, 60, 64, 78, 87, 90, 105, 106, 107, 199**

## **F**

fail, **123, 140, 141**

Fail, **6, 42, 43, 44, 113, 163, 164, 183**

Failed user login, **156**

Failover Test, **145**

Feature acceptance testing, **143**

Feature validation testing, **143**

Feature verification testing, **143**

forbiddenElement, **98, 101, 102, 122, 123**

formal parameter, **5, 6, 31, 32, 33, 34, 50, 132**

formalParameter, **132**

FormalParameterReference, **50, 131, 196**

formalParameterReference {ordered, unique}, **131**

Functionality to Test, **174**

## **G**

General, **179**

generate test case instances, **16, 18**

Generation of Test Sets and Abstract Test Cases, **176**  
 Generic Test Design Capabilities, **60**, **61**  
 GenericTestDesignDirective, **50**, **60**, **67**, **69**, **196**  
 GenericTestDesignTechnique, **50**, **60**, **67**, **71**, **196**  
 Given Requirements on the Test Item, **169**  
 GraphTraversalAlgorithmKind, **149**  
 GraphTraversalStructure, **148**  
 guarantees, **31**, **80**, **82**, **84**  
 Gustaoceptionary Proficiency, **152**  
**H**  
 Human Test Executor, **16**  
**I**  
 ID, **54**, **56**, **57**, **58**, **76**, **82**, **84**, **113**  
 ignoredElement, **101**, **102**, **123**  
 implement automatic test case execution, **16**, **17**, **18**  
 implement onboard test cases, **16**, **17**, **18**  
 implement test components, **16**, **17**, **18**  
 implement tool support for UTP 2, **17**, **18**  
 implicitExpect, **102**, **103**  
 ImplicitExpectationKind, **103**  
 implicitForbid, **103**  
 implicitIgnore, **103**  
 inconclusive, **140**, **141**  
 Inconclusive, **6**, **42**, **43**, **44**, **113**  
 Informative References, **9**  
 input {ordered}, **138**  
 instance, **5**, **6**, **8**, **41**, **46**, **128**  
 instanceOf, **128**, **137**, **138**  
 instanceOf {redefines instanceOf}, **70**, **72**  
 InstanceSpecification, **49**, **50**, **51**, **64**, **65**, **66**, **67**, **68**,  
     **69**, **71**, **72**, **73**, **109**, **112**, **127**, **128**, **129**, **130**, **131**,  
     **132**, **133**, **134**, **135**, **137**, **138**  
 Intake Test, **144**  
 integration test level, **144**  
 Integration Testing Example, **184**  
 Interaction, **78**  
 InteractionFragment, **78**, **84**, **89**  
 InteractionUse, **50**, **91**  
 Interface testing, **145**  
 Internal structure of TestLogStructure Classifier, **130**  
 Invocation Test Log Entry Details, **127**  
 InvocationAction, **49**, **50**, **97**, **98**, **102**  
 InvocationLogEntry, **50**, **131**, **132**, **133**, **134**, **196**  
 InvocationLogEntryStructure, **50**, **130**, **131**, **133**, **196**  
 invocationStructure {redefines instanceOf}, **133**, **134**  
 invocationTarget, **133**, **134**  
 invokedProcedure, **92**  
 invokes, **31**, **33**, **80**  
 is smaller than, **33**, **90**  
 ISTQB Agile Test Set Purpose, **143**  
 ISTQB Library, **142**  
 ISTQB Test Level, **144**  
 ISTQB Test Set Purpose, **144**  
 It is impossible that, **31**  
 It is necessary that, **25**, **29**, **31**, **33**, **37**, **40**, **42**, **43**, **46**,  
     **80**, **82**, **84**, **90**, **97**, **99**, **102**, **106**, **113**

**K**  
 Knowledge of CR-X1072-B, **152**  
**L**  
 Language Architecture, **48**  
 leads to, **36**  
 Load Testing, **145**  
 loggedValue, **135**  
 Login response time, **156**, **158**  
 LoginServer Example, **155**  
 longest, **149**  
 loop, **6**, **32**, **34**, **50**, **78**, **88**, **118**  
 Loop, **50**, **78**, **85**, **87**, **88**, **196**  
 LoopArbitrationSpecification, **50**, **118**, **196**  
**M**  
 Machine Test Executor, **16**  
 Mail address modification, **156**, **158**  
 main, **92**  
 main procedure invocation, **6**, **7**, **33**, **34**, **35**, **80**  
 Manual Test, **144**  
 Mapping Interface Descriptions, **166**  
 Mapping Test Cases and Test Configuration, **167**  
 Mapping the Test Architecture, **166**  
 Mapping the Test Data Specification, **166**  
 Mapping the Test Type System, **165**  
 Mapping to Code, **169**, **173**  
 Mapping to TTCN-3, **165**  
 MatchingCollectionExpression, **196**, **199**, **201**  
 matchingKind, **201**  
 max, **202**  
 maxInclusive, **202**  
 meet, **69**  
 Message, **49**, **98**, **101**  
 MessageEventLogEntry, **50**, **132**, **134**, **196**  
 MessageEventLogEntryStructure, **50**, **131**, **133**, **196**  
 min, **202**  
 Minimal test configuration, **76**  
 minInclusive, **202**  
 model, **28**  
 Model Libraries, **13**  
 Modeling Test Data, **160**  
 Modeling the Behavior of the System, **169**, **170**  
 Modeling the Structure of the System, **169**  
 Modeling the Type System and Logical Interfaces,  
     **159**  
 Morphing, **50**, **104**, **106**, **107**, **196**  
 morphism, **5**, **6**, **7**, **39**, **40**, **42**, **50**, **104**, **106**, **107**  
 Must be used as payload for an expected responses,  
     **201**  
**N**  
 NamedElement, **50**, **51**, **71**, **89**  
 nBoundaryRepresentatives, **64**  
 nCombination, **65**  
 nCombination {redefines nCombination}, **68**  
 negative, **6**, **34**, **50**  
 Negative, **50**, **85**, **87**, **88**, **196**  
 Negative Test, **144**

NegativeArbitrationSpecification, **50**, 118, **119**, 196  
 Nested Classifier not allowed, **82**  
 none, 140, **141**  
 None, **6**, 42, **43**, 44, 113  
 Non-normative data value extensions, **199**  
 Normative References, **9**  
 nRepresentatives, **66**  
 nRepresentatives {redefines nRepresentatives}, **64**  
 NSwitchCoverage, **50**, **67**, 68, 72, 196  
**O**  
 Object Management Group, Inc. (OMG), vii  
 ObjectFlow, 49, 96  
 Objects, 153  
 observedProperty, **135**  
 of, 33, 43, 90  
 OMG specifications, vii  
 One postcondition per test case, **82**  
 One postcondition per test execution schedule, **84**  
 One postcondition per test procedure, **80**  
 One precondition per test case, **82**  
 One precondition per test execution schedule, **84**  
 One precondition per test procedure, **80**  
 OneBoundaryValue, **148**  
 OneRepresentative, **148**  
 Only applicable to UML Behavior building blocks, **89**  
 OpaqueProceduralElement, **50**, 84, **89**, 90, 196  
 OpaqueProceduralElementLogEntry, **50**, 131, **135**, 196  
 Operands shall be empty, **202**  
 Operation, 52  
 or, 43  
 output {ordered}, **138**  
 overrides, **50**, **109**, 196  
 Overview of non-normative ValueSpecification Extensions, **199**, 200  
 Overview of test-specific actions, **36**, 37  
 Overview of the ISTQB library, **142**, 143  
 Overview of the predefined test design technique structures, **148**  
 Owned UseCases not allowed, **82**  
 Owner of Constraint, **97**  
 Owner of Property, **97**  
 Ownership of «TestComponentConfiguration», **75**  
 Ownership of «TestItemConfiguration», **77**  
**P**  
 Package, 51, 54, 58, 106  
 PairwiseTesting, **50**, 65, **68**, 196  
 parallel, **6**, 34, 50, 119  
 Parallel, **50**, 85, 87, **89**, 196  
 ParallelArbitrationSpecification, **50**, 118, **119**, 196  
 parent, **112**  
 part, **76**  
 pass, 93, 123, 140, **141**  
 Pass, **6**, 42, **43**, 44, 113, 159, 164, 183  
 PE end duration, **6**, 33, **35**, 90

PE start duration, **6**, 33, **35**, 90  
 permits to send, 37, 99  
 permittedElement, **98**, 99, 122  
 permutation, **203**  
 postcondition, **6**, 31, **32**, 80, 81, 82, 84  
 precondition, **6**, 31, **32**, 80, 82, 84  
 Predefined context-free test design techniques, **146**, 147  
 Predefined data-related Test Design Techniques, **61**, 62  
 Predefined experience-based Test Design Techniques, **63**  
 Predefined high-level Test Design Techniques, **61**  
 Predefined state-transition-based Test Design Techniques, **62**  
 Predefined Test Design Technique Structures, **148**  
 Predefined Test Design Techniques, **146**  
 Predefined types, **140**  
 Predefined verdict instances, **140**  
 prescribes the execution order of, 32, 33, 80  
 procedural element, 5, 6, 7, 19, 32, 33, 34, 35, 50, 78, 79, 80, 81, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 110, 111, 113, 115, 117, 118, 119, 120, 121, 128, 141  
 Procedural Element Arbitration Specifications, **115**  
 procedural element verdict, 5, 6, 7, 36, 37, 38, 43, 44, 84, 87, 93, 96, 100, 102, 110, 111, 113, 114  
 Procedural Elements, 78, **84**  
 Procedural Elements Overview, **85**  
 proceduralElement, **130**  
 ProceduralElement, 14, **50**, 84, 87, 89, **90**, 196  
 proceduralElement {redefines proceduralElement}, **131**, **133**, **134**, **135**  
 ProceduralElementArbitrationSpecification, **50**, 113, 118, **120**, 196  
 procedure, 5, 6, 7, 8, 32, 33, 34, **35**, 36, 42, 66, 71, 78, 79, 80, 81, 82, 83, 84, 91, 92, 110, 132, 163, 165  
 procedure invocation, 6, **7**, 32, 34, **35**, 50, 85, 91, 120, 121  
 Procedure sequentializes procedural element, **80**  
 ProcedureInvocation, **50**, 79, 81, 83, 85, 87, **91**, 92, 175, 196  
 ProcedureInvocationArbitrationSpecification, **50**, 118, **121**, 196  
 ProcedureInvocationLogEntry, **50**, 132, **133**, 196  
 ProcedureInvocationLogEntryStructure, **50**, 131, **133**, 196  
 ProcedurePhaseKind, **92**  
 Product Manager, **16**  
 Profile Specification [STUB], 13  
 Project Manager, **16**  
 property, 5, **7**, 37, **38**, 59, 64, 66, 68, 72, 96, 97, 120  
 Property, 49, 50, 51, 75, 76, 77, 97, 105, 131, 201  
 provide test data, 16, **18**  
 provides data according to, 40

purpose, **58**

## **Q**

QA Manager, **16**

## **R**

random, **149**

RangeValue, 196, **199**, **202**

Recoverability Test, **145**

referencedBy, **56**, **57**, **59**, **70**, **72**, **113**, **128**

references, **57**

References, 13, **173**, **178**, **194**

refers to, 25, 43

refinement, **7**, 39, 42, 50, 107

Refines, **50**, 105, 106, **107**, 196

Regression Testing, **144**

RegularExpression, **50**, 107, **110**, 196

Relation to keyword-driven testing, 19

requirement, 27

Requirements Engineer, **16**

Requirements Specification, **155**

requires, 31, 80, 82, 84

response, 2, 6, **7**, 19, 36, 37, 38, 93, 100, 102, 103, 107, 108, 110, 199, 200, 201, 202

Restriction of client and supplier, **109**

Restriction of extendable metaclass, **59**

Restriction of extendable metaclasses, **55**, **56**, **57**, **128**, **130**

resultFor, **112**

review test specifications, 16, **18**

role, 91, 92

role {ready-only, union}, **74**

Role only in context of test cases relevant, **92**

RoleConfiguration, **50**, 73, **74**, 75, 77, 196

roleConfiguration {read-only, union}, **77**

RQ-0001, **150**

RQ-0002, **150**

RQ-0003, **150**

## **S**

select test data, 16, 17, **18**

Semantics of Business Rules and Vocabularies, 24

sequence, **7**, 19, 32, 34, **35**, 50, 67, 72, 85, 119, 121, 128

Sequence, **50**, 87, **92**, 175, 196

SequenceArbitrationSpecification, **50**, 118, **121**, 196

setup, **92**

setup procedure invocation, **7**, 35

shortest, **149**

SimpleChecklistBasedStructure, **149**

SimpleErrorGuessingStructure, **149**

Slot, 49, 132

Smoke Test, **144**

Specialization of TestLogStructure Classifier, **130**  
specification, **56**, **57**

Specification of Complex Test Data, **162**

Specification of Dependency client, **136**

Specification of Dependency supplier, **136**

specifies, 25, 40, 106

specifies the configuration of, 29

startAfterPrevious, **90**

State, 97

StateCoverage, **50**, **68**, 196

StateInvariant, 97

StateTransitionTechnique, **50**, 67, 68, 71, 72, 172, 196

stimulus, 5, 7, 37, 38, 93, 98, 99, 107, 201

Stress Testing, **145**

StructuredActivityNode, 49, 50, 86, 87, 88, 89, 92

StructuredClassifier, 51, 76

subresult, **112**

Subsea Production System Example, **174**

subset, **202**

subTestDirective {read-only, union}, **138**

subTestTechnique {union, read-only}, **138**

suggest verdict action, **7**, **38**, 50, 102, 123

suggestedVerdict, **135**

SuggestVerdictAction, **50**, 87, 93, **102**, 196

SuggestVerdictArbitrationSpecification, **50**, 102, 118, **123**, 197

SuggestVerdictLogEntry, **50**, 131, **135**, 197

superset, **203**

Suppliers of a «Morphing» Dependency, **106**

switchStates, **67**

switchStates {redefined switchStates}, **72**

System Designer, **16**

System Operator, **16**

System Test Example, **189**

system test level, **144**

## **T**

targets, 40

TC01

test taste, **154**

TC02

test structure, **154**

TC03

test color, **155**

TDS01, **152**

teardown, **92**

teardown procedure invocation, **7**, 35

technique {read-only, union}, **137**

Terms and Definitions, 13

test action, 5, 6, 7, 8, 19, 31, 32, 33, 34, 36, 37, 38, 43, 44, 45, 79, 80, 93, 94, 95, 96, 97, 98, 100, 102, 111, 121

Test Analysis, 4, **24**, **51**, **157**

Test Architecture, **29**, **73**

Test Architecture and Test Configuration, **161**

Test Architecture Overview, **29**, **73**, 74

Test Behavior, **31**, **78**

test case, 2, 3, 5, 6, 7, 8, 16, 17, 18, 19, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 42, 43, 44, 45, 46, 50, 52, 53, 54, 56, 57, 58, 59, 64, 65, 66, 67, 68, 69, 71, 72, 73, 75, 78, 79, 80, 81, 82, 83, 90, 91, 93, 97, 102, 103, 110, 111,

113, 114, 115, 120, 128, 129, 130, 136, 141, 153, 155, 157, 161, 162, 163, 164, 165, 166, 167, 169, 173, 176, 177, 180, 181, 182, 183, 186, 187, 188, 190, 191, 193, 200, 201

Test case invokes one main procedure, **80**

test case log, **5, 7, 37, 46, 51, 97, 113, 129**

Test Case Overview, **31, 78, 79**

test case verdict, **7, 36, 38, 43, 44, 46, 102, 110, 111, 113, 114, 187, 188**

Test Cases, **31, 153, 172**

test component, **1, 5, 7, 19, 22, 29, 30, 39, 41, 51, 73, 74, 75, 79, 81, 93, 94, 96, 97, 98, 100, 102, 105, 155, 161, 162, 181, 182, 184, 186, 187, 188, 191**

test component configuration, **7, 29, 30, 51, 73, 75, 169, 173**

test configuration, **5, 7, 8, 18, 19, 27, 28, 29, 30, 50, 51, 69, 71, 73, 74, 75, 76, 77, 78, 79, 81, 97, 102, 155, 157, 162, 163, 164, 167, 181, 186, 191**

Test Configuration, **153**

test context, **7, 8, 14, 18, 20, 21, 24, 25, 26, 27, 51, 53, 54, 56, 142, 155, 156, 157, 160, 162, 182, 190**

Test Context Overview, **24, 25, 31, 52**

Test Data, **39, 103**

Test Data Concepts, **39, 40**

Test Design, **4, 27, 59, 152, 161**

test design directive, **7, 8, 27, 28, 50, 51, 59, 60, 67, 69, 71, 146**

Test Design Directive, **51**

Test Design Facility, **60**

Test Design Facility Library, **146**

Test Design Facility Overview, **27, 28**

test design input, **7, 8, 19, 20, 24, 26, 27, 28, 51, 52, 54, 57, 59, 68, 69, 71, 146, 176, 177**

Test Design Inputs, **175**

test design technique, **7, 8, 19, 20, 21, 24, 26, 27, 28, 49, 50, 51, 54, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 146, 148, 149, 159, 172**

Test Designer, **16**

Test Directive Facility, **137**

Test Directives, **136**

Test Evaluation, **42, 110**

test execution schedule, **7, 8, 19, 27, 28, 31, 32, 35, 42, 51, 58, 69, 78, 83, 84, 91, 111, 130**

test item, **1, 2, 5, 6, 7, 8, 19, 21, 22, 26, 27, 29, 30, 32, 36, 37, 38, 39, 43, 44, 51, 52, 57, 66, 73, 77, 79, 81, 93, 95, 96, 97, 98, 100, 101, 103, 141, 150, 155, 156, 159, 161, 162, 169, 173, 180, 181, 183, 184, 185, 186, 190, 202**

test item configuration, **8, 29, 30, 51, 73, 77, 169, 173**

Test Item Controlled Actions, **95**

test level, **1, 8, 21, 24, 25, 26, 54, 142, 155, 157, 180, 184**

test log, **1, 7, 8, 19, 21, 44, 45, 46, 47, 51, 97, 110, 128, 129, 130**

Test Log Entries Details, **126**

Test Log Entries Overview, **125, 126**

Test Log Overview, **44, 46**

test log structure, **8, 44, 45, 46, 51, 128, 129, 130, 136**

Test Logging, **44, 124**

Test Logging Overview, **124, 125**

Test Map, **154**

test objective, **7, 8, 18, 20, 25, 26, 28, 32, 44, 51, 52, 53, 54, 56, 59, 69, 81, 91, 110, 128, 156, 157**

Test Objective Overview, **53, 54**

Test Objectives, **151**

Test Planning, **24, 51, 156**

test procedure, **8, 18, 31, 32, 33, 35, 36, 42, 51, 57, 76, 78, 79, 80, 81, 83, 84, 91, 92, 153, 163, 164, 165**

Test Procedure Arbitration Specifications, **111**

Test procedure operates on test configuration, **79**

Test procedure sequentializes test action, **80**

Test Procedures, **31, 32, 33**

test requirement, **8, 25, 26, 44, 51, 52, 57, 59, 91, 110, 128, 155, 157, 158, 159, 160, 163, 164, 181, 182, 183, 184, 190, 191, 193**

Test Requirement and Test Objective Overview, **25, 26**

Test Requirements, **151**

Test Requirements Realization, **163**

test set, **6, 7, 8, 17, 18, 20, 24, 25, 26, 27, 28, 31, 43, 44, 45, 46, 47, 51, 52, 56, 57, 58, 59, 69, 78, 83, 110, 111, 113, 115, 128, 129, 136, 141, 142, 144, 153**

Test Set "Manual croissants test", **153**

test set log, **8, 46, 47, 51, 129**

test set purpose, **8, 27, 142**

test set verdict, **8, 43, 44, 83, 93, 110, 111, 113**

Test Strategy, **152**

test type, **8, 20, 21, 24, 25, 27, 54, 155, 157, 159, 160, 162**

TestCase, **50, 53, 57, 76, 78, 81, 82, 83, 129, 136, 162, 197**

TestCaseArbitrationSpecification, **51, 113, 114, 123, 197**

testCaseAS, **82**

TestCaseLog, **51, 128, 129, 197**

TestComponent, **14, 51, 73, 75, 76, 77, 105, 162, 169, 197**

testComponent {subsets role}, **75**

TestComponentConfiguration, **51, 73, 74, 75, 170, 197**

testConfiguration, **55**

TestConfiguration, **51, 73, 76, 81, 175, 197**

TestConfigurationRole, **51, 73, 74, 75, 76, 77, 197**

TestContext, **14, 51, 52, 53, 54, 55, 197**

testDesignDirective, **55**

TestDesignDirective, **51, 59, 60, 67, 69, 70, 137, 147, 148, 197**

TestDesignDirectiveStructure, **51, 60, 70, 138, 197**

testDesigningEntity, **70**

testDesignInput, **55**  
 TestDesignInput, **51**, 69, **71**, 169, 170, 197  
 testDesignInput {redefines input}, **70**  
 testDesignOutput {redefines output}, **70**  
 testDesignTechnique, **55**  
 TestDesignTechnique, **51**, 59, 60, 64, 65, 66, 67, 68, **71**, 73, 138, 147, 148, 197  
 TestDesignTechniqueStructure, **51**, 60, **72**, 139, 197  
 TestDirective, **51**, 69, **137**, 197  
 TestDirectiveStructure, **51**, 70, **138**, 197  
 Tester Controlled Actions, **94**  
 TestExecutionSchedule, **51**, 78, 82, **83**, 84, 111, 115, 197  
 TestItem, 14, **51**, 73, 76, 77, 97, 162, 169, 170, 175, 197  
 testItem {subsets role}, **77**  
 TestItemConfiguration, **51**, 73, 74, **77**, 169, 170, 173, 197  
 testLevel, **55**  
 testLog, **55**  
 TestLog, **51**, 127, **128**, 129, 197  
 TestLogElement, **51**, **127**, 128, 130, 197  
 testLogEntry, **90**  
 TestLogEntry, **51**, 127, **130**, 131, 197  
 testLogEntry {ordered, unique}, **129**  
 TestLogStructure, **51**, **130**, 131, 136, 197  
 TestLogStructureBinding, **51**, **136**, 197  
 testObjective, **55**  
 TestObjective, **51**, 52, **56**, 175, 197  
 TestProcedure, **51**, 76, 78, **79**, 81, 82, 83, 92, 197  
 testRequirement, **55**  
 TestRequirement, **51**, 52, **57**, 82, 174, 197  
 testSet, **55**  
 TestSet, **51**, 52, **58**, 59, 111, 115, 129, 197  
 TestSetArbitrationSpecification, **51**, 113, **115**, 197  
 testSetAS, **59**, **84**  
 TestSetLog, **51**, 128, **129**, 197  
 testSetMember, **58**  
 Test-specific Action Arbitration Specifications, **121**  
 Test-specific Actions, 31, **36**, 78, 85, **93**  
 Test-specific actions Overview, **93**, 94  
 Test-specific Contents of Test Context, **53**  
 Test-specific Procedures, **32**, 78  
 TestTechnique, **51**, 71, **138**, 197  
 TestTechniqueStructure, **51**, 72, **139**, 197  
 testType, **55**  
 the, 33, 37, 90, 97  
 the same, 33, 90  
 The Test Item, **150**  
 The TRUST Test Generator, 169, **172**  
 The UTP auxiliary library, **141**, 142  
 The UTP test design facility library, **146**  
 time point, 8, 32, **36**, 46  
 TO00  
 Quality verified, **151**  
 TO01  
 Taste verified, **151**, 154  
 TO02  
 Structure verified, **151**, 154, 155  
 TO03  
 Color verified, **151**, 155  
 toBeCovered, **68**, **72**  
 Tool Vendor, **17**  
 TR01  
 Humans, **151**  
 TR02  
 Waste, **152**  
 Transition, 84, 89  
 TransitionCoverage, **51**, 68, **72**, 197  
 TransitionPairCoverage, **51**, 67, **72**, 197  
 Trigger, 49, 101  
 Type of Argument, **103**  
 Type of elements for the explicit sets, **102**  
 Type of forbidden elements, **98**  
 Type of permitted elements, **99**  
 Type of verdict ValueSpecification, **113**  
 Typical Use Cases of UTP 2, 3  
**U**  
 UML Testing Profile, 15  
 Unit Test Example, **180**  
 Unknown user login, 156  
 update test specifications, 16, **18**  
 Use of «ProcedureInvocation», **80**  
 Use of BehavioredClassifier, **82**  
 Use of permutation matching kind, **201**  
 UseCaseTesting, **51**, 71, **73**, 198  
 User banishing, 156  
 User login, 156, 158  
 User logout, 156, 158  
 UTP 2 Use Cases, 15  
 UTP 2 WG, 5, 6, 7, 8, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43, 44, 46, 47, 152  
 UTP Auxiliary Library, 13, 48, **141**  
 UTP Types Library, 48, 113  
**V**  
 Valid duration, **90**  
 value, 5, 6, 41, **132**  
 valueFor, **132**  
 ValueSpecification, 113, 199, 201  
 verdict, 5, 6, 7, 8, 21, 39, 42, 43, 44, 78, 82, 83, 87, 90, 93, 100, 102, 103, 107, 110, 111, **112**, 113, 114, 115, 117, 118, 119, 120, 121, 122, 123, 124, **128**, 130, **140**, 141, 183  
 Verdict of ArbitrationSpecification, **113**  
 verifies, **51**, 52, **59**, 198  
 Videoconferencing Example, **169**