UML Testing Profile Tutorial

MBT User Conference, 18th of October 2011

Marc-Florian Wendland, Ina Schieferdecker,
Markus Schacher, Armin Metzger

What is the goal of this tutorial?

- Understand what UTP is and what it was made for
- Get an overview of the current status of UTP
- Become acquainted with its main concepts
- Insight into real-world UTP-based projects
- Demonstrate newly incorporated test management capabilities
- Quo vadis, UTP?

Clear-up myths and legends around UTP

Agenda

- Introduction
- UTP en detail
- UTP @ Work
  - Modelling with UTP
  - Model-based test management
  - UTP in a safety-critical domain
- Outlook

Agenda

- **Introduction**
- UTP en detail
- UTP @ Work
  - Modelling with UTP
  - Model-based test management
  - UTP in a safety-critical domain
- Outlook

What is UTP and if so, how many?

- UML natively lacks concepts for testing of systems/software
- Domain-independent specification of test concepts based on UML
- A language for creation, documentation, visualization, specification and exchange of model-based test specifications
- Defines two complementary specifications
  - A native UML profile (for combination with UML)
  - A standalone MOF-based metamodel (using UTP without UML)

History

- 2001: RFP for a test-related UML profile
- 2003: Initial submission elaborated by testers, UML and test solution vendors
  - Industrial members (a.o. Ericcson, Telelogic, IBM, Softeam)
  - Academic members (a.o. Fraunhofer FOKUS, University Luebeck)
- 2005: Final adopted version 1.0 released by the OMG
- June 2010: UTP 1.1 RTF was chartered
- June 2011: UTP 1.2 was chartered

Targets of the UML Testing Profile

- Provide a set of domain-/process-independent, test-related concept for the definition of test models
  - Reuse or combine the UTP with other domain-specific profiles of the OMG like MARTE, SysML, SoaML, …
- Reuse the benefits of model-driven development
  - Raised level of abstraction
  - Use Model-to-Model transformations
  - Use Model-to-Code transformations
- Bridging the gap!

UTP Perception until now

- UTP was/is not widespreadly used in industry
  - Lack of experiences with UML 2
  - Ins
  - Mo
  - La
- Critic
  - Insufficient tool support
  - Missing methodology, guide lines, experience reports …
  - Inadequate readability of the specification document
  - MOF-based metamodel and native UML profile was confusing

UTP was ahead of its time

What is UTP made for…

- Domain-independent test modelling
  - Test basis
  - Test specifiction
- Test case specification
  - Abstract/concrete vs. logical/technical
- Test data specification
- Test deployment
- Test result visualisation
- Combination with other profiles (SysML, MARTE, SoaML)
- …

…and what is out of scope?

- Test case generation
- White-Box approaches
- Audits and Reviews
- Test management (partially addressed)
- Test methodology

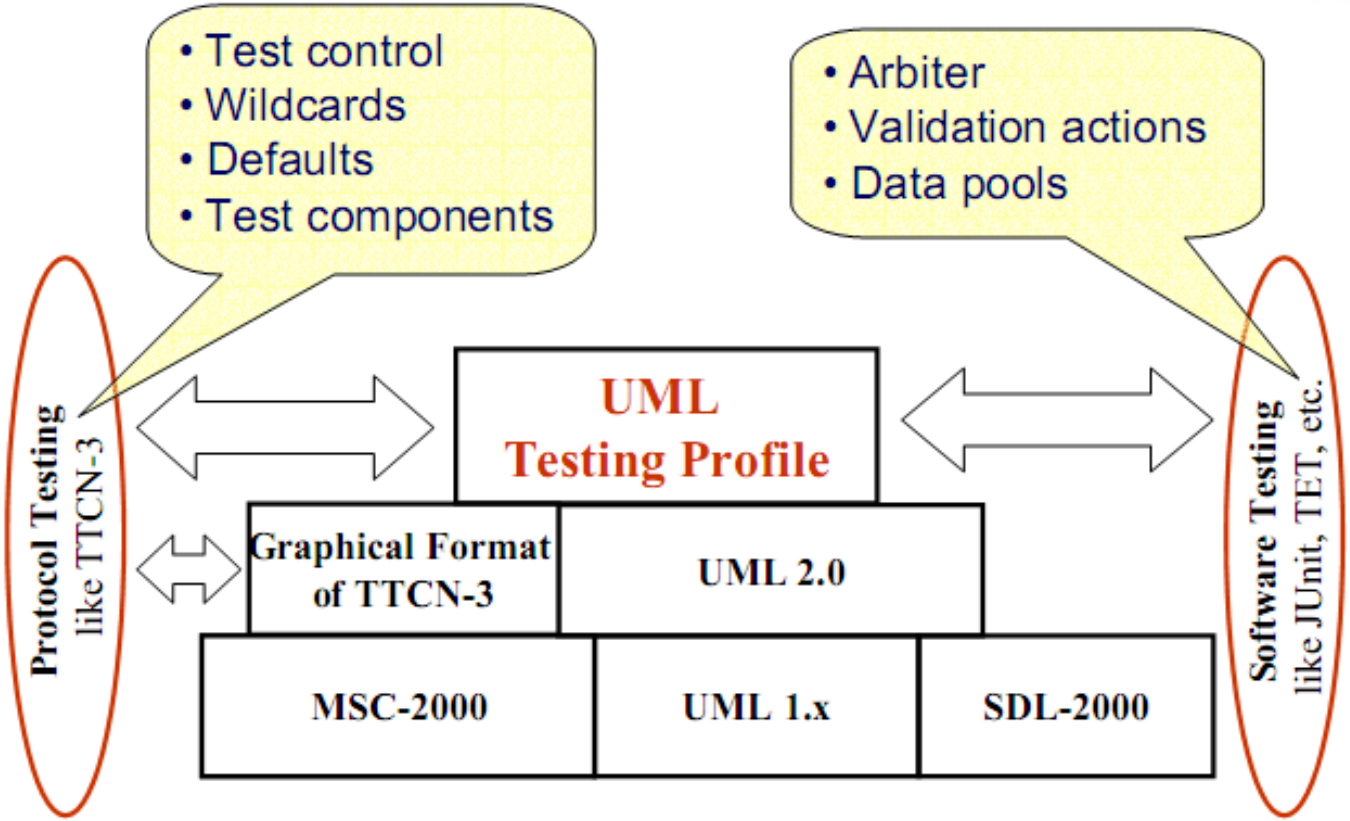
Agenda

- Introduction ✓
- **UTP en detail**
- UTP @ Work
  - Modelling with UTP
  - Model-based test management
  - UTP in a safety-critical domain
- Outlook

What is a UML Profile?

- UML is a GPL
  - Appropriate for object-oriented/component-based analysis/design
  - Some (important) concepts out of scope, e.g. requirememts, testing, service-oriented concepts, embedded and real-time systems
- UML can be leveraged for the design of a domain-specific language (DSL)
  - UML vs. DSL – A false dichotomy?
  - Different extension mechanism
- Profiling is a native UML mechanism
  - Indirect extension of UML Superstructure
  - Allows to add new features/constraints to existing metaclasses

Origin of UML Testing Profile

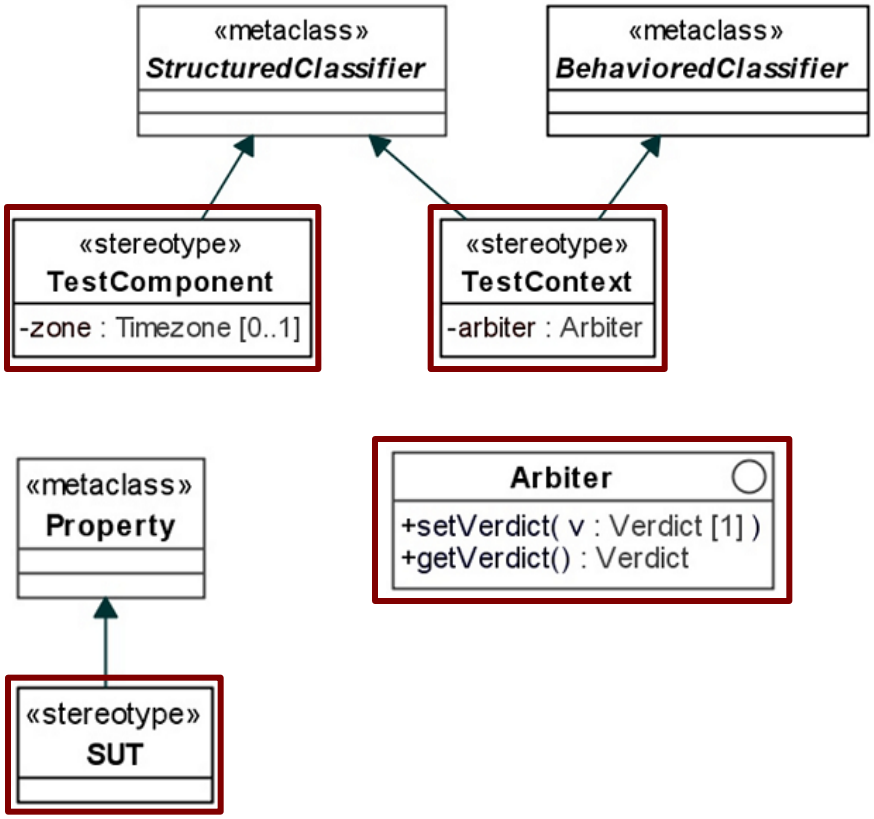


Parts of UML Testing Profile

- Test Architecture
  - Architectural foundation to create test architectures
- Test Behavior
  - Behavioral additions and test-specific actions for test case behavior
- Test Data
  - Concepts to define data partitions, data pools and wildcards
- Timer Concepts
  - Imperative timer mechanism
- Test Management
  - Optional concepts for model-based test management

UTP Test Architecture

- TestContext is the outmost concept in UTP for
  - test case grouping
  - test configuration definition
  - test control specification
- TestComponent is intended to stimulate the SUT and to evaluate its (expected) outcome
- SUT (system under test) represents the test object for a particular test context
- Arbiter is a predefined interface for verdict calculation and assignment

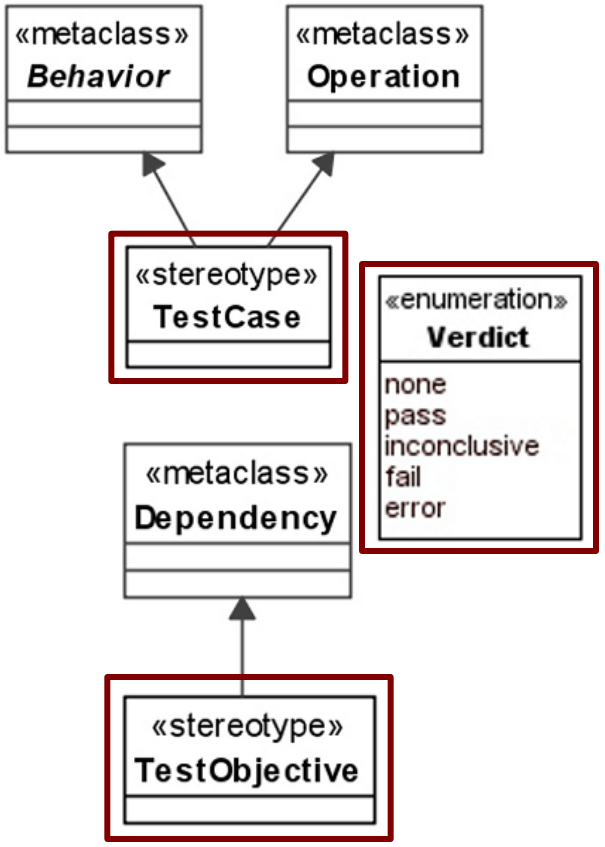


Test Behavior – Test Cases and Objectives

- TestCase marks operations or behaviors as test case specifications. They comprise the respective test steps that represent the interaction between test components and SUT (e.g. the expected test data instances that are exchanged)

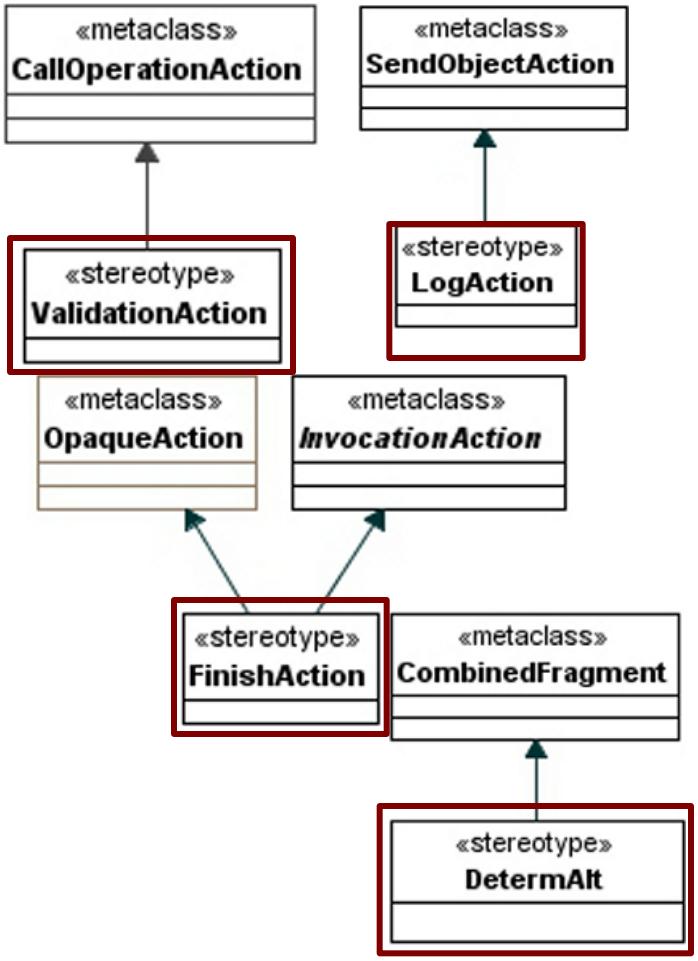
- Verdict reprsents the final conclusion of a test case. The precedence of the predefined verdict kinds is: none < pass < inconclusive < fail < error

- TestObjective describes the purpose for realization and execution of a test case



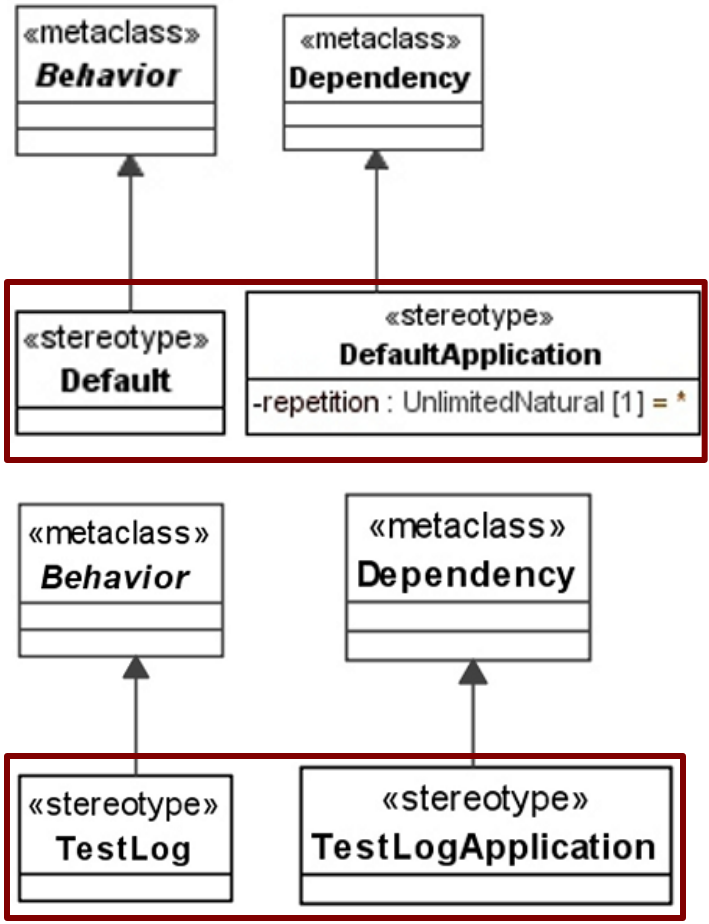
Test Behavior – Test Actions

- ValidationAction is used to set/calculate the verdict of a test case, or a single test step.
- LogAction enables the tester to capture log traces for further analysis
- FinishAction allows the termination of a test component for a particular test case
- determAlt calculates the entry conditions of CombinedFragments in a deterministic way



Test Behavior – Defaults and Logs

- Default behavior for outsourcing the handling of unexpected reactions of the SUT are described as autarcic behavior
- TestLog describes the traces of an actual execution of a test case (e.g. the real instances of data that are exchanged among test components and SUT)
- Respective applications bind both concepts to corresponding test cases or test steps

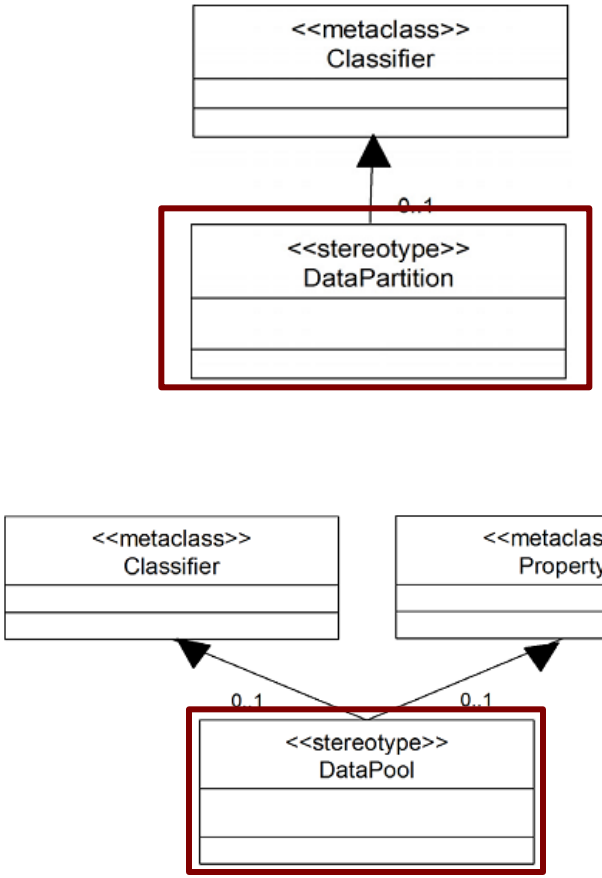


Test Data

- DataPartition allows the definition of logical partitions of test-relevant data types. DataPartitions can be created from scratch or reuse existing type definitions

- DataPool is used for the specification of data tables that can be used for repeated test case execution

- Instances of DataPartition and DataPool represent concrete test data instances, which are used to test case execution

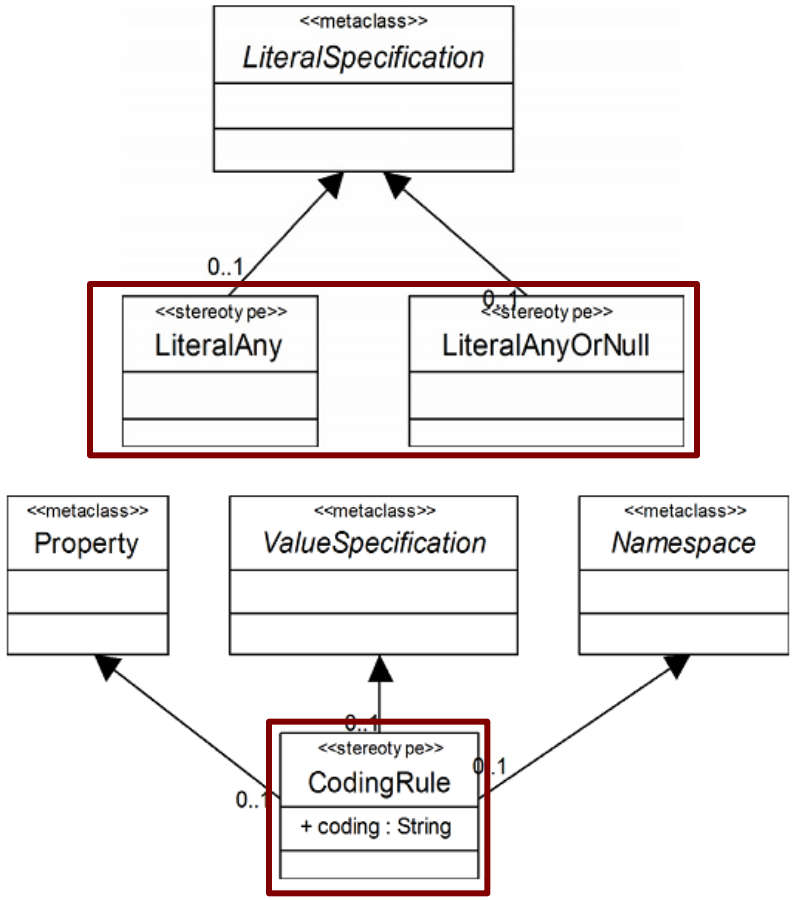


UTP en detail

Test Data

- LiteralAny/LiteralAnyOrNull are used to express wildcards for concret test data instances.
  - Targets an easier creation of test data instances
  - Allows a tester to focus only on the relevant parts of test data instances

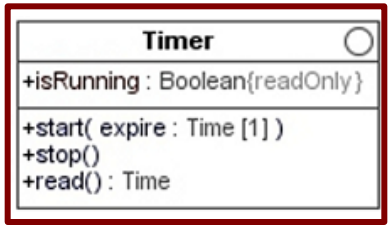
- CodingRule defines how values of messages shall be decoded/encoded for communication between the SUT and test components (e.g.ASN.1, SOAP …)



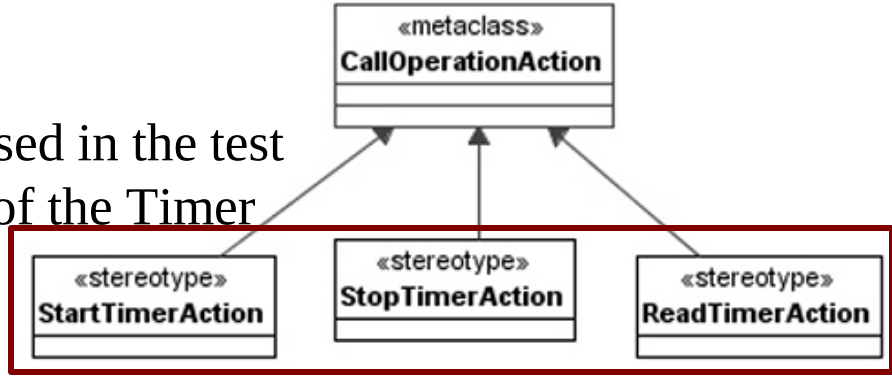
Timer Concepts

- Time is a primitive type for an abstract description of time units within test case executions

- Timer is a predefined interface that represents the timer handling interface of the test execution environment. It is responsible to manage the creation and expiration of timers used by test components within a test case.

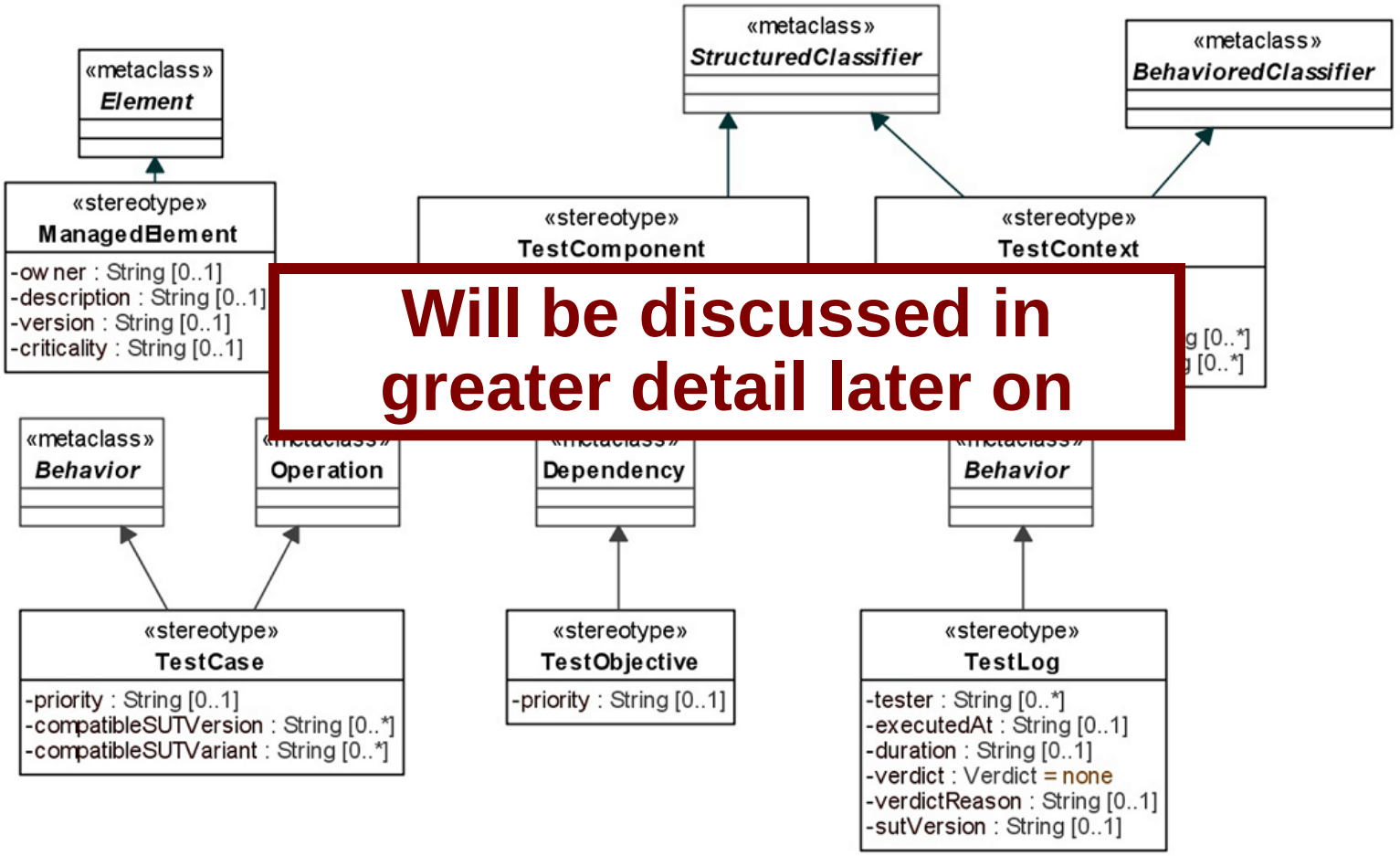


- Start-/Stop-/ReadTimerActions are used in the test case execution to call the operations of the Timer interface



UTP en detail

Test Management



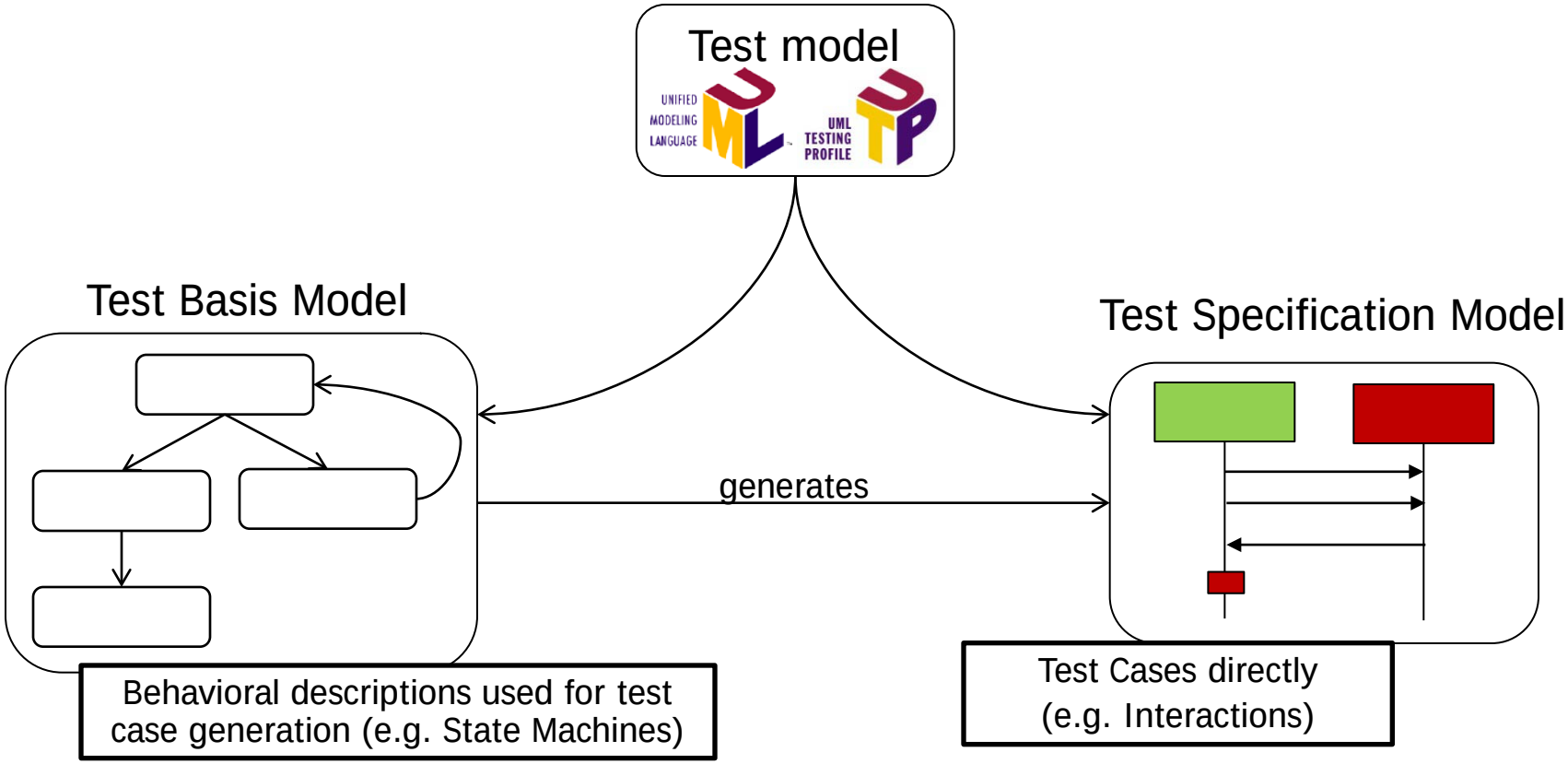
Agenda

- Introduction ✓
- UTP en detail ✓
- **UTP @ Work**
  - Modelling with UTP
  - Model-based test management
  - UTP in a safety-critical domain
- Outlook

Outline

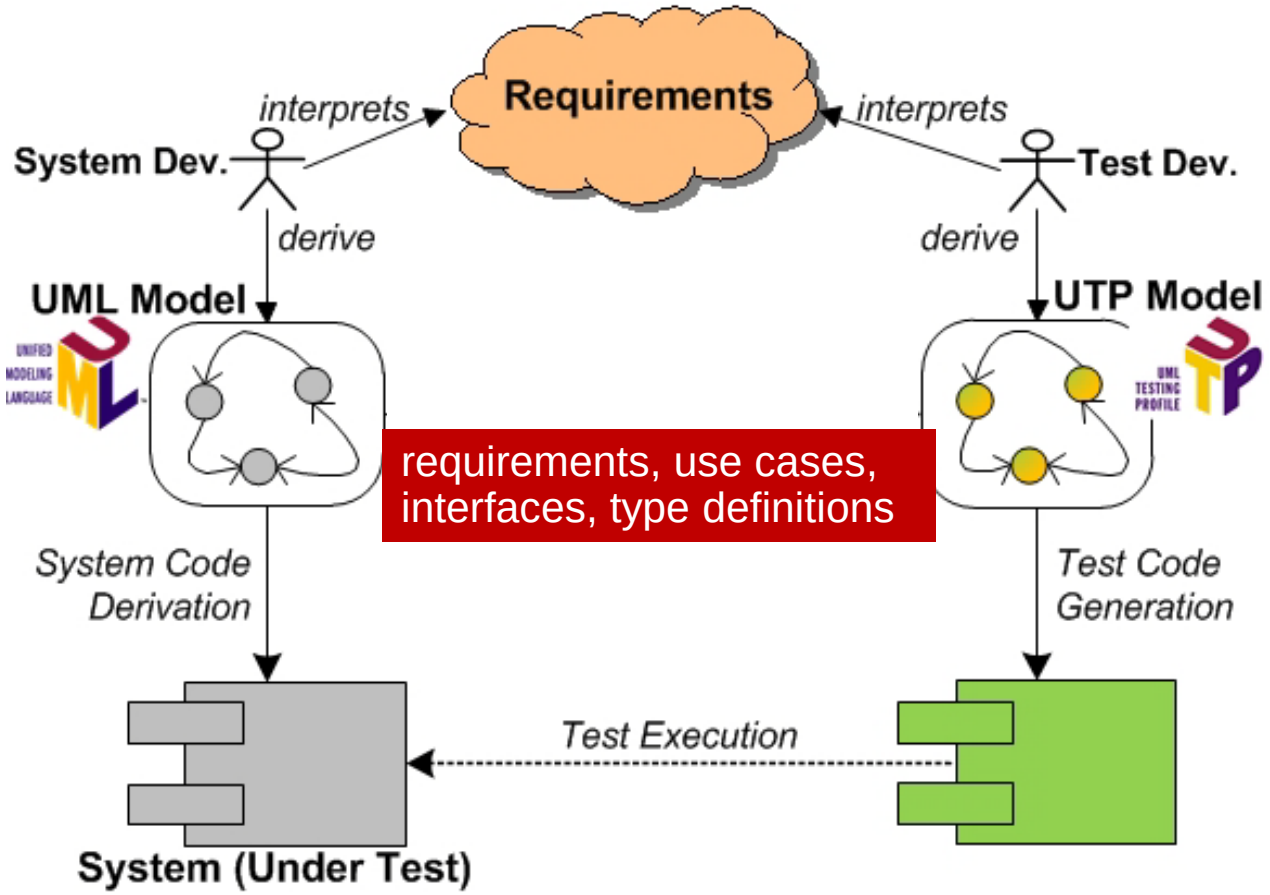
- Modelling with UTP
- Structural test modelling
  - Test architecture
  - Test configuration
- Behavioral test modelling
  - Abstract/concrete vs. logical/technical test cases
- Test data modelling
  - Stimuli and oracles
  - Data partitions and data pools

Views on a Test Model



Modelling with UTP (2)

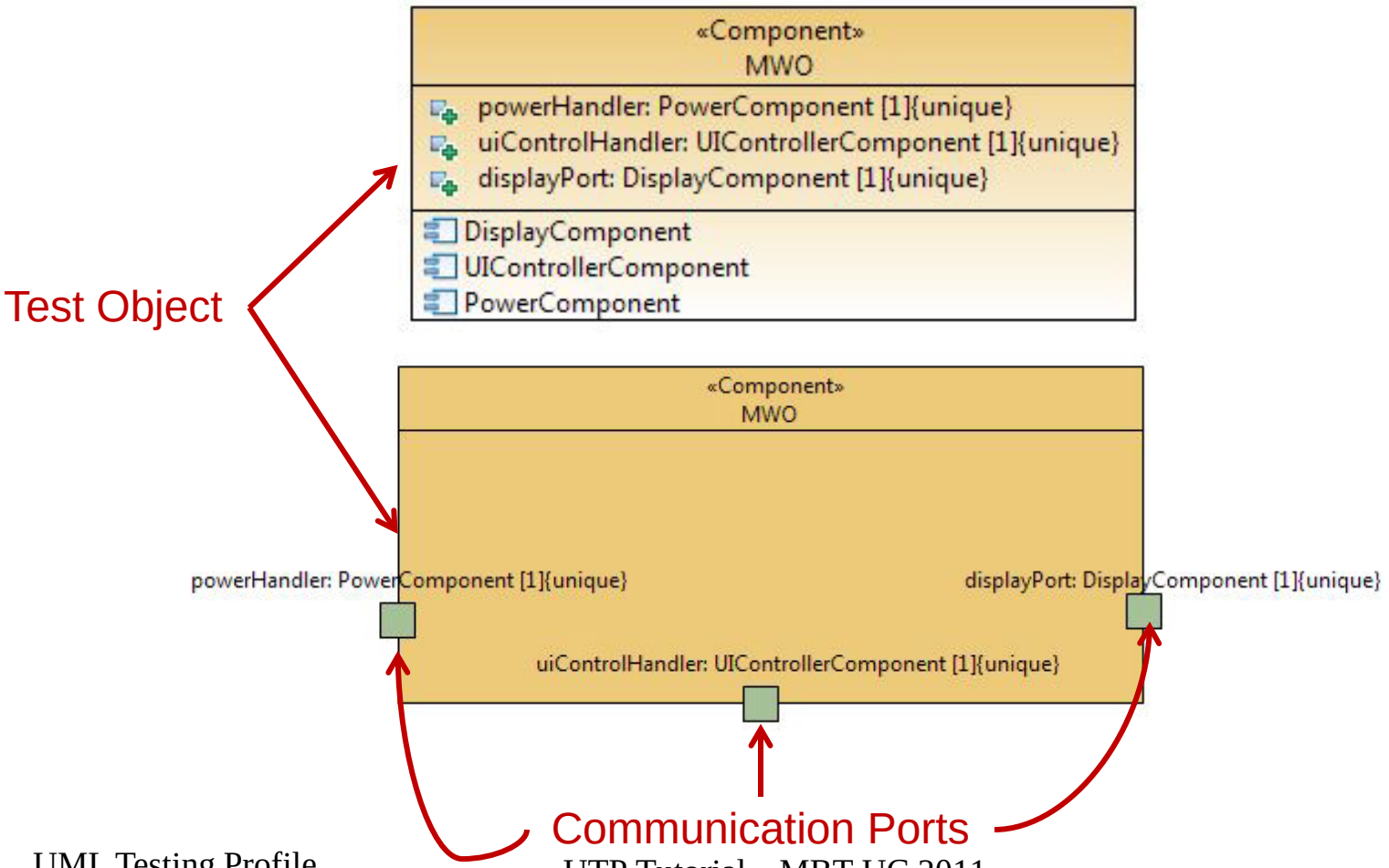
Modelling with UTP (2)



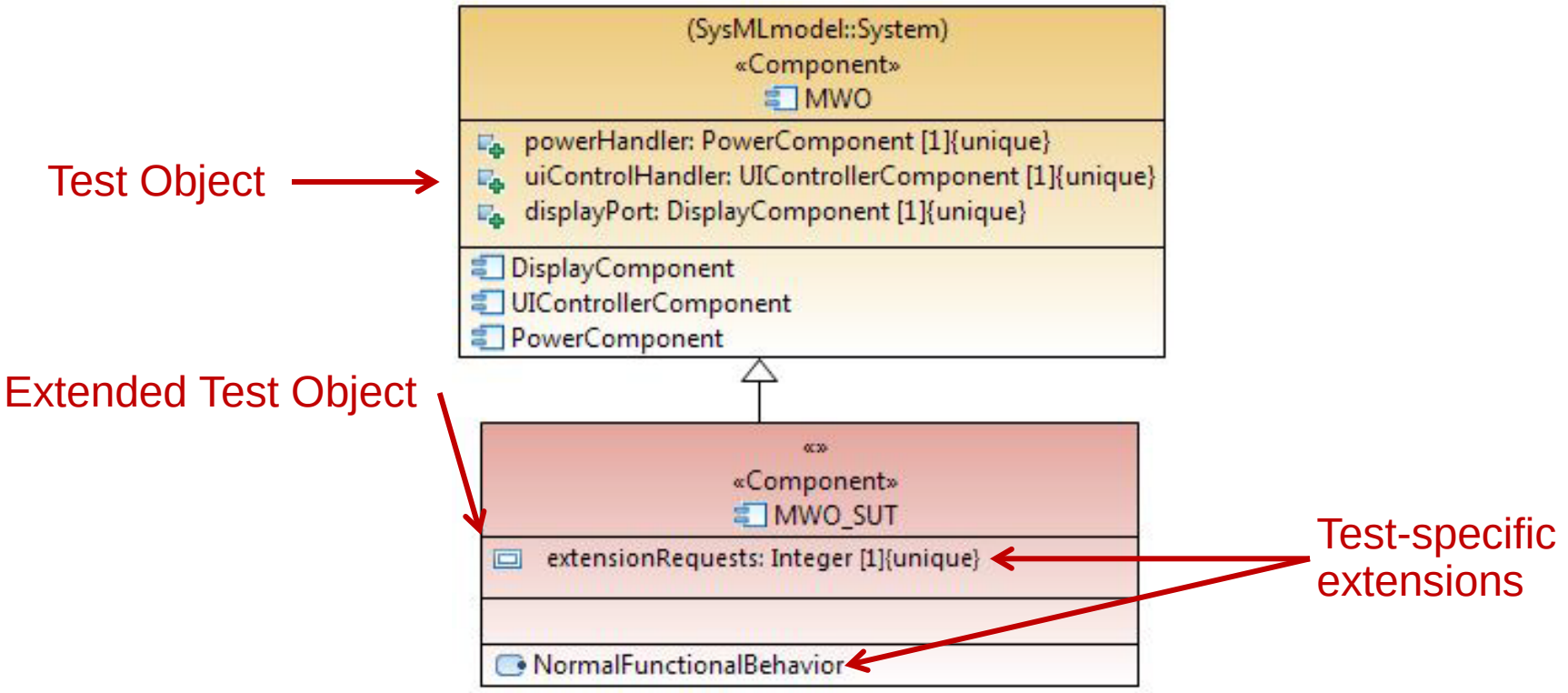
Structural test modelling

- Structural elements are the foundational concepts in UML
  - No disembodied behavior, i.e. behavior exists only in combination with structure
  - Allows the visualiuation of structural/architectural view on the system
- Test structure describes the structure of a system and the test system from a tester‘s point of view
- Analysis and identification of the test objects
- Configuration of environmental model for the SUT

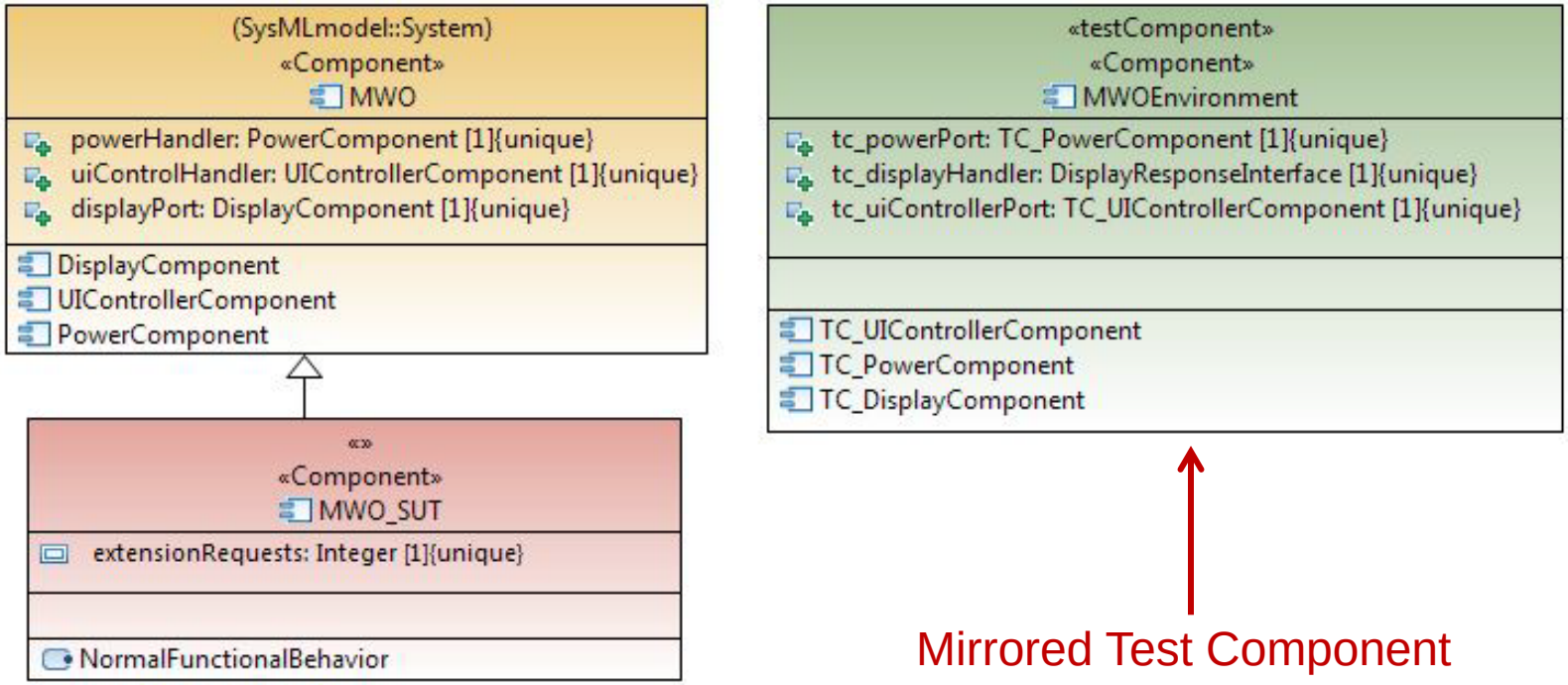
Structural test modelling (2)



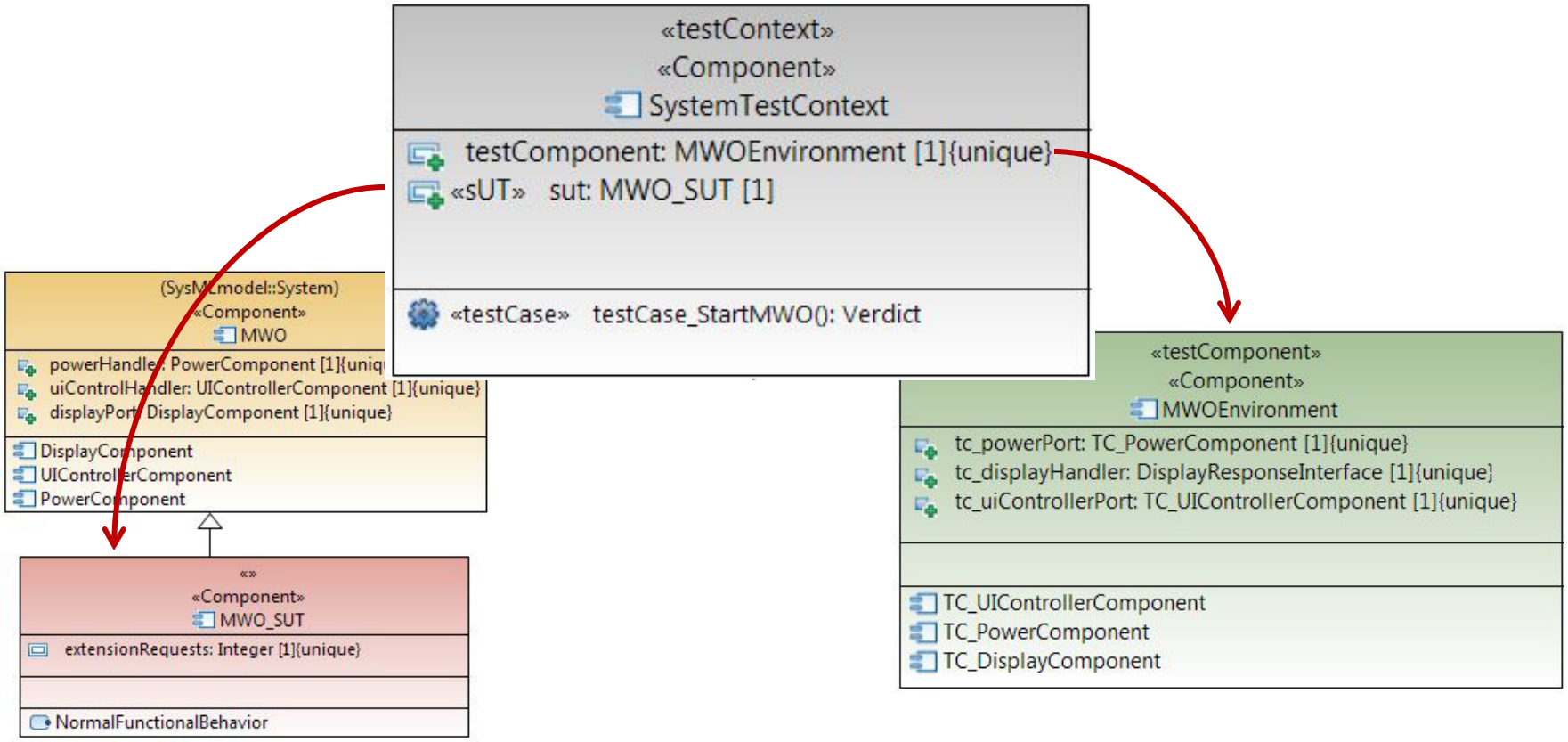
Structural test modelling (3)



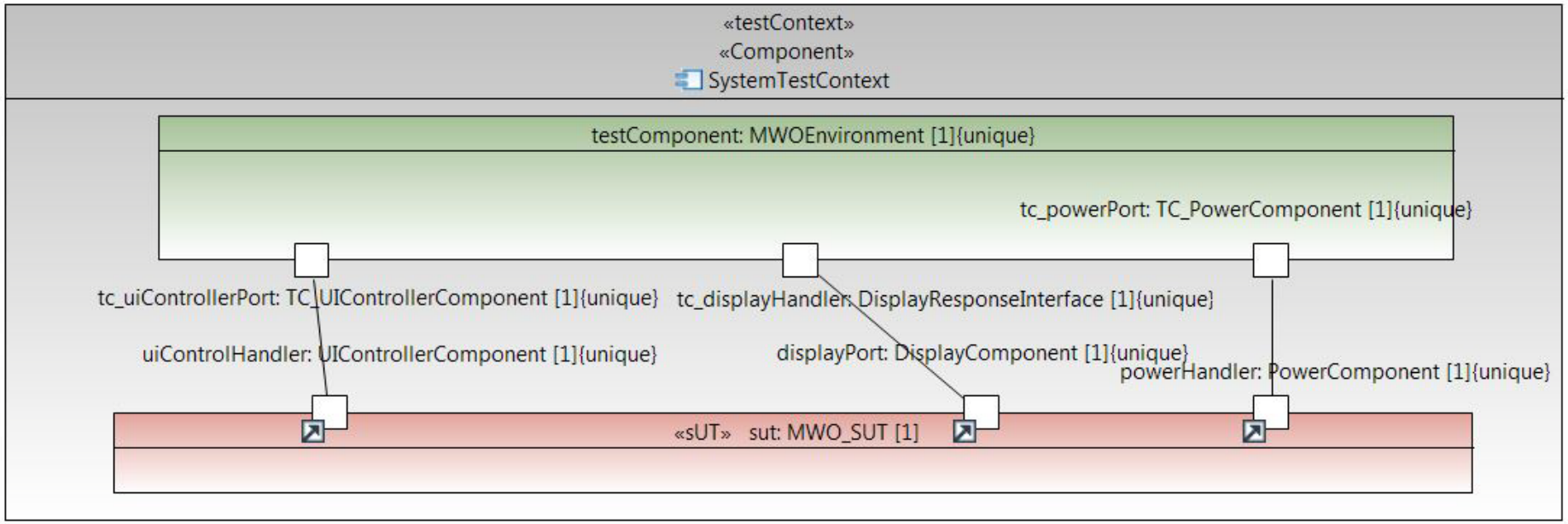
Structural test modelling (4)



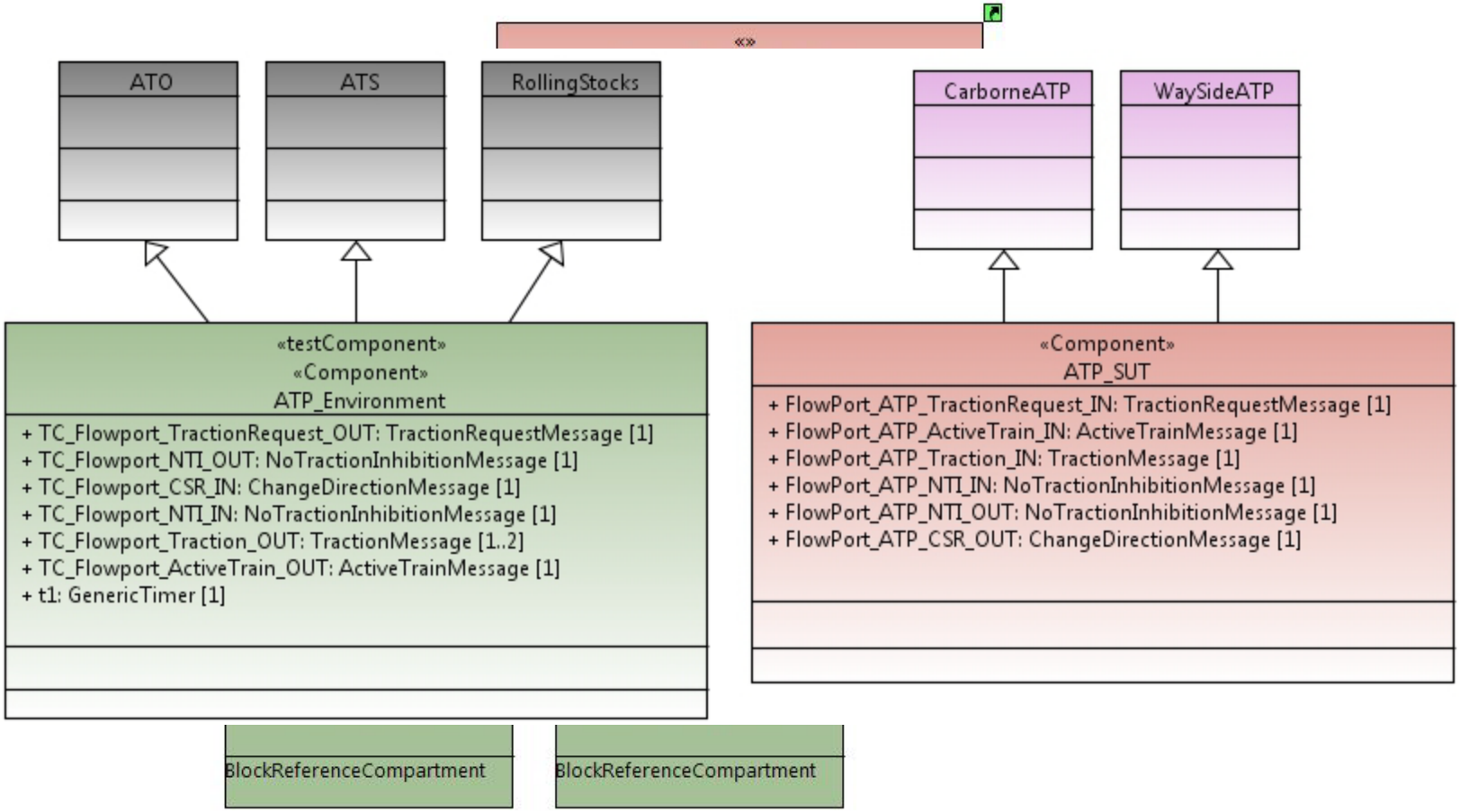
Structural test modelling (5)



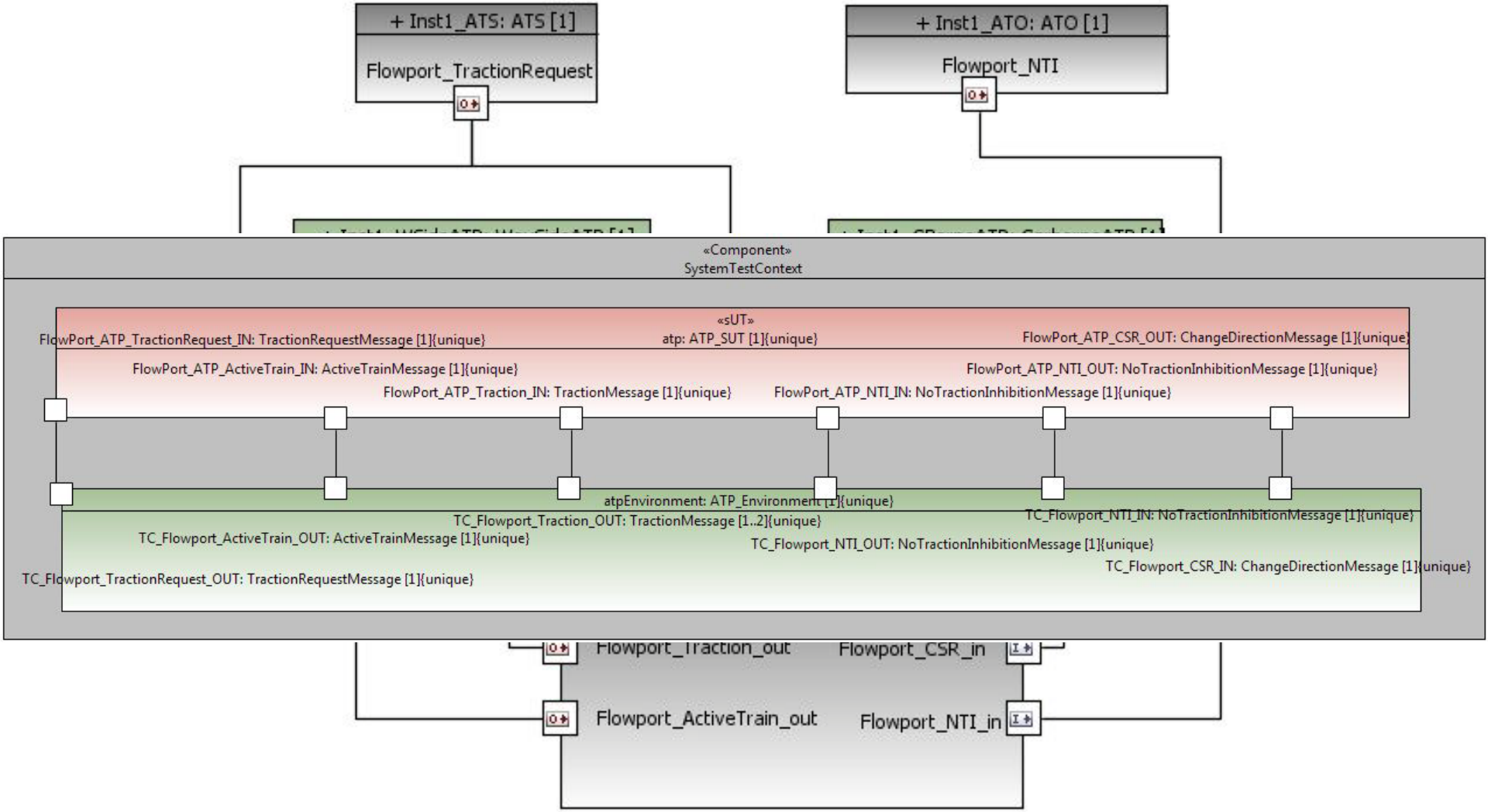
Structural test modelling (6)



Structural test modelling (7)



Modelling with UTP



Behavioral test modelling

- Test behavior provides a tester‘s view of how the system should be used according to its requirements specification
- All UML behavior types usable!
- Test behaviors may vary in their degree of maturity
  - Abstract/concrete vs. logical/technical
- Test behavior expressed as
  - behavioral description of the SUT (within the SUT)
  - behavioral description of test components
  - direct interaction among SUT and test components
  - global representation of the interaction

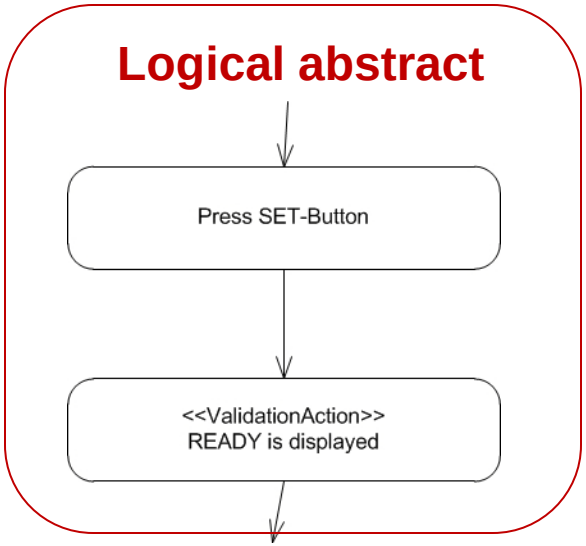
Logical/Technical vs. Abstract/Concrete

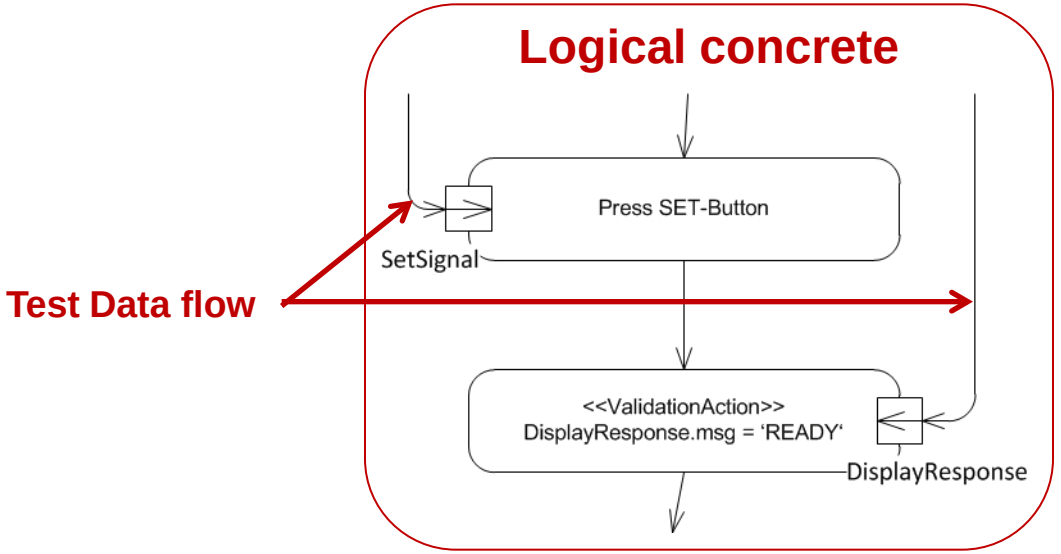
- Most literature distinguish between abstract and concrete test case
  - *Abstract* test cases omits test data
  - *Concrete* incorporates test data
- This distinction is not sufficient!
- Test case term can be furhter refined into
  - Logical (what shall the test verify/check?), and
  - Technical (how shall it communicate with the SUT?) test cases

Each can be abstract or concrete regarding test data!

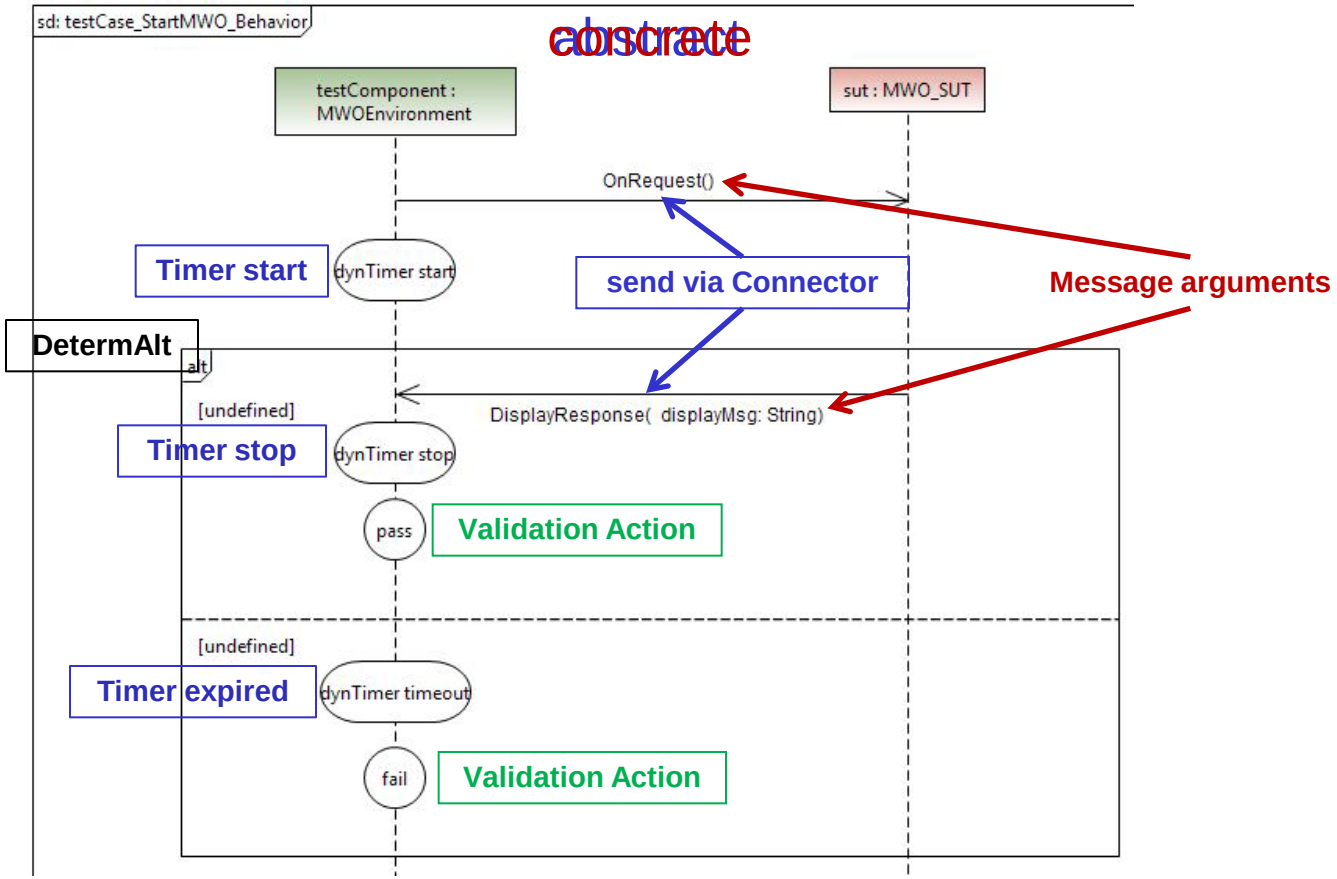
Logical test case specification

- A logical test case describe what a test case shall verify – not how!
- Can be created, read, reviewed by a domain expert

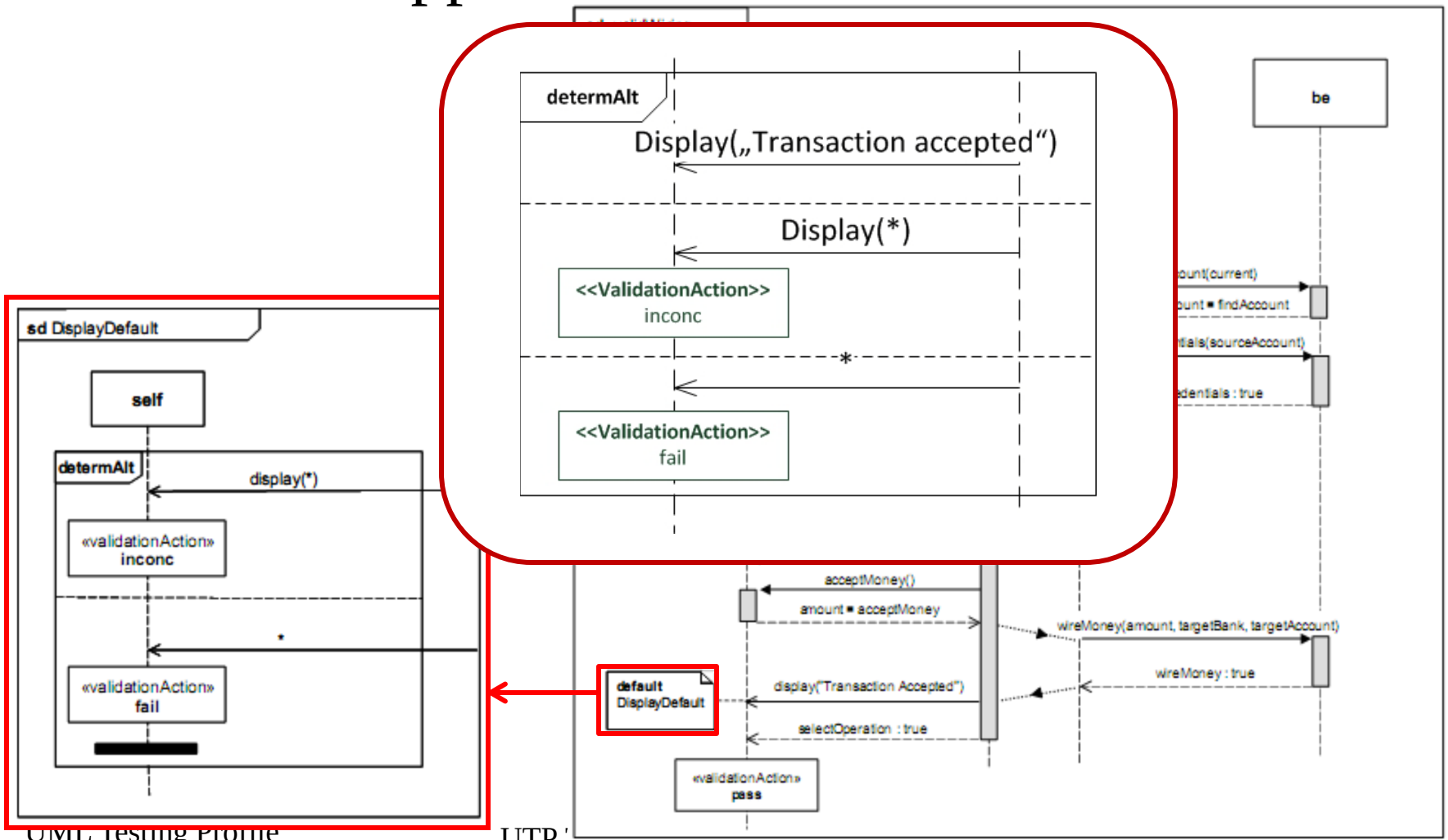




Technical test case specification



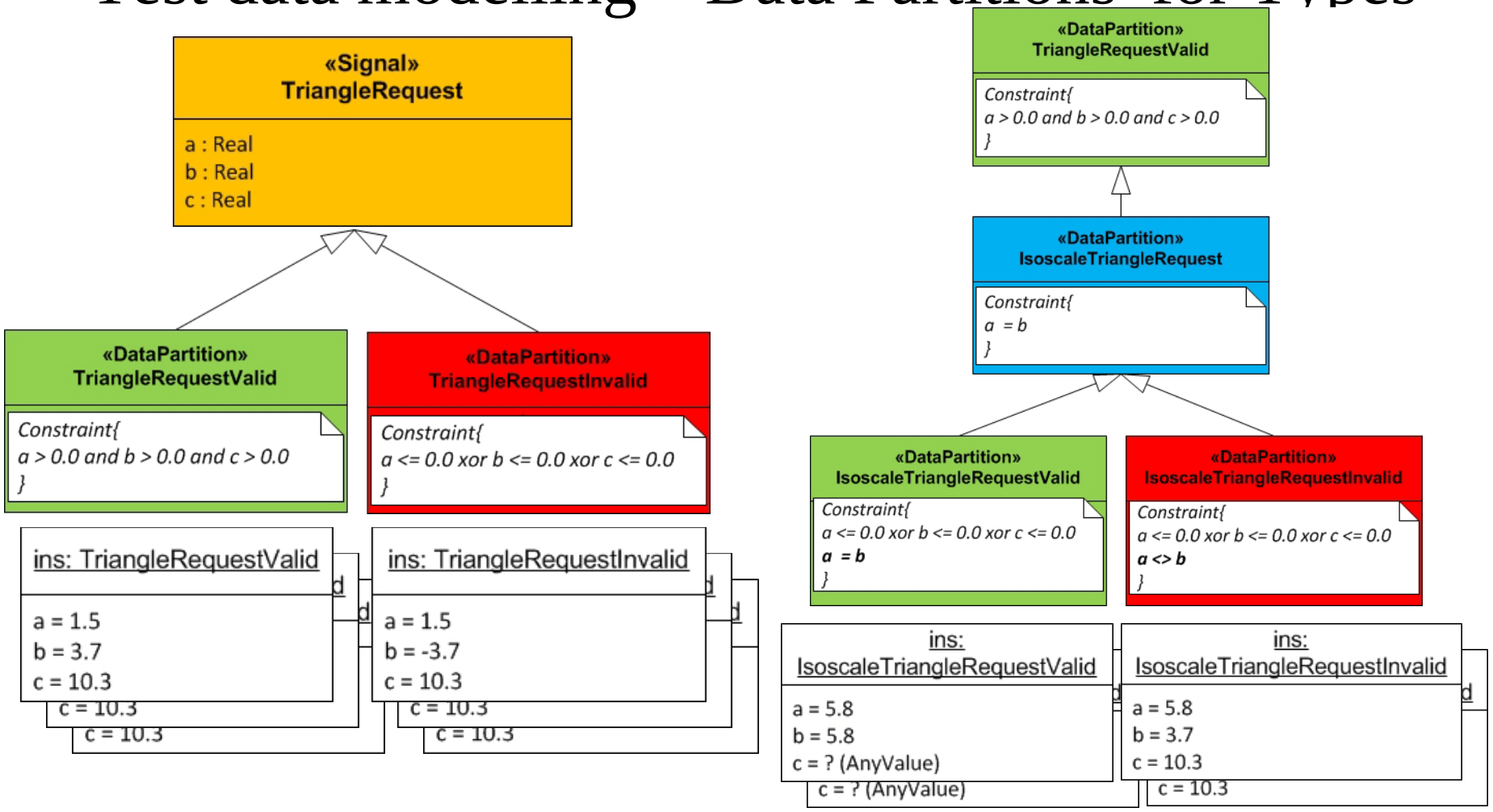
Application of Defaults



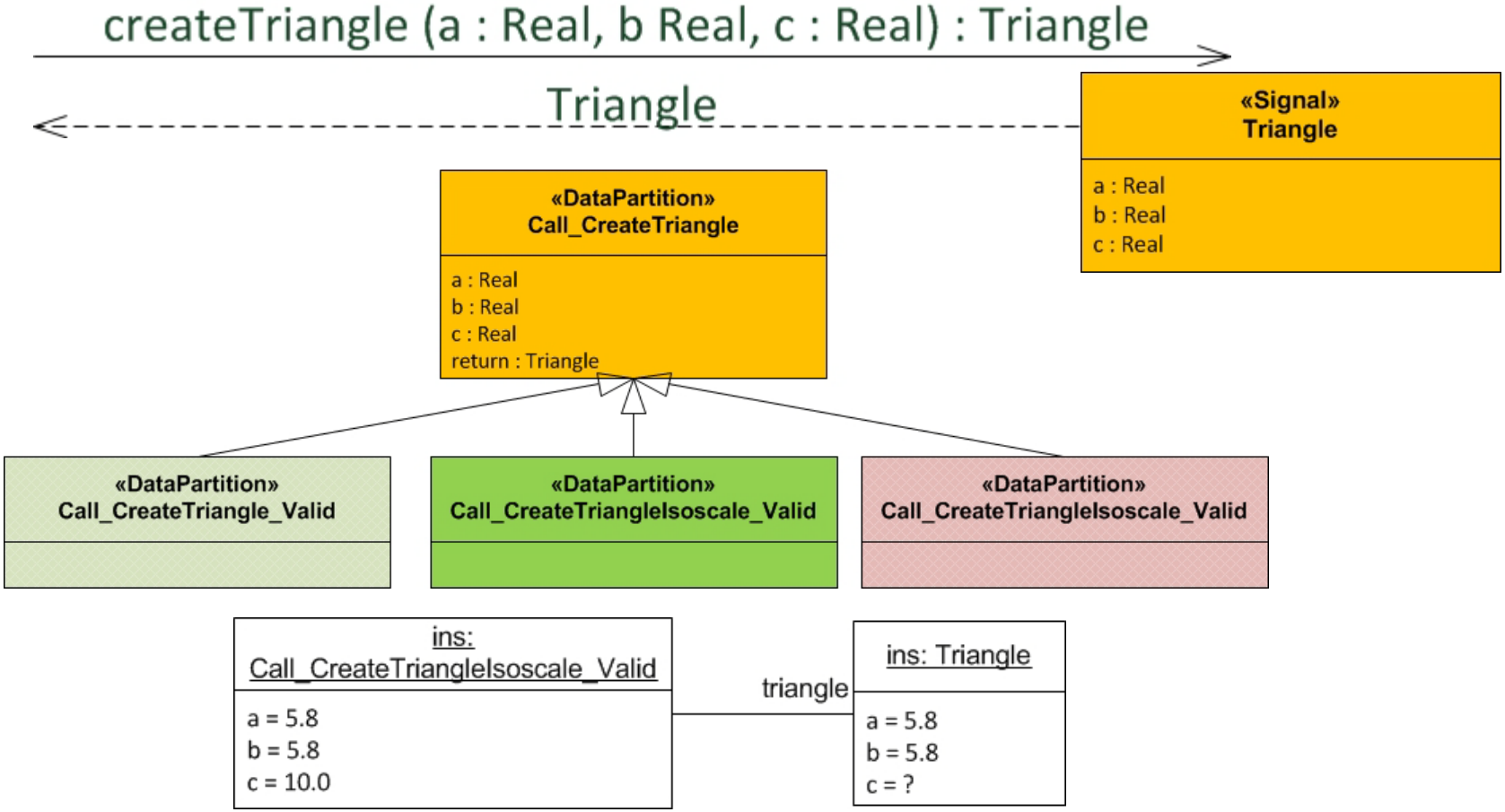
It‘s all about data exchange…

- **Test** data is vital for testing
- „Data that exists (for example, in a database) before a test is executed, and that affects or is affected by the (test) component or system under test.“ [ISTQB]
- UTP *Test Data* can be used to define
  - Data partitions (equivalence class) and respective representatives
  - Data structures (classification trees, equivalence class tables)

Test data modelling – Data Partitions for Types



Test data modelling – Data Partitions for Operations



Test data modelling – Data structures

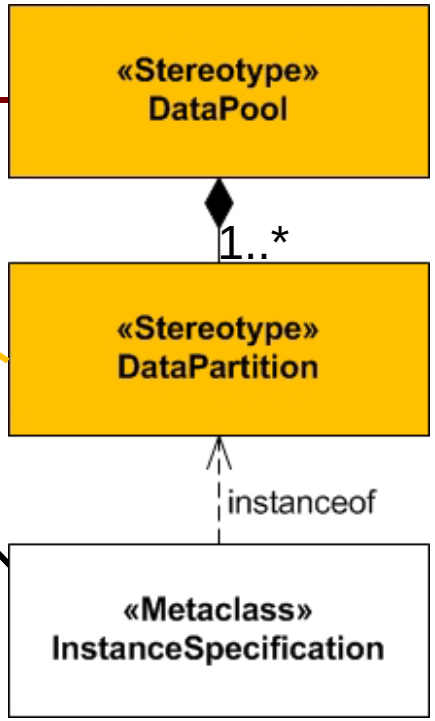
- Equivalence class tables / Boundary value anaylsis
  - Rules for Equivalence Class Table-based test case specification

createTriangle (a : Real, b : Real, c : Real) : Triangle								
Parameter		**a**		**b**		**c**		**Return Value**
Equ. Class		**a > 0**	a <= 0	**b > 0**	b <= 0	**c > 0**	c <= 0	
Test Case #								
1	valid	10,3		5,4		3,2		Triangle(10,5,5)
2	*invalid*		**-3,4**	5,4		3,2		**EXCEPTION**
3	*invalid*	10,0			**-5,0**	5,0		**EXCEPTION**
4	*invalid*	10,0		5,4			**-5,0**	**EXCEPTION**

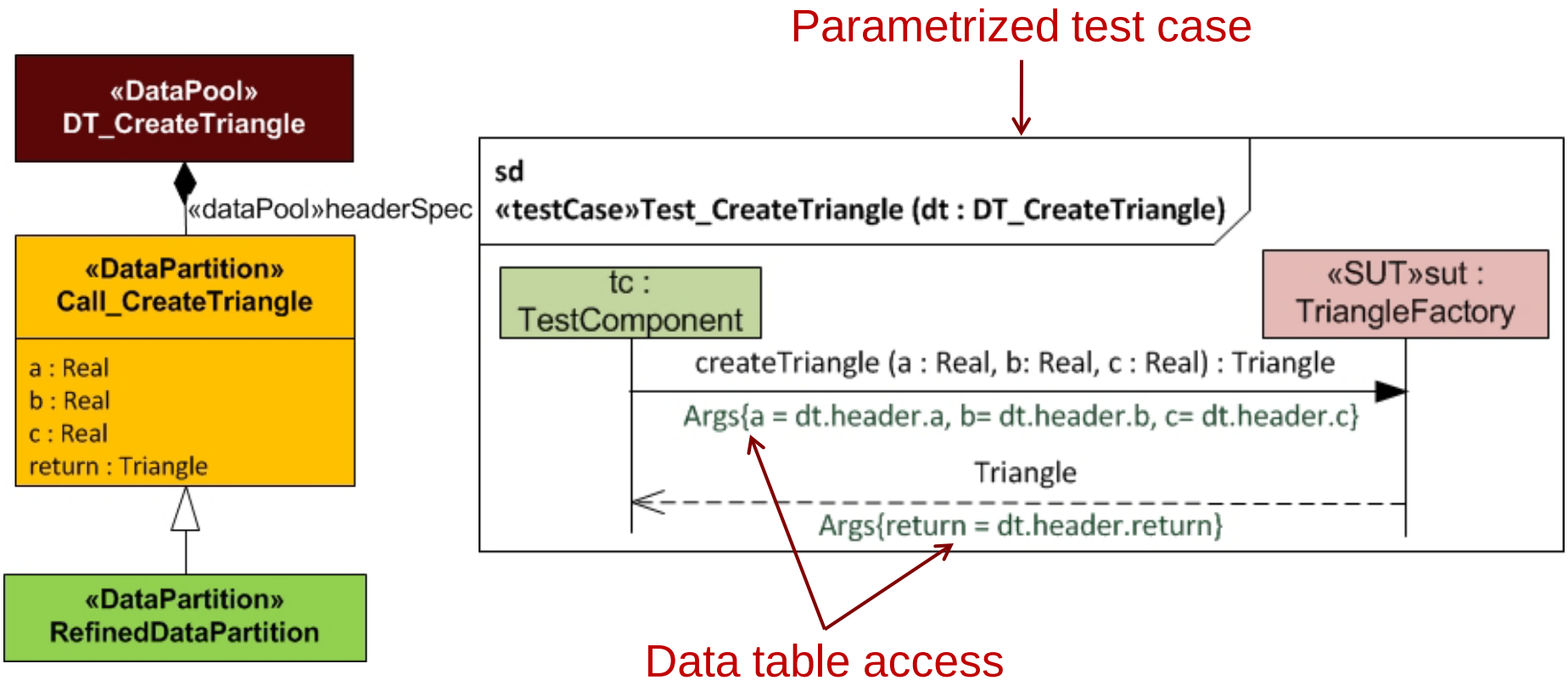
Test data modelling – Data structures

createTriangle (a : Real, b : Real, c : Real) : Triangle

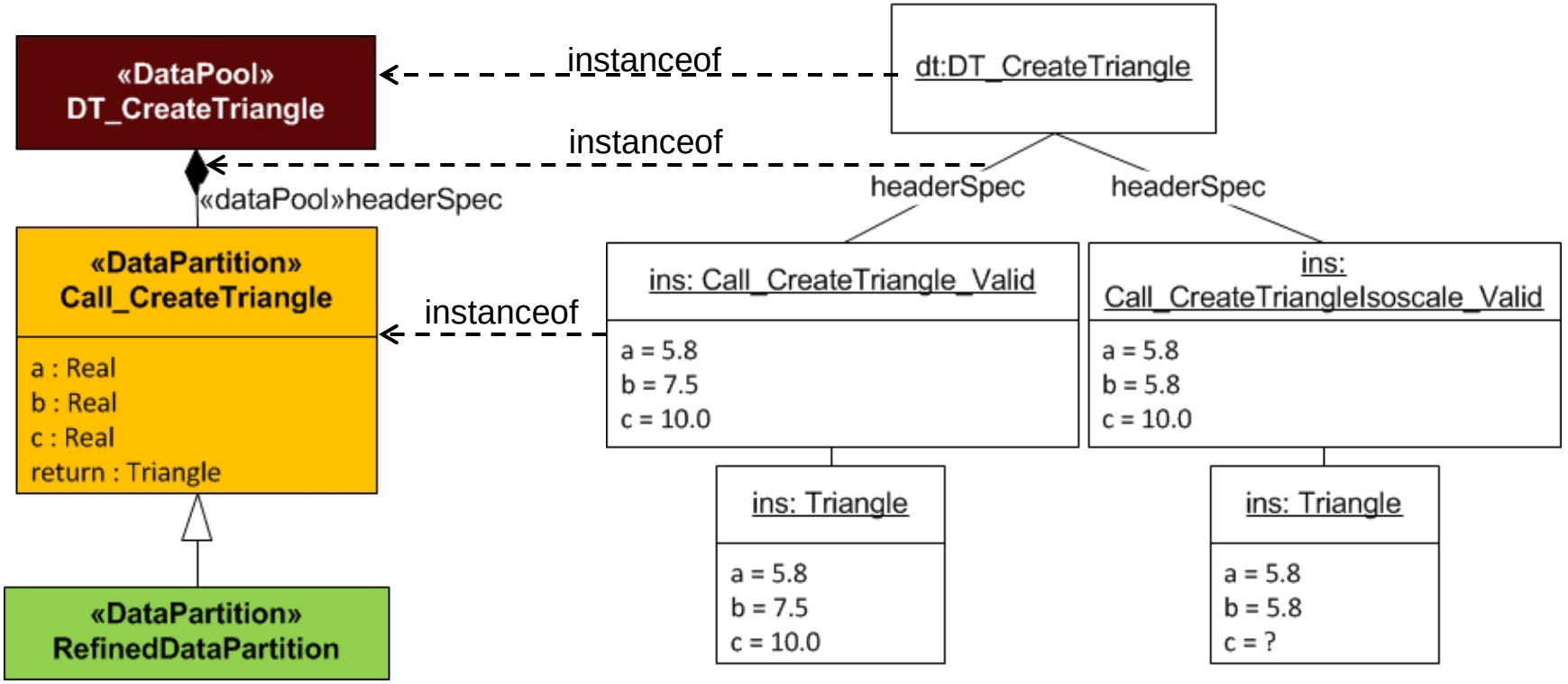
Parameter		a		b		c		Return Value
Equ. Class		a > 0	a <= 0	b > 0	b <= 0	c > 0	c <= 0	
Test Case #								
1	valid	10,3		5,4		3,2		Triangle(10,5,5)
2	*invalid*		-3,4	5,4		3,2		EXCEPTION
3	*invalid*	10,0			-5,0	5,0		EXCEPTION
4	*invalid*	10,0		5,4			-5,0	EXCEPTION



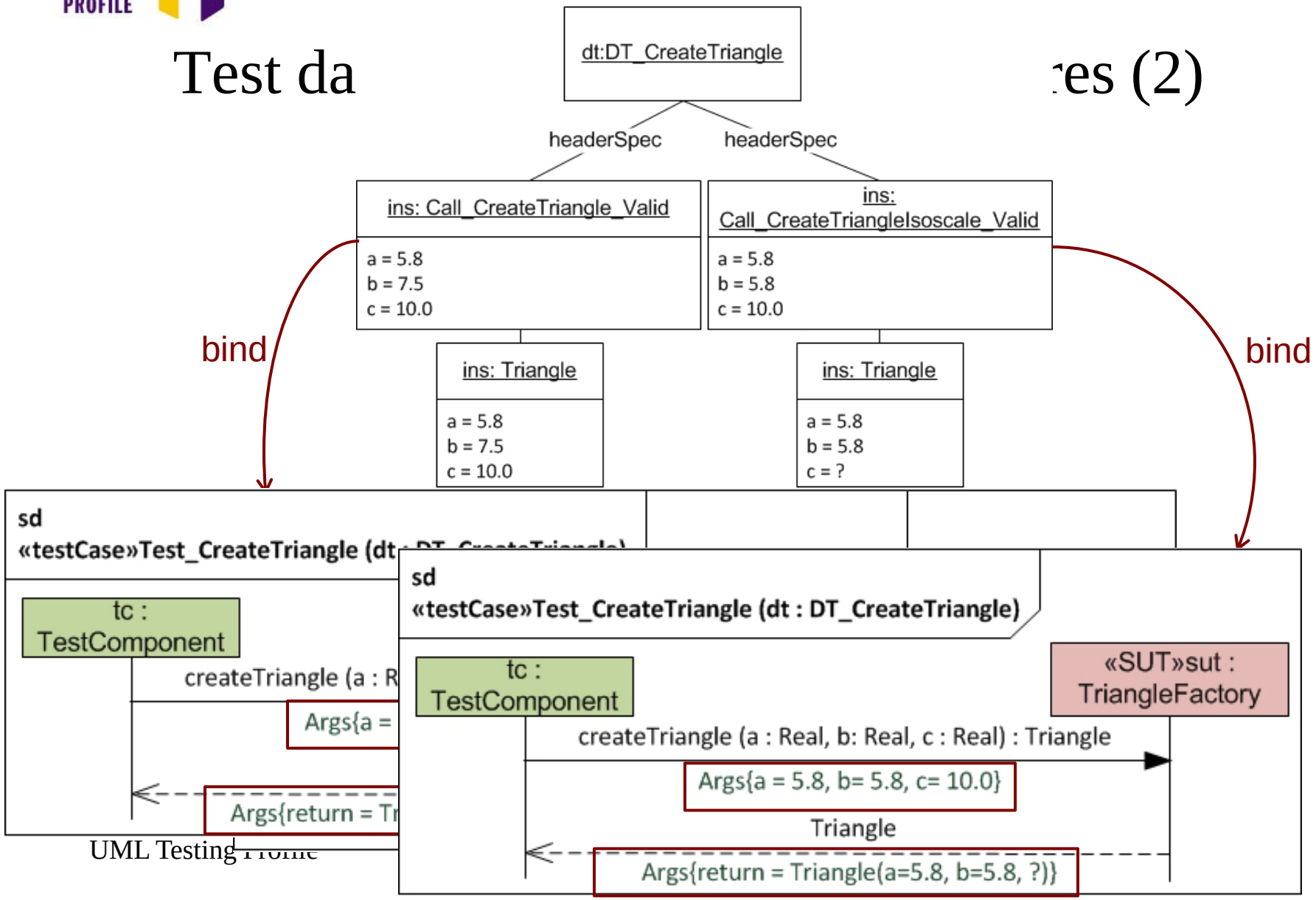
Test data modelling – Data structures (3)



Test data modelling – Data structures (2)



Test da ces (2)


dt:DT_CreateTriangle
headerSpec
headerSpec
ins: Call_CreateTriangle_Valid
a = 5.8
b = 7.5
c = 10.0
ins: Call_CreateTriangleIsoscale_Valid
a = 5.8
b = 5.8
c = 10.0
bind
bind
ins: Triangle
a = 5.8
b = 7.5
c = 10.0
ins: Triangle
a = 5.8
b = 5.8
c = ?
sd
«testCase»Test_CreateTriangle (dt : DT_CreateTriangle)
tc : TestComponent
createTriangle (a : R
Args{a =
Args{return = Tr
sd
«testCase»Test_CreateTriangle (dt : DT_CreateTriangle)
tc : TestComponent
«SUT»sut : TriangleFactory
createTriangle (a : Real, b: Real, c : Real) : Triangle
Args{a = 5.8, b= 5.8, c= 10.0}
Triangle
Args{return = Triangle(a=5.8, b=5.8, ?)}

Agenda

- Introduction ✓
- UTP en detail ✓
- UTP @ Work ✓
  - Modelling with UTP ✓
  - Model-based test management ✓
  - UTP in a safety-critical domain ✓
- **Conclusion and Outlook**