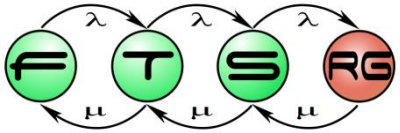
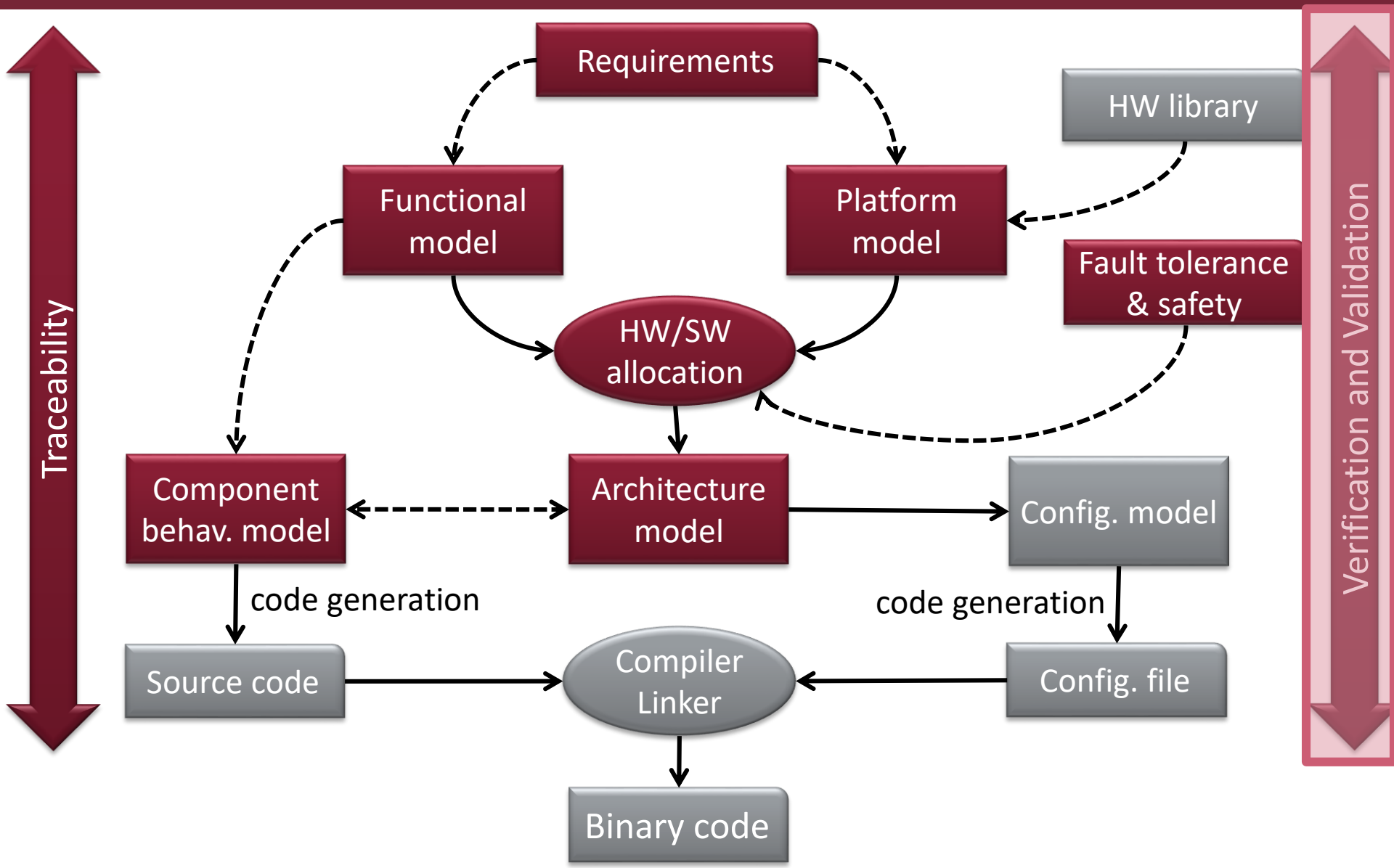
V&V: Model-based testing

Systems Engineering BSc Course



Budapest University of Technology and Economics
Department of Measurement and Information Systems

Platform-based systems design



Learning Objectives

Model-based testing

- Recall what is model-based testing
- List how models can be used in testing

Test modeling

- Explain the concepts in UML 2 Testing Profile
- Apply U2TP to specify configurations and scenarios in test models

Introduction to MBT

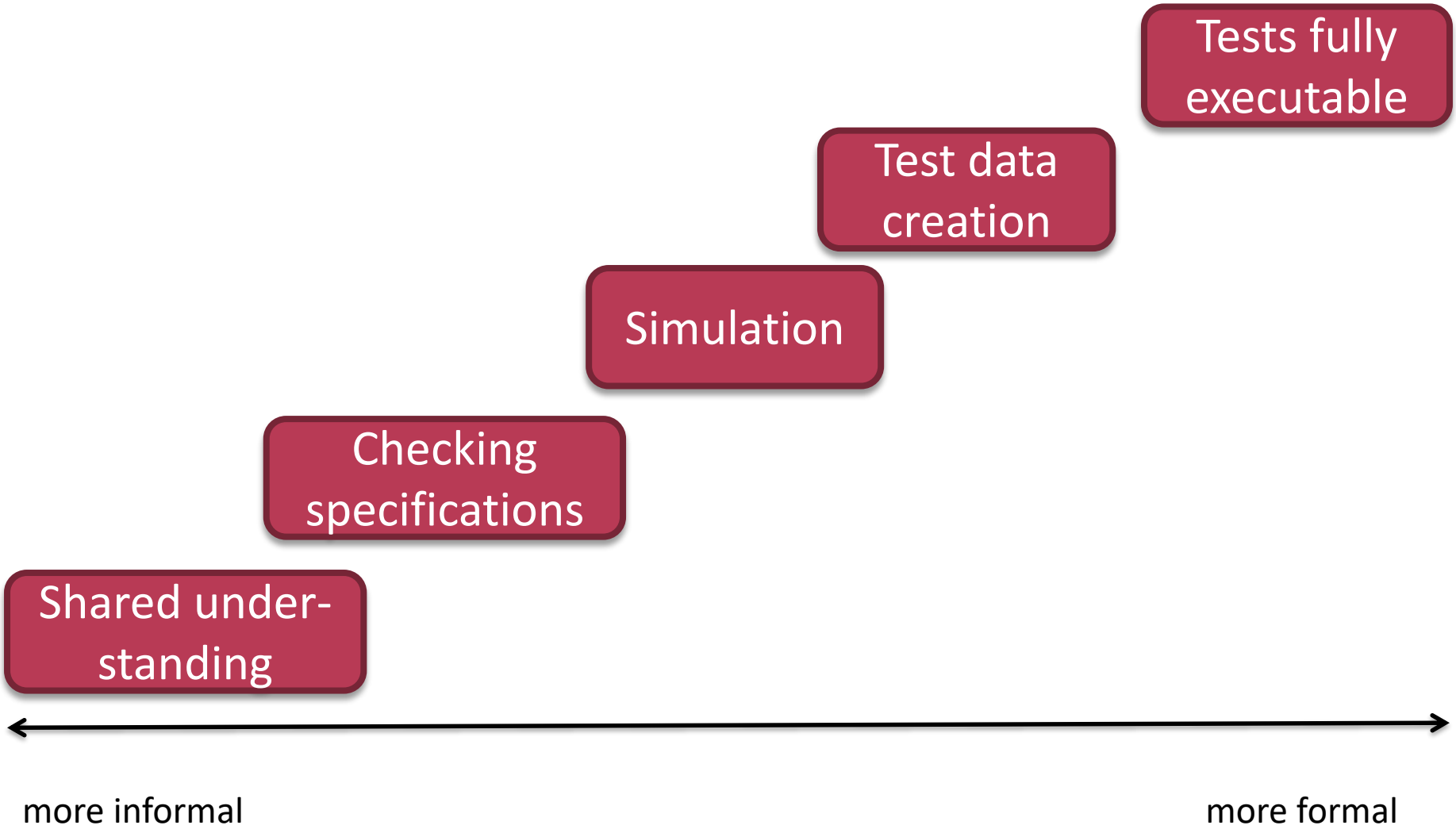
What is model-based testing?

“Testing based on or involving models” [ISTQB]

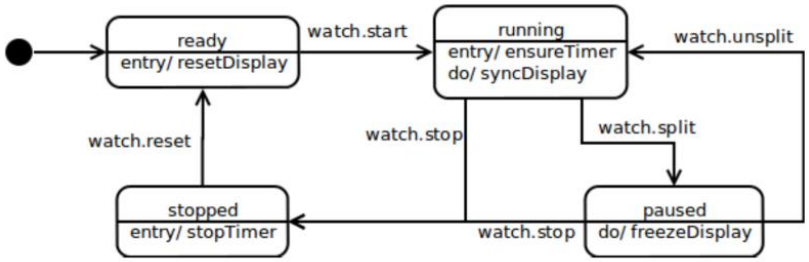
- Not just test generation
- Not just automatic execution
- Not just for model-driven engineering

Source of definition: ISTQB. “Foundation Level Certified Model-Based Tester Syllabus”, Version 2015

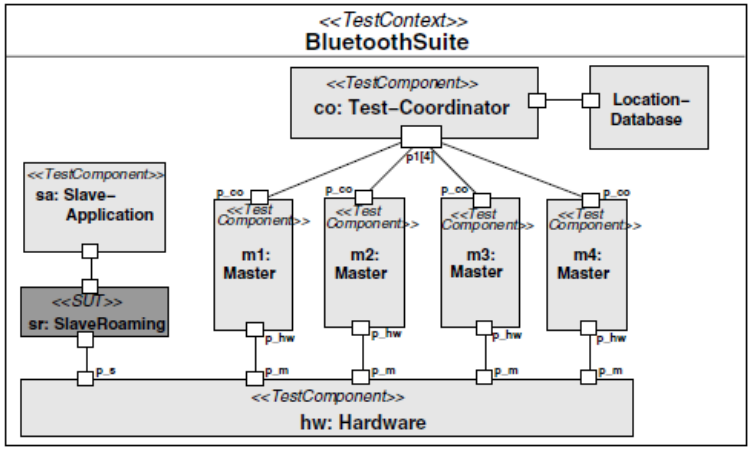
Landscape of MBT goals



Using models in testing (examples)



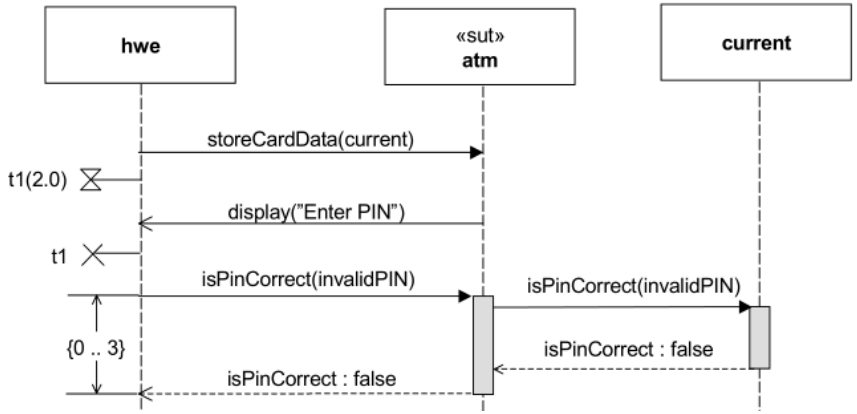
Behavior of SUT



Test configuration

```
timer t;
t.start(5.0);
alt {
 [] i.receive("coffee") {
  Count := Count+1; }
 [] t.timeout { }
}
```

Test sequences



Test sequences

Source: OMG UTP

Benefits of using models

- Close communication with stakeholders
  - Understanding of domain and requirements
- Early testing: modeling/simulation/generation
- Higher abstraction level (manage complexity)
- Automation (different artefacts)

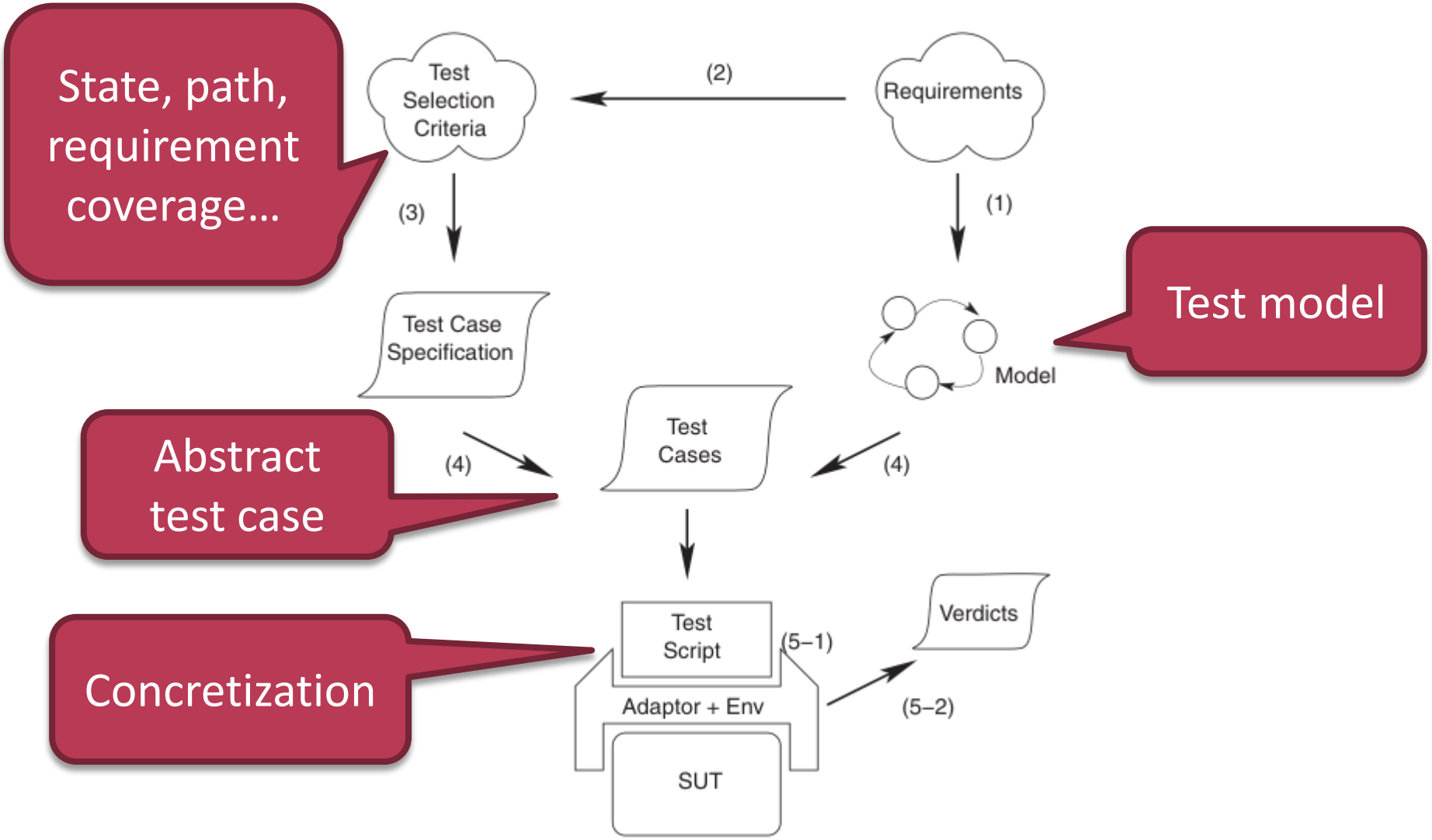
More specific meaning: Test generation

„MBT encompasses the processes and techniques for

- the automatic derivation of abstract test cases from abstract models,
- the generation of concrete tests from abstract tests,
- the manual or automated execution of the resulting concrete test cases”

Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

Typical MBT process



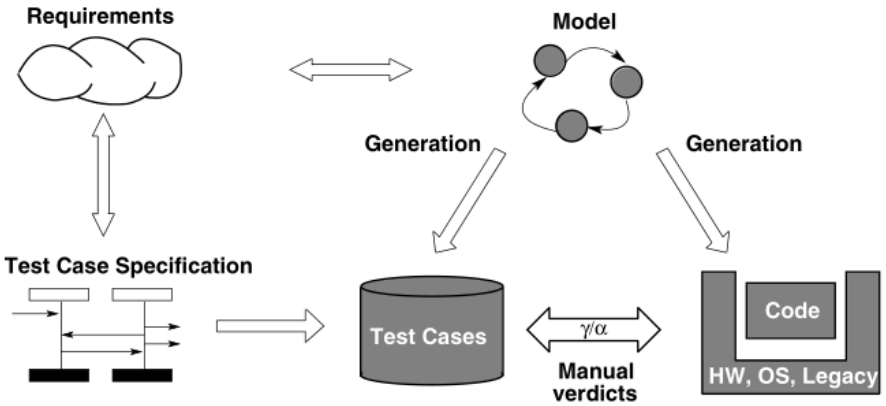
Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

DEMO MBT example

- Create test model using FSMs
- Use GraphWalker to generate test sequences
- Write adaptation to connect to Java code

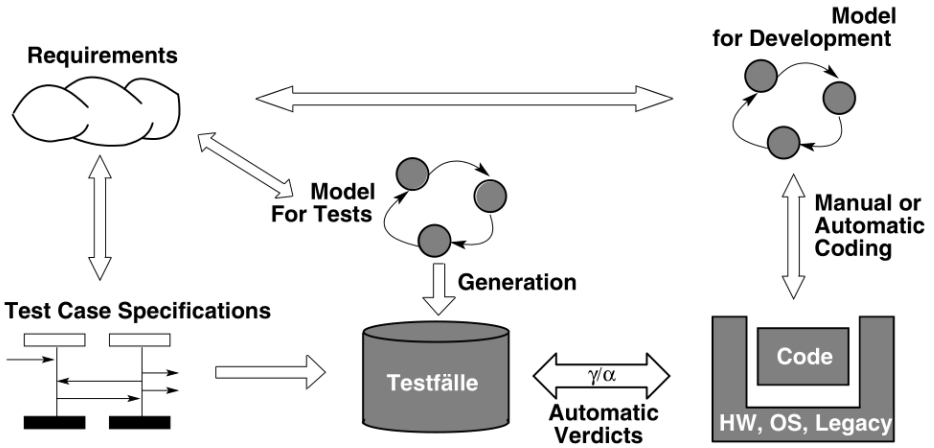
Reuse: Development and Test modeling

What if I have existing design models?



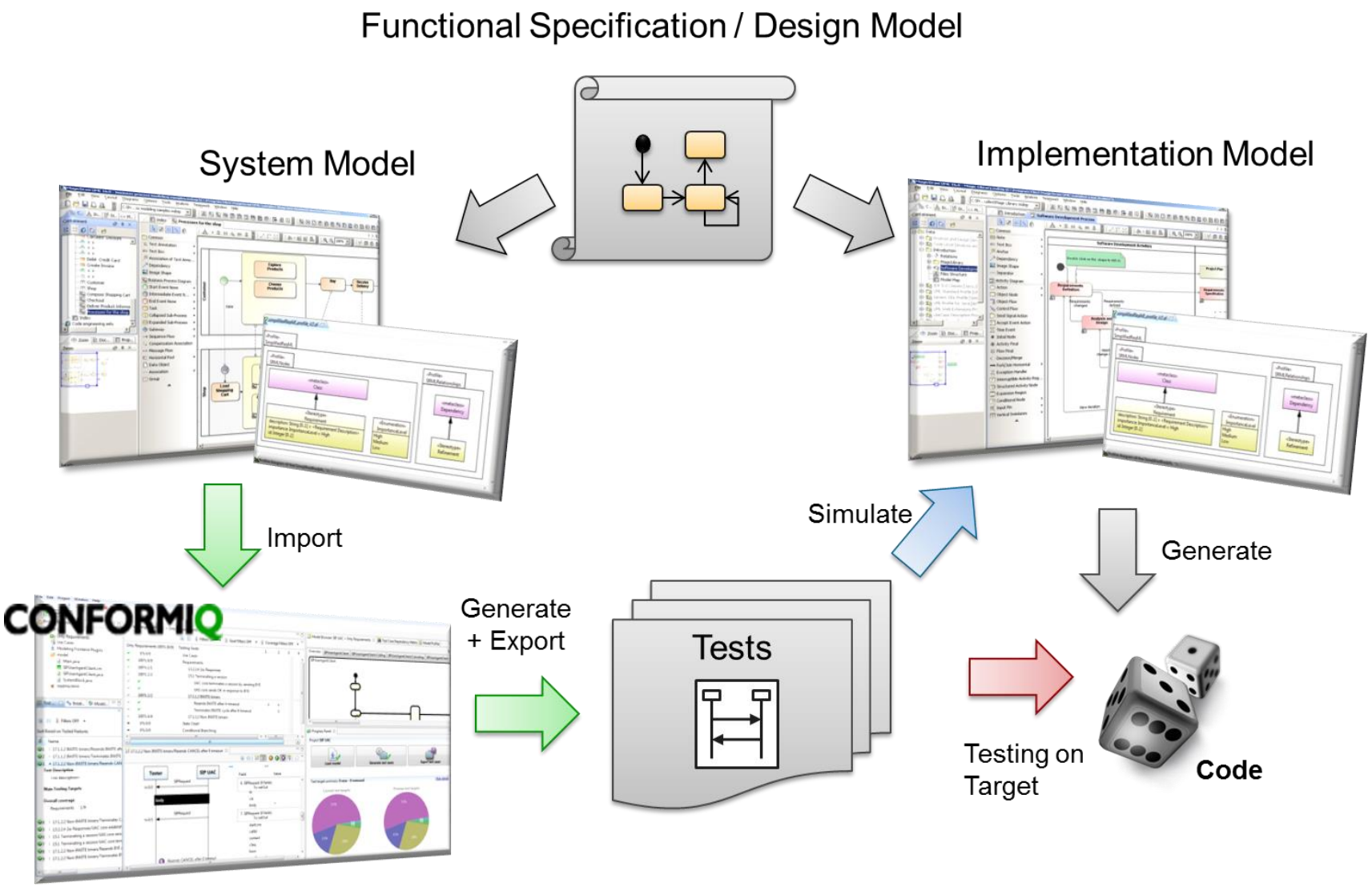
Problem: what do we test here?

Approach: separate dev. and test models



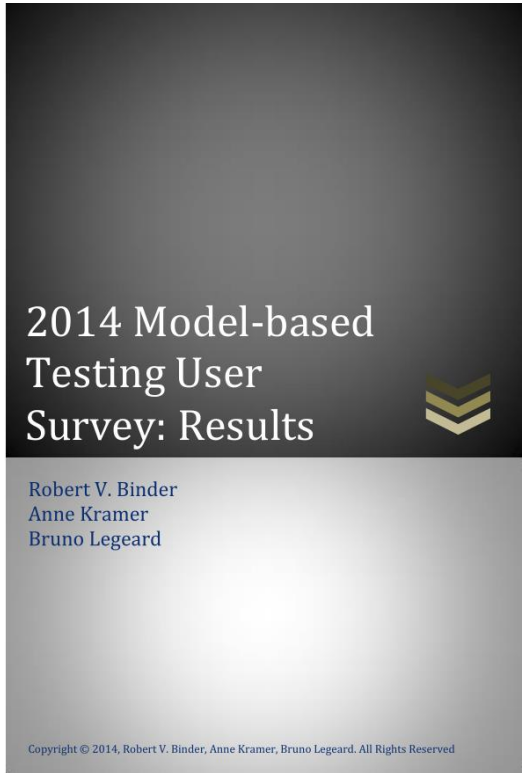
A. Pretschner, J. Philipps. „Methodological Issues in Model-Based Testing", Model-Based Testing of Reactive Systems, 2005.

Example: Model driven workflow



Source: Kimmo Nupponen. "Model driven workflow", 2016.

MBT User Survey 2014



~100 participants
32 questions

Testing levels	
System testing	77,4%
Integration testing	49,5%
Acceptance testing	40,9%
Component testing	31,2%

Generated artifacts	
Automated test scripts	84,2%
Manual test cases	56,6%
Test data	39,5%
Others (docs, test suites…)	28,9%

- "approx. 80h needed to become proficient"
- MBT is effective
- Lots of other details!

Overview: Model-based Testing: Where Does It Stand?

Source: http://model-based-testing.info/2014/12/09/2014-mbt-user-survey-results/

Recap: Tests in finite state machines

- (System modeling VIMIAA00 course)
- Sequence of input events and expected actions
- Model coverage
  - State coverage
  - Transition coverage
- Selecting tests to achieve coverage goals

Note

In the current course we will mainly work on test modeling and not automated test generation (see MSc courses on that topic)

UML 2 Testing Profile (U2TP)

UML 2 Testing Profile (U2TP) v1.1

- UML profile by OMG
- Capture information for functional black-box testing (specification of test artifacts)
  - Mapping rules to TTCN-3, JUnit
- Language (notation) and not a method (how to test)
- Defines stereotypes

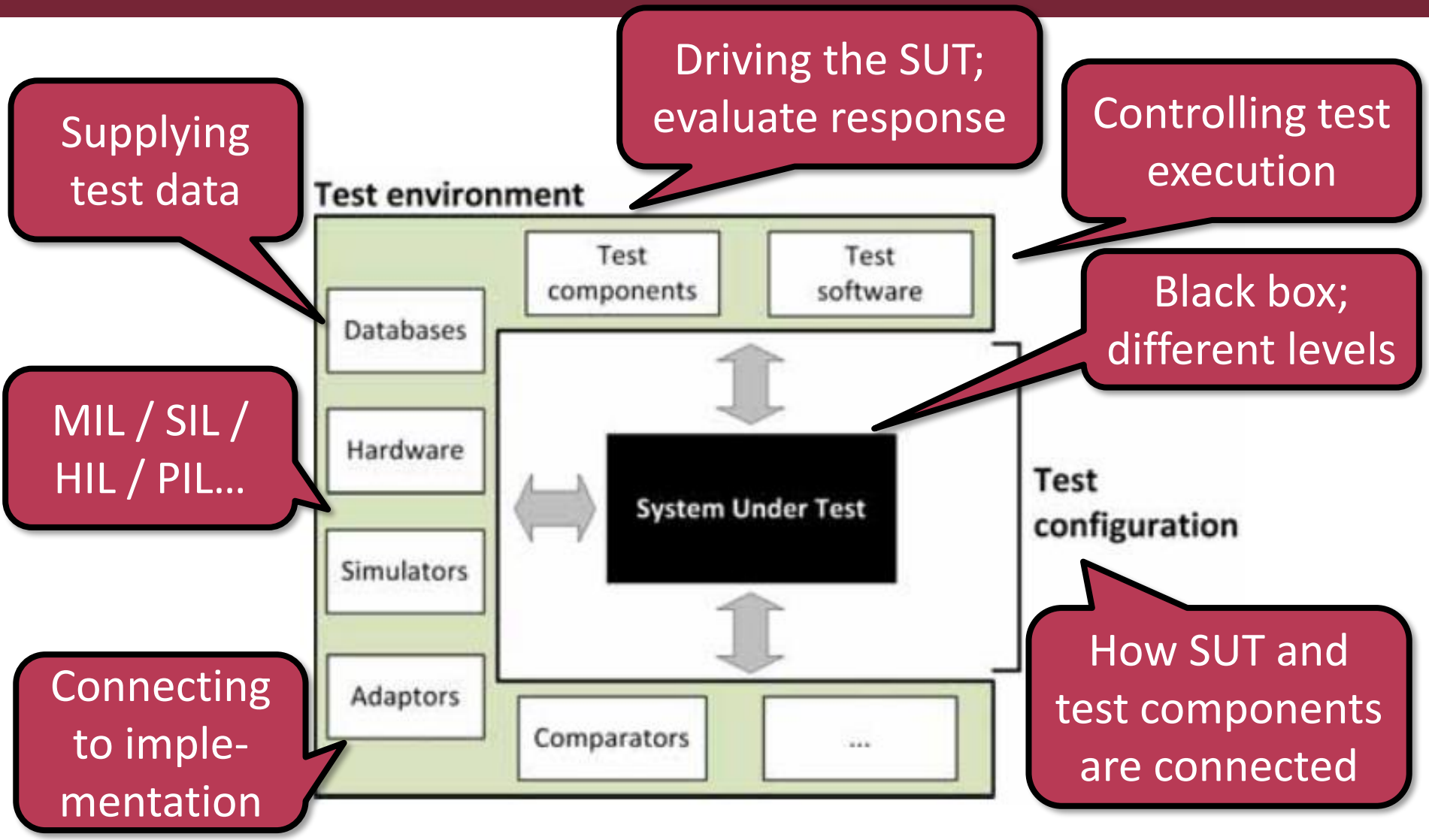
Packages (concept groups)

- Test Architecture
  - Elements and relationship involved in test
  - Importing the UML design model of the SUT
- Test Data
  - Structures and values to be processed in a test
- Test Behavior
  - Observations and activities during testing
- Time Concepts
  - Timer (start, stop, read, timeout), TimeZone (synchronized)

Packages (concept groups)

- **Test Architecture**
  - **Elements and relationship involved in test**
  - **Importing the UML design model of the SUT**
- Test Data
  - Structures and values to be processed in a test
- Test Behavior
  - Observations and activities during testing
- Time Concepts
  - Timer (start, stop, read, timeout), TimeZone (synchronized)

Overview of test architecture



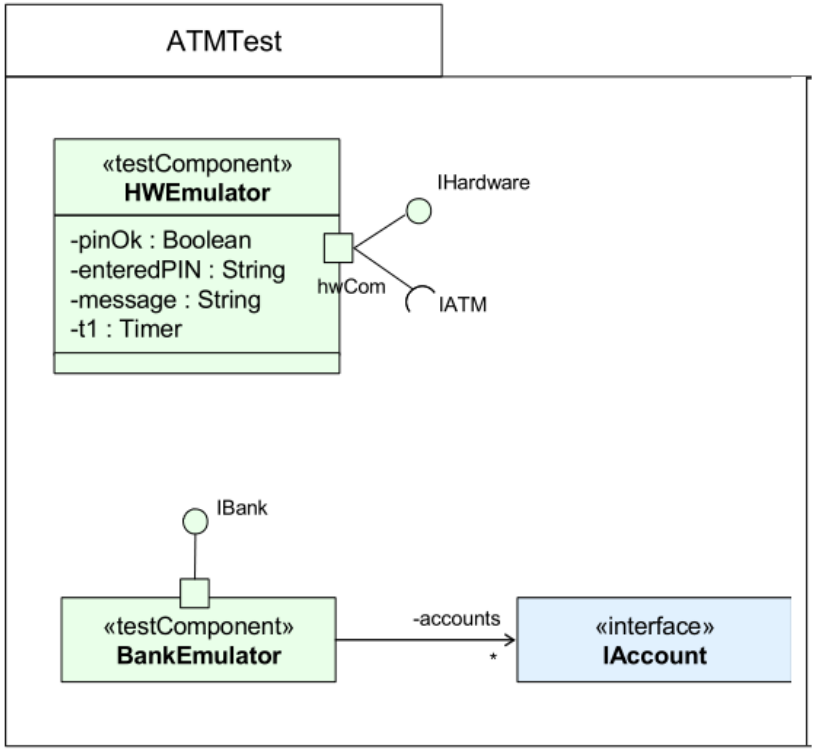
Source: OMG UTP

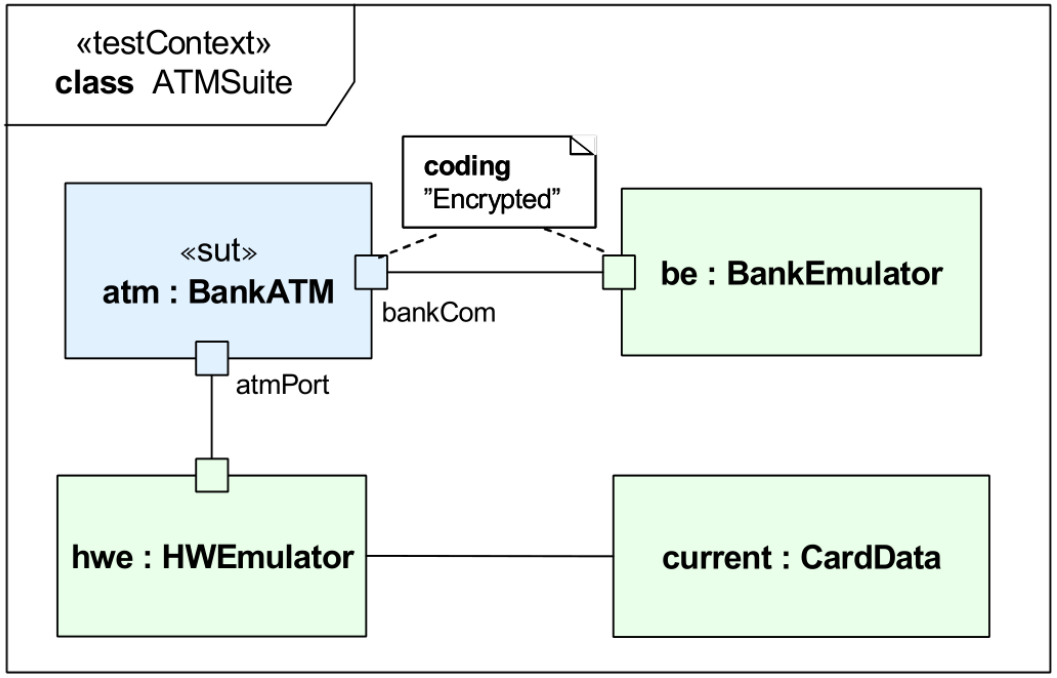
U2TP Test Architecture package

Identification of main components:

- SUT: System Under Test
  - Characterized by interfaces to control and observation
  - System, subsystem, component, class, object
- Test Component: part of the test system (e.g., simulator)
  - Realizes the behavior of a test case
- Test Context: collaboration of test architecture elements
  - Initial test configuration (test components)
  - Test control (decision on execution, e.g., if a test fails)
- Scheduler: controls the execution of test components
  - Creation and destruction of test components
- Arbiter: calculation of final test results
  - E.g., threshold on the basis of test component verdicts

Example: U2TP Test Architecture





Source: OMG UTP

Packages (concept groups)

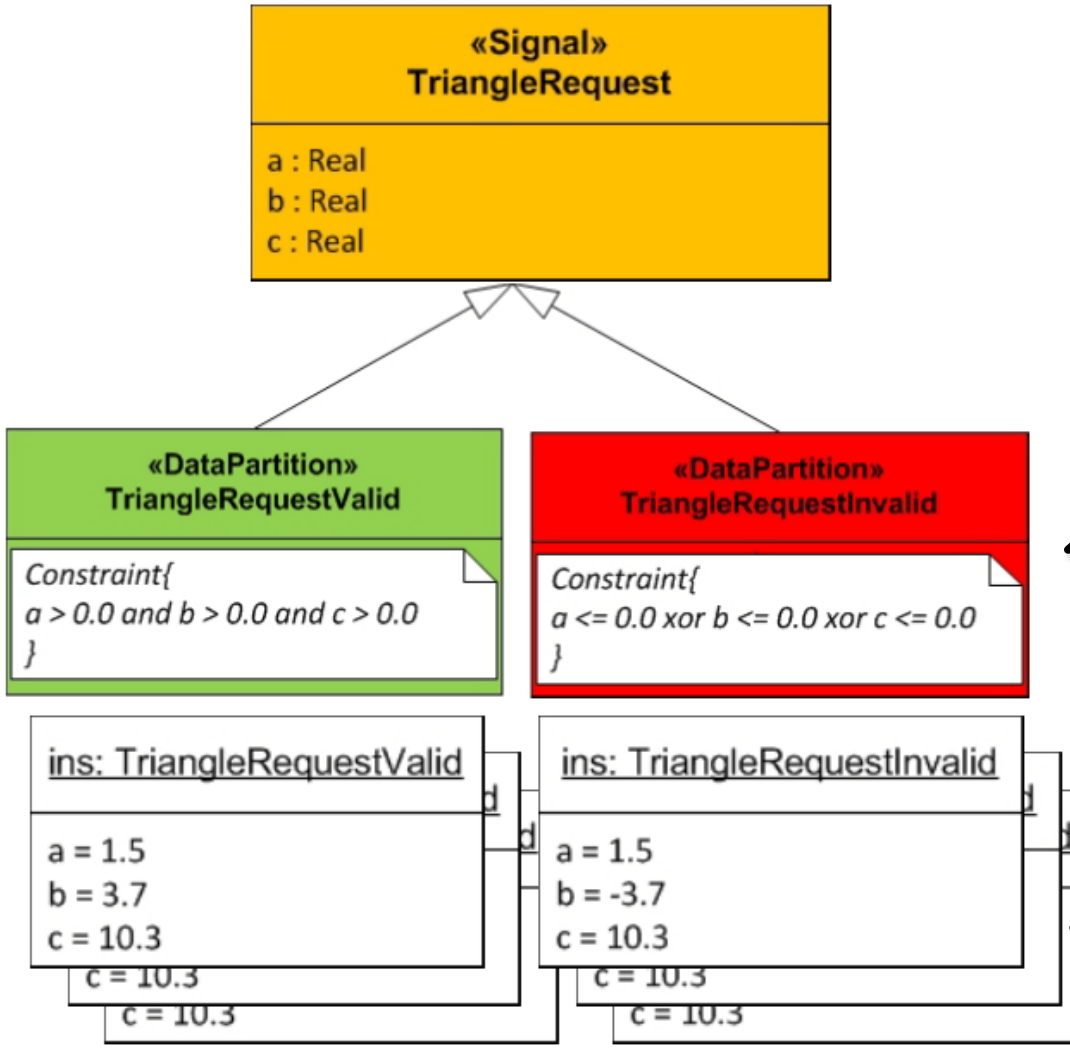
- Test Architecture
  - Elements and relationship involved in test
  - Importing the UML design model of the SUT
- **Test Data**
  - **Structures and values to be processed in a test**
- Test Behavior
  - Observations and activities during testing
- Time Concepts
  - Timer (start, stop, read, timeout), TimeZone (synchronized)

U2TP Test Data package

Identification of types and values for test (sent and received data)

- Test Parameter (Stimulus and observation)
- Abstract test data
  - Wildcards (* or ?)
  - Data Partition: Equivalence class for a given type
- Concrete test data
  - Instances with concrete values
  - Data Selector: Retrieving data out of a data pool

Example: U2TP Test Data



Equivalence classes (abstract)

Concrete values usable in tests

Source: UML Testing Profile Tutorial

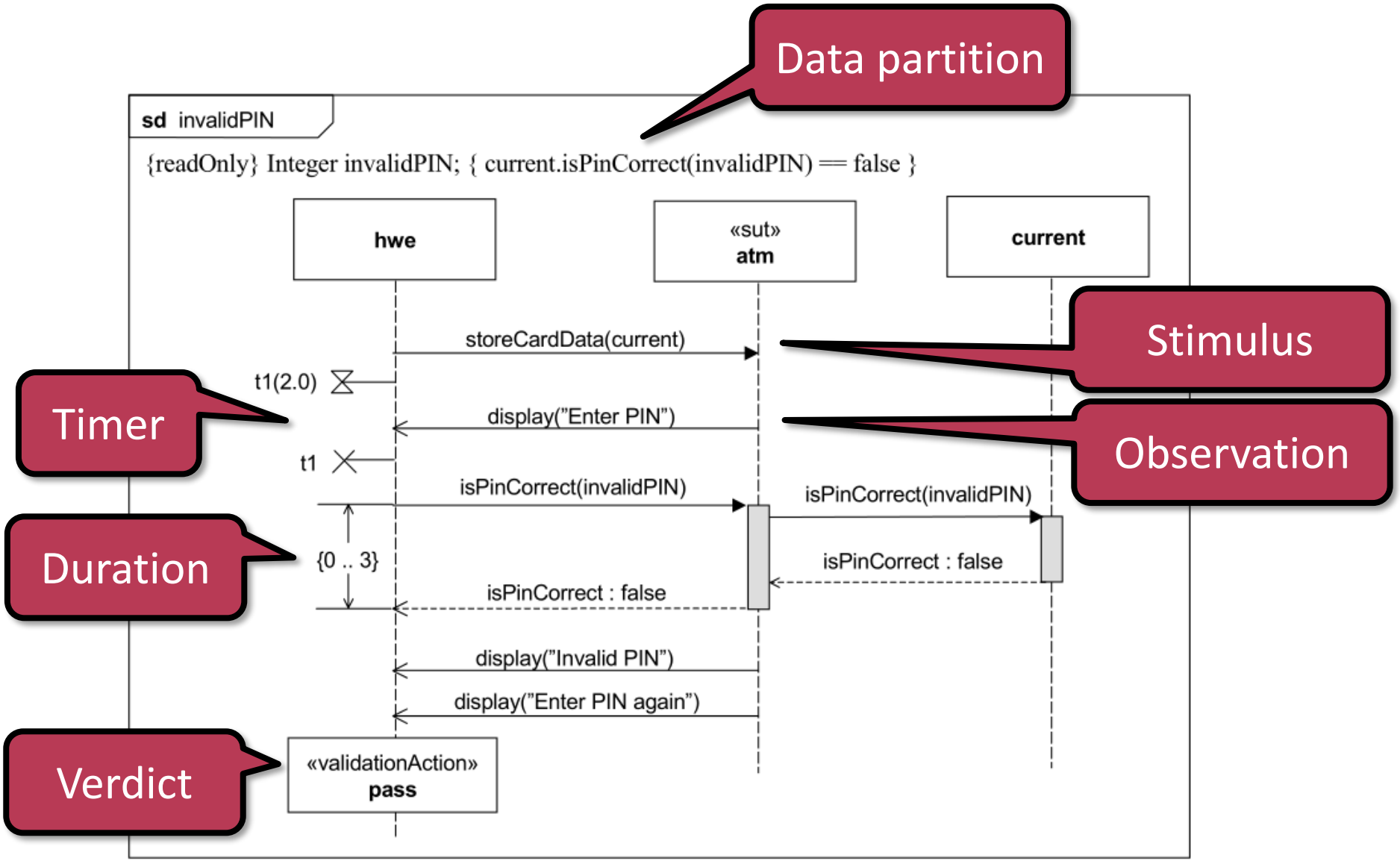
Packages (concept groups)

- Test Architecture
  - Elements and relationship involved in test
  - Importing the UML design model of the SUT
- Test Data
  - Structures and values to be processed in a test
- **Test Behavior**
  - **Observations and activities during testing**
- Time Concepts
  - Timer (start, stop, read, timeout), TimeZone (synchronized)

U2TP Test Behavior package

- Specification of default/expected behavior
- Identification of behavioral elements:
  - Test Stimulus: test data sent to SUT
  - Test Observation: reactions from the SUT
  - Verdict: pass, fail, error, inconclusive values
  - Actions: Validation Action (inform Arbiter), Log Action
- Test Case: Specifies one case to test the SUT
  - Test Objective: named element
  - Test Trace: result of test execution
    - Messages exchanged
  - Verdict

Example: U2TP Test Behavior



Source: OMG UTP

Summary of U2TP concepts

Test Architecture	Test Behavior	Test Data	Time
SUT	Test objective	Wildcards	Timer
Test components	Test case	Logical partition	Time zone
Test suite	Defaults	Coding rules	
Test configuration	Verdicts		
Test control	Validation action		
Arbiter	Test trace		
Utility part	Log action		

Recommended method for using U2TP

1. Define a new package for tests
2. Use interfaces and data types from design model
3. Define test objectives and focus of test
4. Test architecture
    1. Assign SUT to tested component/system
    2. Define test components
    3. Specify test configurations (instances)
5. Test behavior
    1. Design test cases (manually)
    2. Specify defaults and test data

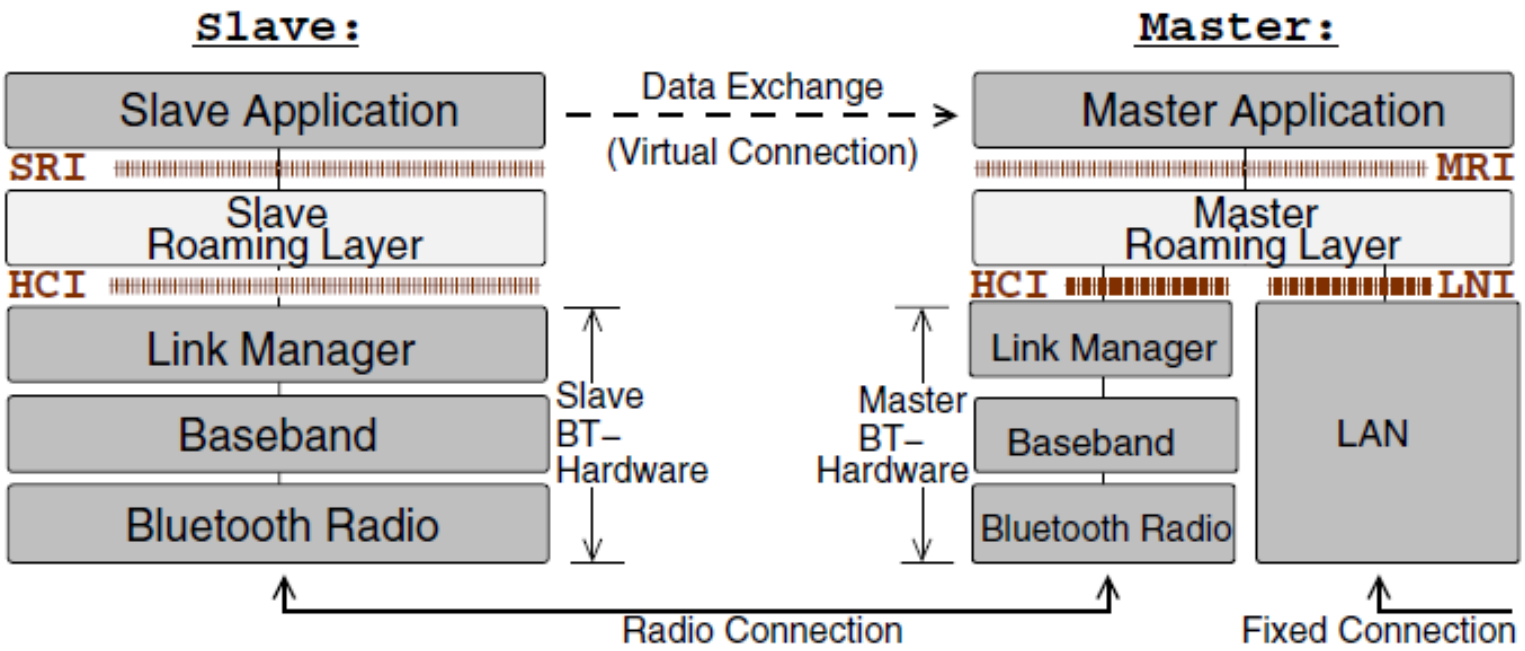
Case study: U2TP Test models for Bluetooth roaming

Source: Zhen Ru Dai et al. “From Design to Test with UML: Applied to a Roaming Algorithm for Bluetooth Devices”, TestCom 2004, pp 22-49

About the case study

- Bluetooth: short-range wireless communication
- Standard: HW (radio, baseband) + SW (protocol)
- Roaming algorithm:
  - Master devices connected to LAN
  - Slave devices move, may loose connection to master
  - Roaming:
    - Check periodically the quality of link to master
    - Select a new master if necessary

Components and protocol stack

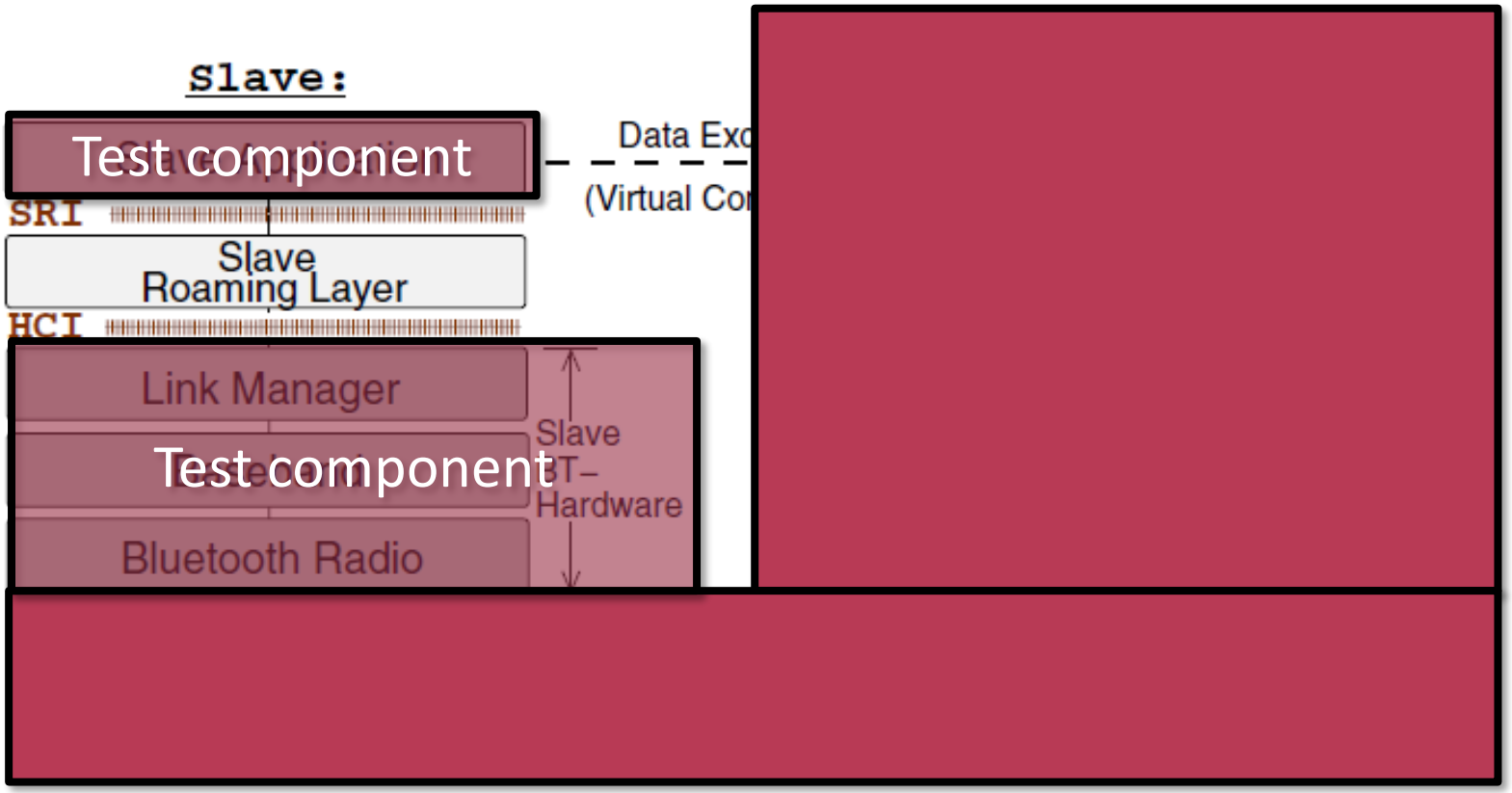


Test objective:

- Slave Roaming Layer functionality
  - Monitoring link quality
  - Connecting to a different master

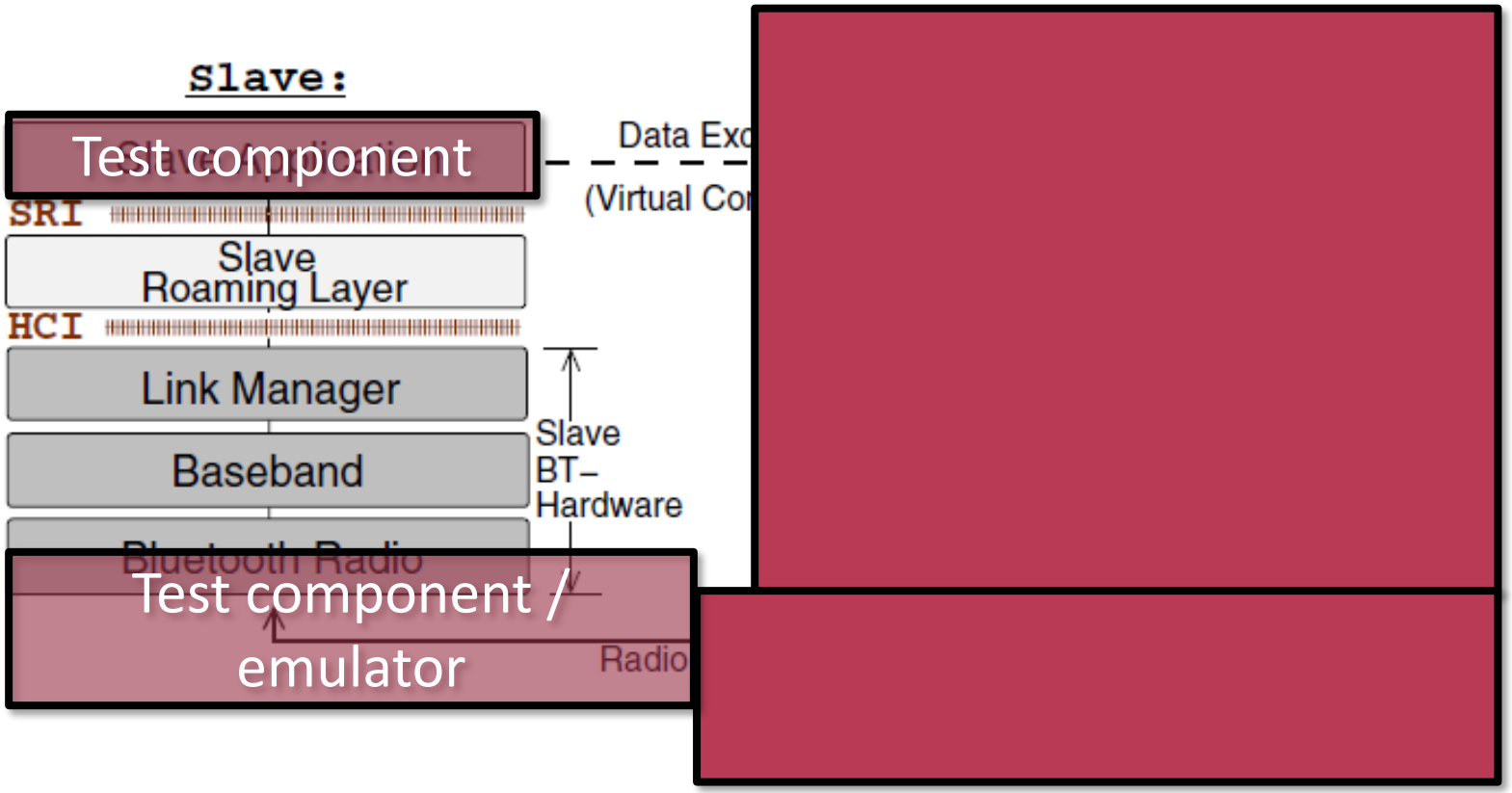
Possible test levels and setups (1)

Component/module test with software



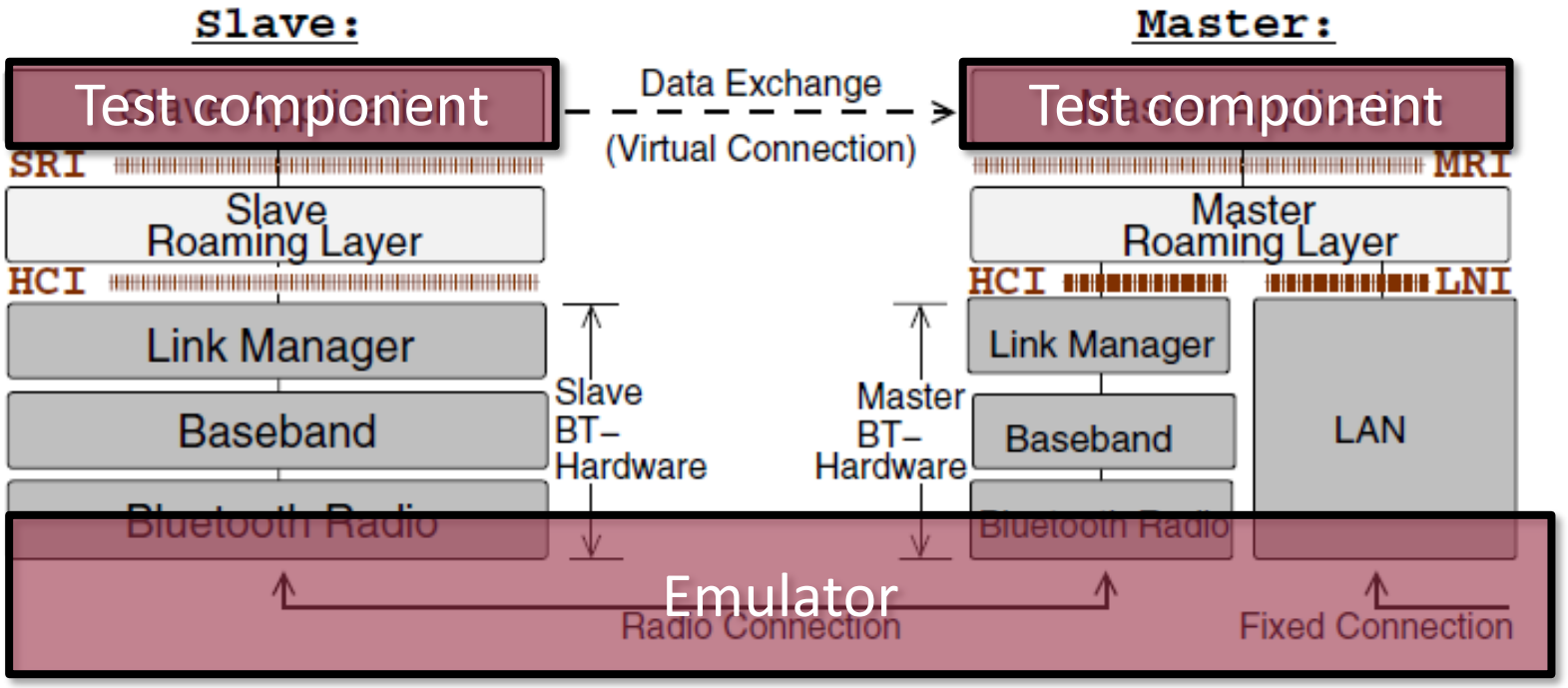
Possible test levels and setups (2)

Integration test with software



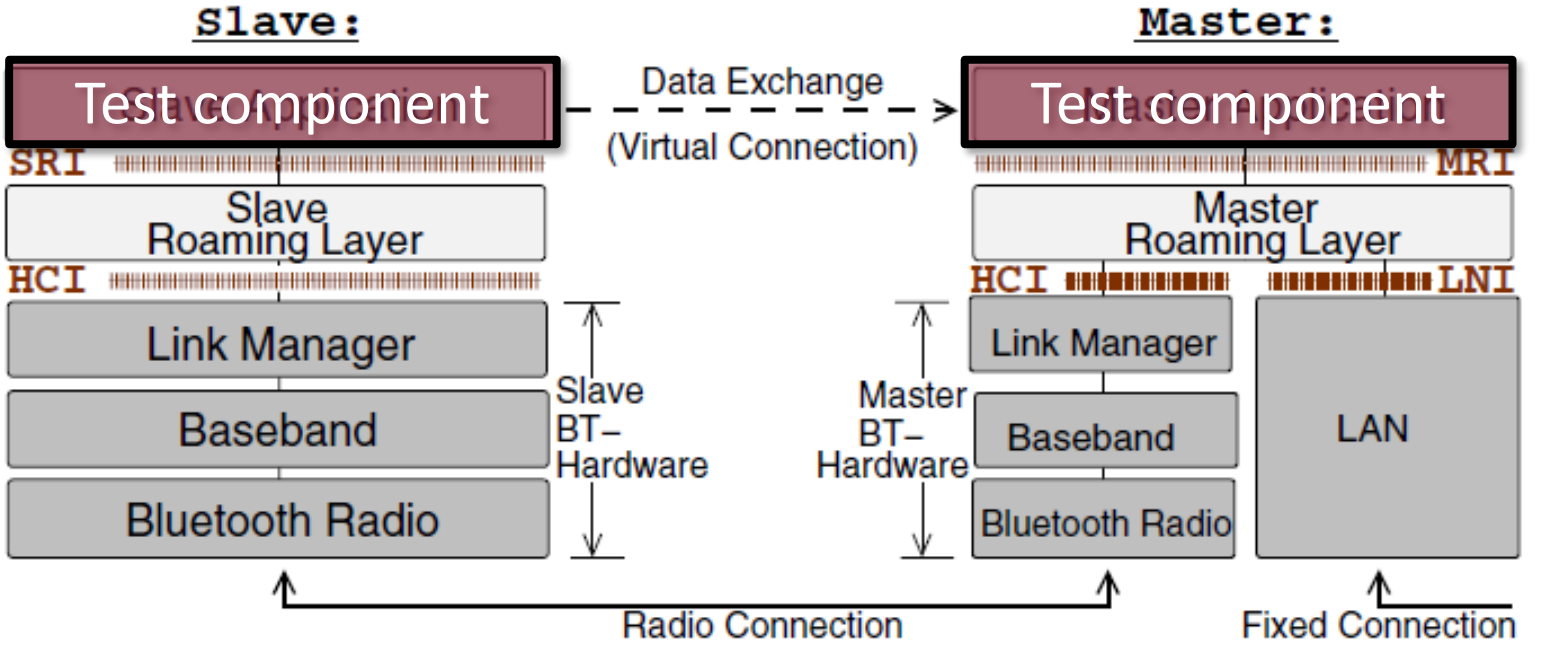
Possible test levels and setups (3)

Integration test with software



Possible test levels and setups (4)

System test with hardware



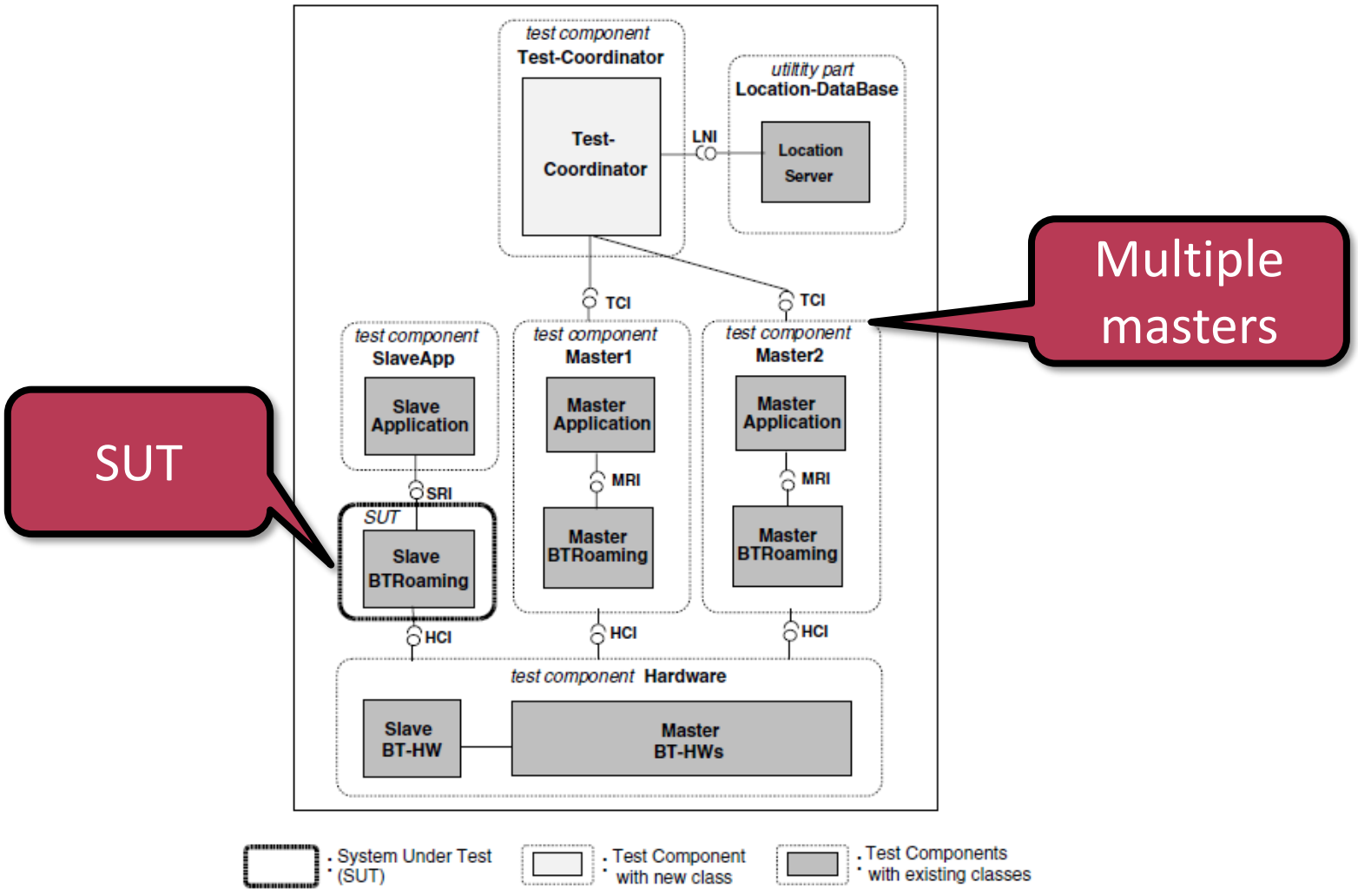
Moving physical devices or wireless test chamber…

Refining test objective

Slave Roaming Layer functionality

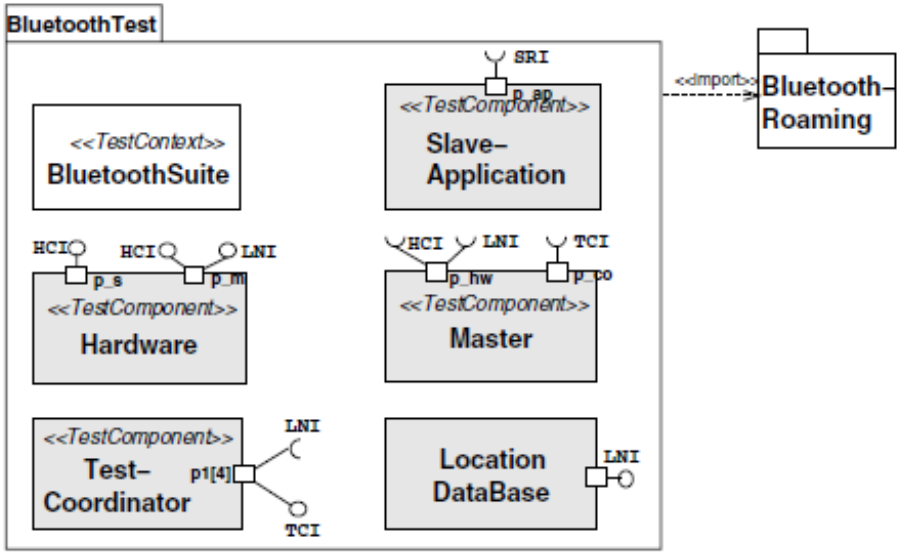
1. “Is the Slave Roaming layer able to choose a new master by looking up its roaming list when the connection with its current master gets weak?”
2. “Does the Slave Roaming layer request a connection establishment to the chosen master?”
3. “Does the Slave Roaming layer wait for a connection confirmation of the master when the connection has been established?”
4. “Does the Slave Roaming layer send a warning to the environment, when no master can be found and the roaming list is empty?”

Selected test configuration

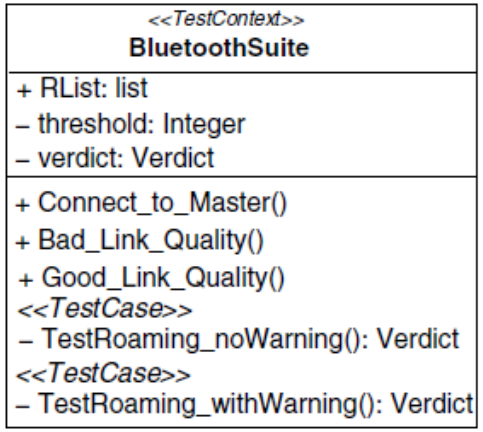


U2TP Test architecture: components

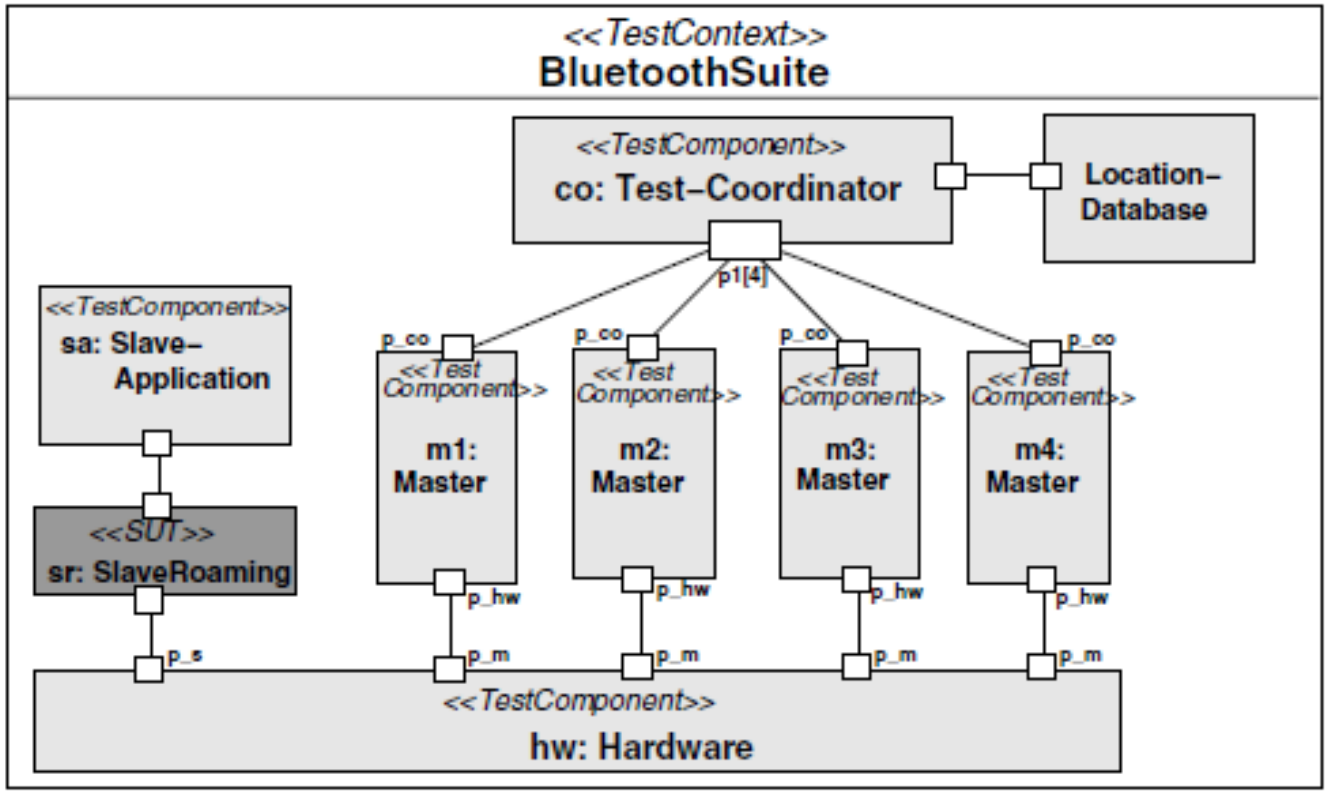
Test package



Test context



U2TP Test architecture: configuration



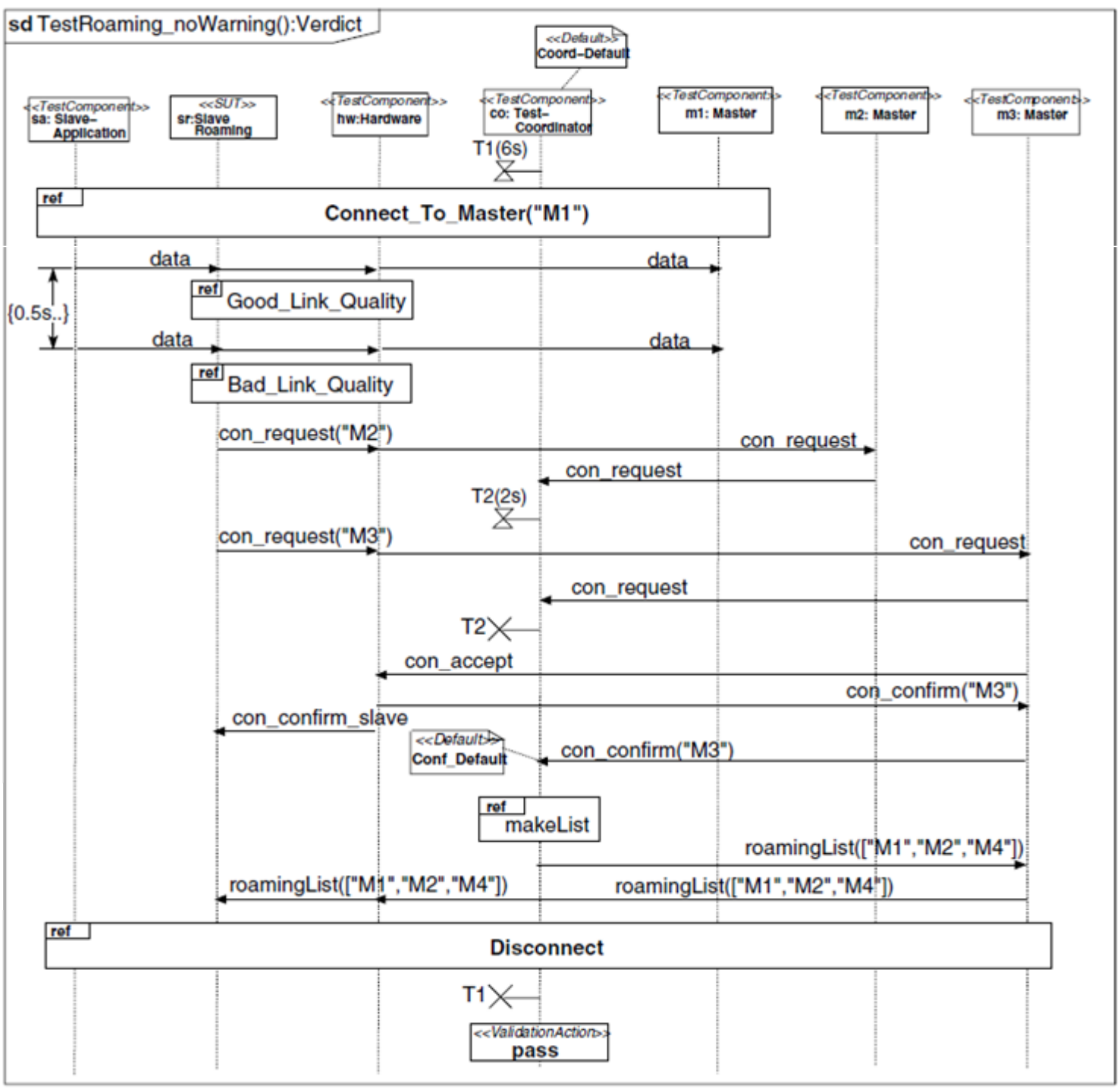
Test configuration

Test behavior

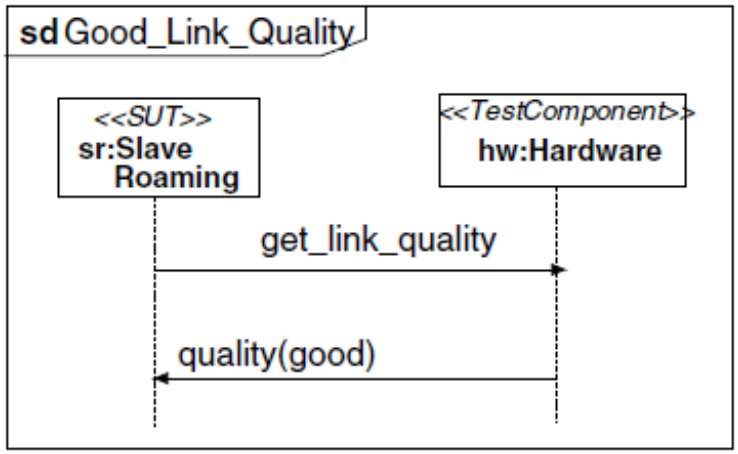
Selecting test scenarios for test objectives

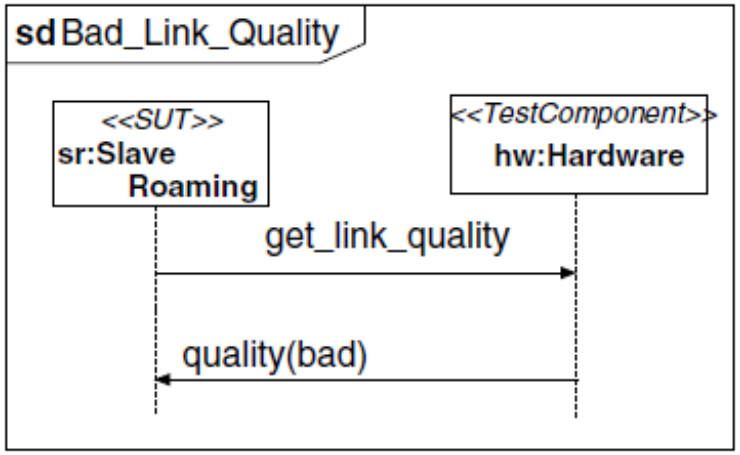
- Objective:
  - Choosing new master when the connection with its current master gets weak
- Scenario 1:
  - “After the exchange of two data packages, the link quality between Slave and its current master *m1* becomes bad. The first alternative master in the roaming list *m2* cannot be reached since the link quality is also weak. Thus, after at most two seconds, a further master *m3* is chosen from the roaming list and the connection is established successfully.”

Test scenario

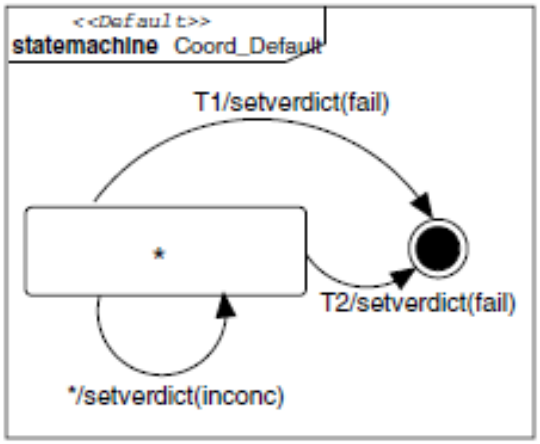


Test scenarios (details)





Sequence diagrams

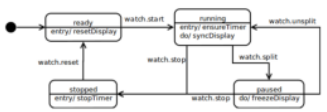


Default behaviors specified to catch the observations that lead to verdicts

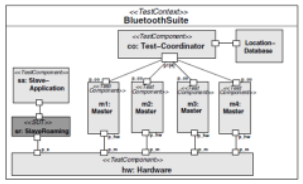
- Here: Processing timer events

Summary

Using models in testing (examples)



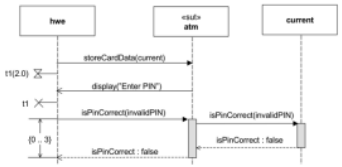
Behavior of SUT



Test configuration

```
timer t;
t.start(5.0);
alt {
 [] i.receive("coffee") {
  Count := Count+1; }
 [] t.timeout { }
}
```

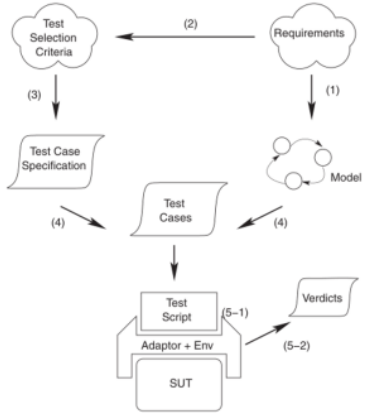
Test sequences



Test sequences

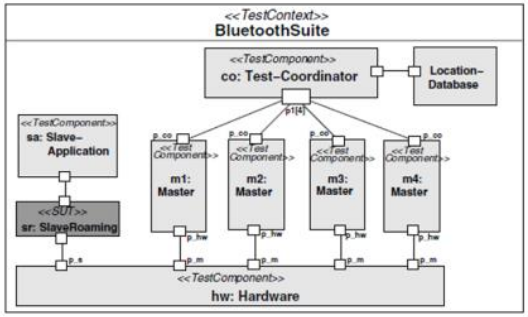
Source: OMG UTP

Typical MBT process



Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

U2TP Test architecture: configuration



Test configuration

Example: U2TP Test Behavior