The UML Testing Profile

- Tutorial at the ECOOP 2004 -

www.fokus.fraunhofer.de/u2tp

Ina Schieferdecker, Technical University Berlin/FOKUS
Øystein Haugen, University of Oslo

June 2004

Agenda

- Motivation
- Basics of Testing
- Overview of the UML Testing Profile
- Example Test Specifications
- The UML Profile and the MOF Model
- The Mappings
- Concluding Remarks
Agenda

- Motivation
  - Basics of Testing
  - Overview of the UML Testing Profile
  - Example Test Specifications
  - The UML Profile and the MOF Model
  - The Mappings
  - Concluding Remarks

The Problem

- Software
  - Increases in complexity, concurrency, and dynamics
  - Quality is key
    - Functionality
    - Performance
    - Scalability
    - Reliability
    - Usability
    - Efficiency
    - Maintainability
    - ...

- Testing is
  - Means to obtain objective quality metrics about systems in their target environment
  - Central means to relate requirements and specification to the real system
Testing Today

• Is
  ▪ Important
  ▪ Means to obtain approval
  ▪ Time critical

• But often
  ▪ Rarely practiced
  ▪ Unsystematic
  ▪ Performed by hand
  ▪ Error-prone
  ▪ Considered being destructive
  ▪ Uncool

  "If you are a bad programmer you might be a tester"

• Conjecture:
  There is a lack of appropriate test methods and techniques

Testing is …

• A technical process
• Performed by experimenting with a system
• In a controlled environment following a specified procedure
• With the intent of observing one or more characteristics of the system
• By demonstrating the deviation of the system’s actual status from the required status/specification.
Integrated Development and Testing

- Developer
- Integrator
- Systems Integrator

Testing throughout the process
Heterogeneity increases

Extreme View

- Developer
- Integrator
- Systems Integrator

Testing tight to Development
Testing tight to Specification

e.g. Java
e.g. TTCN-3
Balanced View

Developer

Integrator

Systems Integrator

Testing tight to Development
e.g. JUnit

Testing tight to Services
e.g. TTCN-3

An Answer: Model-Based View

Developer

Integrator

Systems Integrator

Testing tight to Development
e.g. JUnit

Testing tight to Specification
e.g. TTCN-3

use case diagrams

class diagrams

interactions

state machines
UML and Testing

- UML-based test generation, e.g.
  - Integration testing, Siemens, 2000
  - Component test framework (TINA), FOKUS, 2000
  - Component testing (COTE), INRIA, 2001

- UML-based test notation, e.g.
  - Agedis, EC IST project
  - TeLa, COTE project
  - UML Testing Profile, OMG

UML and Testing

- Model Driven Architecture as new OMG strategy

  - One objective of UML 2.0 is executable UML meaning
    - Code generation
    - Simulation
    - Validation
    - Test generation

  - "...the expanded role of the OMG must be built on rock-solid testing, certification and branding. ..."
Goals of the UML Testing Profile

- Definition of a testing profile to capture all information that would be needed by different test processes
  - To allow black-box testing (i.e. at UML interfaces) of computational models in UML
- A testing profile based upon UML 2.0
  - That enables the test definition and test generation based on structural (static) and behavioral (dynamic) aspects of UML models, and
  - That is capable of inter-operation with existing test technologies for black-box testing
- Define
  - Test purposes for computational UML models, which should be related to relevant system interfaces
  - Test components, test configurations and test system interfaces
  - Test cases in an implementation independent manner

U2TP Partners

- A consortium of testers, UML vendors and users dedicated to make UML applicable for software testing

- **Submitters**
  - Ericsson
  - IBM
  - FOKUS
  - Motorola
  - Rational
  - Softeam
  - Telelogic
  - University of Lübeck

- **Supporters**
  - iLogix
  - ScapaTechnologies
  - IRISA
Agenda

- Motivation
  - Basics of Testing
- Overview of the UML Testing Profile
- Example Test Specifications
- The UML Profile and the MOF Model
- The Mappings
- Concluding Remarks

Types of Testing

<table>
<thead>
<tr>
<th>Level</th>
<th>Accessibility</th>
<th>Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>white box</td>
<td>functionality</td>
</tr>
<tr>
<td>integration</td>
<td>grey box</td>
<td>load/performance</td>
</tr>
<tr>
<td>unit</td>
<td>black box</td>
<td>robustness</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>interoperability</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>usability</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Test Concepts: Black-Box Testing

Test Case

Stimulus

Response

Port

System Under Test (SUT)

• Assignment of a Test Verdict

Test Concepts: Local behavior

• Stimuli are given to the system under test (SUT) at well-defined interfaces
• Observations are taken and compared to the expected ones – the observations are validated and arbitrated
• The arbitration decides upon the further testing and/or the assignment of test results – the verdicts
• A detailed test procedure consisting of various stimuli and observations is called test case
• Several test cases constitute a test suite
• The relation between test cases and their execution is defined in a test control
• Typically, a test case describes a tree structure: one stimulus – several possible observations
• Alternatives and defaults are used for an efficient description of such trees
Test Concepts: Distributed Behavior

- Typically, distributed SUTs require distributed test setups.
- Such test setups are realized via test components which have to be coordinated and synchronized.
- All test components have a local verdict, which contributes to the overall verdict.
- The initial test configuration defines the test components used initially for a test case – the test configuration can change dynamically.
- Test components may use utility parts to use/access information outside the test system.

Test Concepts: Data

- Both, stimuli and observation are described as test data.
- Test data for stimuli and observations can be unspecific in their properties and values – data pools and data partitions are used.
- In addition, test data for observations can be unspecific in certain elements – wildcards are used.
- Coding rules describe the encoding/decoding used on interfaces to the SUT.
Agenda

- Motivation
- Basics of Testing
- Overview of the UML Testing Profile
- Example Test Specifications
- The UML Profile and the MOF Model
- The Mappings
- Concluding Remarks

The Testing Profile Roots

- Test control
- Wildcards
- Defaults
- Test components

- Arbiter
- Validation actions
- Data pools

Protocol Testing

TTCN-3

Software Testing

JUnit, TET, etc.

Graphical Format of TTCN-3

MSC-2000

UML 2.0

UML 1.x

SDL-2000

UML Testing Profile
1st Root: TTCN-3

- The new standardised test specification and test implementation language
  - Developed from 1999 – 2002 at the European Telecommunications Standards Institute (ETSI)
- Developed based on experiences from previous TTCN editions
  - Removal of OSI specific concepts; Improvement of concepts; Introduction of new concepts
- Applicable for all kinds of black-box testing for reactive and distributed systems, e.g.,
  - Telecom systems (ISDN, ATM, GSM, UMTS); Internet (IP, IP based protocols and applications); Software systems (Java, XML); Middleware platforms and component-based systems (CORBA, .Net, EJB)

```
: testcase myTestcase () runs on MTCType system TSIType
{  
  mydefault := activate (OtherwiseFail);
  verdict := pass;
  connect (PTC_ISAP1:CP_ISAP1
  mtc:CP_ISAP1);
  map (PTC_ISAP1:ISAP1, system:TSI_ISAP1);
  PTC_ISAP1.start(func_PTC_ISAP1());
  PTC_MSAP2.start(func_PTC_MSAP2());
  Synchronization();
  all component done;
  log("Correct Termination");
}
```
2nd Root: UML 2.0

- Developed by OMG (Object Management Group)
  - UML 2.0 Infrastructure RFP
    - metamodel restructuring in order for Core to be reusable by other OMG languages
  - UML 2.0 Superstructure RFP
    - new and improvement/extension of UML concepts
  - UML 2.0 OCL RFP
    - defining an OCL metamodel
  - UML 2.0 Diagram Interchange RFP
    - ensuring diagram interchange between different tools

UML 2.0 Improvements

- More unified conceptual base
  - Parts in Internal structure, Collaborations, Use cases and indirectly in Interactions
- More unified semantics
  - Higher precision
- Improved expressiveness
  - Structured Classes, Sequence Diagrams and Statemachines
  - Activities merged with actions
  - Collaborations aligned with structured classes
  - Patterns (templates) and frameworks support

- More powerful and expressive than UML 1.4
- Tighter and more consistent than UML 1.4
- Executable UML becomes possible
UML 2.0 Profiles

- Use of UML in
  - Analysis
  - Design/implementation
  - Directly executable notation (e.g., xUML)
  - Architecture description
  - Process engineering, workflow
  - Website structures
  - Data Modeling

- with obviously different (and inconsistent) semantics

- UML has many “semantic-free zones”, so called “semantic variation points”
  - E.g., detailed semantics of state machines, ...

> Profiles

- Specializations of UML by stereotypes, providing special semantics

<<TestContext>>

ATM

UML 2.0 Profile Walkthrough (1)

- Define profile(s)
  - based on reference metamodel
  - may use other packages for its definition

```
<<profile>>
U2TP

<reference/>
Java

<profile>
U2TP

<profile>
Java

<reference>
<import>
Meta model

<metaclass>
StructuredClassifier

<profile>
U2TP

<stereotype>
TestComponent

zone: TimeZone[0..1]
```

Extension
**UML 2.0 Profile Walkthrough (2)**

- Specify model
  - based on UML metamodel

![Diagram of BankTest class definition]

**UML 2.0 Profile Walkthrough (3)**

- Apply profile(s) to model
  - make it possible to apply stereotypes of the profile to the model elements

![Diagram with profile and apply stereotypes applied]
UML 2.0 Profile Walkthrough (4)

- Apply stereotypes to model elements as desired

Concepts of the Testing Profile

- Test architecture
  - Test structure, test components and test configuration

- Test data
  - Data and templates used in test procedures

- Test behavior
  - Dynamic aspects of test procedures

- Test time
  - Time quantified definition of test procedures
Test Architecture Realization

- System Under Test (SUT)
- Test components
- Test context with test configuration and test cases
- Test verdict arbitration with arbiter
- Test coordination with scheduler

Test Data Realization

- Individual coding rule definition
- Wildcards * and ?
- Concrete test data with data pool, data partition and data selector

Test Behavior Realization

- Test objectives
- Test cases
- Test verdicts: pass, fail, inconclusive
- Defaults behaviors on different levels
- Utility part

Test Time Realization

- Clock
- Timezone definition for synchronizing test components
- Timer operations
Concepts beyond **TTCN-3**

- Unification of test cases:
  - Test case as a composition of test cases
  - Test behavior defines the execution of a test case
- Separation of test behavior and verdict handling
  - Arbiter is a special component to evaluate the verdict
  - Validation actions are used to set the verdict
- Abstract test cases that work on data partitions rather than individual data
  - Data partitions to describe value ranges for observations and stimuli
- Test architecture with test deployment support
  - Part of the test specification is the definition of deployment requirements for a test case

Concepts beyond **UML**

- Defaults within test behavior
  - Concentration on main flow of test behavior
  - Default hierarchy to handle different concerns
- Wildcards within test data
  - Flexible definition of value sets
- Timers and time constraints
  - Time controlled test behavior
- Arbitration and verdicts
  - Assessment of test behavior
- Coding attributes
  - Encoding/decoding for data exchange with the SUT
Agenda

- Motivation
- Basics of Testing
- Overview of the UML Testing Profile
  - Example Test Specifications
- The UML Profile and the MOF Model
- The Mappings
- Concluding Remarks

The Example
UML Model of the System Level Architecture

System Integration Test
System Test
Unit Test

Unit-Level Test
Unit-Level Test

Money

IMoney
- fAmount : Integer
- fCurrency : String
- equals(m:Object): Boolean

Money
- fAmount : Integer
- fCurrency : String
- Money(a: Amount, c:String)
- add(m: Money): IMoney

MoneyBag
- fAmount : Integer
- fCurrency : String
- MoneyBag
- contains(m: IMoney): Boolean
- add(m: IMoney): IMoney

Classes to be tested

A Unit-Level Test Suite

"Test" package

MoneyUnitTest

Unit-level test suite

Money

Unit-level test cases

«TestContext»
MoneyTest

«<import>»

addSameMoney():Verdict
addDifferentMoney():Verdict
A Test Case

- Test suite object performing the test case
- Class instances of the SUT
- Return of test verdict

System-Level Test

- ATM
- HWControl
- Bank
- Money
- SWIFTNetwork

- `sd addSameMoney():Verdict`
- `self`
- `add(money2)`
- `get: money2`
- `return pass`
System Level Test

System-level test suite with public and private test cases

Test component

Miscellaneous classes

Test Configuration

SUT part

Coding rule part

Test component parts

Utility part

ECOOP, Oslo, June 2004 © U2TP Consortium
Test Control (execution of test suite)

- Referring test case behaviors

A Test Case

- Data partition
- Setting a timer
- Stopping a timer
- Duration constraint
- Setting arbitrated verdict

Stimulus
- Observation
- Function return value
A Test Case with Default (extracts)

```
sd validWiring

hwe

<Sup> atm

atmPort

bankCom

amount = acceptMoney

display("Transaction Accepted")

selectOperation : true

default 

DisplayDefault

<validationAction>

pass

sd validWiring

validWiring

<Sup> atm

atmPort

bankCom

amount = acceptMoney

display("Transaction Accepted")

selectOperation : true

default 

DisplayDefault

<validationAction>

pass
```

Defaults

Defining an event-specific default

```
sd DisplayDefault

self

alt

display(*)

? - pinOk : Boolean

-enteredPIN : String

-message : String

-t1 : Timer

«testComponent»

HWEmulator

default

HWEmulator::hweDefault

IHardware

? - pinOk : Boolean

-enteredPIN : String

-message : String

-t1 : Timer
```

Applying a component-specific default

```
HWEmulator

default

HWEmulator::hweDefault

IHardware
```

Wildcards

```
? - pinOk : Boolean

-enteredPIN : String

-message : String

-t1 : Timer

«testComponent»

HWEmulator

default

HWEmulator::hweDefault

IHardware
```

Defining a component-specific default

```
HWEmulator

default

HWEmulator::hweDefault

IHardware
```
System-Integration-Level Test

ATM

HWControl

Bank

Money

SWIFTNetwork

System-Integration-Level Tests

SWIFTSuite

SWIFTTest

SWIFTNetwork

SwiftNetwork

Bank

ATM

TestData

default

TransactionController::tcDefault

IHardware

Hardware

IAccount

ACTM

IAccount

completed()

TestData

User-defined

arbiter
**Test Configuration**

![Diagram of Test Configuration](image)

**Test Data**

![Diagram of Test Data](image)
Load Test

```
loadTest(int maxUsers, float p)
```

```
loop
  Wiring(dp)
    im.T1(testduration)
    im.T2(dp.getDistributionInterval())
    im.T2
    Wiring(dp)
    im.T1
    im.T2(dp.getDistributionInterval())

  validationAction pass
```

Arbitration

Load Test Contd.

```
Wiring(DataPool p)
```

```
<<create>>
runUSTrxn(p.getUSTrxnData())
runEUTrxn(p.getEUTrxnData())
```

ValidationAction pass
User-Defined Arbiter

```plaintext
numPassed = 0  
numOther = 0

Running

setverdict(result)

[result != pass] / numOther++

[result == pass] / numPassed++

completed

(numPassed > pc) >> pc

(numPassed > numOthers) > pc

=validationAction

pass

=validationAction

fail

Final test
result

Threshold calculation
```

Agenda

- Motivation
- Basics of Testing
- Overview of the UML Testing Profile
- Example Test Specifications
  - The UML Profile and the MOF Model
- The Mappings
- Concluding Remarks
The U2TP Profile and Standalone Metamodel

- Organization of concepts
  - Test Architecture
  - Test Behavior
  - Test Data
  - Time

- The UML-based profile
  - Used by UML tools to provide test specification via profiled UML elements.

- The MOF-based standalone metamodel
  - Used by MOF-based tools and repositories to manage and manipulate artifacts created using the profile.

The Profile – Excerpt of Test Architecture

Structured classifier for test components and test suites

Predefined interface

A characteristics of properties, typically in a test configuration
The Profile – Excerpt of Test Behaviour

- **Dependency** (from Dependencies)
- **Operation** (from Collaborations)
- **Behavior** (from BasicBehaviors)
- **TestCase**
- **TestObjective**
- **TestLog**
- **Default**
- **DefaultApplication**
- **Behavior** (from BasicBehaviors)

- **Could be anything and to be attached to a test suite/test case.**
- **To be defined as a operation or behavior.**
- **Just behaviors**
- **To be applied to Package, Classifier, Behavior, Interaction Fragment, State, Region or Activity.**

The Profile – Excerpt of Test Data

- **Classifier** (from Kernel)
- **Operation** (from Kernel)
- **Property** (from Kernel)
- **Classifier** (from Kernel)
- **DataPool**
- **DataPartition**
- **DataSelector**

- **Data collections**
- **Partitioning of Data**
- **Access to Data**
MOF Walkthrough

- Meta Object Facility
- Foundation for the OMG metadata architecture
  - Design and implement metamodels and models
    - Use UML classes for specification
    - Discover and manipulate metadata
    - Find and manage metadata repositories
  - Provides reflective meta-object interfaces for introspection and update of distributed metadata
  - Provides a MOF-to-IDL mapping to automate generation of concrete object interfaces for specific metamodels
  - Provides a MOF-to-XML mapping to automate generation of XML schema and documents

The MOF Metamodel
**Agenda**

- Motivation
- Basics of Testing
- Overview of the UML Testing Profile
- Example Test Specifications
- The UML Profile and the MOF Model
  - The Mappings
- Concluding Remarks

**Mappings**

- To enable the direct execution of U2TP specifications
- To reuse existing test infrastructures

  - Mappings to
    - The JUnit test framework
      - An open source test technology for Java
      - Black-box tests on unit level
    - The Testing and Test Control Notation, TTCN-3
      - A generic test technology by ETSI/ITU-T
      - Black-box/grey-box tests on unit, component, integration and system level
Mappings

The mappings define possible, but not the only mappings.

Principles of Mapping to JUnit

<table>
<thead>
<tr>
<th>Direct mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test control</td>
</tr>
<tr>
<td>Test case</td>
</tr>
<tr>
<td>Test objective</td>
</tr>
<tr>
<td>Verdict</td>
</tr>
<tr>
<td>Arbiter</td>
</tr>
<tr>
<td>Validation action</td>
</tr>
<tr>
<td>Data pool</td>
</tr>
</tbody>
</table>
### Principles of Mapping to JUnit

<table>
<thead>
<tr>
<th>Implicit mapping</th>
<th>as part of the test methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus, observation, coordination</td>
<td></td>
</tr>
<tr>
<td>SUT and utility part</td>
<td>any accessible Java class</td>
</tr>
<tr>
<td><strong>Not supported</strong></td>
<td></td>
</tr>
<tr>
<td>Test configuration and test components</td>
<td></td>
</tr>
<tr>
<td>Wildcards and matching</td>
<td></td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
</tr>
<tr>
<td>Test trace and log</td>
<td></td>
</tr>
<tr>
<td>Time concepts</td>
<td></td>
</tr>
<tr>
<td>Coding rules</td>
<td></td>
</tr>
</tbody>
</table>

#### Example for Mapping to JUnit

```java
public class MoneyTest
    extends TestCase {
    public void addSameMoney() {
        Money money1 = new Money(20, "USD");
        Money money2 = new Money(50, "USD");
        money1.add(money2);
        assertTrue(money1.equals(new Money(70, "USD")));
    }
    ...
}
```
## Direct mapping

<table>
<thead>
<tr>
<th>Category</th>
<th>TTCN-3 Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test control</strong></td>
<td>control of a test module</td>
</tr>
<tr>
<td><strong>Test case</strong></td>
<td>testcase</td>
</tr>
<tr>
<td><strong>Stimulus, observation, coordination</strong></td>
<td>communication to SUT and to test components</td>
</tr>
<tr>
<td><strong>Defaults</strong></td>
<td>altsteps and activate/deactivate operations</td>
</tr>
<tr>
<td><strong>Wildcards and matching</strong></td>
<td>template and pattern mechanisms</td>
</tr>
<tr>
<td><strong>Test components</strong></td>
<td>component types and instances</td>
</tr>
<tr>
<td><strong>Test configuration</strong></td>
<td>create, connect, map ... operations</td>
</tr>
</tbody>
</table>

## Direct mapping

<table>
<thead>
<tr>
<th>Category</th>
<th>TTCN-3 Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arbiter</strong></td>
<td>user defined verdict handling (and user defined verdict type)</td>
</tr>
<tr>
<td><strong>Validation action</strong></td>
<td>(external) data function</td>
</tr>
<tr>
<td><strong>Utility part and data pool</strong></td>
<td>(external) constants or module parameters</td>
</tr>
<tr>
<td><strong>Coding rule</strong></td>
<td>encoding attribute</td>
</tr>
<tr>
<td><strong>Timer</strong></td>
<td>timer</td>
</tr>
<tr>
<td><strong>Logging</strong></td>
<td>log operation</td>
</tr>
</tbody>
</table>
Principles of Mapping to TTCN-3

### Implicit mapping

<table>
<thead>
<tr>
<th>Test objective</th>
<th>comment for a test case in a special attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUT</td>
<td>characterized via the test system interface only</td>
</tr>
</tbody>
</table>

### Not supported

- Timezone
- Test trace

---

Example for Mapping to TTCN-3

```plaintext
....
type port hwCom_PType
procedure {...}
....
type component HWEmulator_CType{
  port atmPort_PType hwCom;
  var boolean pinOk;
  var charstring enteredPIN;
  var charstring message_; 
  timer t1;
}
```
Example for Mapping to TTCN-3

```tcl
function invalidPIN_hwe ... {
  ...
  hwCom.call(
    storeCardData:{current},nowait);
  t1.start(2.0);
  hwCom.getreply(
    display_:{"Enter PIN");
  t1.stop;
  hwCom.call(
    isPinCorrect:{invalidPIN},3.0) {
    [] hwCom.getreply(
      isPinCorrect:{?} value false) {}
  } hwCom.getreply(
    display_:{"Invalid PIN");
  hwCom.getreply(
    display_:{"Enter PIN again"});
  setverdict(pass); }
```

 Agenda

- Motivation
- Basics of Testing
- Overview of the UML Testing Profile
- Example Test Specifications
- The UML Profile and the MOF Model
- The Mappings
  - Concluding Remarks
Implementations under Development

- Eclipse Project Hyades on an Open Source Trace and Test Framework
  - The test part is based on the U2TP standalone metamodel
- Telelogic Tau G2
  - The test part is based on the U2TP profile
- Microsoft Visual Studio
- ITEA Project on Advanced Test Methods and Tools TTmedal

Compliance Points

1. UML Profile for Testing: a compliant implementation supports the UML profiling mechanism, the UML entities extended by the Testing Profile, and the stereotyped entities of the UML Testing Profile.

2. MOF-based Metamodel for Testing: the compliant implementation supports all of the entities in the MOF-based metamodel.

3. Notation: If graphical notation is used, the compliant implementation recognizably supports the notation defined by the Testing Profile specification.

4. XMI/DTD: An XMI compliant implementation of the Testing Profile and/or MOF metamodel provides the UML XMI exchange mechanism.

5. Static Requirements: The compliant implementation checks the specified constraints automatically.

- Compliance requires meeting 1 and/or 2. Points 3-5 are optional and can be claimed in any combination.
At the End: a Standardized Testing Profile

- One test notation for many testing applications
- Universally understood syntax and operational semantics
- Off-the-shelf tools
- Cheaper education and training costs
- Exchange and reuse of test suites
- Easier maintenance of test suites

- Transparency for the test process, increase of the objectiveness of tests and comparability of test results
- Direct support for test design
- Integration in the system development process

Summary

- UML Testing Profile provides specification means for test artifacts of systems from various domains
- Enhances UML with concepts like test configuration, test components, SUT, verdict and default
- Seamlessly integrates into UML: being based on UML metamodel, using UML syntax

- Adopted at OMG Technical Meeting, Paris, June 2003

http://www.fokus.fraunhofer.de/u2tp/
Thank you for your attention!

Any further questions?