Integrated System and Test Development –
The UML 2.0 Testing Profile

Ina Schieferdecker
Technical University Berlin/FOKUS
Outline

• Introduction
• The Testing Profile Concepts
• An Example
• Making Use of Testing Profile Specifications
Motivation

- Integrated Development and Testing

Early and continuous consideration of test aspects
Motivation

• 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. …“
Test Artefacts in UML

• Possible
  • Internal structure for test configuration
  • Class diagrams for test structuring
  • sequence diagrams for test purpose definition

• However, test case specifications ?
  i.e. test specific concepts which have evolved over time
  • Points of control and/or observation and coordination points
  • Test components and SUT
  • Default behaviors
  • Verdict handling

➢ UML is not yet ready for testing
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
The Testing Profile Roots

- Test control
-Defaults
-Test components

- Arbiter
-Data pools

UML 2.0

MSC-2000

UML 1.x

SDL-2000

Protocol Testing like TTCN-3

Software Testing like JUnit, TET, etc.
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.set(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");
}
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 which can use a set of stimulus 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
2nd Root: UML 2.0

  - 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

<<TestSuite>>
ATM
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
Concepts of the Testing Profile

- **Test architecture**
  - Test structure, test components and test configuration
- **Test data**
  - Test data and templates used in test procedures
- **Test behavior**
  - Dynamic aspects of test procedures
- **Test time**
  - Time quantified definition of test procedures
## Testing Profile Concepts

<table>
<thead>
<tr>
<th>Architecture concepts</th>
<th>Behavior concepts</th>
<th>Data concepts</th>
<th>Time concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUT</td>
<td>Test objective</td>
<td>Wildcards</td>
<td>Timer</td>
</tr>
<tr>
<td>Test components</td>
<td>Test case</td>
<td>Data pools</td>
<td>Time zone</td>
</tr>
<tr>
<td>Test context</td>
<td>Defaults</td>
<td>Data partitions</td>
<td></td>
</tr>
<tr>
<td>Test configuration</td>
<td>Verdicts</td>
<td>Data selectors</td>
<td></td>
</tr>
<tr>
<td>Test control</td>
<td></td>
<td>Coding rules</td>
<td></td>
</tr>
<tr>
<td>Arbiter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility part</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example

- System-integration level tests
  - US Bank
  - SWIFTBureau
  - SWIFTNet
  - EU Bank

- System-level tests
  - SWIFTBureau
  - US Bank Network
  - EU Bank Network

- Unit-level tests
  - SWIFTBureau
  - US Bank SSSB Client
  - EU Bank SSSB Client

OTC Market Makers
Clearing Company
The Example Packages

System Test

ATM

HWControl

Bank

Money

SWIFTNetwork

«import»

«import»

«import»

«import»

«import»
System Level Test

"Test package" test suite with test cases

- import

Test component

ATMTest

«testContext»

ATMSuite
- verdict : Verdict
- amount : IMoney
- targetBank : SwiftId
- targetAccount : String
- sourceAccount : String

«testCase» + validWiring() : Verdict
«testCase» + invalidPIN() : Verdict
«testCase» - authorizeCard() : Verdict

Test component

«testComponent»

BankEmulator
- pinOk : Boolean
- enteredPIN : String
- message : String
- t1 : Timer

«interface»

IAccount

Miscellaneous
Test Configuration

- «testContext»
  - ATMSuite

- SUT property
  - «sut»
    - atm : BankATM

- Coding rules
  - Utility property
    - be : BankEmulator
    - current : CardData

- Test component property
  - hwe : HWEmulator

- Connections
  - atmPort
  - bankCom
  - validWiring() : Verdict
  - invalidPIN() : Verdict
  - authorizeCard() : Verdict
Test Control

```
{
  testContext
  ATMSuite
  -verdict : Verdict
  -amount : IMoney
  -targetBank : SwiftId
  -targetAccount : String
  -sourceAccount : String

  testCase +validWiring() : Verdict
  testCase +invalidPIN() : Verdict
  testCase -authorizeCard() : Verdict

  sd ATMSuite

  Referring test case behaviors

  ref
  verdict = invalidPIN

  [verdict == pass]
  ref
  verdict = validWiring

  [verdict == fail]
```
A Test Case

```
{readOnly} Integer invalidPIN; { current.isPinCorrect(invalidPIN) == false }

isPinCorrect(invalidPIN)
{0 .. 3}

isPinCorrect : false

display("Invalid PIN")

display("Enter PIN again")

isPinCorrect : false

display("Enter PIN")

storeCardData(current)

t1(2.0)
```

### SUT and Test Component Lifelines

**Stimulus**

**Observation**

**Timing**

**Arbitrated verdict**
A Test Case with Default (Extract)

The diagram illustrates a test case for a financial application involving interaction between a machine (hwe) and an ATM (atm). The scenario involves a transaction being accepted, with default actions being executed.

- **sd validWiring**: Indicates the valid wiring scenario for the test.
- **hwe**: Represents the machine entity involved in the transaction.
- **atm**: Represents the ATM entity.
- **default DisplayDefault**: Indicates the default action for the transaction.
- **amount = acceptMoney**: The amount to be accepted.
- **display("Transaction Accepted")**: The action performed upon transaction acceptance.
- **selectOperation : true**: Indicates the selection of the operation.
- **default DisplayDefault**: Represents the default display action.
- **default DisplayDefault**: Represents the default display action.
- **default DisplayDefault**: Represents the default display action.
- **<validationAction> pass**: Represents the successful validation action.

The diagram also shows the flow of actions and interactions, with arrows indicating the direction and nature of these interactions.
Defaults

Applying a component-specific default

Defining an event-specific default

self

sd DisplayDefault

alt

display(*)

«validationAction»

inconc

«validationAction»

fail

If

hweDefault

t1 / setverdict(fail);

ejectCard / setverdict(fail);

display(msg) / if (msg == Connection lost) then setverdict(inconc); else setverdict(fail);

«testComponent»

HWEmulator

-pinOk : Boolean
-enteredPIN : String
-message : String
-t1 : Timer

hweDefault 

HWEmulator::hweDefault

DisplayDefault
The Mappings

• To enable the direct execution of U2TP specifications by reusing existing test infrastructures

• Mappings to
  • The JUnit test framework
    • An open source test technology for Java
    • Black-box tests on unit level
    ➢ Only selected concepts of U2TP can be mapped
  • 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
    ➢ Almost all concepts can be mapped
The mappings define possible, but not the only mappings.
Example for Mapping to TTCN-3

... type port hwCom_PType

procedure {...}

... type component

HWEmulator_CType{

port atmPort_PType hwCom;
var boolean pinOk;
var charstring enteredPIN;
var charstring message_;
timer t1;
}

ATMTest

«testComponent »

HWEmulator

pinOk : Boolean
enteredPIN : String
message : String
t : Timer
Example for Mapping to TTCN-3

```tTCN
function invalidPIN_hwe ... {
    hwe
    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) {}
        display_:{"Invalid PIN"},
        display_:{"Enter PIN again"})
    hwCom.getreply(
        display_:{"Invalid PIN"},
        display_:{"Enter PIN again"})
    setverdict(pass); }
```

```tTCN
sd invalidPIN
{readOnly} Integer invalidPIN; { current
hwe
storeCardData(current)
display("Enter PIN")
isPinCorrect(invalidPIN)
{0 .. 3}
isPinCorrect : false
display("Invalid PIN")
display("Enter PIN again")
«validationAction»
pass
```
Deriving Tests

• From Design to Test with UML:
  • A methodology to derive tests from UML system designs
  • By Zhen Ru Dai, Jens Grabowski, Helmut Neukirchen, and Holger Pals
# A Methodology using the Testing Profile

<table>
<thead>
<tr>
<th>Test Architecture Concepts</th>
<th>Test Behavior Concepts</th>
<th>Test Data Concepts</th>
<th>Time Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUT</td>
<td>Test objective</td>
<td>Wildcards</td>
<td>Timer</td>
</tr>
<tr>
<td>Test components</td>
<td>Test case</td>
<td>Data partition</td>
<td>Time zone</td>
</tr>
<tr>
<td>Test context</td>
<td>Defaults</td>
<td>Coding rules</td>
<td></td>
</tr>
<tr>
<td>Test configuration</td>
<td>Verdicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arbiter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility part</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mandatories**

**Optionals**

**Derivables**
Methodology Overview

• Test preparations
  • Define a test package with a test suite.
  • Determine SUT and import its implementation.
  • Determine test objectives.

• Test configuration
  • Assign roles: SUT, test components, utility part.
  • Attach stereotypes.

• Test cases
  • Group instances.
  • Attach stereotypes.
  • Define re-usable test fragments to functions.
  • Assign verdicts.
  • Specify timers.
  • Specify default behaviors
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

➢ Direct support for test design
➢ Integration with the system development process
➢ OMG standard since June 2004
Implementations under Development

• Eclipse Project **Hyades** on an Open Source Trace and Test Framework
  • The test part is based on the U2TP specification
• Telelogic **Tau G2**
• Artisan **Software Real-Time Studio**
• Microsoft **Visual Studio**
• ITEA Project on Advanced Test Methods and Tools
  **TTmedal**
Thank you for your attention!

→ www.fokus.fraunhofer.de/u2tp

Questions?