TTCN-3 Test Generation from System Models

Ina Schieferdecker
Technical University Berlin/Fraunhofer FOKUS
schieferdecker@fokus.fraunhofer.de

and the FOKUS team:
Maik Busch, Ranganai Chaparadza,
Zhen Ru Dai, Andreas Hoffmann,
Laurette Lacmene, Titus Ngwangwen,
Guy Collins Ndemb, Diana Serbanescu,
Justyna Zander-Nowicka

Contents

- Motivation
- The Basics: TTCN-3
- The Approach: MDA++
- Some more details
- Summary
Motivation

Integrated Development and Testing

- Early and continuous consideration of test aspects
Positioning

- Model-based, but practical work
- Not on-the-fly testing, but explicit test specifications
  1. tests can be generated along different methods and strategies and require typically selection/configuration and parameterization
  
  in order to make tests repeatable, a separate test specification is needed
  
  2. a test model developed independently from the system model is a good choice
  
  this test model should be based on common test concepts and by that reduce the efforts in developing/generating the tests

Contents

- Motivation
- The Basics: TTCN-3
- The Approach: MDA++
- Some more details
- Summary
The Test Technology TTCN-3

- Testing and Test Control Notation
- TTCN-3 is standardized by ETSI (ES 201 873 series) and the ITU-T (Z.140 series)
- Areas of Testing
  - Regression Testing
  - Conformance and Functionality Testing
  - Interoperability and Integration Testing
  - Load/ Stress Testing
- Applications
  - Mobile communications (GSM, GPRS, UMTS, TETRA)
  - Wireless LANs and MANs (Hiperlan, Hiperaccess), cordless phones (DECT)
  - Broadband technologies (B-ISDN, ATM)
  - Internet protocols (IPv6, SIP, Voice over IP)
  - Middleware platforms (CORBA, CCM, EJB, WebServices)
  - Smart Card Testing, MOST, CAN, Powertrain

Main Aspects of TTCN-3

- Central concepts
  - Configuration: Static and dynamic concurrent test configurations with test components
  - Communication: Various communication mechanisms (synchronous and asynchronous)
  - Control: Test case execution and selection mechanisms
  - Matching: Expected vs. received data
  - Parameterization: Flexibility along test campaigns
  - Modules: Structuring and reuse
- Extendibility via attributes, external function, external data
- Well-defined syntax, static and operational semantics
- Different presentation formats
The TTCN-3 Folklore

TTCN-3: Test Specification and Execution

B) Component Provider

- Development Process
  - Specification
  - Product/Component Specification
- Functionality Interop.Integr.
- Solution Deployment

C) Solution Provider

- Test Execution
  - Specification
  - Standardization
  - Product/Component Specification

A) Forum

- The testing middleware
  - unifying the documentation and definition of tests
  - unifying the tests in IT, Internet, Telco, and embedded systems (supporting their convergence)
  - unifying the test infrastructure
Our tool set – free for education

Contents

- Motivation
- The Basics: TTCN-3
- The Approach: MDA++
- Some more details
- Summary
Model Driven Architecture

- Focusing on model-based system development (system centric)
- Relations/Mappings
- Code Generation
- Specify Systems/Applications
- QVT Specify Transformations

Leveraging Testing into the Model-Based Paradigm

- Meta-meta-model
  - MOF Model
- Meta-model
  - UML Meta Model
  - Test Meta Model
- Models
  - UML Model
  - Test Suite
- Data
  - (Testable) System
A MOF-MM for TTCN-3

U2TP has been defined to capture all information that would be needed by different test processes within UML.

- It is 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
- It is an official OMG standard since summer 2005
Using MDA Concepts for Testing

Scenario 1:
PSM-based Test Derivation
Scenario 2:

PIM-based Test Derivation with Single PST

Scenario 3:

Manual PIM-based Test Derivation with Single PST
Detailed Approach:
Integrated System and Test Development

UML2 for System and Test Development
Contents

- Motivation
- The Basics: TTCN-3
- The Approach: MDA++
- Some more details
- Summary

System Model to Test Model Transformation
System Model to Test Model Transformation

- Based on the meta-models defined for both eUML and eTML, transformation rules are proposed that allow the automated generation of eTML test model skeletons out of EUML models.

- The transformation rules are defined in a model-driven way, i.e. the information contained in a source meta-model is transformed into a target meta-model.

- The advantage of that approach is that the correctness of model instances inside the source repository is ensured by the conformity to the corresponding meta-model and its invariants.

PIM to PIT Transformation

- Derivation of test models from system models
- Includes also test behaviour generation
- Includes generation of graphical information for new test model elements
Transformation Rules for Test Architecture

- All elements of the eUML model become part of the eTML model.

- Create an overall test package. **Import** the system model package into the test model package. Name the overall test package according to the rule “TestPackage_SystemName” with **SystemName** as the name of the system model.

Transformation Rules for Test Architecture

- Create a package inside the TestPackage_SystemName. Name this package according to the rule “SystemNameSuites” with **SystemName** as the name of the system model.

- Create a class stereotyped with “<TestContext>” inside the SystemNameSuites. Name this class according to the rule “SystemNameSuite_Nr” with **SystemName** as the name of the system model.
Transformation Rules for Test Architecture

- Derive test configuration in the form of composite diagram from the system configuration, specifically from the eUML collaboration diagram. Put it inside the previously created SystemNameSuite_Nr class.
- The SUT is derived both from a class or a part.
- Test components are derived both from a class or a part.

![Diagram showing test components and system components](image1)

Transformation Rules for Test Architecture

- Traverse the classes of test configuration assigning for each of them <<TestComponent>> or <<SUT>> stereotypes appropriately so that more composite structure diagrams can be obtained (if needed).
- Rotate the role of SUT for each class defined in the system configuration. Only testable classes may be assigned as SUT. Algorithm of assigning the <<SUT>> stereotype is provided by estimation if a class is testable. A class is testable, if it provides an interface with public methods returning values.

![Diagram showing test components and system components](image2)
Create additional classes stereotyped with <<TestContext>> inside the SystemNameSuites package. Name these classes according to the rule "SystemNameSuite Nr" with SystemName as the name of the system model. The number of them depends on number of test configurations that can be obtained.

Transformation Rules for Test Architecture

Create additional classes (for each part stereotyped with <<TestComponent>> in the composite structure diagram). Name these classes according to the rule "ClassNameEmulator" with ClassName as the name of the class coming from system model.
Transformation Rules for Test Behaviour

- A testcase is derived from an existent activity diagram of the eUML model. Herein, the original activity diagram with all its swim-lanes and behavior are adopted. Name the diagram according to the rule “DiagramName_testCase” with DiagramName as the name of the diagram coming from system model.

PIT to PST Transformation

- Transforms eTML models to TTCN-3 models
- Test behaviour transformation rules for activities
- Some adjustments to mapping rules to provide information the test adapter (and codec) needs for test execution
Generation of TTCN-3 Module, SUT and MTC Component Types out of Test Context

module PizzaServiceSuite { 
group Suite { 
  type component MTC_CType [] 
  
  type component PizzaService_SUT_CType ( 
    port usePS_PType providePS; 
    port provideDN_PType useDN; 
  ) 
}

Generation of Signatures and their Parameters out of Interface Operations

signature notifyOnPendingOrder() return PendingOrder; 
signature getMenu() return Menu; 
signature confirmOrder[in LocalContactData localContactData] return Order; 
signature getCustomerOrders[in integer customerId] return order; 
signature removeAllMenuItems(); 
signature updateMenuItems [in charstring actionSelector, in MenuItem menuItems] ;
Generation of Test Cases
out of Activities stereotyped with <<TestCase>>

testcase PizzaShopBehavior_testCase() runs on MTC_CType system PizzaService_SUT_CType {

Generation of Test Cases Bodies
out of Activities stereotyped with <<TestCase>>

testcase PizzaShopBehavior_testCase() runs on MTC_CType system PizzaService_SUT_CType {

```java
function PizzaService_setup_Configuration()
    in PizzaService_SUT_CType pizaaService,
    inout CustomerEmulator_CType customerEmulator,
    inout DelivererEmulator_CType delivererEmulator;
    customerEmulator := CustomerEmulator_CType.create;
    delivererEmulator := DelivererEmulator_CType.create;
    map (pizaaService: providePizaa, customerEmulator: useSet);
}
```

Connection to TTworkbench

- Issue:
  - Transformers are based on Medini rep.
  - TTworkbench is based on Eclipse EMF
- Approach:
  - Decoupling of the Medini TTCN-3 repository and TTworkbench
  - Serialiser for the Medini TTCN-3 repository writing TTCN-3 files to TTworkbench

Test Code Generation with TTworkbench

- Generating Java test code from TTCN-3 with TTCN-3 compiler of TTworkbench
- Compiling the Java files to jar files ready for test execution
• Test adapters (including codec) are needed for test execution
• Usually implemented in Java (but also C etc. possible)
• Implement the mechanisms and en/decoding needed for the communication with the SUT
• Support for EJB and WSDL
• Some influence on the TTCN-3 generation (parameters and signatures needed for test execution in the test adapter)
Test Execution

Test Deployment & Execution

Compiled .jar File

TTworkbench - TTman

Communication (EJB / WSDL)

SUT

Test Logging, Tracing & Evaluation

TTworkbench – Logged Test Traces

Evaluating the Test Results
## Contents

- Motivation
- The Basics: TTCN-3
- The Approach: MDA++
- Some more details
- Summary

---

**Integrated Modelling and Testing**

![Diagram showing integrated moddeling and testing](image)

- System & test centric view which is supported by tools

---

© FOKUS, 2006  
IPA Lenterdagen on Testing, Apr. 2006  
www.fokus.fraunhofer.de/motion
Conclusion

- MDA techniques are applicable to the automated derivation of test cases
  - Application of PIM and PSM abstraction also to test development
  - Transformers for both system development and test derivation based on MOF technology
- Seamless integration of test into system development with UML2 easier
  - Integrated UML2 based tool chain at platform-independent level
  - Joint UML2 based editor and joint repository for system and test models
- Automated generation of test models with test behaviour from UML based system models is possible
  - System and test behaviour based on activity diagrams
  - Test derivation rules and transformers for test behaviour based on activity diagrams is quite complex and need some further refinement
- Developed a test framework with seamless tool chain for testing EJB and WS systems
  - Based on UML2, TTCN-3, MOF infrastructure (applying Medini and EMF) and Eclipse
  - New approach for Medini-Eclipse integration (repository serialisation)
  - Proof of test concepts and demonstration with PizzaShop example

Experiences

- Derivation of test behaviour
  - Mapping rules and transformer implementation get complex because of test behaviour generation
  - Behaviour in system model must be well-defined and completely described for test behaviour generation
  - Otherwise, generated test behaviour needs to be completed manually
- Test execution
  - Aligning the transformer and the adapter for test execution takes some effort, e.g.
  - Mapping between TTCN-3 signatures and "real" signatures of SUT required
  - Common meta-model for system and test development needs to be quite stable
  - Changes to meta-model require considerable effort for adjusting the transformers
  - Using the same language for both system and test development is really beneficial
    - Allows for early and seamless integration of tests into the system development
Thank You!

Any Further Questions?

Please consider
3rd TTCN-3 User Conference
Berlin, May 31 – June 2, 2006

TTCN-3 could be the target, however ...

- TTCN-3 with its roots in telecommunication
  (and Europe)
  is hardly accepted
  in the software/IT environment

- Hence, idea to develop a
test specification approach within UML (U2TP) and

- extend the MDA paradigm
  with a test development chain (MDA++)
Extend TTCN-3 with an UML access

Test Design with UML 2.0 Testing Profile

- U2TP has been defined to capture all information that would be needed by different test processes
- It is 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
- It is an official OMG standard since summer 2004
Comparing U2TP and TTCN-3

<table>
<thead>
<tr>
<th></th>
<th>U2TP</th>
<th>TTCN-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Design</td>
<td>✓</td>
<td>(—)</td>
</tr>
<tr>
<td>Test Specification</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Test Execution</td>
<td>(—)</td>
<td>✓</td>
</tr>
<tr>
<td>Test Meta Modelling</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Format</td>
<td>Graphical</td>
<td>Textual and graphical</td>
</tr>
<tr>
<td>Transformation</td>
<td>U2TP to TTCN-3 (✓)</td>
<td>TTCN-3 to U2TP ✓</td>
</tr>
</tbody>
</table>