Testing framework for Platform Independent Models with TTCN-3

Gábor Bátori
gabor.batori@eth.ericsson.se
Topics

• Motivations
• xUML and Simulation architecture
• TTCN-3 language
• Test configuration
• Transformations
• Test behavior
Motivations

• Fuzzy textual specifications
• Graphical models
  – No way of early model verification
  – The graphical notation set is very rich
• Executable Specifications
  – Completely describe the model’s dynamic behavior
  – Verification with simulation
  – Creating tests independently of the problem modeling
  – Model transformation for code generation (Model Driven Architecture)
Goals

- Executing large number of test cases on Executable Models
- Regression testing on model level
- Testing the whole model or only smaller parts
- Using special test language → UML doesn’t applicable for this purpose
- Early-phase tests help the implementation level test development
Model Driven Architecture (MDA) and xUML

- **PIM (Platform Independent Model)**
  
  *Expresses only business functionality and behavior*

- **PSM (Platform Specific Model)**

  *Defines also technology-specific details*

- **Domain Partitioning**
  - Reusable, well-defined description of a problem

- The domains are in client-server relation

- Relationships between domains are well defined with bridges
Domains and bridges

Application

Service 1
- Security

Service 2
- Communication

Architectural
- Distributed

Implementation 1
- C++

Implementation 2
- TCP / C++

Bridges

On-line Bank
Domains and bridges

• Subject matter partition:
  – DOMAINS
    Separate, independent subject matters
  – Subsystems

• Relationship:
  – BRIDGES
    Relationships between domains
How does translation work?

- OOA models can be simulated before implementation
- OOA models are translated by Model Compilers
How does simulation work?

- Create a UML Virtual Machine
  - As input, the Virtual Machine requires an xUML model
  - A special platform with general capabilities
  - We trust on the functionality of the VM
- The simulation runs on this Virtual Machine
- The simulated model is testable
  - Test interface definitions?
How does simulation work?

**UML model**
- Bank
- Account
- Transfer
- Customer

**State Chart**

**UML Virtual Machine**

**Simulation Architecture**
What do we have?

Kennedy – Carter toolset

- **iUML** - modeler
  - simulator, Simulation architecture
- **iCCG** - Configurable Code Generator
  for creation new architectures

**Test Description**

- **TTCN-3** (Testing and Test Control Notation) a standardized test language
Introduction to TTCN-3

- Flexible Language
- Applicable to the specification of all types of reactive system tests.
- The core language has a C like textual syntactic (easy to read for programmers as well)
- Operations for procedure-based and message-based communication
- Specification of dynamic concurrent testing configurations
- Data and signature templates with matching mechanism
TTCN-3 language overview

• TTCN-3 static definitions
  – Data types, special types
  – Templates and constants
  – Test components, test port definitions

• Dynamic behavior descriptions
  – Functions, test cases
  – Test configurations
  – Behavior trees, timers, verdicts
  – Executing and controlling test cases
TTCN-3 Executor

- Imported ASN.1 Modules
- TTCN-3 Abstract Test Suite
- Other Presentation Formats
- Base Library
- Compiler
- Source Code
- Test Port(s)
- Compiler
- Executable Test Suite
- User Code
- Skeleton

ECOOP 2004, OSLO
Basic language elements

- Module header
  - Module name, object identifier, module (external) parameters

- Definitions part
  - Data types and function signatures
  - Constants
  - Templates
  - Port and component types
  - Functions
  - Test Cases

- Control part
  - Control of execution of test cases
Basic language elements

- **Module header**
  - Module name, object identifier, module (external) parameters

- **Definitions part**
  - Data types and function signatures
  - Constants
  - Templates
  - Port and component types
  - Functions
  - Test Cases

- **Control part**
  - Control of execution of test cases
**A TTCN-3 example**

```tcl
testcase ConnectionTest() runs on MTCType
    timer T1:=2;
    ttcn_port.send(Connect);
    T1.start;
    alt{
      []ttcn_port.receive(Connection_Acknowledgement){setverdict(pass);
      []T1.timeout{setverdict(fail);
    }
}
```

**Test Data on PIM level**

```tcl
template Connect_Message Connect:=
    connection_type:=persistent
```

**Test Data on implementation level**

```tcl
template Connect_Message Connect:=
    connection_type:=persistent,
    ip_address := "1.1.1.1",
    tcp_port := 1000
```
Obvious question: Why not connect these technologies?

• Interface definition between the two technologies
• Hide the communication problems from the testers and analysts
Testing and debugging executable models

- PIMs suffer from testability problems in the area of
  - Observability
  - Controllability
- We need an interface on UML level to reach the inner part of the model
Structure of the PIM tester

- **PIM tester**
  - TTCN-3
    - Test data
    - Test behavior

- **Test Domain**
  - Domain operations

- **Tested Domain(s)**

- **Bridges**

- **Logical layer**
  - Physical layer

- **TTCN-3 Executor**

- **xUML**

- **UML Virtual Machine**

- **Communication Interface**
Transformation of the xUML model

- The Platform Independent model is the source of the test definition
- Based on the bridge definition of the tester domain

```
<table>
<thead>
<tr>
<th>xUML PIM model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model transformation</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Basic test data</td>
</tr>
<tr>
<td>TTCN-3 Static Definition</td>
</tr>
<tr>
<td>TTCN-3 Test Port</td>
</tr>
<tr>
<td>Test Port for Sim. Architecture</td>
</tr>
</tbody>
</table>
```
Transformation of the xUML model

- Black-box testing of the model
- Message-based communication between the tester and the model
- Using special TAGs to note the border of the model
- Creating sending and receiving messages from the UML model
Transformation of the xUML model

- Based on the PIM model of the analysed problem
- Code generation for TTCN-3 and UML Virtual Machine
- Client-server based communication interface
Behavior of the PIM tester

TTCN-3

Test data
Test behavior

TTCN-3 Port

map( mtc:uml_mport, system:uml_mport );
uml_mport.send(packet1);
alt{
    []uml_mport.receive;
}

Tester Domain

Tester Domains

Bridges

Terminators

Domain operations

TTCN-3 Port
UML Port
Behavior of the PIM tester

void UML__PORT::outgoing_send(const Packet& send_par)
{
    packet=EncodePacket();
    sendto(send_sock,packet,...);
}
Behavior of the PIM tester

void DecodegetPacket(char* message){
    ...
    D6TERM4_Packet(6,0,NULL,user_name);
}
Behavior of the PIM tester
Ongoing work

- Model level test of an Alarm Mediation application
- Derive static test data definitions automatically
- Describing test data manually based on the derived definitions
- Run the implementation level test on the final product
Summary

• TTCN-3 allows:
  – Automatic execution of large number of test cases
  – Regression test on different version of the model
• Compiler provides automatic creation of:
  – Static data definition for TTCN-3
  – Port definition for TTCN-3
  – Port definition for Simulation Architecture
• Test development simultaneously with modeling
Thank you for the attention!
Appendix A

TTCN-3 language elements
An example: “Hello, World!” in TTCN-3

module MyExample{
    type port PCOType message{
        inout charstring;
    }
    type component MTCType{
        port PCOType MyPCO;
    }
    testcase HelloW() runs on MTCType{
        MyPCO.start;
        MYPCO.send("Hello,World!");
        MyPCO.stop
        setverdict(pass);
    }
    control{
        execute(HelloW());
    }
}
Structured Types Example – record, record of

```plaintext
//record type definition
type record MyMessageType1
{
    integer field1,
    boolean field2
}
type record MyMessageType2
{
    MyMessageType1 field1,
    boolean field2
}
//list type definition
type record of MyMessageType2 MyListType;
```
Data Templates

- Special data values for sending and receiving messages
- Templates can be created for simple types as well as compound types

```plaintext
template MyMessageType1 MyMessage:=
{
    field1 := 2,
    field2 := true
}

// Example use of this template (in the dynamic part of the test description)
PORT1.send(MyMessage);
```
Appendix B

Translation of the xUML model
TTCN-3 Static definition

- Module declaration
- TTCN-3 structures represent the parameters of terminators and domain scoped operations
- Port declaration for sending and receiving messages
- Main component declaration
TTCN-3 Static definition

define bridge SNMP:SNMP_APP1_Trapv1_received
input \n    in_enterprise:OID,\n    in_agent_address:IP_Address,\n    in_generic_trap_type:Generic_Trap_Type,\n    in_specific_trap_type:Integer,\n    {in_variable_list}:Name_Value_Pair
Output
...

\begin{Verbatim}
type record Trapv1_received_T{
    charstring in_enterprise,
    charstring in_agent_address,
    Generic_Trap_Type in_generic_trap_type,
    integer in_specific_trap_type,
    Name_Value_Pair_list in_variable_list
};
\end{Verbatim}
TTCN-3 Static definition

define bridge SNMP:SNMP_APP1_Trapv1_received
input \n    in_enterprise:OID, \n    in_agent_address:IP_Address, \n    in_generic_trap_type:Generic_Trap_Type, \n    in_specific_trap_type:Integer, \n    {in_variable_list}:Name_Value_Pair
Output
...

type enumerated Generic_Trap_Type{
    coldStart(0),
    warmStart(1),
    linkDown(2),
    linkUp(3),
    ...
};
TTCN-3 Static definition

define bridge SNMP:SNMP_APP1_Trapv1_received
input \n    in_enterprise:OID,\n    in_agent_address:IP_Address,\n    in_generic_trap_type:Generic_Trap_Type,\n    in_specific_trap_type:Integer,\n    {in_variable_list}:Name_Value_Pair
Output ...

\n\n\n\ntype record Name_Value_Pair {
    charstring name,
    charstring value_
};
\ntype record of Name_Value_Pair
Name_Value_Pair_list;
TTCN-3 Test Port Definition

- Automatically generated from the UML model
- Socket based communication with the xUML model
- Automatic encoding and decoding of messages

```c
void UML__PORT::outgoing_send(const UserInfoRequest__T& send_par){
  char outbuf[1024];
  int len = 0;
  char tmp_text[1024];
  strcpy ( &outbuf[len], "UserInfoRequest_T," );
  len += strlen("UserInfoRequest_T," );
  strcpy(&outbuf[len], send_par.in__user__name());
  len += strlen(send_par.in__user__name());
  nbytes = sendto (send_sock, outbuf, ...);
}
```

```c
define bridge CHATAPP:USERD1_UserInfoRequest
  input \
    in_user_name:Text
  Output
  ...
```
Test Port Definition for UML

- Automatically generated from the UML model
- Socket based communication with the xUML model
- Automatic encoding and decoding of messages

```c
void DecodeUserInfoRequest_T(char* message){
  char token[256];
  char* in_user_name;
  sscanf(message, "%[^,],%s", token, message);
  in_user_name = (Text)cgen_malloc(strlen(token) + 1);
  strcpy( in_user_name, token );
  D2_USERD1_UserInfoRequest(2,0,NULL, in_user_name);
}
```

define bridge CHATAPP:USERD1_UserInfoRequest
input \
  in_user_name:Text
Output
...