FROM TDL TO TTCN-3

Philip Makedonski (University of Göttingen)
Martti Käärik (Elvior OU)
Overview

What is TDL?
- Test Description Language
  - Design, documentation, and representation of formalised test descriptions
  - Scenario-based approach
  - Standardised at ETSI by TC MTS
    - STF 454 (2013)
    - STF 476 (2014)
    - STF 492 (2015-2016)
    - STF 522 (2017)

What is TTCN-3?
- Testing and Test Control Notation
  - Specification and implementation of all kinds of black-box tests
  - Platform independent link between modelling and execution
  - Component-based approach
  - Standardised at ETSI by TC MTS
  - 15+ years of maintenance work

Mapping TDL to TTCN-3
- Establish a connection between TDL and TTCN-3
  - generation of executable tests from test descriptions
  - standardised, ensuring compatibility and consistency
  - re-use existing tools and frameworks for test execution
  - re-use existing TTCN-3 assets (data, behaviour)
What is TDL?

- Test Description Language
  - Design, documentation, and representation of formalised test descriptions
  - Scenario-based approach
- Standardised at ETSI by TC MTS
  - STF 454 (2013)
  - STF 476 (2014)
  - STF 492 (2015-2016)
  - STF 522 (2017)
What is TDL?

- Design, documentation, representation?
- ease development and review
- improve productivity and quality
- both industry and standardisation
- reduce implementation details
System Requirements Specification

MBT Workflow

Model

TPLan

TDL

Manual Workflow

Test Generator

TDL-TO

Test Design Level

Test Code Generator

TTCN-3 (or Other Execution Language)

Requirements Level

Adaptation

Executable Tests

Test Implementation Level

Test Execution Level
What is TDL?

• Scenario-based?
  • describe interactions with a system
  • attach test objectives to scenarios
  • derive and automate tests
• Reactive, distributed, real-time
  • common black-box testing concepts
  • domain adaptation
  • agile development
What is TDL?

- Standardised?
  - canonical reference
  - stable documentation
  - clear semantics
  - interoperability and independence
  - updated with user needs
  - maintenance commitment
What is TDL?

- Contributions from:
  - Siemens AG, Ericsson Hungary
  - Fraunhofer FOKUS, ETSI CTI
  - CEA, University of Göttingen
  - OU Elvier, Cinderella ApS
- Guidance:
  - Steering Group, TC MTS
What is TDL?

Part 1: MM Meta-Model and Semantics

Part 2: GR Graphical Syntax

Part 3: XF Exchange Format

Part 4: TO Structured Test Objective Specification
What is TDL?

Part 5: UML Profile for TDL

Part 6: Mapping to TTCN-3

Part 7: Extended Test Configurations
What is TDL?

A 'GateType' represents a type of communication points, called 'GateInstance's, for exchanging information between 'ComponentInstance's. A 'GateType' specifies the 'DataType's that can be exchanged via 'GateInstance's of this type in both directions.

Generalization
- PackageableElement

Properties
- `dataType: DataType [1..*] {unique}`
  The 'DataType's that can be exchanged via 'GateInstance's of that 'GateType'. The arguments of 'Interactions' shall adhere to the 'DataType's that are allowed to be exchanged.

Constraints
- There are no constraints specified.
What is TDL?

A 'GateType' represents a type of communication points, called 'GateInstance's, for exchanging information between 'ComponentInstance's. A 'GateType' specifies the 'DataType's that can be exchanged via 'GateInstance's of this type in both directions.

6.4.2 GateType

Concrete Graphical Notation

```
GATETYPE.NAMELABEL
Data Type: DATATYPE.LISTLABEL
```

Radio

Data Type: Message, Signal

Formal Description

```
context GateType
GATETYPE.NAMELABEL ::= self.name
DATATYPE.LISTLABEL ::= self.dataType.name->separator(,)
```

Comments

No comments.
What is TDL?

- TDL main ingredients
  - Test data
  - Test configuration
  - Test behaviour
  - Test objectives
  - Time
What is TDL?

- TDL main ingredients
  - Test data
  - Test configuration
  - Test behaviour
  - Test objectives
  - Time
Main Ingredients

- Test data
  - data definition and data use
  - abstract types and instances
  - composed by using parameters
  - functions and actions
  - mappable to concrete data
  - variables and special values
Test Data

Type Login;
Login correct;
Login incorrect;

Use "data.ttcn3" as DATA ;
Map correct to "johnny_correct" in DATA as correct_ttcn3;
Map incorrect to "johnny_incorrect" in DATA as incorrect_ttcn3;

template Login johnny_correct := {
    user := "johnny",
    password := "apple",
    hint := "seed",
    id := 1000
}
template Login johnny_incorrect := {
    user := "johnny",
    password := "orange",
    hint := "second favourite fruit",
    id := 2000
}
type record Login {
    charstring user,
    charstring password,
    charstring hint,
    integer id
} with {
    encode "xpath=//div[@id='login']";
    encode (user) "relative=/div/dd[3]";
    encode (password) "relative=/div/dd[4]";
};
Test Data

Type Login;
Login correct;
Login incorrect;

Use "data.ttcn3" as DATA ;
Map correct to "johnny_correct" in DATA as correct_ttcn3;
Map incorrect to "johnny_incorrect" in DATA as incorrect_ttcn3;
Test Data

Simple Data Type

Login

Simple Data Instance

correct

Simple Data Instance

incorrect

Data Element Mapping

correct_ttcn3 := "johnny_correct"

Parameter Mapping

Data Resource Mapping

DATA

Resource URI

"data.ttcn3"

Data Element Mapping

incorrect_ttcn3 := "johnny_incorrect"

Parameter Mapping

success_ttcn3 := "johnny_success"

Parameter Mapping

failure_ttcn3 := "johnny_failure"
Main Ingredients

- Test configuration
  - typed components and gates
  - timers and variables
  - connections among gates
  - component roles
Test Configuration

Gate Type $gt$ accepts $Login$, $Response$;

Component Type $ct$ having {
  gate $g$ of type $gt$;
}

Test Configuration $tc$ {
  create $Tester$ tester of type $ct$;
  create SUT $sut$ of type $ct$;
  connect tester.$g$ to $sut.g$;
}

$a failure response is sent$ when incorrect login is provided

"ensure that $success_ttcn3$ := "johnny_success"

"data.ttcn3"

"failure_ttcn3" := "johnny_failure"
Main Ingredients

- Test behaviour
  - defines expected behaviour
  - failure upon deviations by default
  - actions and interactions
  - alternative, parallel, iterative, conditional
  - defaulting, interrupting, breaking
Test Behaviour

Test Description \( \text{td} \) (\( p \) of type \( \text{Login} \)) uses configuration \( \text{tc} \) {
  tester.g sends incorrect to sut.g;
  alternatively {
    sut.g sends failure to tester.g with {
      test objectives : tp;
    };
    set verdict to pass;
  } or {
    sut.g sends success to tester.g;
    set verdict to fail;
  }
}

or simply (relying on the default semantics):

Test Description \( \text{td\_default} \) (\( p \) of type \( \text{Login} \)) uses configuration \( \text{tc} \) {
  tester.g sends incorrect to sut.g;
  sut.g sends failure to tester.g with {
    test objectives : tp;
  };
}
Main Ingredients

• Test objectives
  • may be attached to
    • behaviour (atomic or compound)
    • whole test description
  • contain description and reference
**Test Objectives**

Test Objective \( tp \) {
  description : "ensure that when incorrect login is provided a failure response is sent";
}

Test Description \( td \) (\( p \) of type \( \text{Login} \))
  uses configuration \( tc \) {
    tester.g sends incorrect to sut.g;
    alternatively {
      sut.g sends failure to tester.g;
      set verdict to pass;
    } or {
      sut.g sends success to tester.g;
      set verdict to fail;
    }
  }
}

} with {
  test objectives : \( tp \);
}
Overview

What is TDL?

- Test Description Language
  - Design, documentation, and representation of formalised test descriptions
  - Scenario-based approach
- Standardised at ETSI by TC MTS
  - STF 454 (2013)
  - STF 476 (2014)
  - STF 492 (2015-2016)
  - STF 522 (2017)

What is TTCN-3?

- Testing and Test Control Notation
- Specification and implementation of all kinds of black-box tests
- Platform independent link between modelling and execution
- Component-based approach
- Standardised at ETSI by TC MTS
  - 15+ years of maintenance work
What is TTCN-3?

- Testing and Test Control Notation
- Specification and implementation of all kinds of black-box tests
- Platform independent link between modelling and execution
- Component-based approach
- Standardised at ETSI by TC MTS
- 15+ years of maintenance work
What is TTCN-3?

- Black-box tests?
  - functional, conformance, interoperability, robustness, load
  - standardisation and certification
- Used in various domains
  - telecommunications
  - automotive
  - railway
  - financial
  - medical
What is TTCN-3?

- Platform independent?
  - standardised core language
  - standardised interfaces
  - not tied to application or interface
  - not tied to tooling
- Requirements
  - test suite
  - compiler / interpreter
  - adapters and codecs
  - execution environment
TCI - TTCN-3 Control Interface
TRI - TTCN-3 Runtime Interface

Test System User

Test Management
TTCN-3 Test System
Test Logging

Component Handling

CoDec

System Adapter
Platform Adapter

System Under Test SUT

TCI
TRI
Adapted from Grabowski et al., 2014
What is TTCN-3?

- Component-based?
  - describe behaviour of test system
  - one or more test components
  - interconnected among each other
  - mapped to unified SUT interface
MTC - Main Test Component
PTC - Parallel Test Component
MTC - Main Test Component
PTC - Parallel Test Component
What is TTCN-3?

- Test suite ingredients
  - Data
    - basic, structured, and special types
    - constants, templates, expressions
  - Configuration
    - components, ports, connections
    - dynamic management
  - Behaviour
    - test cases, functions, altsteps
    - defaults and timers
    - optional test execution control
What is TTCN-3?

//enumerated data type
type enumerated MSGKind {question, answer}

//structured data type
type record MSG {
    MSGKind kind,
    charstring content
}

//a question template
template MSG readyQuestion := {
    kind := question,
    content := "Ready?"
}

//a generic question template
//any question is fine
template MSG anyQuestion := {
    kind := question,
    content := ?
}

//a generic question template
//the content shall be provided upon use
template MSG p_Question (charstring c) := {
    kind := question,
    content := c
}

//a generic answer template
//any content is fine
template MSG anyAnswer := {
    kind := answer,
    content := ?
}

//a generic answer template
template MSG p_Answer (charstring c) := {
    kind := answer,
    content := c
}
What is TTCN-3?

//simple port
type port MSGPort message {
    inout MSG
    //may also support transmission of other types
}

//simple component
type component Client {
    timer patience;
    port MSGPort clientPort
    //may also define multiple ports, variables, timers
}

//simple test case
testcase TC_isServiceReady() runs on Client {
    clientPort.send(p_Question("Ready?"));
    alt {
        □ clientPort.receive(p_Answer("Yes!")) {
            setverdict(pass);
        }
        □ clientPort.receive(p_Answer("No!")) {
            setverdict(fail);
        }
    }
}
What is TTCN-3?

//simple timed test case
testcase TC_isTimedServiceReady() runs on Client {
    clientPort.send(p_Question("Ready?")));
    patience.start(10.0);
    alt {
        clientPort.receive(p_Answer("Yes!"))) {
            setverdict(pass);
        }
        clientPort.receive(p_Answer("No!"))) {
            setverdict(fail);
        }
        patience.timeout {
            setverdict(fail);
        }
    }
    patience.stop;
}
What is TTCN-3?

//distributed test case
testcase TC_distributed() runs on Client
    system Service {
        //create components
        var Client client1 := Client.create;
        var Client client2 := Client.create;
        //map / connect components
        map(system:servicePort, client1:clientPort);
        map(system:servicePort, client2:clientPort);

        //start initiate behaviour of components
        client1.start(f_isReady());
        client2.start(f_isReady());

        //wait for components to complete their execution
        all component.done;
    }

//handle timeouts and incoming questions
altstep impatientYesMan() runs on Client {
    clientPort.receive(p_Question(?)) {
        clientPort.send(p_Answer("Yes!"))
        repeat;
    }
    patience.timeout {
        setverdict(fail);
    }
}

//reusable behaviour
//can be executed multiple times
function f_isReady() runs on Client {
    clientPort.send(p_Question("Ready?"));
    patience.start(10.0);
    activate(impatientYesMan());
    alt {
        clientPort.receive(p_Answer("Yes!")) {
            setverdict(pass);
        }
        clientPort.receive(p_Answer("No!")) {
            setverdict(fail);
        }
    }
}
Overview

What is TDL?

- Test Description Language
  - Design, documentation, and representation of formalised test descriptions
  - Scenario-based approach
  - Standardised at ETSI by TC MTS
    - STF 454 (2013)
    - STF 476 (2014)
    - STF 492 (2015-2016)
    - STF 522 (2017)

What is TTCN-3?

- Testing and Test Control Notation
  - Specification and implementation of all kinds of black-box tests
  - Platform independent link between modelling and execution
  - Component-based approach
  - Standardised at ETSI by TC MTS
    - 15+ years of maintenance work

Mapping TDL to TTCN-3

- Establish a connection between TDL and TTCN-3
  - generation of executable tests from test descriptions
  - standardised, ensuring compatibility and consistency
  - re-use existing tools and frameworks for test execution
  - re-use existing TTCN-3 assets (data, behaviour)
Mapping TDL to TTCN-3

- Establish a connection between TDL and TTCN-3
- generation of executable tests from test descriptions
- standardised, ensuring compatibility and consistency
- re-use existing tools and frameworks for test execution
- re-use existing TTCN-3 assets (data, behaviour)
ETSI ES 203 119-6 V1.1.1 (2018-06)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Part 6: Mapping to TTCN-3

ETSI STANDARD

ES 203 119-6

Test Implementation Level

Test Execution Level

Requirements Level

Test Design Level

MBT Workflow

System Requirements Specification

Model

Test Generator

TDL

Test Code Generator

TTCN-3

Adaptation

Executable Tests

Manual Workflow

Requirements Level

Test Design Level

Test Implementation Level

Test Execution Level

ETSI ES 203 119-6

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Part 6: Mapping to TTCN-3

ETSI STANDARD
Mapping TDL to TTCN-3

- **Challenges**
  - different levels of abstraction
  - different perspectives
  - different assumptions
    - behaviour
    - configurations
    - data
    - time

**ETSI ES 203 119-6 V1.1.1 (2018-06)**
Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 6: Mapping to TTCN-3
Mapping TDL to TTCN-3

- Levels of abstraction
  - TTCN-3
    - low - close to implementation
    - sufficient for automated execution
    - still abstracts away some details
  - TDL
    - high - test purposes (TO-extension)
    - medium - test design and description
    - low - some implementation details
    - focus on relevant parts at every level
Mapping TDL to TTCN-3

• Perspectives

• TTCN-3
  • test-system centric (test system view)
  • test components
  • unified SUT interface (ports)

• TDL
  • system centric (global view)
  • tester and SUT components (roles)
  • describes entire scenario
Mapping TDL to TTCN-3

• Assumptions: Data
  • TTCN-3
    • comprehensive type system
    • powerful template mechanism
    • extensive matching operators
  • TDL
    • mappable symbolic elements
    • types and instances
    • wildcards
    • limited direct data manipulation
    • nested arguments for data use
Document content not visible in the image.
Mapping TDL to TTCN-3

- **Mapping: Data Definition**
  - data mappings within TDL required
    - also for all members
    - substituted by respective targets
  - basic generation in case absent
    - charstrings, records, templates
    - functions for functions and actions
  - annotations override assumptions
    - also for variables and parameters
//data types
Type SESSIONS (id1 of type Integer, id2 of type Integer);
Type MSG (ses of type SESSIONS, content of type String);

//data instances
SESSIONS s1(id1 = 1, id2 = 2);
SESSIONS s2(id1 = 11, id2 = 22);
MSG msg1(ses = s1, content = m1);

Component Type ct having {
  //variables
  variable v1 of type MSG with {VALUE;};
  variable v2 of type MSG;
  gate g of type gt;
}

//data types
type record SESSIONS {
  integer id1,
  integer id2
}
type record MSG {
  SESSIONS ses,
  charstring content
}

//templates
template SESSIONS s1 := {id1:=1, id2:=2}
template SESSIONS s2 := {id1:=11, id2:=22}
template MSG msg1 := {ses := s1, content := "m1"}

//value -> constant
const SESSIONS c_s1 := {id1:=1, id2:=2}
const MSG c1 := {ses := c_s1, content := "c1"}

type component ct {
  //variables
  var MSG v1;
  var template MSG v2;
  port gt g;
}
Mapping TDL to TTCN-3

• **Mapping: Data Use**
  • treatment as values or templates
    • temporary templates
    • using valueOf
  • modification for arguments
    • inline for first level
    • iterative for nested arguments
• special values
  • AnyValue -> ?
  • AnyValueOrOmit -> * (optional), ?
  • OmitValue -> omit
Mapping TDL to TTCN-3: Data use

Test Description `td` uses configuration `tc` {
  //one level arguments
  tester.g sends msg1(ses = s2) to sut.g;

  //nested arguments
  tester.g sends msg1(ses = s1(id1 = 111)) to sut.g;

  //nested arguments with value
  tester.g sends msg1(ses = c_s1(id1 = 111)) to sut.g;
}

function `td_tester()` runs on `ct` {
  //one level arguments
  g.send(modifies msg1 := {ses := s2});

  //nested arguments
  template SESSIONS t_s1 modifies s1 := {id1:=111};
g.send(modifies msg1 := {ses := t_s1});

  //nested arguments with constants
  template SESSIONS t_c_s1 := c_s1;
template SESSIONS t_c_s1_m modifies t_c_s1 :=
     {id1:=111};
g.send(modifies msg1 := {ses := t_c_s1_m});
}
Mapping TDL to TTCN-3

- Assumptions: Configurations
  - TTCN-3
    - dynamic instantiation / management
    - MTC, PTCs, system interface
    - mapping vs connecting ports
    - connection and mapping restrictions
  - TDL
    - static configuration defined upfront
    - holistic view, multiple SUTs
Test Configuration
ClientAndServer

TESTER
Server:CNode

SUT
Gateway:GWNode

TESTER
Client:CNode

SUT
Firewall:FWNode

Test System

System Under Test

Test Component
Server

Test Component
Client
Mapping TDL to TTCN-3

- **Mapping: Configurations**
  - port types for each gate type
  - infer unified system interface
  - types for MTC, system components
  - types for tester components
  - creating components
  - map and connect ports
  - respect restrictions in TTCN-3
  - some ports may need to be cloned
Mapping TDL to TTCN-3: Configurations

Gate Type defaultGT accepts
   ACK, PDU, PDCCH, C_RNTI, CONFIGURATION ;

Component Type defaultCT having {
   gate g of type defaultGT;
}

Test Configuration defaultTC {
   create Tester SS of type defaultCT;
   create SUT UE of type defaultCT ;
   connect UE.g to SS.g ;
}

type port defaultGT_to_map message {
   //this is a port type for SUT-Tester connections
   inout charstring, PDCCH /* ACK, PDU, C_RNTI, CONFIGURATION ; */
}

type port defaultGT_to_connect message {
   //this is a port type for Tester-Tester connections
   inout charstring, PDCCH /* ACK, PDU, C_RNTI, CONFIGURATION ; */
}

type component MTC_CT {
   //component type for MTC
   //variable for the PTC(s) --TESTER component(s) in TDL
   var defaultCT TESTER_SS;
}

type component defaultCT {
   port defaultGT_to_map g_to_map;
   port defaultGT_to_connect g_to_connect;
}

function defaultTC() runs on MTC_CT {
   // Test Configuration defaultTC, mappings, connections
   TESTER_SS := defaultCT.create;
   map (TESTER_SS:g_to_map, system:g_to_map);
}
Mapping TDL to TTCN-3

- **Assumptions: Behaviour**
  - TTCN-3
    - test system view
    - independent concurrent execution
    - explicit synchronisation
    - strictly local behaviours
  - TDL
    - global view
    - global or local ordering
    - implicit or explicit synchronisation
    - global combined behaviours
Mapping TDL to TTCN-3: Views

TTCN-3 view: behaviour defined for a **component and its ports**
Mapping TDL to TTCN-3: Views

TDL view: behaviour defined for all components
Mapping TDL to TTCN-3: Ordering

TTCN-3 assumption: order of sending msg1 and msg2 is undefined
Mapping TDL to TTCN-3: Ordering

TDL global ordering assumption: msg1 always occurs before msg2
In an implementation a global scheduler shall keep everything in order.
Mapping TDL to TTCN-3: Ordering

TDL local ordering assumption: order of msg1 and msg2 is undefined
Mapping TDL to TTCN-3: Ordering

TDL local ordering assumption: order can be specified explicitly
Mapping TDL to TTCN-3

- **Mapping: Behaviour**
  - capture tester perspective only
  - only locally ordered so far
  - functions for each component
  - combined behaviours
    - split for each participating component
  - interactions
    - split into test and/or receive
  - deviations from behaviour
    - altsteps activated as defaults
Mapping TDL to TTCN-3: Behaviour

Test Description Implementation TD_7_1_3_1
uses configuration defaultTC {

    SS.g sends pdcch (c_rnti=ue) to UE.g;
    SS.g sends mac pdu to UE.g;
    UE.g sends harq ack to SS.g with {
        test objectives : TP1 ;
    };

    set verdict to PASS ;
    SS.g sends pdcch (c_rnti=unknown) to UE.g;
    SS.g sends mac pdu to UE.g;

    alternatively {
        UE.g sends harq ack to SS.g ;
        set verdict to FAIL ;
    } or {
        gate SS.g is quiet for five ;
        set verdict to PASS ;
    } with {
        test objectives : TP2 ;
    }
}

altstep to_handle_deviations_from_TDL_description_AS () {
    any port.receive {
        setverdict(fail);
        mtc.stop;
    }
    //if nothing happens, a timer shall be started
    //before every receive instruction
    //and the timer must be here
    //or we can leave the timeout for
    //the execute instruction called with the optional
    //timer parameter - but in this case
    //the final verdict will be 'error'
}

altstep quiescence_handler_AS (timer quiescence) {
    //for all quiescence that is not connected to a gate
    any port.receive{
        setverdict(fail);
        mtc.stop;
    }
    quiescence.timeout {
        setverdict(pass);
    }
}
Mapping TDL to TTCN-3: Behaviour

Test Description Implementation TD_7_1_3_1 uses configuration defaultTC {

    SS.g sends pdcch (c_rnti=ue) to UE.g;
    SS.g sends mac_pdu to UE.g;
    UE.g sends harq_ack to SS.g with {
        test objectives : TP1;
    };

    set verdict to PASS;
    SS.g sends pdcch (c_rnti=unknown) to UE.g;
    SS.g sends mac_pdu to UE.g;

    alternatively {
        UE.g sends harq_ack to SS.g;
        set verdict to FAIL;
    } or {
        gate SS.g is quiet for five;
        set verdict to PASS;
    } with {
        test objectives : TP2;
    }
}

function behaviourOfTESTER_SS() runs on defaultCT {
    timer quiescence;

    activate(to_handle_deviations_from_TDL_description_AS());

    g_to_map.send(modifies pdcch := \{c_rnti := ue\})
    g_to_map.send(mac_pdu);
    g_to_map.receive(harq_ack);
    setverdict(pass);
    /*Test Objective Statisfied:  TP2 */

    g_to_map.send(modifies pdcch := \{c_rnti := unknown\});
    g_to_map.send(mac_pdu);
    quiescence.start(five);
    alt{
        [] g_to_map.receive(harq_ack){
            setverdict(fail);
        }
        quiescence_handler_AS(quiescence);
        /*Test Objective Statisfied:  TP2 */
    }
}
Mapping TDL to TTCN-3: Behaviour

Test Description Implementation TD_7_1_3_1 uses configuration defaultTC {

SS.g sends pdcch (c_rnti=ue) to UE.g;
SS.g sends mac_pdu to UE.g;
UE.g sends harq_ack to SS.g with {
    test objectives : TP1 ;
};

set verdict to PASS ;
SS.g sends pdcch (c_rnti=unknown) to UE.g;
SS.g sends mac_pdu to UE.g;

alternatively {
    UE.g sends harq_ack to SS.g ;
    set verdict to FAIL ;
} or {
    gate SS.g is quiet for five ;
    set verdict to PASS ;
} with {
    test objectives : TP2 ;
}
}

testcase TD_7_1_3_1() runs on MTC_CT
    system defaultCT
    {
        activate(to_handle_deviations_from_TDL_description_AS());
        defaultTC();
        TESTER_SS.start(behaviourOfTESTER_SS());
        all component.done;
    }

Mapping TDL to TTCN-3

- Assumptions: Time
  - TTCN-3
    - timers and timer operations
    - realtime extension
  - TDL
    - timers and timer operations
    - time operations (wait, quiescence)
    - time labels and time constraints
Mapping TDL to TTCN-3

[@T_received < T_sent + 3]
Mapping TDL to TTCN-3

- Mapping: Time
  - all concepts expressed by timers
  - local time keeping per component
  - time constraints challenging
Mapping TDL to TTCN-3: Time

function behaviourOfTESTER_tc1() runs on ct {
    timeKeeper.start(forever)
    g.send(msg1);
    // Time label
    var float T_sent := timeKeeper.read;
    g.receive(msg2);
    var float T_received := timeKeeper.read;
    // Time constraint
    if (T_received > T_sent + 3) {
        setverdict(fail);
        mtc.stop;
    }
    // ...
}
function behaviourOfTESTER_tc1() runs on ct {
    //...
    //Wait
    timer T1_wait_1;
    var default wh := activate(Wait_handler_AS());
    T1_wait_1.start(2.5);
    T1_wait_1.timeout;
    deactivate(wh);
    g.send(msg3);
    //Quiescence
    timer T1_quiescence_1;
    T1_quiescence_1.start(2.5);
    alt {
        [] T1_quiescence_1.timeout {setverdict (pass);}
        [] any port.check(receive) {setverdict (fail);}
    }
    }
}
altstep Wait_handler_AS() {
    //for suppressing handling of unexpected behaviour
    [] any port.check(receive) {repeat;}
}
Mapping TDL to TTCN-3

• Everything else
  • packages -> modules
  • element imports -> imports
  • annotations ->
    • comments
    • special instructions
    • code (TTCN3Code)
  • test objectives -> comments
  • comments -> comments
Concluding remarks

- Rapid initial growth
  - becoming more and more stable
- Open-source project for essential tool support
  - accelerate adoption, validate standards
- Custom tools can be put together in a matter of hours
  - basic, yet capable, make early adoption easier
- Advanced solutions still require additional effort
  - not immediately necessary to get started with using TDL
Concluding remarks

- Mapping may seem straightforward at first
  - but things can get very complicated the closer one looks
  - both languages have evolved to become rather complex

- Identify assumptions and semantic gaps
  - some restrictions may not be immediately obvious
  - some concepts may not be mappable at all in a useful way
  - adaptations to both languages can make mappings easier
  - some assumptions may need to be challenged

- A standardised mapping defines baseline expectations
  - tool- and user-specific can be optionally applied on top
IMPLEMENTING THE STANDARDISED MAPPING OF TDL TO TTCN-3

Philip Makedonski (University of Göttingen)
Jens Grabowski (University of Göttingen)
Summary

What is TDL?
- Test Description Language
  - Design, documentation, and representation of formalised test descriptions
  - Scenario-based approach
  - Standardised at ETSI by TC MTS
    - STF 454 (2013)
    - STF 476 (2014)
    - STF 492 (2015-2016)
    - STF 522 (2017)

What is TTCN-3?
- Testing and Test Control Notation
  - Specification and implementation of all kinds of black-box tests
  - Platform independent link between modelling and execution
  - Component-based approach
  - Standardised at ETSI by TC MTS
  - 15+ years of maintenance work

Mapping TDL to TTCN-3
- Establish a connection between TDL and TTCN-3
  - generation of executable tests from test descriptions
  - standardised, ensuring compatibility and consistency
  - re-use existing tools and frameworks for test execution
  - re-use existing TTCN-3 assets (data, behaviour)
What would you want to see in TDL?
What would you want to see in TDL?

Test Description Language (TDL) is a new language for the specification of test descriptions and the presentation of test execution results.

There is a methodology gap between the high-level expression of what needs to be tested, i.e., the test purposes described in prose or Test Purpose Language (TPLan), and the complex coding of the executable tests in Testing and Test Control Notation version 3 (ITCN-3). TDL fills that gap.

tdl.etsi.org
From TDL to TTCN-3

Philip Makedonski
Martti Käärik