An Automated, Flexible Testing Environment for UMTS

Jan Bredereke, Holger Schlingloff

University of Bremen, TZi
Germany
Project Frame

- **UMTS** protocol stack
- **Siemens AG**, Salzgitter: development in SDL
- **TZi**: black-box testing
Overview

• underlying testing tool RT-Tester

• testing environment for SDL
  – automated interfacing to SDL
  – flexible maintenance of test specification

• application: what did we find?
Specification Based Testing Approach

requirements specification (CSP)

test case generator

test cases

test driver/evaluator

test results

system specification (SDL)

development platform

executable code

Jan Bredereke, Univ. of Bremen
Testing Tool RT-Tester

- test specification in **timed CSP**

- **execution of test specifications:**
  1. automatic transformation into **graph of state transitions**
  2. on-the-fly, automatic, real-time generation / execution / monitoring / evaluation of test sequences

- tests of **arbitrary length/duration**

- both **functional** and **hard real-time** properties

- various visualization possibilities
Problem & Approach

- complex interface, still evolving

⇒

- automation of testing environment:
  - interface in terms of SDL, no low-level descriptions
  - automated consistency check of interface
  - interface code generation

- flexible test specification:
  - rules for modularizing test specifications
Manual Interfacing to SDL

TZi: human specifier

CSP specification

CSP compiler

RT-Tester runtime system

sockets

 Siemens: human specifier

SDL specification

SDL compiler

executable implementation

target machine
Automated Interfacing to SDL

TZi: human specifier

CSP specification annotated with parameter mapping to SDL → CSP compiler → executable test cases → RT-Tester runtime system

interface generator tool

CSP - socket interface

interface error report

SDL - socket interface

SDL compiler → executable implementation → target machine

sockets

UMTS standard

Siemens: human specifier

SDL specification

RT-Tester runtime system

CSP compiler

SDL compiler

target machine
Example: Matching CSP and SDL Parameters

**CSP**

```
nametype Rb_identity = {0 .. maxRb_count}
datatype Rlc_data = dummy_value
channel rlc_tr_data_req :

Rb_identity .

Rlc_data
```

**SDL**

```
synotype RB_Identity = integer
    constants 0:MaxRb_Count
endsynotype;
synonym RLC_MAX_SDU_SIZE integer = 512;
newtype RLC_SDU_A
carray(RLC_MAX_SDU_SIZE, octet)
endnewtype;
newtype RLC_SDU struct
    RB_Id        RB_Identity;
    RLC_SDU_Data RLC_SDU_A;
    length       integer;
endnewtype;
signal RLC_TR_DATA_Req( RLC_SDU);
```
Example: Matching CSP and SDL Parameters

CSP

```
nametype Rb_identity = {0 .. maxRb_count}
datatype Rlc_data = dummy_value
channel rlc_tr_data_req:
  pragma SDL_MATCH
      PARAM 1!RB_Id
      Rb_identity .
  pragma SDL_MATCH
      TRANSLATE dummy_value 0x99 * 16
  pragma SDL_MATCH
      PARAM 1!RLC_SDU_Data SUBSET_USED
      Rlc_data
  pragma SDL_MATCH
      SKIP 1!length DEFAULT_VALUE 16
```

SDL

```
syntype RB_Identity = integer
    constants 0:MaxRb_Count
endsyntype;
synonym RLC_MAX_SDU_SIZE integer
    = 512;
endsynonym;
newtype RLC_SDU_A
    carray(RLC_MAX_SDU_SIZE, octet)
endsnewtype;
newtype RLC_SDU struct
    RB_Id   RB_Identity;
    RLC_SDU_Data RLC_SDU_A;
    length integer;
endsnewtype;
signal RLC_TR_DATA_Req(RLC_SDU);
```
Overview

• underlying testing tool RT-Tester

• testing environment for SDL
  – automated interfacing to SDL
  – flexible maintenance of test specification

• application: what did we find?
Flexible Maintenance of Test Specification

• **late changes** to requirements

• **variants** of test suites:

  1. **adjust test coverage**
     - selected signal parameters
     - stimuli: random $\rightarrow$ increased probabilities $\rightarrow$ deterministic

  2. **component / integration** tests
     - different protocol layers
     - parallel instances of same layer

 3. **active / passive** tests

$\Rightarrow$ a **family** of test suites
Rules for Modularizing Requirements

• **separate**: *signature / behaviour* of module

• **identify** req. s that will **change together**, put into **one module** specifically, **separate**:
  – tester specific issues / application
  – timer handling / application
  – protocol layers
  – stimulus generation / test observation
Separate: Test Stimulus Generation / Test Observation

TESTSPEC

RT-Tester Test Driver

wrong_reaction
no_reaction
stimulus_overrun

RLC
(Radio Link Control layer)

System Under Test

Jan Bredereke, Univ. of Bremen
Separate: Test Stimulus Generation / Test Observation

TESTSPEC

RANDOM_TESTGENERATOR(EventSet)

RLC_OBSERVER

RT-Tester Test Driver

wrong_reaction
no_reaction
stimulus_overrun

RLC (Radio Link Control layer)

System Under Test

Jan Bredereke, Univ. of Bremen
Overview

• underlying testing tool RT-Tester

• testing environment for SDL
  – automated interfacing to SDL
  – flexible maintenance of test specification

• application: what did we find?
  – ambiguities in the standards
  – unexpected behaviour of SUT
Found: Ambiguities in the Standards
Interface of RLC Layer

MAC (Media Access Control)

RLC (Radio Link Control)

RRC (Radio Resource Control)

other

service access point

logical channels

Jan Bredereke, Univ. of Bremen
Found: Ambiguities in the Standards (2)
Properties of Service Access Points and Logical Channels

- forward instantaneously / buffer?
- queueing discipline?
- min./max. delay between delivery and availability?
- handling of unspecified reception?
Found: Unexpected Behaviour of SUT

Tester

```
crlc_config_rq.rbSetup
mac_status_ind
mac_status_ind
mac_data_req
```

SUT: RLC

```
}

RLC setup phase

nonsense input

even more nonsense output!

Jan Bredereke, Univ. of Bremen
Found: Unexpected Behaviour of SUT (2)

multiple RLC protocol machines in parallel
Summary

- **automatic generation** of interface

- **design rules** for flexible, modular family of test suites

profitable when:

- complex **requirements from 3rd party**

- **interface is a draft** most of the time

- **after completion** of requirements, short **time-to-market** is crucial