Specification-Based Testing of the UMTS Protocol Stack

Jan Bredereke and Bernd-Holger Schlingloff
Bremen Institute of Safe Systems
University of Bremen, Germany
{bredereke,hs}@tzi.de

Quality Week, San Francisco, May 31st, 2001

Bremen Institute of Safe Systems

- Technology transfer institute within Bremen university
- 35 scientists, 5 professors, 1.1 M Euro annual revenues
- Application of formal methods in industrial contexts
- Verification projects: Airbus, German Aerospace (EADS), OHB Satellites, DLR, SiemensVT, INSY Rail, *Siemens Telecom*, ...

Introduction

Current methods for testing embedded real-time control software:

- Structural or code-based
- Specification based

Application to the *Radio Link Control* (RLC) protocol layer of the *Universal Mobile Telecommunication System* (UMTS).

- Siemens AG, Salzgitter: code development
- Technologie-Zentrum Informatik (TZi), Bremen: testing support

UMTS: distributed development of standards, user equipment and base stations.
Consistency checks cannot be based on particular implementation!
$\Longrightarrow$ **specification based testing rather than structural testing!**

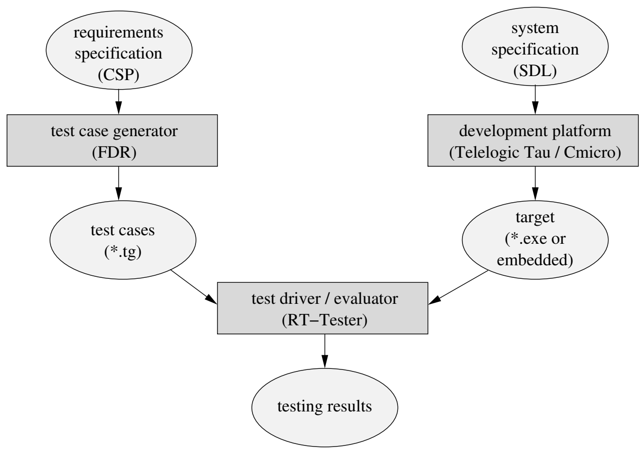
Specification Based Testing

Specification based methods:
treat system under test (SUT) as *black box*
focus on *properties* of the system.

Advantages:

- Concentration on functionality aspects
- Exhibition of ambiguities in the requirements
- Detection of misinterpretations, omissions and missing cases
- Generation of arbitrary length test scripts
- Reusability and maintainability

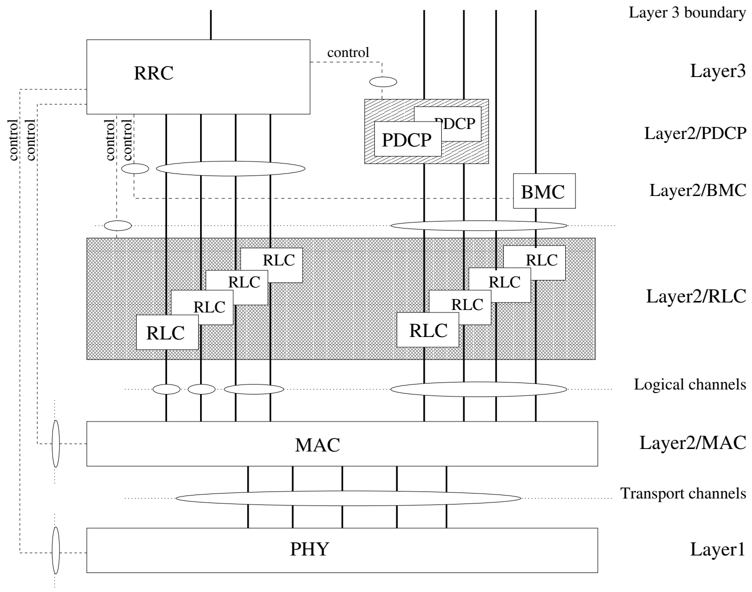
Testing Approach



High degree of automation reduces the overall development time!

UMTS Protocol Architecture

UMTS standard as developed by 3GPP consortium:



UMTS Protocol Architecture (cont.)

- *Layer 1: physical layer*; hardware services provided by the chip-set
- *Layer 2: data link layer*; provides point-to-point connection concept
  - *Medium Access Control (MAC)*; provides unacknowledged transfer of service data units, reallocation of parameters such as user identity number and transport format, and local measurements such as traffic volume and quality of service indication
  - *Radio Link Control (RLC)*; segmentation and reassembly of long data packets from higher layers into fixed width protocol data units, respectively. This includes flow control, error detection, retransmission, duplicate removal, and similar tasks
  - *Packet Data Convergence Protocol (PDCP)*; *Broadcast / Multicast Control (BMC)*
- *Layer 3: network layer*; provides network services such as establishment / release of a connection, hand-over, broadcast of messages in a geographical area, and notification of information to specific users. Radio Resource Control (RRC) assigns, configures and releases wireless bandwidth (codes, frequencies etc.)
- *Layer 4+: application layers*; e.g. Call Control (CC) and Mobility Management (MM)

The RLC Layer of UMTS

Tasks:

- Correction of *transmission errors*
- *Segmentation* of variable-length data packets received from the upper layer into fixed-length PDUs
- *Reassembly* of received PDUs according to the attached sequence numbers
- Optional *cipher mechanism* preventing unauthorized access to message data

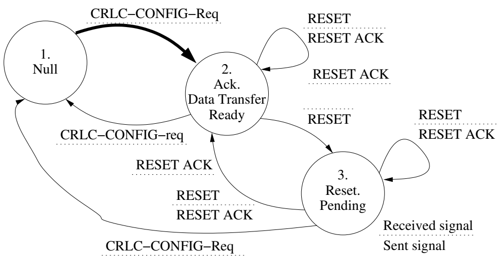
Three modes of data transfer:

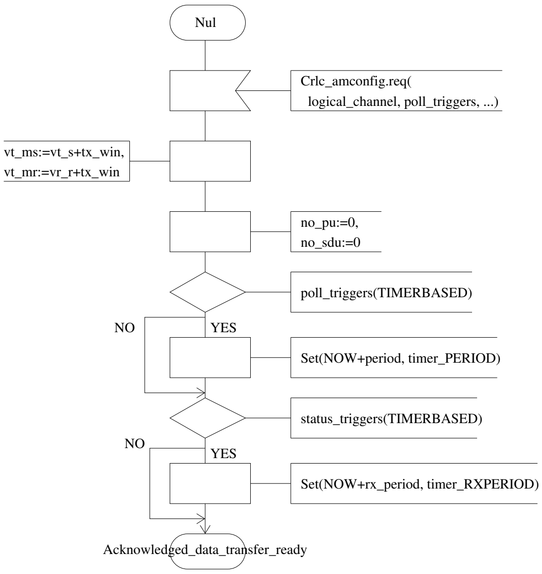
- *Acknowledged mode* (error-free transmission guaranteed)
- *Unacknowledged mode* (immediate; deletion of erroneous and duplicate packets)
- *Transparent mode* (unchanged data transfer)

Possibility of several RLC instances at the same time!

Implementation in SDL

3GPP standard is given in a *mixture of formalisms*: plain English, annotated figures, bitmap-tables, state transition diagrams, message sequence charts, SDL diagrams





Formal CSP Specifications

CSP (Communicating Sequential Processes):

- Abstract description of *requirements*
- Especially designed to describe the behaviour of *parallel*, *communicating*, *embedded real-time* systems
- Operators reflecting the *structure* of requirements,
  e.g., *sequential* and *parallel* composition, *choice*, *iteration* and *hiding*
- Communication by (synchronous) *exchange of events*, real-time modelling with *timers*
- Rich and well-established *theory* ([Hoare 1978], [Hoare 1985], [Roscoe 1997])

Example: Vending machine

```
include "timers.csp"

pragma AM_INPUT
channel coin, buttonCoffee, buttonTea
nametype MonEv = { coin, buttonCoffee, buttonTea }

pragma AM_OUTPUT
channel coffee, tea
nametype CtrlEv = { coffee, tea }

OBSERVER = (  (coin -> HAVE_COIN)
           [] (buttonCoffee -> OBSERVER)
           [] (buttonTea -> OBSERVER))
HAVE_COIN = (  (coin -> HAVE_COIN))
            [] (buttonTea -> AWAIT({tea}) ; OBSERVER)
            [] (buttonCoffee -> AWAIT({coffee}); OBSERVER)

RANDOM_STIMULI = (|~| x: MonEv @ x -> PAUSE; RANDOM_STIMULI)

TEST_SPEC = RANDOM_STIMULI [| MonEv |] OBSERVER
```


CSP Testing Tool RT-Tester

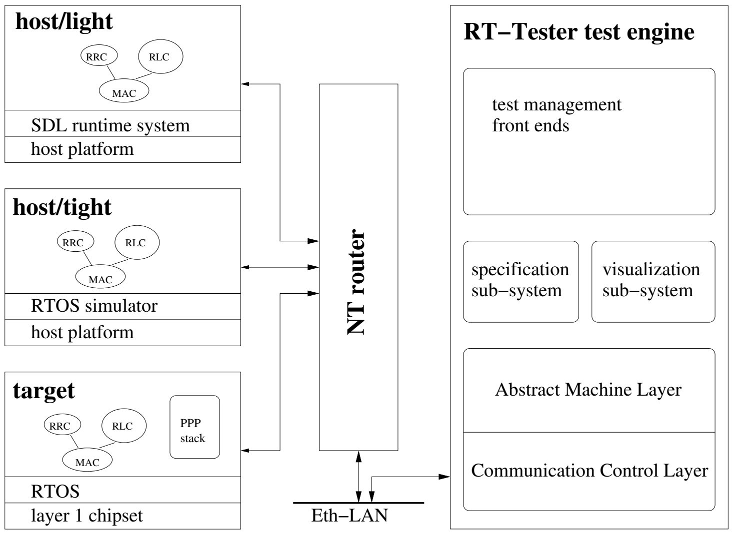
Joint development; now distributed by Verified Systems International GmbH, Bremen.

- Hardware-in-the-loop and component *black box tests*
- Tests of *arbitrary length* with steadily increasing coverage
- Testing of both *functional* and *hard real-time* properties
- Various specification and visualization possibilities

Execution of formal CSP test specifications proceeds in two stages:

- Automatic transformation of CSP input into *graph of admissible state transitions*
- *Generation*, *execution*, *monitoring* and *evaluation* of test sequences
  (automatically and in real time)

Configurations for Testing during Development



Dealing with a Moving Target

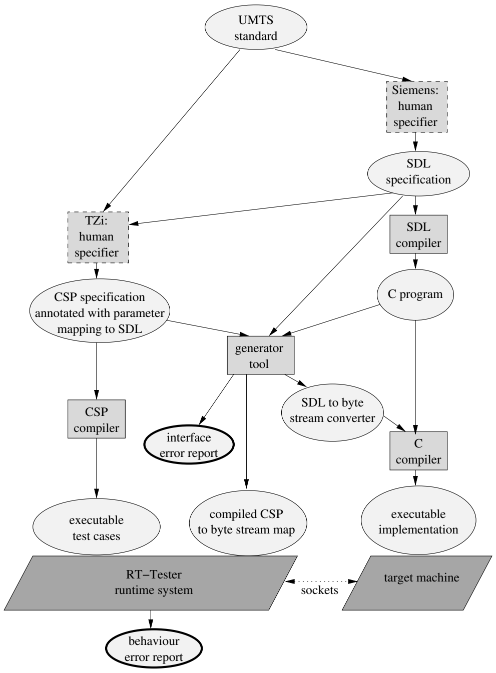
Problem: Both standard and implementation still subject to considerable change!

- Parameters of service primitives
- Details of internal data structures
- Behaviour of the protocol machines, in particular for error handling
- Machine representation of data at the interfaces

Solution: highly flexible testing environment

- *Interface definitions in terms of SDL* signals and data structures instead of low level descriptions,
- *Automated consistency check* between the SDL description of the interface and the formal CSP specification of the interface
- *Modularization* of formal behaviour specifications into largely independent functional requirements.

Automated interfacing of SDL and CSP



An example for matching CSP channels with SDL signals

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

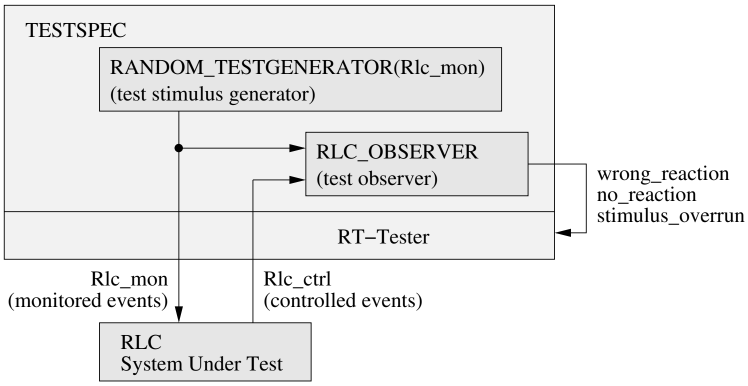
```
syntype RB_Identity = integer
        constants 0:MaxRb_Count
endsyntype;
synonym RLC_MAX_SDU_SIZE integer = 512;
newtype RLC_SDU_A
        carray(RLC_MAX_SDU_SIZE, octet)
endnewtype;
newtype RLC_SDU struct
        RB_Id RB_Identity;
        RLC_SDU_Data RLC_SDU_A;
        length integer;
endnewtype;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

Compiling Target and Test Scripts

- The mapping of *CSP events* to *byte strings* for the interface adapter in the RT-TESTER runtime system is produced automatically from the CSP specification.
- The *SDL specification* is compiled into a *C language program*, including C language header files and a *C language template file* for the input and output of SDL signals from and to the environment, respectively.
- This template is filled by automatically inserting code for the test interface adapter. In particular,
  - the *SDL signal* and data type definitions are *matched* with the annotated *CSP channel* parameters, and
  - the *byte representation* is determined from the generated C language header files and corresponding C language data types and parameter names
- The *template and C files* are compiled into an *executable implementation* for the target machine, and the *CSP specification* is compiled for the RT-TESTER *runtime system*.

Modular Structure of the Formal Behaviour Specification

Example: separation into *test stimulus generator* and *test observer*



Formal CSP Specification of RLC Connection Initialization

CSP code for part of the initialization of a connection in acknowledged mode:

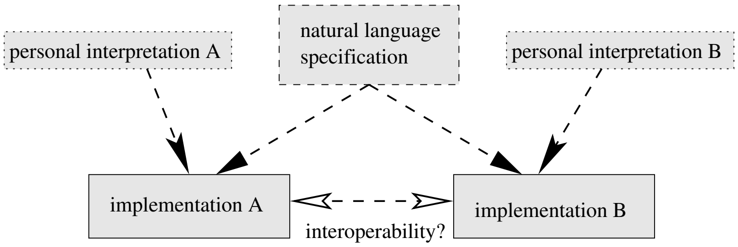
```
-- The null state of the RLC (AM) entity:
RLC_AM_NULL(instance_id) = instate_rlc_am_null.instance_id ->
(crlc_config_req.rbSetup.1.instance_id?dummy ->
        RLC_CONFIG_AM(instance_id,rlc_to_rrc)
 [] ([] x : diff(Rlc_mon,
            {| crlc_config_req.rbSetup.1.instance_id |}) @
        x -> RLC_AM_NULL(instance_id))
 [] ([] x : Rlc_ctrl @ x -> warn_spontaneous_event -> RLC_AM_NULL(instance_id))
)
```

Comment: The observer for the RLC instance with the number `instance_id` starts in the state `RLC_AM_NULL` and waits for a configuration request event. If the event occurs, the instance goes to the next state. All other events to the SUT are ignored. Any spontaneous output from the SUT would be an error and is flagged.

Management of Test Specifications

- **Separation of testing concerns:**
  - different layers of the UMTS protocol stack
  - data type definitions, channel definitions, and behaviour definitions
  - each property of the SUT in a separate requirements module
- **Implementation of various testing strategies via CSP:**
  - *completely random* selection of test stimuli
  - random choices with different *probabilities*
  - fixed, explicitly specified *trace* of events
- **Family of test specifications in a directory tree structure**
  - specification modules, stimulus generators, test suites etc. separated
  - need for intelligent configuration facilities

Testing Results 1: Ambiguities in the Standards Document



Example: In the implementation, a split of RLC SAP depending on the destination of a PDU was made (which is obvious but not standardized); as a consequence the RLC can be used only with MAC and upper layers which add the same parameter and which use the same SAP split.
Documentation of these design decisions can help in the integration phase.

Testing Results 1: Ambiguities in the Standards Document (cont.)

No precise definition of the properties of a SAP in the standard

- Buffer lengths, queueing discipline (FIFO, . . .)
- Queue delivery dependences between different SAPs
- Minimum/maximum delays between delivery and availability
- Handling of signals that cannot be received

Design decisions on these items might set a de-facto standard.
Documentation of the ambiguities can help to adapt to alternate implementations.

Testing Results 2: Deviations in the Behaviour

- Example for an *unexpected (spurious) reaction*:
  - an RLC protocol machine is created by the event `crlc_config_req.rbSetup` from the upper layers
  - becomes operational after receiving the event `mac_status_ind` from the underlying MAC layer
  - immediately after that, the test stimulus generator randomly generates another (nonsensical) `mac_status_ind` event
  - the RLC layer sometimes, but not always, reacted by generating a data packet, i.e., with a `mac_data_req` event
- There were also *interactions between different instances* of the RLC protocol machines; *deadlock situations* in spite of error recovery mechanisms appeared
- *Safe return* to an initial state was not always guaranteed

Summary

- For quality assurance, specification of *requirements* should be in a formal *(unambiguous)* language and *independent* from the implementation
- *Automatic generation and execution of tests* from these specifications with appropriate testing tools can reveal subtle (otherwise undetectable) errors
- *Automating the connection* between the *system under test* and the *requirements specification* allows maximal flexibility in the design process