Exploring Model-Driven Language Implementation of a Test Description Language

Florian Philipp, Philip Makedonski, Jens Grabowski

Georg-August-Universität Göttingen

Outline

Model-driven language implementation

Grammarware (terms after [KLV05])

- Traditional approach to language design
- Develop grammar first, derive data model from abstract syntax tree (AST)

Modelware

- Data model developed first
- Model as standardized common ground between tools
- Meta-model common ground between models
- Advantages:
  - Multiple representations (textual, graphical, binary)
  - Compatibility with other models (e.g. UML profile)
  - Reduced development cost through code generators, etc.

Test Description Language

This work:

- Model based methodology for TDL implementation
- Explores capabilities and limits
- Assessment of tools and frameworks
- Prototype for TDL
- Parallel to work of STF: Similar concept, different details

Outline

Behind the scenes

Based on Eclipse Modeling Framework (EMF) [SBMP08]

- Meta-model: EMF Ecore
- Model
  - Purely message based
  - Minimal type system
  - Test scenarios, reusable test steps
  - Structured interaction flow: loops, alternatives, threads
- Static semantics: Object Constraint Language (OCL) [Obj12]
- Operational semantics
  - Model-to-model transformation
  - System under test and I/O simulated
  - Implemented in Kermeta [JBF11]

Different representations

1. Textual syntax
    - Comprehensive
    - Usage: Authoring, documentation and discussion of details
    - Implementation: Eclipse Xtext [EV06]
2. Visualization
    - Based on UML sequence diagram / Message Sequence Chart (MSC)
    - Usage: Overview of interaction flow
    - Implementation: Xpand [Kla08] Model-to-text (M2T) transformation to SVG
3. Textual narrative
    - Usage: Short summary of interactions
    - Implementation: Xpand M2T transformation

Outline

Textual syntax

```
scenario Hello_World {
  import "Hello_World.tdt"
  system Echo_Service
  sequence {
    loop (4) {
      message hello { receive on port1 ref hello_world_content }
      alt {
        case {
          message echo {
            send on port1 after message hello range [,10] /*seconds*/
            ref hello_world_content
          }
        }
        case {
          message error { send on port1 ref error_content }
        }
      }
    }
  }
}
```

Listing 3.1: Extended Hello World example source code

Visualization

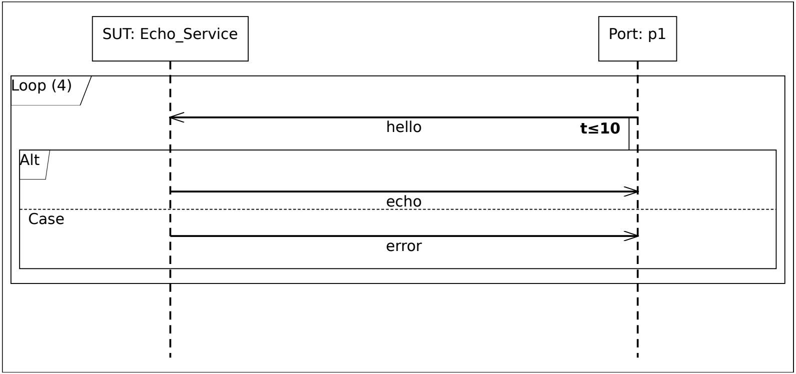

Figure 1 : Extended Hello World example diagram

Narrative

```
Receive hello_world_content then at most 10 s after
hello_world_content send hello_world_content.

Alternatively send error_content.

Repeat the last step 4 times.
```

Listing 3.2: Extended Hello World example narrative

Outline

Test case in a nutshell

Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 1, Section 7.2.3.9 "‘AM RLC / Polling for status"’ [3rd]

- Station Subsystem (SS $=$ tester) sends Protocol Data Units (PDUs)
- User Equipment (UE $=$ system under test) responds with PDUs
- Tester verifies responses and usage of the polling bit
- Repetition to test handling of time-outs and different configurations, etc.

Original test description

St	Procedure	Message Sequence		TP	Verdict
		U - S	Message		
0	During the whole test sequence, the SS should not allocate UL grants unless when explicitly stated so in the procedure.	-	-	-	-
1	The SS transmits 4 AMD PDUs such that 1 AMD PDU is sent every second radio frame, each containing an RLC SDU of 2 560 bits. (Note 2)	<--	AMD PDU (SN=0) AMD PDU (SN=1) AMD PDU (SN=2) AMD PDU (SN=3)	-	-
-	EXCEPTION: In parallel to the events described in step 1A, the step specified in Table 7.2.3.9.3.2-2 should take place.	-	-	-	-
1A	The SS waits for 100 ms after the first DL AMD PDU has been transmitted in step 1, then starts assigning UL grants (UL grant allocation type 2) in every second radio frame of size 2 600 bits. (Note 1) (Note 2)	-	-	-	-
2	Check 1: Does the UE transmit an AMD PDU with a SN in range 0 to 3 and P=1? Record time T_B. Check 2: Is ($T_B - T_A$) = *t-PollRetransmit*?	-->	AMD PDU	2	P
2A	The SS starts the UL default grant transmission	-	-	-	-
3	Upon receiving the Poll, the SS transmits an RLC Status Report.	<--	STATUS PDU	-	-
4	Check: Does the UE retransmit an AMD PDU within 1 sec?	-->	AMD PDU	2	F

Figure 2 : LTE test description excerpt (3 tables, 28 steps total)

TDL transcription

```
scenario AM_RLC_Polling_for_Status {
  import "AM_RLC.tdt"
  import "UL_Grant_Type2.tdl"
  /* more imports */
  system UE
  sequence {
    annotate {
      annotation "Table" : "7.2.3.9.3.2-1"
      annotation "Note" : "Allocate␣no␣UL␣grants␣unless␣stated"
      annotation "Time␣Unit" : "ms"
  }
  parallel {
    thread {
      annotate { annotation "Step" : "1" }
      loop (4 time range [20,29]) {
        template amd_pdu_sn0 {
          receive on SS ref AMD_PDU_RLC_SDU_2560
        }
      }
    }
    thread {
      annotate { annotation "Step" : "1A" }
      teststep UL_Grant_Type2 { after message amd_pdu_sn0 range [100,] }
    }
  /* continues: 206 lines of code */
```

Listing 4.1: 3GPP LTE source code

Visualization

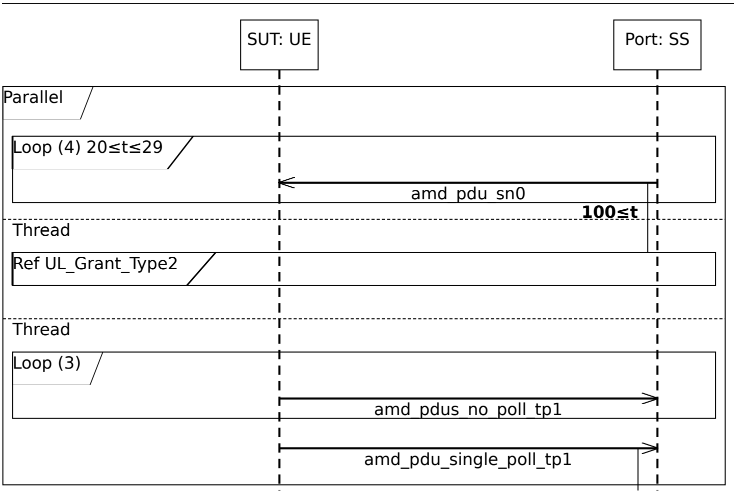

Figure 3 : 3GPP LTE diagram excerpt

Narrative

```
Receive amd_pdu_sn0 4 times every 20 till 29 ms.
In parallel at least 100 ms after amd_pdu_sn0 execute UL_Grant_Type2.
In parallel send amd_pdus_no_poll_tp1 3 times then send
amd_pdu_single_poll_tp1 (TP: 1).

Between 400 ms and 420 ms after amd_pdu_single_poll_tp1 send
amd_pdu_step2 (TP: 2).
In parallel execute UL_Grant_Default.

Receive rlc_status_after_poll.

[continues: 42 lines]
```

Listing 4.2: 3GPP LTE narrative

Conclusion

- Prototypical implementation of TDL successful
  - EMF Ecore-based model
  - Reference implementation of interpreter
  - Xtext-based concrete syntax
  - SVG-based visualization
  - Textual narrative serialization in prose format
- Approach tested on LTE and IMS test descriptions
  - On the whole successful
  - Precise reproduction limited by missing parts of model, not approach itself
- Open issues
  - Graphical concrete syntax and editor
  - Viable alternatives to Kermeta for operational semantics
  - Alignment with result of TDL standardization

Literature I

[3rd] 3rd Generation Partnership Project (3GPP).
Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 1: Protocol conformance specification (Release 11).
3GPP specification.
`http://www.3gpp.org/ftp/Specs/2012-12/Rel-11/36_series/36523-1-b11.zip`.

[EV06] Sven Efftinge and Markus Völter.
oAW xText: A framework for textual DSLs.
In *Workshop on Modeling Symposium at Eclipse Summit*, volume 32, 2006.

[JBF11] Jean-Marc Jézéquel, Olivier Barais, and Franck Fleurey.
Model driven language engineering with kermeta.
In *Generative and Transformational Techniques in Software Engineering III*, pages 201–221. Springer, 2011.

Literature II

[Kla08] B. Klatt.
Xpand: A closer look at the model2text tranformation language.
In *12th European Conference on Software Maintenance and Reengineering*, 2008.

[KLV05] Paul Klint, Ralf Lämmel, and Chris Verhoef.
Toward an engineering discipline for grammarware.
ACM Transactions on Software Engineering and Methodology (TOSEM), 14(3):331–380, 2005.

[Obj12] Object Management Group (OMG).
Object constraint language, v2.3.1.
ISO adopted standard, 2012.
ISO 19507:2012 `http://www.omg.org/spec/OCL/2.3.1`.

[SBMP08] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro.
EMF: Eclipse Modeling Framework.
Pearson Education, 2008.

IMS NNI case study

IMS NNI Interoperability: Simple trace

Step	Direction									Message	Comment
	User A	UE A	User B	UE B	IMS A	IBCF A	IBCF B	IMS B	AS B		
5						←				**100 Trying**	**IBCF_A responds with a 100 Trying provisional response**
6						→				**INVITE**	**IBCF_A forwards INVITE to IBCF_B**
7							←			**100 Trying**	**IBCF_B responds with a 100 Trying provisional response**
8							→			**INVITE**	**IMS_A forwards INVITE to IMS_B**
9								←		100 Trying	IMS_B responds with a 100 Trying provisional response
											INVITE triggers the OIP IFC in IMS_B
10								→		**INVITE**	**IMS_B forwards the INVITE to IMS_B AS**
11									←	100 Trying	AS optionally responds with a 100 Trying provisional response
12									←	INVITE	IMS_B AS returns, possibly modified, INVITE to IMS_B
13								→		100 Trying	IMS_B responds with a 100 Trying provisional response
14								←		INVITE	IMS_B forwards the INVITE to IMS_A
15							→			100 Trying	IMS_A responds with a 100 Trying provisional response
16							←			INVITE	IMS_B forwards INVITE to IBCF_B
17						→				100 Trying	IBCF_A responds with a 100 Trying provisional response

Figure 4 : IMS test description excerpt (79 steps total)

TDL transcription

```
scenario UC_08_R {
  import "IMS.tdt"
  system UserB
  sequence {
    annotate { annotation "Table" : "4.4.7.3" }
    message INVITE_Step_6 {
      receive on IBCFB
      annotate { annotation "Step" : "6" }
      ref INVITE
    }
    message TRYING_provisional_response_Step_7 {
      send on IBCFB
      annotate { annotation "Step" : "7" }
      ref TRYING_100
    }
    message INVITE_forward_Step_18 {
      send on IBCFB
      annotate { annotation "Step" : "18" }
      ref INVITE
    }
  /* continues: 125 lines of code */
```

Listing 6.1: IMS NNI source code

Visualization

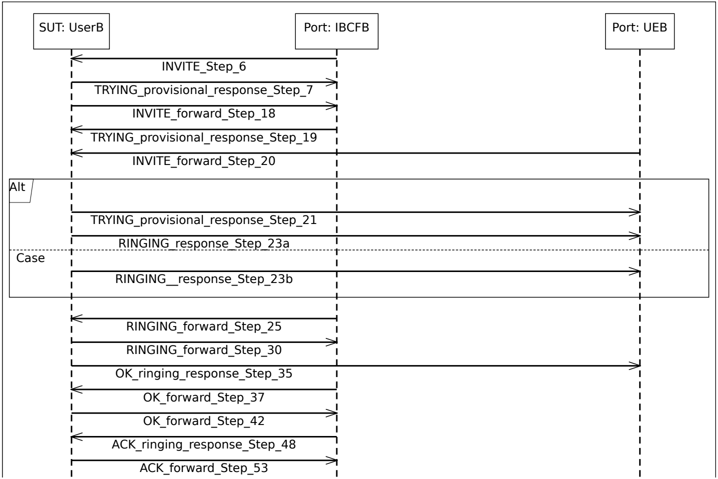

Figure 5 : IMS NNI diagram excerpt

Narrative

```
Receive INVITE on IBCFB.

Send TRYING_100 on IBCFB.

Send INVITE on IBCFB.

Receive TRYING_100 on IBCFB.

Receive INVITE on UEB.

Send TRYING_100 on UEB then send RINGING_180 on UEB. Alternatively send
RINGING_180 on UEB.

[Continues: 21 lines]
```

Listing 6.2: IMS NNI narrative