Enabling Interface Validation through Text Generation

Håkan Burden, Rogardt Heldal & Peter Ljunglöf
Computer Science and Engineering
Gothenburg, Sweden
How to validate an interface?
MicroOven

**setTimer** (time:int)

**setTimer** (min:int, sec:int)

cancel()

**confirmTimer**()

**startCooking**()

**rejectCooking**()

**confirmStartCooking**()

**confirmRejectCooking**()
How to validate an interface?

Use a state machine!
(It’s formal - It’s UML)
When a new timer is generated, the timer is confirmed. If a random boolean is true, then cooking is started; otherwise, it is rejected.

If cooking is rejected, it can be confirmed or failed. If cooking is started, it can be confirmed or failed.
Not always that easy

1. Competence in tools
2. Competence in models
3. Competence in paradigm

Arlow, Emmerich and Quinn, 1999,
“Literate Modelling
– Capturing Business Knowledge with the UML”
Even software developers struggle
• 25 interviews at Ericsson, Volvo Cars and the Volvo Group
Q: “So do you overload the interface? Throw in a signal just in case?”

A: “Yes, that is what we do. At least I do it ... and then you end up with the problem of not knowing which signal it is you should actually use.”
“The tools are too unintuitive ... the threshold for learning how to use them is high”
Prototype solution

Bridgepoint UML tool:

- Components
- Interfaces
- State machines
- **Model transformation language**
Natural Language Generation

“everybody knows how to consume text”
GenerateTimer:

time = randomInt;
setTimer(time);

ValidateTimer:

...
01 .select many definedSignals
02 related by interface -> C_EP[R4003] -> C_AS[R4004]
03 <table border="0">
04 <hr>Unused signals in MicroOven:</hr>
05 .assign unusedSignals = definedSignals - usedSignals
06 .if (not_empty unusedSignals)
07 .for each signal in unusedSignals
08 .invoke paramText = GetParamData
09 <tr><td><i>${signal.name (paramText)}</i></td></tr>
10 .end for
11 .else
12 <b>All defined signals are used.</b>
13 .end if
Intended usage of MicroOven:

1: **Tester** sends `setTimer(time:int)`

*MicroOven* responds with `confirmTimer()`

2: **Tester** sends `startCooking()`

*MicroOven* responds with `confirmStartCooking()`  
*MicroOven* responds with `confirmRejectCooking()`

Unused signals in *MicroOven*:

`setTimer(min:int, sec:int)`

`cancel()`
Conclusions

Intended usage of MicroOven:
1: Tester sends setTimer(time:int)
   MicroOven responds with confirmTimer()
2: Tester sends startCooking()
   MicroOven responds with confirmStartCooking() 2: Tester sends rejectCooking()
   MicroOven responds with confirmRejectCooking()

Unused signals in MicroOven:
   setTimer(min:int, sec:int)
   cancel()

Models and tools challenging for SE
Prototype: NLG from interfaces
Future Work

Intended usage of MicroOven:
1: Tester sends setTimer(time:int)
MicroOven responds with confirmTimer()
2: Tester sends startCooking()
   2: Tester sends rejectCooking()
MicroOven responds with confirmStartCooking() MicroOven responds with confirmRejectCooking()

Unused signals in MicroOven:
setTimer(min:int, sec:int)
cancel()