Analytical SW Design in Philips Healthcare

Erik Oerlemans
June 2009
Introduction

- The ASD methodology
- Where used?
- The benefits (quality++)
- Experiences (good/bad)
- Compliance Testing and Certification

Unfortunately no qualitative figures at this moment
The ASD methodology

- Specification of behavior of control SW
- Based on academic analytical models
- Theory hidden behind practical tools
- Formal specifications → opportunities for:
  - automated model consistency checks
  - automated code generation
  - automated test cases generation
The ASD methodology
The ASD methodology

Stepwise refinement
The ASD methodology

Offered interface

(Sub)Component

<<Behavior>>

Used interfaces

(Sub)Component

(Sub)Component

ASD spec for offered interface

ASD design for Component

ASD specs for used interface

Model Checker Tool

Check consistency

Check consistency
The ASD methodology

Code Generator Tool

Offered interface

ASD spec for offered interface

Generate test cases

C#

ASD design for Component

Generate C# code

C#

ASD specs for used interface

Generate test cases

C#

(Sub)Component

<<Behavior>>

(Sub)Component

(Sub)Component

(Sub)Component
The ASD methodology

• Sequence based specification:
  – Start with initial state
  – List all possible stimuli
  – Define per stimulus the response and the next state
  – Repeat for all new states until no more new states found
  – Separate control from data (from beginning)
The ASD methodology

<table>
<thead>
<tr>
<th>State / Stimulus</th>
<th>Response</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: init</td>
<td>StartFailed</td>
<td>0: init</td>
</tr>
<tr>
<td>Start</td>
<td>StartFailed</td>
<td>0: init</td>
</tr>
<tr>
<td>Start</td>
<td>Starting</td>
<td>1: Starting</td>
</tr>
<tr>
<td>INT</td>
<td>Blocked</td>
<td>-</td>
</tr>
<tr>
<td>1: Starting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>Illegal</td>
<td>-</td>
</tr>
<tr>
<td>INT</td>
<td>Started</td>
<td>2: Started</td>
</tr>
<tr>
<td>INT</td>
<td>StartFailed</td>
<td>0: init</td>
</tr>
<tr>
<td>2: Started</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>Illegal</td>
<td>-</td>
</tr>
<tr>
<td>INT</td>
<td>Blocked</td>
<td>-</td>
</tr>
</tbody>
</table>
The ASD methodology

**Client**
- Response = return value
- Stimulus = API call

**Server**
- Response = Call-back
- Stimulus = INT Server event

**Component**
- Decoupling Queue

**Diagram Notes:**
- Stimulus = API call
- Response = return value
- Response = Call-back
- Stimulus = INT Server event
The ASD methodology

ASD Specs (Intf, Dsgn, UM) → Convertor → CSP specs → Model Checker → All possible paths checked

CSP specs → Convertor → TML specs → Sequence Extractor → Sequences → Code Generator → C# implementation

CSP specs → Model Checker

C# Code Generator

Test case programs
Where used?

Front-End

BE-FE control protocol

Suitable for ASD

FE-IP control protocol
suitable for ASD

Image stream
not suitable for ASD

Back-End

BE-IP control protocol
suitable for ASD

Image stream
not suitable for ASD
Where used?

Front-End

BE Design model

Model Check

Interface model

BE-FE control protocol

IP

Back-End

Code Generator

C# Back-End SW and test case programs

Currently about 5% of BE SW is ASD generated code
ASD in BE, some figures

- 37 ASD interfaces
- 8 ASD components (designs)
- In total over all designs:
  - 320 states
  - 210 stimuli
  - 325 responses
- Total generated code: 18000 Lines Of Code
- Total checked execution paths: 140 million

Suppose you needed to test these paths manually …..
The benefits

Quality ++, efficiency +/- (due to learning)

+ impossible to forget something
+ mistakes are found by Model Checker
+ code is consistent with interface spec
+ test cases are consistent with interface spec
+ incremental development is possible

- all stimuli/responses must be known at design time (per increment)
- integration with legacy code is extra work
  - Too small → too much overhead: Select large part for ASD
- no graphical editor (yet)
- Long learning curve (after that, efficiency ++)

+ Supplier is working on tool improvement
Experiences

• Specification
• Model checking
• Consistency with Word document
• Code generation
Experiences

• Specification
  – System design phase → many details
  – Dev effort shifts to early phase → no waterfall model
  – Work very precise and detailed (attitude change)
  – Restrict yourself to design patterns that allow model checking (client/server, queue, static component creation, …) → attitude change
  – Tooling is improving steadily
Experiences

• Model checking
  – Partly integrated with ASD editor (separate srv)
  – Very cryptic error messages (CSP based)
  – Initially much support from supplier is required, long learning curve
  – Millions of paths → long processing time (hours)
  – Finds every single detailed error
  – Model check time not in line with perceived complexity (stateless messages)
  – Tooling will be improved coming years
Experiences

• Consistency with Word document
  – ASD file needs a descriptive document (Word)
  – Concepts, requirements, parameters
  – Keep consistent manually
  – Iterative:
    • Word → ASD → ModelCheck → ASD → Word
  – Small steps
  – Data model only in Word doc
    • No state behaviour depending on param values
Experiences

• Code generation
  – Fast (comparable with C# compilation times)
  – Satisfies static rules checker (TICS)
  – Correct (no errors found during testing)
  – Uses 1 thread per component (for queue handling); to avoid deadlocks
  – Supports C, C++, C# (Java under development)
  – All infrastructure code automatically generated (e.g. thread handling, component creation, register callbacks)
ASD in Philips Healthcare - summary

- Used for formal interface specification
- Used for design models + code generation
- Used in small, yet significant part of SW
- Method guarantees high quality SW
- Tooling must be and will be improved
- Will be used in more projects this year
- Upscale is needed to increase efficiency
Thank you for your attention.

Questions?
Compliance Testing and Certification

• Formal
  – One test environment
  – Consistent with interface specification
• Independent organization
• Certificates
  – Deliverable to system group
  – Prerequisite for system integration
  – Centrally maintained
Compliance Testing of Back-End

Compliance Test Framework

- ASD interface specifications
- Test Case Generator
- Test Cases
- Subsystem interface adapters (ASD)
- Logging
- Test Engine

Back-End

- BE Test interface
- IBeFe
- IBeIp
- BE Software

Philips Healthcare Best, Business Unit CardioVascular X-Ray, Erik Oerlemans, June 2009
Compliance Testing tool chain

- Under construction (coop Verum – Philips)
- Test Case Generator
  - Based on statistical testing
  - # possible scenarios = infinite
  - Test time = limited
  - Select set of scenarios that gives statistical value for
    - Reliability (probability of no errors in field)
    - Confidence (how sure about reliability)
  - High values → many test cases (thousands)
Compliance Testing - Experiences

- Usage Model
- Tool chain
- Test environment (exec, adapters)
Compliance Testing - Experiences

• Usage Model
  – Make once per subsystem
  – Consider all possible interleavings of messages over all interfaces → much manual work
  – Behavior must be made deterministic (interface spec is not deterministic)
  – Test interface follows from Usage Model
  – Start with “happy flow” and add exceptional paths in incremental steps
  – Model checking two ways: Correct & Complete

• Tool chain
• Test environment (exec, adapters)
Compliance Testing - Experiences

• Usage Model

• Tool chain
  – Made once, reused for all subsystems
  – Prototype available, improved version in 2010

• Test environment (exec, adapters)
Compliance Testing - Experiences

• Usage Model
• Tool chain
• Test environment (exec, adapters)
  – Separate version per subsystem
  – Available for BE, in progress for FE