Executable UML for Model Driven Architecture

Raising the level of abstraction

Executable UML update

- Raising the level of abstraction – Some history & benefits.
- Executable UML and Model Driven Architecture (MDA)
- Open translation of Executable UML models.
  (= model compilers)
- More of everything!!! ...with less effort = lower costs ("extra all" 😊)

Raising the level of abstraction

- PMI's becomes reusable large-scale components, possible to reuse across different platforms.
- Our Model Compilers becomes reusable large-scale assets for our platforms, our competitive advantage.
- Executable UML models are runnable and testable without generating any code (~like PowerPoint).
- Platform Independent Models (PIM) translated to target code by Model Compilers for different platforms.
- The Shlaer-Mellor Method for OOA evolves to Executable UML and an Action Language for UML.
- Model Driven Architecture (MDA) introduced. Define the importance of PMI vs. PIM evaluation.
- Assembly code dominates the embedded arena.
- Models get tied to the underlying execution environment (platform), hard to reuse.
- Increase of Model Based Development, followed by Java, Ada95 and C++ (planning some pictures and then write all code by hand).
- Writing & maintaining software gets more efficient.
- SW programming tooling & compiler improvements. (more, quicker, better)

Higher levels of abstraction - Benefits

- Executable UML and MDA allows us to:
  - Express more functionality with less effort.
  - Focus on modeling functionality (not code).
  - Test more with less effort.
  - Reuse more with less effort.
  - Reduce the number of faults. (ref. snapshot TR’s)

- Fly higher, better, faster and safer:
  - Higher degree of automation.
  - Higher degree of reuse.
  - Higher quality. (ref. Covert for SW Quality & Security Analysis)

Higher degree of automation example

Sample Plex-C code in AXE

Sample model in Executable UML

- All required by the Plex-C designer is to mark the variable as reload.
- The Plex-C compiler and the AXE.
- The AXE re-casts the variable as reload.
- In AXE all data marked as reload is retained on system restart.
- All required by the model designer is to mark the variable as reload.
- The Model Compiler handles the rest.
- All data marked as reload is retained from the file system on system restart.
- Verified on CPP node Q2-09.
What is Executable UML?

- Executable UML is a graphical specification language, combining a well-defined subset of UML with executable action semantics and rules for timing.
- An evolutionary stage of the Shlaer-Mellor Method.
- Executable UML Specifications are platform independent, can be run, tested and debugged much like a program but long before any code is generated.
- Executable UML models are translated into design by application independent Model Compilers.
- Executable UML = executable models without generating code (= execute models right out of the repository).
- In Mentors xtUML: \( x = \text{Executable} \) and \( t = \text{Translatable} \)

Executable UML Specifications

- Executable UML
- Model Driven Architecture (MDA)

UML Forum - Robot Contest in Tokyo

- Robot contest in full swing
- UML Forum/Tokyo
- "Robots - the most advanced"
- Leon Starr - www.modelint.com

xtUML Translation according to MDA

- Platform Independent Models (PIM) are translated into design through Platform Specific Models (PSM).
- The application models are free from implementation (design) details and fully reusable across different existing and future platforms and for different SW/HW design alternatives.
- Application independent metamodel of the software architecture. The expert system for how to generate the most efficient code for the target platform. Embeds target language and platform expertise, best design practices etc. Part of the reusable design base for the target platform.
Translation to different platforms
Translating the same xtUML model using different Model Compilers
Erlang Plex-C for APZ 21230/-33
C/C++ for Windows, OSE & Linux Java

Translation to different platforms
VHDL for simulation in ModelSim

UDPSH on nxtOSEK LEGO
No room for UDPSH + TestBench + nxtOSEK in 64K RAM.

xtUML Modeling & Translation
xtUML Model Translation
Host or Target Platform
Platform Specific Code

Performance Benchmarking
- Performance is key!
- MDD is of limited use if it doesn’t meet the performance requirements (budget) for real-time critical applications.
- Low performance of the generated code drives the manufacturing costs for HW.
- This is the main reason to why performance benchmarking is one of the first things our Design Units do.
AAL2 Performance Comparison:
Hand-coded vs. BridgePoint xtUML (AAL2 Setup & Release)

Benchmarking newly developed SW
The Radical SW Pilot @ Ericsson PDU CPP

- A new SW function was developed in multiple tracks:
  - Hand-written C++ (synchronous design)
  - Modelled using Executable UML (Shlaer-Mellor UML+AL)
  - Modelled using UML with C++ (not ready yet)

- The C++ was functionally tested both on host and target.

- The Executable UML model was developed in BridgePoint xtUML and was functionally tested on host using a modelled test environment of Scenario Players controlled by a Conductor.

- 100% of the code was generated from the xtUML model.
  (some minor glue code was written by hand)

Model Compiler Enhancements
Ericsson Proprietary Model Compiler

- AAL2 Pilot enhancements:
  - Architectural optimizations (reduces memory costs)
  - Improved state machine scheduling (improves performance)
  - Support for Linux and nxtOSEK

- Radical SW Pilot enhancements:
  - Auto-alignment of data struct members (reduces memory costs)
  - Single job execution (reduces memory costs)
  - Architectural optimizations (reduces memory costs)
  - "Follow the leader" scheduling (improves performance)
  - Synchronous state machine execution (improves performance)
  - Support for Model Based Test Coverage

UDPSH BridgePoint xtUML version:
Model Based Test Coverage

Tested using Model Based Regression Test
(modelled test scenario players)

The else-branch coverage indicates the amount of alternative execution paths tested in the Action Language code.
## UDP/SH Traffice Load Test Cases

<table>
<thead>
<tr>
<th>Test #</th>
<th>Pre-condition 1</th>
<th>Pre-condition 2</th>
<th>Load Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P2a: sess in new gr</td>
<td>TC1: 1500 sess/sec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>P2b: sess in exist gr</td>
<td>TC2: 2500 sess/sec</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>P3: 1500 clients</td>
<td>P2a: sess in new gr</td>
<td>TC1: 1500 sess/sec</td>
</tr>
<tr>
<td>4</td>
<td>P3: 1500 clients</td>
<td>P2b: sess in exist gr</td>
<td>TC2: 2500 sess/sec</td>
</tr>
<tr>
<td>5</td>
<td>5 clients in 50 groups</td>
<td>P2a: sess in new gr</td>
<td>TC1: 1500 sess/sec</td>
</tr>
<tr>
<td>6</td>
<td>5 clients in 50 groups</td>
<td>P2b: sess in exist gr</td>
<td>TC2: 2500 sess/sec</td>
</tr>
<tr>
<td>7</td>
<td>5 clients in 1500 groups</td>
<td>P2a: sess in new gr</td>
<td>TC1: 1500 sess/sec</td>
</tr>
<tr>
<td>8</td>
<td>5 clients in 1500 groups</td>
<td>P2b: sess in exist gr</td>
<td>TC2: 2500 sess/sec</td>
</tr>
<tr>
<td>9</td>
<td>5 clients in 3000 groups</td>
<td>P2a: sess in new gr</td>
<td>TC1: 1500 sess/sec</td>
</tr>
<tr>
<td>10</td>
<td>5 clients in 3000 groups</td>
<td>P2b: sess in exist gr</td>
<td>TC2: 2500 sess/sec</td>
</tr>
</tbody>
</table>

## UDP/SH Traffic Load Test Case

5 clients setting up sessions in 10 different groups with 50 sessions each = 2500 sessions per sec.

<table>
<thead>
<tr>
<th>Client #</th>
<th>Pre-condition 1</th>
<th>Pre-condition 2</th>
<th>Load Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TC1: 1500 sess/sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TC2: 2500 sess/sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TC3: 600 sess/sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## UDP/SH Performance Benchmarking Test Case 3

5 clients setting up sessions in new groups, load is 2500 sessions/sec

Worst-case CPU load for these 10 test cases

- CPU Load
  - Hand-coded C++
  - BridgePoint UML+AL
  - UML with C++

## UDP/SH Performance Benchmarking Test Case 9

5 clients setting up sessions in new groups, load is 600 sessions/sec

Worst-case memory cost for these 10 test cases

- Memory Utilization:
  - Node 1
  - Node 2

## HOT POTATO test: Sending data packets between state machines

Full tests was performed on different HW platforms & OS'es. (PowerPC vs. Intel and OSE vs. Linux SW/HW platforms)

## HOT POTATO signalling benchmark: BridgePoint vs. a UML with C++ tool

Signalling between modelled state machines using 100 Byte packages.

(platform is 8572 with Monta Vista Linux)
HOT POTATO signalling benchmark: BridgePoint vs. a UMLwithC++ tool

Signalling between modelled state machines using 100 Byte packages.
(platform is ”monster machine” 5570/Sles/gcc64)

Recorded time in us

BridgePoint single thread
BridgePoint dual thread
UMLwithC++ dual thread
UMLwithC++ single thread

dual core
single core

HOT POTATO signalling benchmark:
BridgePoint vs. a UMLwithC++ tool

UMLwithC++ tool runtime call graph

Model Compiler for multi-core SW
Baseband Research @ Lindholmen

Research project just started:
Translation of xtUML models onto a new multi-core ASIC

Model Compiler for multi-core SW
Baseband Research @ Lindholmen

Research project just started:
Translation of xtUML models onto a new multi-core ASIC

The green parts are planned to be handled (automated) by the Model Compiler

More of everything with Executable UML & MDA
(”extra all” 😊 )

History of xtUML success stories

- RBS: PP Adapter – adapting existing RoseRT legacy to new HW platform
  - Developed by a trainee in parallel with an RBS hand-coding team.
  - Running all 30+ Use Cases on target in the Cello lab before the hand-coding team had written a single line of code. (= agile)

- RNC: Load Control – preventing overload situations in the RNC
  - Re-engineered by thesis students at RNC Design.
  - Worked on 1st attempt on target. (”has never happened before”)
  - Benchmark against existing 10 year old & optimized RoseRT solution.

- DUPL: The AAL2 pilot
  - Developed by senior engineers not programmed for ~15 years.
  - No faults on target.
  - Benchmarked against existing 10 year old & optimized solution.
  - Load module sent to Croatia for testing

- PDU CPP: Radical SW pilot – the UDPSH function
  - Worked on target right away.
  - Equals performance and memory utilization with the newly handwritten C++ version.
  - Has been run on different HW with OSE and Linux. (as a UDPSH + TestBench monolith)
History of xtUML success stories

- "Surprisingly" high quality of software.
  - Typically no faults on target.
  - Coverity did not find any problems in generated code.
- Equal in performance with hand-optimized code.
  - Results from benchmarking of CPU load.
- Re-factoring is easier than expected.
  - Need a modelled test bench for re-testing the re-factored model.
- Full control of the entire code generation process.
  - Flexibility to alter code generation whenever needed.
  - Not dependent on tool vendor.

Conclusions

- Increasing the level of abstraction not necessarily means getting a reduction in performance. (= old myth killed)
- It depends primarily on how good the Model Compiler is. (provided that the model not is "badly" designed)
- We must learn to "trust" & develop Model Compilers and learn how to develop & provide additional support & value for our proprietary platforms and reduce the costs for HW.
- It's time to stop just talking about it, because...

“It takes a whole new way of thinking to solve the problems that were created by the old way of thinking”

Albert Einstein