Evolving / Formalising / Automating an Embedded Real-Time Software Architecture

–

The Leap from Hand-Coding to Automatic Code Translation with Retained Control over the Code Generation Process

Erik Wedin
Senior Software Systems Architect
Saab Bofors Dynamics, Linköping, Sweden
www.saab.se/dynamics
Contents

• Saab Presentation

• Background & OOD Architecture Evolution

• The Transition to Open Code Translation

• Ada Model Compiler

• Experiences

• Questions & Answers
Saab is one of the world’s leading high-technology companies, with its main operations focusing on defence, aviation and space.

Employees 14 000
Turnover MSEK 17 000
Saab Bofors Dynamics

A vital part of Saab’s defence activities

<table>
<thead>
<tr>
<th>Employees</th>
<th>1,460</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turnover</td>
<td>SEK 2,850 million</td>
</tr>
<tr>
<td>Exports</td>
<td>SEK 2,000 million</td>
</tr>
<tr>
<td>Backlog of orders</td>
<td>SEK 8,800 million</td>
</tr>
<tr>
<td>Exports</td>
<td>SEK 7,300 million</td>
</tr>
</tbody>
</table>

Missiles

Support weapons

New Quality Standard certification for:
SS-EN ISO 9001: 2000 + Tick IT + AQAP110
ISO 14001
Complete Missiles Solution

- Develops advanced missile systems for the Swedish Defence Forces and other national defence forces
- Participates in international projects
System Characteristics

- Embedded real-time systems – but also planning systems and simulators
- Specially-designed hardware and computer platforms – but also COTS
- Autonomous systems ⇒ reliable, predictable
- Complex communications – internally and externally
- Application functionality
  - guidance, navigation and control; signal and image processing; data fusion ⇒
    - complex mathematics, computationally demanding
  - mission management
  - telemetry
- Hard real-time requirements – missed deadline ⇒ may result in loss of the system
  - communication & execution
- Periodic mixed-frequency execution – but also aperiodic
- Safety-critical – usually solved in hardware
How did it all start?
OOD Architecture Evolution – System I

- OOD Architecture Generation
  - G1
- Development Time-Frame ~1990-
- Analysis & Modelling
  - Shlaer Mellor OOA
  - Teamwork
- Implementation Approach
  - manual, rule-based
  - class templates
- Platform
  - embedded real-time
  - workstations and parallel computing platform
  - Ada run-time – Sun OS & another COTS OS
- OOD Architecture Profile
  - Ada83
  - monolithic (execution control and application tightly connected)
OOD Architecture Evolution – System II

- OOD Architecture Generation
  - G2
- Development Time-Frame ~1994-
- Analysis & Modelling
  - Shlaer Mellor OOA
  - Teamwork/ObjectTeam
- Implementation Approach
  - manual rule-based
  - class templates – aperiodic, periodic, passive
- Platform
  - embedded real-time on in-house hardware & COTS workstations
  - Ada run-time – bareboard & Sun OS/Solaris
- OOD Architecture Profile
  - Ada83
  - separation of execution control and application, periodic execution
OOD Architecture Evolution – System III

- OOD Architecture Generation
  - G3
- Development Time-Frame ~1997-
- Analysis & Modelling
  - Shlaer Mellor OOA
  - BridgePoint
- Implementation Approach
  - manual rule-based
  - class templates – aperiodic, periodic, passive
- Platform
  - embedded real-time on COTS hardware (demonstrator) & existing software architecture of another company (product)
  - Ada run-time – bareboard & simulation models on COTS HW and OS
- OOD Architecture Profile
  - Ada83 + Ada9x extensions
  - background processes, multiprocess protection
OOD Architecture Evolution – System IV

- OOD Architecture Generation
  - G4
- Development Time-Frame ~1998-
- Analysis & Modelling
  - Shlaer Mellor OOA (UML-notation)
  - BridgePoint
  - successful separation of subject matters ⇒ IO, communication, execution, application
- Implementation Approach
  - manual rule-based
  - class templates – aperiodic, periodic, passive
- Platform
  - embedded real-time on in-house hardware
  - multiple processors
  - no run-time – bareboard & several simulation models on COTS HW and OS
  - extreme timing requirements
- OOD Architecture Profile
  - ANSI C
  - multiprocessor distribution, multiprocess protection
Experiences –
Inconsistent Software Development Information

Documents

Source Code

Models

RT-R04:8010(Ö) Issue 2

UNCLASSIFIED
© 2004 Saab Bofors Dynamics AB
eriwe/RTPL Erik Wedin
2004-10-25

This document and the information contained herein is the property of Saab Bofors Dynamics AB and must not be used, disclosed or altered without Saab Bofors Dynamics AB prior written consent.
Experiences – Multi-Platform Development

**System Specification**

**Host Environment**

**Build Tools**
- Compiler
- Application SW
- RTS
- SUN Solaris
- SUN Sparc Workstation
- Network

**Software Platform**
- Implementation:
  - Compiler
  - Application SW
  - RTS
  - SUN Solaris
  - SUN Sparc Workstation

**Hardware Platform**
- Network

**Target Environment**

**Compiler**
- Application SW
- RTS
- SUN Solaris
- SUN Sparc Workstation
- Network

**Repartitioning of Functionality Between Processors**
- FPGA
- PowerPC
- Memory
- PCI

**Manual Rule-Based Translation**

This document and the information contained herein is the property of Saab Bofors Dynamics AB and must not be used, disclosed or altered without Saab Bofors Dynamics AB prior written consent.
Translation Study Rationale

- Exploit previous good project-experiences and fix found problem areas
- Remove tedious/boring/error prone hand-translation of OOA-models
- Presumably small step to automate the manual implementation approach with the existing OOD Architecture
- Optimise OOD Architecture reuse
- Raise the level of abstraction
- Enable domain reuse
Automatic Code Generation Requirements

- **Source code representation**
  - readable code
    - shall look like hand-coded
  - clear connection between source code and models
    - eases the step to the translation technology – design (model) ↔ implementation (code) ↔ executing system
  - full control over generation process – “Down to the last semicolon”
    - control all features of the architecture to match critical system requirements

- **Translation approach**
  - everything should be generated, not only code skeletons
  - ability to mix hand-coded code and automatic approach
  - ability to go back to rule-based hand-code approach – without having to rewrite the code
    - risk reduction: convince management to use this approach; tool support failure

- **Long-term maintenance issues**
  - the code will be delivered to customers without ability to regenerate the code
  - long-term maintenance may be on a code-level
  - i.e. see points above about Source code representation and Translation approach
Ada Model Compiler
Development Starting Point

- Class templates
  - Passive
  - Aperiodic
  - Periodic
- Implementation rules
- Ada mechanisms
  - Execution Control
  - Time
  - Class Mechanisms
- Architectural design concept and documentation

**Identical architecture for ANSI-C**
OOD Architecture Evolution – Translation Study Result

- OOD Architecture Generation
  - G5
- Development Time-Frame ~2001-2002
- Analysis & Modelling
  - Executable UML
  - BridgePoint
- Implementation Approach
  - automated manual coding
- Platform
  - Ada run-time – workstation/SUN Solaris & PC/Windows
- OOD Architecture Profile
  - Ada95
  - OOD-architecture metamodel
  - full relationship support added (previously in attributes)
  - non-optimised – may generate unused code
  - support for Executable UML

Signal Handling <<realised>>

Missile Management

Missile Test

Digital IO

Simulated Environment

<<realised>> Model Compiler

Ada

Pilot Missile System
Metamodels

Extended Architecture Metamodel

Extended Executable UML Metamodel

Interaction Metamodel
(Developed by Saab Bofors Dynamics)

Executable UML Metamodel
(Delivered with BridgePoint)

(Implicit) Bridging Metamodel
(Developed by Saab Bofors Dynamics)

Relationships between metamodel classes

Mapping

OOD Architecture Metamodel
(Developed by Saab Bofors Dynamics)
Interaction Metamodel
Interaction Metamodel

Mapping to Executable UML Metamodel
Architecture Metamodel
Architecture Metamodell – OOD System Subsystem

- Execution Ordering
  - Process order within Program
  - Class order within Process

Execution Ordering

- Process order within Program
- Class order within Process

RT-R04:8010(Ö) Issue 2
UNCLASSIFIED
© 2004 Saab Bofors Dynamics AB
erwe/RTPL Erik Wedin
2004-10-25

This document and the information contained herein is the property of Saab Bofors Dynamics AB and must not be used, disclosed or altered without Saab Bofors Dynamics AB prior written consent.
Bridging Metamodel

<<Subsystem>>
Bridging {BR, 100-111}

<<Subsystem>>
OOA_of_OOA_Classes {OOA, 1-9}
System Design – Marking Tool
Model Compilation

- Executable UML modelling
  - each domain model is exported from BridgePoint Model Builder

- Static system structure creation
  - predefined directory structure
  - generation of user-specified marking files

- Generation database creation
  - structure from the Executable UML Metamodel, the OOD Architecture Metamodel, the Interaction Metamodel and the Bridging Metamodel
  - each domain model populates the Executable UML Metamodel

- Model consistency check
  - checks that the model constructs are supported by the model compiler

- Implicit bridging
  - mapping of Executable UML Metamodel instances to Executable UML Metamodel instances from different domains
  - resulting in fully populated Bridging Metamodel
Model Compilation

• Implicit marking – Phase 1
  – insertion of built-in non-deployment/non-partitioning dependent architecture design decisions
  – basic population of the OOD Architecture Metamodel
  – mapping Executable UML Metamodel instances to OOD Architecture Metamodel instances

• First pass action translation
  – analysis of
    ▪ interactions – attribute access, operation invocation, event generation, relationship navigation, instance creation/deletion, instance selection, synchronous service invocation, external entity event generation, external entity bridge invocation, …
    ▪ dependencies, data type usage, …
  – for the purpose of model compiler optimisations
  – populates the Interaction Metammodel and the OOD Architecture Metammodel
  – resulting in fully populated Interaction Metammodel
Model Compilation

- **Explicit marking – Phase 1**
  - insertion of user-specified deployment/partitioning design decisions
  - populates the OOD Architecture Metamodell

- **Implicit marking – Phase 2**
  - insertion of built-in deployment/partitioning dependent architecture design decisions
  - mapping Executable UML Metamodel instances to OOD Architecture Metamodel instances
  - populates the OOD Architecture Metamodell

- **Explicit marking – Phase 2**
  - insertion of user-specified deployment/partitioning design decisions
  - populates the OOD Architecture Metamodell
  - resulting in fully populated OOD Architecture Metamodell
Model Compilation

- Dynamic system structure creation
  - performed if necessary
  - directory structure based on system deployment marking
- Code generation
  - structural translation
  - second pass action translation
- Report generation
- Ada compilation

- Supported platforms
  - ACT GNAT Pro
    - PC – Windows
    - PowerPC – Bareboard
    - SUN Sparc – Solaris
  - Green Hills AdaMULTI
    - PowerPC – Bareboard
    - PowerPC – Integrity (RTOS)
Resulting System Directory Structure
Pilot Missile System – Guidance Class

Guidance

G_G: unique_id
Demanded_X_Acceleration: real
Demanded_Y_Acceleration: real
Update_Timer: inst_ref<Timer>
current_state: state<State_Model>

Calculate_Demanded_Accelerations(X:real,Y:real): void

1. Idle
   entry/
   G_G1:Start_Guidance [Initial_X_Acc, Initial_Y_Acc]

2. Initialization
   entry/
   self.Demanded_X_Acceleration = rcvd_evt.Initial_X_Acc;
   self.Demanded_Y_Acceleration = rcvd_evt.Initial_Y_Acc;
   create event instance Update_Guidance of G_G2:Update_Guidance() to self;
   self.Update_Timer = TIM:timer_start_recurring
      (microseconds : 200000,
       event_inst : Update_Guidance);
   generate G_G2:Update_Guidance() to self;

3. Free_Flight
   entry/
   G_G2:Calculate_Demanded_Accelerations
      (X : self.Demanded_X_Acceleration,
       Y : self.Demanded_Y_Acceleration);
   G_G2:Update_Guidance

4. Autodestruct
   entry/
   Result = TIM:timer_cancel(timer_inst_ref : self.Update_Timer);
   // Not modelled: Perform some autodestruction manoeuvre.
   G_G3:Autodestruct

RT-R04:8010(Ö) Issue 2

UNCLASSIFIED
© 2004 Saab Bofors Dynamics AB
erwe/RTPL Erik Wedin
2004-10-25

This document and the information contained herein is the property of Saab Bofors Dynamics AB and must not be used, disclosed or altered without Saab Bofors Dynamics AB prior written consent.
package MM_Process.PM_Missile_Management.Guidance is

-- Exported Constants and Types
----------------------------------------------
type Event_Data_Structure is
record
  Initial_Y_Acc : PM_Missile_Management_Domain.Acceleration_Range := 0.0;
  Initial_X_Acc : PM_Missile_Management_Domain.Acceleration_Range := 0.0;
end record;

-- OOD Constants and Types
----------------------------------------------
No_Timer     : constant         := 0;
Maximum_Number_Of_Timers : constant Natural := 1;

type Timer_Identification is range
No_Timer .. Maximum_Number_Of_Timers;

type Instance_Event_Name is
(Update_Guidance, Start_Guidance, Autodestruct, No_Event);

type Instance_Event is
record
  Name : Instance_Event_Name := No_Event;
  Data : Guidance.Event_Data_Structure;
end record;

type Instance_State is
(Initialization, Free_Flight, Idle, Autodestruction);

type Instance_Attributes is
record
  Demanded_X_Acceleration : Standard_Types.Real := 0.0;
  Demanded_Y_Acceleration : Standard_Types.Real := 0.0;
  Update_Timer : Timer_Identification := No_Timer;
end record;

type Instance_Attributes_Reference is access all
Instance_Attributes;

-- Exported Class Mechanism
package Instances is new
OOD.OOA_Instances
(Maximum_Number_Of_Instances => 1,
Instance_State => Instance_State,
Instance_Attributes => Instance_Attributes,
Instance_Attributes_Reference =>
  Instance_Attributes_Reference);

end MM_Process.PM_Missile_Management.Guidance;
Resulting Source Code – Guidance Class

-- Action Bodies

procedure Initialization_Action

(The_Event : in     Instance_Event;
  Instance  : in out Instances.Identification) is
  Attribute           : Instance_Attributes_Reference := Instances.Attributes(Instance);
  Update_Guidance_Evt : Instance_Event;
  Guidance_Event_Data : Event_Data_Structure;
begin
  Attribute.Demanded_X_Acceleration := The_Event.Data.Initial_X_Acc;
  Attribute.Demanded_Y_Acceleration := The_Event.Data.Initial_Y_Acc;

  Update_Guidance_Evt := (Update_Guidance, Guidance_Event_Data);

  Timer.Set_Recurring
    (At_Time => OOD.System_Time.Clock,
     Period => OOD.System_Time.To_Time_Span(Standard_Types.Real(200000)/1.0E6),
     With_Event => Update_Guidance_Evt,
     Timer => Attribute.Update_Timer,
     For_Instance => Instance);

  Event_Handler.Generate
    (Event => (Name => Update_Guidance,
                Data => Guidance_Event_Data),
     To_Instance => Instance);
end Initialization_Action;
System Execution & Analysis – Log Analysis Tool
OOD Architecture Evolution – System V

- OOD Architecture Generation
  - G6
- Development Time-Frame ~2003-
- Analysis & Modelling
  - system-level – Statemate models
  - software-level – Executable UML & BridgePoint
- Implementation Approach
  - Statemate models mapped to Executable UML models
  - open code translation from Executable UML
  - Ada95 Model Compiler
- Platform
  - existing software architecture of another company
  - safety-critical software
- OOD Architecture Profile
  - isolated from the safety-critical software parts
  - Ravenscar-compliant (limited tasking profile)
  - to be SPARK-compliant (Ada-subset & annotations ⇒ formal analysis & proofing)
MDA Metamodel-Perspective

Extended Executable UML Metamodel

- Statemate Metamodel
- Interaction Metamodel
- Executable UML Metamodel
- (Implicit) Bridging Metamodel

Mappings

Extended Executable UML Metamodel

- Mappings to Statemate to Executable UML Model Mapping Guidelines (Defined by Saab Bofors Dynamics)
- Mappings to Ada95 Model Compiler (Developed by Saab Bofors Dynamics)
- Mappings to Ada95 Compiler

Mappings

Extended Executable UML Metamodel

- Mappings to Ada95 Model Compiler (Developed by Saab Bofors Dynamics)
- Mappings to Ada95 Compiler

Extended Executable UML Metamodel

- Mappings to OOD Architecture Metamodel
- Mappings to Ada95 Metamodel

Extended Executable UML Metamodel

- Mappings to OOD Architecture Metamodel
- Mappings to Ada95 Metamodel

Extended Executable UML Metamodel

- Mappings to OOD Architecture Metamodel
- Mappings to Ada95 Metamodel

Extended Executable UML Metamodel

- Mappings to OOD Architecture Metamodel
- Mappings to Ada95 Metamodel

Extended Executable UML Metamodel

- Mappings to OOD Architecture Metamodel
- Mappings to Ada95 Metamodel

Extended Executable UML Metamodel

- Mappings to OOD Architecture Metamodel
- Mappings to Ada95 Metamodel
Executable UML and Open Code Translation Experiences

- Formalisation of our knowledge in building our systems – reuse
- Model Compiler design
  - the architecture is the demanding part – formalising/automating is pretty simple and straightforward
  - OOD-architecture already given
  - experiences from manual approach invaluable
  - full control over the code generation
  - relationships the hardest part – new design
  - updates – prototype code by hand, test and then formalise in archetypes
- Object Oriented Analysis and Modelling
  - the hardest part
- Integrated into the regular software development process MIL-STD-498 / ISO12207
  - mapping between Executable UML activities and MIL-STD-498/ISO12207
- Separation of analys and design
  - feels natural; in line with previous work
  - easy to add system-wide changes/optimisations
- Reverting to manual approach is simple
  - generated code looks like hand-coded
  - all generated code can be used
  - the models can be used
- Consistency in software information
  - documentation ↔ model ↔ code
  - same information – different representation
- Relatively easy to switch target language
  - architecture for ANSI C exists
  - architecture for VHDL studied
- Inclusion of hand-written code
  - tested and works; suitable for algorithms
Other Experiences

- Lacking efficient tool support for model verification
  - test harness definition
  - test panel (GUI) construction for interactive testing
  - test script construction for automated testing
- Executable UML lacks composite type support – records, arrays
  - natural way to think in, for software and systems engineers
    - e.g. a position (X, Y, Z) as an attribute in a class
    - vectors and matrices are always used
- Limitations in the action language
  - weak typing
  - definition of abstract data types – type and associated operations
  - definition of infix operators
- Integration into systems, methods and processes is a Key Issue!
- COTS model compiler MC2020 for C++ also used recently
Challenges to Introduce Executable UML

- **Software**
  - main-stream “full” UML often preferred – hierarchy, powerful state machines, …
  - affects – organisation, software development process, tool costs, training, code-centred view, …
  - “look-and-feel” and functionality of BridgePoint – not modern; does not support “full” Executable UML

- **Systems**
  - sometimes lacking usage of a well-defined development process
  - other tools and methodology
    - algorithmic-centred development tools – MATLAB, SimuLink, …
  - functional decomposition is “natural”, object-orientation is not
  - UML is considered to be a “low-level” software notation/language
    - “how can you model a complete hardware/software system in UML?”

- **Electronics**
  - sometimes lacking usage of a well-defined development process
  - functional decomposition is “natural”, object-orientation is not
  - sometimes struggling to reach a reasonably abstract level in VHDL
Challenges to Introduce Executable UML

- General
  - program workshare – total development responsibility ⇔ partial development responsibility
  - customer requirements
    - automatic code generation not allowed
    - intellectual property-claims to the Model Compiler and/or the OOD-architecture
    - ...
  - integration with software development and architecture approaches of partners
  - further development of existing systems – implemented in source code with “informally” modelled models or no models
    - Alt. I - reverse engineer code to Executable UML models
    - Alt. II - produce new Executable UML models of existing functionality
Vision: Multi-System Translation

System 1 Design
- Marking 1
- Bridge Mapping 1

System Specification
- Executable UML Domain

System 3 Design
- Marking 3
- Bridge Mapping 3

Software Architecture
- Ada MC Main + BP Gen
  - GNAT Pro
    - Application SW
      - GNAT Ada RTS
        - SUN Solaris
      - SUN Sparc Workstation
    - Network

Software Platform
- Hardware Platform
  - FPGA Application
    - FPGA
    - Memory

Build Tools
- Implementation
  - Ada/VHDL MC Msl + BP Gen

Implementation
- AdaMULTI
  - Application SW
    - GH Ada RTS
      - BSP
    - Memory
    - PowerPC
    - PCI
  - FPGA Application
    - Hardware

Precision
# Technology Studies and Technology Development

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>Ada code generation from Teamwork/ObjectTeam</td>
<td>MSc Thesis Work</td>
<td></td>
</tr>
<tr>
<td>1997</td>
<td>VHDL code generation from BridgePoint</td>
<td>MSc Thesis Work</td>
<td>✔️</td>
</tr>
<tr>
<td>2000</td>
<td>Development of a Log Analysis Tool (first try, Java)</td>
<td>Intern Work</td>
<td>✔️</td>
</tr>
<tr>
<td>2001</td>
<td>SMUG 2001</td>
<td>Shlaer/Mellor User Group</td>
<td>✔️</td>
</tr>
<tr>
<td>2001-2002</td>
<td>Open Code Translation from Executable UML Models</td>
<td>Internal Study</td>
<td>✔️</td>
</tr>
<tr>
<td>2002</td>
<td>SMUG 2002</td>
<td>Shlaer/Mellor User Group</td>
<td>✔️</td>
</tr>
<tr>
<td>2003</td>
<td>Development of a Marking Tool (SQLite+TCL/Tk)</td>
<td>Intern Work</td>
<td>✔️</td>
</tr>
<tr>
<td>2003-2004</td>
<td>Implicit Bridging and Open Code Translation</td>
<td>MSc Thesis Work</td>
<td>✔️</td>
</tr>
<tr>
<td>2003-2004</td>
<td>Open VHDL Code Translation from Executable UML Models</td>
<td>FMV-funded Feasibility Study – joint study between Software and Electronics Engineering</td>
<td>✔️</td>
</tr>
<tr>
<td>2004</td>
<td>Development of a Log Analysis Tool (SQLite+TCL/Tk)</td>
<td>Intern Work</td>
<td>✔️</td>
</tr>
</tbody>
</table>
# Technology Studies and Technology Development

<table>
<thead>
<tr>
<th>Year</th>
<th>Activity</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003-…</td>
<td>Continuous development of the Ada95 Model Compiler, including the Marking/Bridging and Log Analysis tools</td>
<td>Integrated in current programs</td>
</tr>
<tr>
<td>2004</td>
<td>SMUG 2004</td>
<td>Shlaer/Mellor User Group conference</td>
</tr>
<tr>
<td>Starts 2004</td>
<td>Open Code Translation of a Distributed Real-time System from Executable UML</td>
<td>MSc Thesis Work</td>
</tr>
<tr>
<td>Planned to start 2004</td>
<td>Modelling and Generation of Test Cases for Generated Ada95 Applications</td>
<td>MSc Thesis Work</td>
</tr>
<tr>
<td>Proposed</td>
<td>Integration of Software and Systems Engineering Model-Based Development</td>
<td>Internal Study – joint between Software and Systems Engineering</td>
</tr>
<tr>
<td>Proposed</td>
<td>Open VHDL Code Translation from Executable UML Models – VHDL Architecture/Model Compiler Development</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>Software/Hardware Co-design – Ada95/VHDL Model Compiler Integration</td>
<td></td>
</tr>
</tbody>
</table>
Questions & Answers?