MODEL-DRIVEN ARCHITECTURE OCH xtUML I PRAKTIKEN

Erik Wedin, Senior Specialist – Software Systems Architecture, Saab Bofors Dynamics
erik.wedin@saabgroup.com
November 20 2008
SESAM HÖSTSEMINARIE 2008
CONTENTS

SAAB Presentation
Background
MDA and xtUML
MESC – Modular Evolutionary System Core
System Development Using MDA and xtUML Based on MESC
Metrics
Conclusion
References & Extras
Saab serves the global market with world-leading products, services and solutions ranging from military defence to civil security. Saab has operations and employees on all continents and constantly develops, adopts and improves new technology to meet customers’ changing needs.

<table>
<thead>
<tr>
<th>Employees</th>
<th>13,500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turnover, MSEK</td>
<td>21,063</td>
</tr>
</tbody>
</table>

Annual sales are EUR 2.3 billion. Research and development corresponds to about 20 percent of annual sales.
A vital part of Saab’s defence activities with approximately 1,100 employees, consolidated in the business segment Systems and Products.

Key financial ratios, Saab Bofors Dynamics:

<table>
<thead>
<tr>
<th>Turnover</th>
<th>SEK 3,300 million</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exports</td>
<td>SEK 2,600 million</td>
</tr>
<tr>
<td>Backlog of orders</td>
<td>SEK 7,900 million</td>
</tr>
<tr>
<td>Exports</td>
<td>SEK 6,600 million</td>
</tr>
</tbody>
</table>

Certification to the following standards:
- ISO 9001 + ISO 14001
- Tick IT + AQAP 2110
COMPLETE MISSILES SOLUTIONS

Develops advanced missile systems for the Swedish Defence Forces and other national defence forces
Participates in international projects
INTERNATIONAL COLLABORATIONS

IRIS-T
Meteor
TAURUS
NLAW
ASRAD-R
RBS15
SOME SYSTEMS DEVELOPED USING MODEL-BASED DEVELOPMENT
INTEGRATED SOFTWARE/ELECTRONICS/SYSTEM DEVELOPMENT – THE NEXT STEP

Executable & Translatable Specifications (Models)

- Automated Implementation
  - High-Level Language (Ada95, C, …)
  - High-Level Language (VHDL, …)

- Compiler
  - Assembly Code

- Assembler
  - Object Code

- Synthesizer
  - Netlist File

- Placer & Router
  - Programming File

System
- Processor
- Programmable Logic Device
DIFFERENT WAYS OF USING UML

Sketches
- informal capture of ideas and concepts, coding may proceed directly
- visualises a solution and enables communication among persons

Blueprints
- specifies software structure
- near one-to-one mapping to code
- code frames may be generated – code is directly added in the model or in separate files
- if generated code is modified by hand $\Rightarrow$ model $\neq$ code $\Rightarrow$ need for “round-trip” i.e. generate model from code; only sensible if code and model has a near one-to-one relationship

Executable models $\equiv$ Executable specifications
- separates the application from the implementation, i.e. the platform-independent parts from the platform-specific parts
- the model does not necessarily have a one-to-one relationship with the code
- what xtUML and MDA and the rest of the presentation is all about
WHY ARE WE USING MDA AND xtUML?
THE POTENTIAL AND PROMISES OF MDA AND xtUML

- Executable models/specifications (with an abstract and executable action language)
  - Early validation of specifications – execute the models while building them, without code generation
  - Raise the abstraction level to a suitable level for each subject matter
  - Reuse of applications across architectures and platforms
    - capture corporate knowledge of the company’s applications – without muddling them with implementation details

- Translation of executable models/specifications to implementations
  - Repartition functionality across a platform, e.g. between software and hardware
  - Retarget functionality to new/modified platforms
  - Reuse architectural patterns across applications
    - capture corporate knowledge of the company’s architectures and platforms
  - Consistency between models, documentation and code

- Integration of systems, software and electronics engineering disciplines
  - Raises the productivity
  - Shortens the development time
  - Increases product quality
  - Increases flexibility in building systems

⇒ Raises the productivity
⇒ Shortens the development time
⇒ Increases product quality
⇒ Increases flexibility in building systems

{ Gives cost effectiveness + competitive edge

(And it’s a lot of fun too! 😊) }
EXECUTABLE UML

Executable UML is a profile (subset) of UML 2.0, including an abstract action language, adhering to the now standardised Action Semantics – defined by Stephen Mellor/Marc Balcer in 2002.

Enables development of Software and Hardware platform-independent specifications of the problem.

A standardised UML action language syntax is about to be defined – Executable UML is the basis for that via Stephen Mellor.

Supports the OMG Model-Driven Architecture (MDA) initiative:
- PIM – Platform-Independent Model - models the solution of a problem
- PSM – Platform-Specific Model - models the details of the implementation
- Separation of Subject Matters ➔ Abstraction & Reuse of models (not code)
- A Model Compiler weaves the models together, guided by marks, and translates them into an implementation at design-time not at specification-time

xtUML is Mentor Graphics’ implementation of Executable UML.

Executable UML (xtUML) models ≡ Executable Specifications
- can be executed and simulated (platform-independently) – *without generating code*
- can be translated to one/several implementation(s) onto one/several specific software/hardware platform(s) – *without changing the models*
## UML – UNIFIED MODELLING LANGUAGE

<table>
<thead>
<tr>
<th>Diagram Type</th>
<th>Usage</th>
<th>BridgePoint Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Large-scale components and their interfaces</td>
<td>✓</td>
</tr>
<tr>
<td>Package</td>
<td>Groupings of elements and dependencies between them</td>
<td>✓</td>
</tr>
<tr>
<td>Class</td>
<td>Classes, attributes, associations and generalisations</td>
<td>✓</td>
</tr>
<tr>
<td>State machine</td>
<td>Behaviour over time in response of events</td>
<td>✓</td>
</tr>
<tr>
<td>Use case</td>
<td>Requirements in terms of interactions between a system an its users</td>
<td>✓</td>
</tr>
<tr>
<td>Activity</td>
<td>Parallel and sequential behaviours connected by data and control flows</td>
<td>✓</td>
</tr>
<tr>
<td>Sequence</td>
<td>Synchronous/asynchronous interactions between objects</td>
<td>✓</td>
</tr>
<tr>
<td>Communication</td>
<td>Communications between objects (previously “collaborations”)</td>
<td>✓</td>
</tr>
<tr>
<td>Composite structure</td>
<td>Internal structure, ports, collaborations and structured classes</td>
<td>To be supported</td>
</tr>
<tr>
<td>Deployment</td>
<td>Deployment onto nodes in a specific implementation</td>
<td></td>
</tr>
<tr>
<td>Interaction</td>
<td>Overview of activities and interactions</td>
<td></td>
</tr>
<tr>
<td>Object</td>
<td>Selected instances of classes and values of attributes at run-time</td>
<td></td>
</tr>
<tr>
<td>Timing</td>
<td>Timing in a variation of sequence/interaction diagrams</td>
<td></td>
</tr>
</tbody>
</table>
BRIDGEPOINT – xtUML VERIFICATION PERSPECTIVE
BRIDGEPOINT – xtUML BUILD PERSPECTIVE
COMPONENT DIAGRAMS

- Defines components and their interfaces
  - ports
  - provided interfaces
  - required interfaces
- Defines how components are connected through interfaces
- Components can be nested in components
- Component types
  - formal – fully xtUML modelled = Domain ⇔ Separate Subject Matter
  - informal – non-modelled/realised, e.g. existing legacy design/code
SYSTEM CONSTRUCTION

There are two ways of connecting components

Explicit interfacing
- connection of interfaces between components at *specification-time* during system-level modelling
- pros: easy to understand, visible in the models, directly supported in the component diagrams
- cons: explicit interface calls in the component may compromise the component’s integrity/subject matter (e.g. calls to the GUI in an application that needs to present information)

Implicit interfacing (Model weaving)
- mapping between components of component-internal model constructs
- applied at *design-time* not at *specification-time*
- pros: non-intrusive (i.e. no explicit references in the model); very powerful when applied to certain component interactions (e.g. between an application and a GUI presenting information from the application); state-of-the-art MDA compliance
- cons: sometimes hard to understand; not clearly visible in today’s tooling
xtUML’s MDA PROCESS

1. Requirements Gathering (Use Cases, Activity Diagrams & Sequence Diagrams)
2. System-Level Design (Component Diagram)
3. Build Class Models (Class Diagram & Descriptions)
4. Build State Models (State Machines & Tables)
5. Organise Control Interactions (Class Communication Diagram)
6. Execute Instance Interactions (Sequence Diagrams)
7. Build Procedure Models (Action Language)
8. Specify Architecture
9. Build/Buy Model Compiler (Software Architecture)
10. Translate Models (Marking & Bridging)
11. Realise Components
12. Integrate and Test

Legend
- = PIM-Related Activities
= Architecture-Related Activities
= System Design Activities
Re用自己的软件架构

- 利用BridgePoint的开放翻译接口
- 内部MC是宝贵的战略资产

**Ada MC**
- 从头开始开发，基于前项目ODD架构
- 在多个项目中使用/重用
- 为特定项目需求量身定制

**VHDL MC**
- 架构设计，但尚未形式化在一个MC中

**SPARK Ada MC**
- SPARK Ada = 安全的Ada子集 + 注释中的注解
- 生成可以接受正式验证的代码来证明运行时错误的缺失
- 适合高性能和安全关键系统
- 处于开发中，于2008年用于一款产品

**Commercial MCs**
- MC2020 - C++
  - 通过显式桥接、持久性等扩展
- MC3020 - ANSI C
  - 用于产品中部分的选择部分，主要使用Ada MC
MESC – MODULAR EVOLUTIONARY SYSTEM CORE

- A reusable model-based architecture for tactical simulators and planning systems
- Developed by Saab Bofors Dynamics
- Developed according to MDA in xtUML
- Strict subject matter separation between domains/components
- Focus on reuse
- Maintained separately from the demonstrator- and product-projects that are using it
BACKGROUND

Legacy (autonomous missions) limited concept evaluation capability

Goal (autonomous missions, with update capability) improved evaluation capability

MESC PURPOSE

- Capture synergies during development of demonstrators for KEPD350/RBS15 and ECOM
- Support future demonstrator development efforts (ability to change, extend and build new demonstrators in the future)
- Focus on Planning and Control functions
- Reuse and Flexibility

Evolved RBS15/KEPD350
- (Support FM)

The slide is courtesy of Joachim Wickman and the Saab Bofors Dynamics MESC project
The slide is courtesy of Joachim Wickman and the Saab Bofors Dynamics MESC project.
THREE PILLARS OF MESC

Devices / Actor Composition

- An Actor is composed by Devices
- Devices are configurable
- Sensor, Warhead, Guidance, Control, Vehicle Dynamics, Link etc.
- Devices connected via Ports (information) – configurable structures
- Properties and Action control available for Rule Definition

Plans

- Routes (Waypoints, Transitions)
- Time Schedules (Time, Time Interval)
- Branched and Recursive Routes and Time Schedules

Rules

- Action (single evaluation) or Behaviour (continuous evaluation)
- Connected to Plans, Transitions, Waypoints, Times and Time Intervals
- Activates/deactivates other rules, devices and branches

The slide is courtesy of Joachim Wickman and the Saab Bofors Dynamics MESC project
FUNCTION (COMPONENT) COVERAGE

- Weapon system logic
- Mission Planning Control
- Route generation/Optimisation
- Rules

- Vehicle construction/Management
- Vehicle dynamic/control models
- Effector/vulnerability models
- Sensor/signature models

- Electronic warfare
- Environment/sourcedata
- Time management
- Evaluation

- Communication
- Resource management
- Situation Awareness
- Decision support

- Scenario control
- User interface
- Distribution
- Architecture

- Other (Mathematics, persistence, parsers etc)

The slide is courtesy of Joachim Wickman and the Saab Bofors Dynamics MESC project
MESC MISSION PLANNER COMPONENT
SYSTEMS DEVELOPED/BEING DEVELOPED BASED ON MESC

- Tactical simulator/planning system demonstrators
  - Demo “Markstridsdagar”
  - RBS15 MEPS demonstrator
  - KE PD 350 MiPaC demonstrator
  - ECOM MiPC demonstrator
  - Scenario Editor/ADSS Demonstrator

- Tactical simulator product in development

- One tactical simulator/planning system demonstrator is scheduled
ONE DEMONSTRATOR-SYSTEM OVERVIEW

The slide is courtesy of Joachim Wickman and the Saab Bofors Dynamics MESC project
DEMONSTRATOR PLATFORM

Operating System
• Windows

Framework
• SAFIR from Saab Systems
  • Framework for building command, control and communication systems
  • Includes GUI toolkit, data distribution components etc.

Programming Language
• C++

Model Compiler/Architecture
• MC2020 (C++) from Mentor Graphics
  • simple implicit bridging for actor construction
  • extended with automatic generation of interface definitions (DOU-files) to the DOB which is used to generate interface APIs by the DOB build process, i.e. interfaces are always consistent
  • extended with persistence to XML-files
SIMULATOR APPLICATION SAMPLE
SIMULATOR PRODUCT PLATFORM

- **Operating System**
  - Linux Red Hat

- **Framework**
  - DOB data distribution component from Saab Systems
  - The DOB is open-source originating from SAFIR
  - QT GUI toolkit

- **Programming Language**
  - Ada 2005 (application)
  - C++ (GUI)

- **Model Compiler/Architecture**
  - baseline Ada Model Compiler from Saab Bofors Dynamics
  - extended with implicit bridging – actor construction and application to DOB, i.e. decoupling of the application models from the GUI and planning system
  - extended with automatic generation of interface definitions (DOU-files) to the DOB which is used to generate interface APIs by the DOB build process, i.e. *interfaces are always consistent*
  - extended with persistence
## SIMULATOR APPLICATION – CURRENT MODEL METRICS

<table>
<thead>
<tr>
<th>Components</th>
<th>Instance State Machines</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelled - Total/MESC</td>
<td>States</td>
<td>666</td>
</tr>
<tr>
<td>Realised</td>
<td>State Transitions</td>
<td>1076</td>
</tr>
<tr>
<td>Data Types</td>
<td>Events</td>
<td>613</td>
</tr>
<tr>
<td>Interface Operations/Signals</td>
<td>Class State Machines</td>
<td>1</td>
</tr>
<tr>
<td>Provided</td>
<td>States</td>
<td>2</td>
</tr>
<tr>
<td>Required</td>
<td>State Transitions</td>
<td>2</td>
</tr>
<tr>
<td>Subsystems</td>
<td>Events</td>
<td>2</td>
</tr>
<tr>
<td>Classes</td>
<td>Attributes</td>
<td>2833</td>
</tr>
<tr>
<td>Attributes</td>
<td>Lines of Action Code</td>
<td>59745</td>
</tr>
<tr>
<td>Operations</td>
<td>Max Number of Instances</td>
<td>TBD</td>
</tr>
<tr>
<td>Associations/Relationships</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>Generalisation</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Associative</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Derived</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### SIMULATOR APPLICATION – CURRENT IMPLEMENTATION METRICS

<table>
<thead>
<tr>
<th>Generated Ada Code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>2884</td>
</tr>
<tr>
<td>Packages</td>
<td>969</td>
</tr>
<tr>
<td>Ada Tasks/Processes</td>
<td>5</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>263779</td>
</tr>
<tr>
<td>(excl. comments + empty lines)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Realised Ada Code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>655</td>
</tr>
<tr>
<td>Packages</td>
<td>318</td>
</tr>
<tr>
<td>Ada Tasks/Processes</td>
<td>0</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>84009</td>
</tr>
<tr>
<td>(excl. comments + empty lines)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Translation Time (Complete Retranslation)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- BridgePoint v1.5.4 on Windows XP</td>
<td></td>
</tr>
<tr>
<td>- Pentium Core 2 Duo ~2 GHz</td>
<td></td>
</tr>
<tr>
<td>- 2GB RAM</td>
<td></td>
</tr>
<tr>
<td>- Model Compiler and generated code reside on network-</td>
<td>~27 min</td>
</tr>
<tr>
<td>mounted drives</td>
<td></td>
</tr>
</tbody>
</table>
LEVEL OF MODEL REUSE

RBS15 MEPS Demonstrator

ECOM MiPC Demonstrator

Simulator Product

KEPD 350 MiPaC Demonstrator

Scenario Editor/ADSS Demonstrator

>80%
CONCLUSIONS

- MDA and xtUML delivers on their promises as illustrated by the example systems

- Reuse of application models across architectures and platforms – e.g. MESC components
  - captures corporate knowledge of the company’s applications – without muddling them with implementation details
  - enables rapid development of demonstrators: < 3 months development time for an air-defence demonstrator in 2007

- Reuse of architectural patterns across applications – e.g. Ada Model Compiler
  - captures corporate knowledge of the company’s architectures and platforms

- Integration of systems and software engineering disciplines
  - systems and software engineering specifies functionality – software engineering realises it in software in a fully automated way
REFERENCES

Mellor/Scott/Uhl/Weise – “MDA Distilled – Principles for Model Driven Architecture”, 2004
  • overall principles of MDA

  • the official book about Executable UML

Leon Starr - “Executable UML, a case study”,
  • a hands-on example project

  • book about Information Modelling with Executable UML

www.omg.org/mda
  • the official Object Management Group MDA home page

  • Anders Eriksson, Saab Bofors Dynamics – “Experiences from the First Step in Designing an Architecture Executing Executable UML Semantics in Programmable Logic Using VHDL”
EXTRA SLIDES
TECHNICAL SYSTEM CHARACTERISTICS

- Embedded real-time systems and planning systems and simulators
- Specially-designed hardware and computer platforms – but also COTS and workstations/PCs
- 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
  - simulation
- 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, but not in the near future
NON-TECHNICAL CHARACTERISTICS

- Long development programs
  - the hardware becomes obsolete before production
  - application functionality lives longer than the platforms

- Very long maintenance commitments – sometimes 20-30 years
  - capability to maintain/upgrade existing systems

- Restrictions in the selections of development environments
  - e.g. limited types of processors suited ⇒ limits the number of languages/tools to use

- Low production volumes
  - development has to be profitable

- International (and national) collaborations
  - flexibility in adapting to other companies’ architectures and platforms
  - flexibility in adapting to other companies’ development approaches and cultures
MDA Metamodel-Perspective – An Example From a Project

- Statemate Metamodel
- Extended xtUML Metamodel
  - Interaction Metamodell
  - xtUML Metamodel
  - Marking Metamodel
- Mappings
- OOD Architecture Metamodel
- Ada95 Metamodel
- Assembler Metamodel

More
Platform-Independent

- Statemate to Executable UML Model Mapping Guidelines
  (Manual Translation, Defined by Saab Bofors Dynamics)

- Ada95 Model Compiler
  (Automated Translation, Developed by Saab Bofors Dynamics or Commercial)

- Ada95 Compiler
  (Automated Translation)

More
Platform-Dependent
CORE EXECUTABLE AND TRANSLATABLE UML DIAGRAMS

Component (Domain) Diagram
- Components, Interfaces, Functions

Domain Package Diagram
- Data Types, Functions, Subsystems, External Entities

Class Diagram
- Classes (Attributes, Derived Attributes, Operations), Associations

Class/Instance State Machine
- Events, States, State Transitions

State Transition Table
- Events, States, State Transitions

Procedure
- Action Language

Action
- Action Language Statement

☆ = Contains Action Language

Model Compiler/Architecture
Ada95 VHDL

Application
Comm IO

E1 E2 E3 E4
S1 S2 IG IG CH
S2 CH S1 IG S3
S3 IG IG S1 IG

xxxx xxxxx x
xxxxxxx
xxxxxxx xxxxxx xxxxxx x
xxxxxxx
xxxx xxxxx
SYSTEM-LEVEL MODELLING

Component and Interface diagrams

«component»

Component Package

«interface»

Interface Package
INTERFACE DIAGRAMS

- Defines interfaces and their contents
  - operations  – synchronous communication/interaction
  - signals     – asynchronous communication/interaction

```
<table>
<thead>
<tr>
<th>Interface 1</th>
<th>Interface 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation 1(P1:integer):void</td>
<td>Operation 3():void</td>
</tr>
<tr>
<td>Operation 2():void</td>
<td></td>
</tr>
<tr>
<td>signals</td>
<td>signals</td>
</tr>
<tr>
<td>Signal 1(P2:real)</td>
<td>Signal 2()</td>
</tr>
<tr>
<td></td>
<td>Signal 3()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation 4(P3:integer, P4:integer):...</td>
</tr>
</tbody>
</table>
```
COMPONENT DIAGRAMS

- Sequence diagrams can be constructed between components
- Components can be executed in the Verifier
- Components can be nested in components
REUSE (EXAMPLES)

Air defence

- Air Defence logic
  - Effector/Vulnerability Model
  - Sensor/Signature Model
  - Dynamic Model
  - Distribution

Missile

- Missile logic
  - Vulnerability/Effector Model
  - Signature/Sensor Model
  - Dynamic Model
  - Distribution

The slide is courtesy of Joachim Wickman and the MESC project
MESC ACTOR COMPOSITION COMPONENT

Actor Property Access (M)  Actor Mission Control (M)  Actor Setup And Management (M)  Device Registration And Service (M)  Actor_Composition

Scenario (M)  Mission Plan Execution (M)  XML (Plan Creation (M)  Mode Timer (M)  String Utilities (M)  GUI Selectables (M)  Architecture (M)  Template Logger (M)  Device Plugin (M)
MESC RULE COMPONENT

Rule Conditions (M)

Rule Evaluation (M)

Architecture (M)
Model Timer (M)
Actor Property Access (M)
String Utilities (M)
GUI Selectables (M)
XML (M)
Template Logger (M)