Applying MDA in an Avionics Systems Environment

Ian Wilkie, Kennedy Carter

October 2005
Adopting MDA at BAE Systems

- Re-engineering of existing component, and automatic Ada Code Generation into existing execution environment
- Domain Reuse
  - Reuse of Ada Code Generation Rules
- Automated translation of structured analysis models in Teamwork tool to UML models in xUML tool
- RTSA to UML Tool Migration
Challenges for the Project - Obsolescence

■ Tool
  ■ The *Teamwork* CASE tool is unsupported.

■ Method
  ■ Real-Time Structured Analysis is no longer mainstream.

■ Platform
  ■ Need to ensure that the software remains maintainable and upgradeable during 25 years.

■ Skills
  ■ Experienced Ada programmers are an endangered species.
Project Goals

- Assess feasibility of migrating of existing Teamwork Models into the new tool environment to allow ongoing maintenance of Real-Time Structured Analysis models as profiled UML models.

- The re-engineering of a significant component into executable UML (xUML) models.

- Demonstration of automatic code generation of Ada code from re-engineered xUML Models.

- Demonstration that software can be built with heterogeneous mixture of legacy and re-engineered components.
Adopting MDA

Model Driven Architecture and xUML

www.kc.com
The project decided to take the approach of Model Driven Architecture (MDA) which involves building Platform-Independent Models (PIMs) from which we derive Platform-Specific Models (PSMs) and/or Platform-Specific Implementations (PSIs).

The models are represented using the notation known as the Unified Modeling Language (UML).

Both the MDA process and the UML notation are owned by the non-profit consortium known as the Object Management Group (OMG).
Executable UML (xUML)

- xUML is an implementation of the MDA process
- xUML is based around the idea of building PIMs that are precise, complete and executable (and therefore testable)
- xUML uses a subset of the UML formalism in order to achieve the required precision

“It’s like being a cook. You have to know how to use virtually all of the possible ingredients - just not all at the same time.”

Conversation with Jim Rumbaugh 1999

- xUML provides a specific view of the notion of “Platform” that is inherent in MDA.
- xUML also provides a coherent Process for the development of systems of both small and large scale and of various degrees of complexity.
Adopting MDA

Component Re-Engineering
Re-Engineering Objectives

A demonstration that a significant software component can be re-engineered in xUML and that Ada code can be automatically generated from it.

The automatically generated code is integrated back into the Radar System.

Function and performance matches the code that it replaces.
The Data Processor is a multi-processor, multi-tasking application. It uses a bespoke mailbox mechanism for inter-task communication.

<table>
<thead>
<tr>
<th>VM1</th>
<th>VM2</th>
<th>VM3</th>
<th>VM4</th>
<th>...</th>
<th>VMn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Machines</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Passive Objects (POs)

Ada Types

Drivers & OS
An Ada avionics component was re-engineered with xUML. Ada code was generated from xUML, and integrated with existing code.

Re-engineered Component
Domains

- Air To Surface Tracking
- Existing Radar Software
- Modelled in xUML
- Legacy code
- Passive Objects (POs)
- Ada Types
- Drivers & OS
Platform Independent Model Structure

15

Air To Surface Tracking

Existing Radar Software

Domain Model

(part of)

State Model

(part of)

Class Model

Scan

attributes

scanId:Integer \{="11"\}
cycleTime:Real

maximumTracks:Integer

currentActiveGroup:Integer

noOfGroupsRemaining:Integer

associationCount:Integer

trackUpdatesCompleteCount:Integer

createScan
doTargetAssociation

designCompletedUpdates

storeAssociations

identifyNonDirectAssociations

updateAssociationStatus

Reduced Correlation Matrix

attributes

matrixId:Integer \{="11"\}

noOfGroups:Integer

noOfTargets:Integer

noOfWindows:Integer

scanId:Integer

createRed

doRemoveDirect

doSetup

doCorrelate

doOutputUnas

7. Waiting For Updates To Complete
ectry /

6. Updates Complete
ectry /

[] = S3:processCompletedUpdates []
generate S4:waitForEndOfGroup() to this

allUpdatesComplete()

waitForEndOfGroup()
Interfaces with Legacy Code

Interfaces to legacy code units are modelled using <<terminator>> classes.
Adopting MDA

Process Migration

www.kc.com

The information contained in this document is the property of Kennedy Carter Limited. No part of it may be reproduced or used except by written permission. The copyright and foregoing restrictions on reproduction and use extend to all the media in which the information may be embodied.
Process Migration: Elaboration vs. Translation

Before

- manually build a Platform Independent Model...
- PIM (UML)
- manually build a Platform Specific Model...
- PSM (OOD)
- manually code a Platform Specific Implementation
- PSI (Ada)
- Process definition
- Design policies
- Coding rules

After

- manually build a Platform Independent Model...
- PIM (xUML)
- manually specify mappings onto the platform...
- PIM-PSI Mappings (xUML)
- automatically generate a Platform Specific Implementation using PIM-PSI mappings
- PSI (Ada)

Requirement Change impact

Technology Change impact
Formalised Application and Design Expertise

- manually build a Platform Independent Model...
  - Rules and policies for organising aircraft
  - Rules and policies for organising software
  - Platform Specific Model...

- PIM (xUML)
- Automatically generate a Platform Specific Implementation using PIM-PSI mappings
- PSI (Code)
Formalise the Formalisms

The Ada experts define mappings onto Ada...

The radar experts instantiate the "xUML Metamodel” classes...

...the Ada mappings are applied to the Radar PIM objects to generate the Radar Ada
Generate the Generator

**xUML to Code Mappings in ASL**

```plaintext
# Generate the operations.
[] = CCG_XUML403::ADA_genComment(file, "Operations")
first = TRUE
{operations} = this->R215->R215
for operation in {operations} do
  if (first) then
    first = FALSE
  else
    [] = FORMS0007:KC_TA0_formatNewLine[file]
  endif
[] = OP401:ADA_genSubProgramDecl[file] on operation
endfor
```

**xUML Metamodels**

(part of) xUML Metamodel

- **Class**
- **Attribute**
- **SignalGeneration**

**Project Domain Models**

<table>
<thead>
<tr>
<th>Scan</th>
<th>{no=2, kl=S}</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanId: Integer</td>
<td>{#=&quot;1&quot;}</td>
</tr>
<tr>
<td>cycleTime: Real</td>
<td></td>
</tr>
<tr>
<td>maxNumTracks: Integer</td>
<td></td>
</tr>
<tr>
<td>currentActiveGroup: Integer</td>
<td></td>
</tr>
<tr>
<td>noOfActiveGroups Remaining: Integer</td>
<td></td>
</tr>
<tr>
<td>associationCount: Integer</td>
<td></td>
</tr>
<tr>
<td>trackUpdatesCompleteCount: Integer</td>
<td></td>
</tr>
</tbody>
</table>

**Project Code**

```plaintext
-- Attribute access

function READ_scanId(INSTANCE_PTR : in PTR_TYPE) return XUML.INT is
  return INSTANCE_PTR.all.scanId;
end READ_scanId;

procedure WRITE_scanId(INSTANCE_PTR : in PTR_TYPE; VALUE : in XUML.INT) is
  INSTANCE_PTR.all.scanId := VALUE;
  WRITE_scanId;

function READ_cycleTime(INSTANCE_PTR : in PTR_TYPE) return XUML.REAL is
  return INSTANCE_PTR.all.cycleTime;
end READ_cycleTime;

procedure WRITE_cycleTime(INSTANCE_PTR : in PTR_TYPE; VALUE : in XUML.REAL) is
  INSTANCE_PTR.all.cycleTime := VALUE;
  WRITE_cycleTime;
```
Maintainability vs. Executability

Before

manually build a Platform Independent Model...

PIM
(UML)

manually build a Platform Specific Model...

PSM
(OOD)

manually code a Platform Specific Implementation

PSI
(Ada)

Built for maintainability and executability

In classic approaches, the PSI (code) must be built to be maintainable, typically by incorporating layering and encapsulation...

...which have a detrimental effect on speed and size of the executing system.

After

In translation-based approaches, the maintained entity (the PIM) is built for maintainability with layering and encapsulation...

...while the executable entity (the PSI) is optimised for execution efficiency.

manually build a Platform Independent Model...

PIM
(xUML)

automatically generate a Platform Specific Implementation using PIM-PSI mappings

Built for executability

Built for maintainability
Re-Engineering : Findings

“Working with graphical representations meant that software structure was easily understood and maintained.”

“Problems discovered during test were rectified in the PIM and the code regenerated.”

“No problems were detected attributable to code generator.”

“Code worked first time after Platform Independent Model modifications.”
Adopting MDA

Tool Migration

www.kc.com
Migrating from RTSA Tool to UML Tool

RTSA model held in Teamwork

Dataflow diagrams to UML class diagrams

UML model held in xUML Tool

Mealy state machines to Moore state machines
Automated Migration of Avionics Teamwork models to UML Tool

RTSA Model (in Teamwork) map CDIF to CDIF MM

CDIF Metamodel map CDIF to RTSA MM

RTSA Metamodel

xUML Tool Database

RTSA Model

populate <<RTSA>>xUML from xUML MM

xUML Metamodel

map RTSA MM to xUML MM

map CDIF to RTSA MM

map CDIF to CDIF MM

Developer

build, maintain and check RTSA Model

RTSA Model

population
Example Teamwork RTSA Model

1. Calculate Engine Run Damage
2. Reset Engine Run Damage
3. Update Total Engine Damage
4. Display Engine Damage

Engine Speed

Engine

Pilot

DFD 0 v1
The RTSA Metamodel

This model describes RTSA independently of the tool in which it is held...
The RTSA to xUML Mapping

The inter model mapping is expressed as ASL operating on the meta-models.

This sample of such ASL shows an operation provided by the "Data Flow Diagram" class in the RTSA meta-model...

```java
# Map the data flow diagram.
[] = DFD_TME2:mapDataFlowDiagramToXuml[sa_model, this.diagramNumber] on this

# Map the data dictionary entries.
{data_dictionary_entries} = sa_model -> R26
for data_dictionary_entry in {data_dictionary_entries} do
  [] = DDE6:mapDdeOnDfdToXuml[this] on data_dictionary_entry
endfor

# Map the data dictionary entry references.
{dde_references} = {data_dictionary_entries} -> R18.Data_Dictionary_Entry_Reference
for dde_reference in {dde_references} do
  [] = DDER2:mapDdeReferenceOnDfdToXuml[this] on dde_reference
endfor

# Map the nodes.
{nodes} = this -> R5.Node
for node in {nodes} do
  [] = N3:mapNodeToXuml[sa_model, this] on node
endfor

# Map the flows.
{process_inputs} = this -> R5.Node_on_DFD -> R1.Process_Input
for process_input in {process_inputs} do
  [] = PI3:mapProcessInputToXuml[] on process_input
endfor

{process_outputs} = this -> R5.Node_on_DFD -> R8.Process_Output
for process_output in {process_outputs} do
  [] = PO3:mapProcessOutputToXuml[] on process_output
endfor
```

Data Flow Diagram

- attributes
  - diagramNumber {i=(1)}
  - generationNumber
  - diagramTitle {i=(2)}
  - modelName {R=(R10, R27)}

- operations
  - genXmlElement
  - mapDataFlowDiagramToOod
  - mapDataFlowDiagramToXuml
  - checkDataFlowDiagram
  - createDataFlowDiagram
  - reconcile
  - getOffPagePlaceholderNumber
  - constructDataFlowDiagram
  - constructTopLevelDataFlowDiagr..
Example RTSA Metamodel Objects

metamodel objects for the SA model

metamodel classes
**Example RTSA Metamodel Objects**

<table>
<thead>
<tr>
<th>Domain: SAM</th>
<th>Class: Terminator</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeId</td>
<td>terminationName</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain: SAM</th>
<th>Class: Data_Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeId</td>
<td>storeName</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain: SAM</th>
<th>Class: Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeId</td>
<td>processName</td>
</tr>
<tr>
<td>4</td>
<td>Display_Engine_Damage</td>
</tr>
<tr>
<td>3</td>
<td>Update_Total_Engine_Damage</td>
</tr>
<tr>
<td>2</td>
<td>Reset_Engine_Run_Damage</td>
</tr>
<tr>
<td>1</td>
<td>Calculate_Engine_Run_Damage</td>
</tr>
</tbody>
</table>

**DFD 0 v1**

1. Calculate Engine Run Damage
2. Reset Engine_Run Damage
3. Update Total Engine Damage
4. Display Engine Damage

- Engine
- Engine Speed
- Engine Run
- Pilot
- Engine Damage
The metamodel mappings are applied to generate an xUML model...

DFD 0 v1

- T Engine
  - R001: 1 Engine_Parameters
  - R002: 1 P1_Calculate_Engine_Run_Damage
  - R003: 1 S_Engine_Run_Damage
  - R004: 1 P2_Add_Engine_Damage_To_Total

- P1_Calculate_Engine_Run_Damage
  - Engine.Run.Damage
  - Engine.Run
  - Engine.Run.Damage

- S_Engine_Run_Damage
  - Engine.Run.Damage
  - Engine.Run.Damage

- P2_Add_Engine_Damage_To_Total
  - Total_Engine_Damage

- P3_Display_Total_Engine_Damage
  - Total_Engine_Damage

- Engine Speed
  - 1 Engine.Run.Damage
  - 1 Engine.Run
  - 1 Engine.Run.Damage
  - 1 Display_Engine_Damage

- Pilot
  - 1 Total_Engine_Damage
  - 1 Total_Engine_Damage
  - 1 Total_Engine_Damage

- Engine Damage
  - 1 Total_Engine_Damage
Mapping RTSA Models to UML Models
Spot the Difference: Mapping xUML Models to Ada Code

**xUML Model**

- **Scan**
  - scanId: Integer \{0\}
  - cycleTime: Real
  - maxNumTracks: Integer
  - currentActiveGroup: Integer
  - noOfGroupsRemaining: Integer
  - associationCount: Integer
  - trackUpdatesCompleteCount: Integer

**Application experts build application PIMs**

**xUML Metamodel**

**Use PIM model to populate xUML metamodel**

**Design experts specify mapping rules from UML metamodel onto platform-specific technologies**

**xUML to Ada Metamodel Mappings**

**Apply xUML-Ada Mappings to xUML metamodel objects**

**Ada Code**

- **Operations**

  ```ada
  procedure updateAssociationStatus
  (instance PTR : in PTR_TYPE) is
  begin
    PTR_TYPE := INSTANCE_PTR;
    null;
  end updateAssociationStatus;
  ```

  ```ada
  procedure identifyNonDirectAssociations
  (instance PTR : in PTR_TYPE) is
  begin
    null;
  end identifyNonDirectAssociations;
  ```

**The platform-specific implementation (code) is automatically generated by applying the xUML-Ada mapping rules**
MDA Formalises Intellectual Property

This approach ensures that all the business intellectual property is formalised, and made reusable, in xUML.
Benefits of Adopting MDA

- Overcome the Method and Tool Obsolescence Issue.
- **Simplify the Software Development Process** by treating Application Analysis and Software Design as totally independent, decoupled operations.
- **Protect intellectual investments** in xUML Models against Platform Obsolescence.
- **Capture the expertise of the best Computer Scientists and Ada Software Engineers** in an automated tool.
- Adapt to a modern technology which fully supports maintenance and enhancements over the next 25 years or more.
- By addressing architectural issues separately from specification of behaviour, address issues of **performance on any platform**.
Further Information

For more information contact:

- lan.Wilkie@kc.com
- Wolfgang.George@kc.com
- www.kc.com
Applying MDA in an Avionics Systems Environment

Ian Wilkie, Kennedy Carter

www.kc.com

THE END: QUESTIONS?

October 2005