xUML and AADL

xUML Overview

Chris Raistrick

www.kc.com
A Platform Independent Model (PIM) is a technology agnostic model of some aspect of the system under study.

A PIM contains no information about any of the following:
- Hardware Architecture
- Operating System
- Programming Language
- Database Technology
- Internal Communication Technology

It is therefore much simpler than a Platform-Specific Model (PSM).

Use of Executable UML (xUML) allows construction of PIMs that are:
- Precise
- Complete

PIMs built using xUML can be:
- Executed to demonstrate compliance with functional requirements
- Automatically translated into a complete Platform Specific Implementation using a suitable model translator
- Used as executable specifications, forming the basis for contract-based procurement
The MDA Process Overview

MDA with xUML Development

- Built by application experts
- Build PIM (xUML)
- Define PIM-PSI Mapping Rules
- PIM-PSI Mapping Rules
- Translate PIM To PSI
- PSI (AADL/C++/Ada/…)

- Built by technology experts

Built by technology experts

Built by application experts
Service Layers on the Domain Model

Network Level Services

Mission Level Services

Store Control Services (API)

Store OSI Services (API)

Specific Comms Services

UML Execution Services
Classes

- Classes identify the things that exist with a domain. Ideally, they represent things in the “real world” of that domain.
- They establish the vocabulary of the domain, or area of expertise.
- For example, the “Effectors” domain might contain this class...

Weapon
Attributes

- Attributes specify what we know about each thing (or class).
- They are analogous to data.

**Weapon**

<table>
<thead>
<tr>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>weaponId: Integer {l=(*)1}</td>
</tr>
<tr>
<td>maxRange: Integer</td>
</tr>
</tbody>
</table>
Operations

- Operations specify what we can do to each thing.
- They are analogous to code.

<table>
<thead>
<tr>
<th>Weapon</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>weaponId: Integer ( l=(*1) )</td>
<td></td>
</tr>
<tr>
<td>maxRange: Integer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>createWeapon(in maxRange, newWeapon)</td>
<td></td>
</tr>
<tr>
<td>findPendingTargets()</td>
<td></td>
</tr>
<tr>
<td>engageTarget()</td>
<td></td>
</tr>
</tbody>
</table>
Every operation has a method, comprising the ASL to specify that operation. Use of ASL makes the model executable while preserving platform independence.
**Associations**

Associations capture real-world connections between classes.

![Image of a jet plane]

<table>
<thead>
<tr>
<th>Weapon</th>
<th></th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>weaponId: Integer {l=*1}</td>
<td>targetId: Integer {l=*1}</td>
</tr>
<tr>
<td></td>
<td>maxRange: Integer</td>
<td>currentWeaponId: Integer {R=(R1)}</td>
</tr>
<tr>
<td>operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>createWeapon(in maxRange, newWeapon)</td>
<td>createTarget(in hostility, newTarget)</td>
</tr>
<tr>
<td></td>
<td>findPendingTargets()</td>
<td>determineRangeFromOwnShip(rangeFromOS)</td>
</tr>
<tr>
<td>0..1</td>
<td>R2</td>
<td>0..*</td>
</tr>
<tr>
<td>has pending</td>
<td>is pending for</td>
<td>has pending</td>
</tr>
<tr>
<td>0..1</td>
<td>R1</td>
<td>0..1</td>
</tr>
<tr>
<td>is current for</td>
<td>pendingWeaponId: Integer {R=(R2)}</td>
<td>hostility: Text</td>
</tr>
</tbody>
</table>

```python
Weapon
- weaponId: Integer {l=*1}
- maxRange: Integer

operations
- createWeapon(in maxRange, newWeapon)
- findPendingTargets()

Target
- targetId: Integer {l=*1}
- currentWeaponId: Integer {R=(R1)}
- pendingWeaponId: Integer {R=(R2)}
- hostility: Text

operations
- createTarget(in hostility, newTarget)
- determineRangeFromOwnShip(rangeFromOS)
```
The Class Collaboration Diagram shows the interfaces within the system, and between the system and the outside world. In this view, “Actors” are represented as <<Terminator>> classes.
Signals on The Class Collaboration Diagram

The Class Collaboration Diagram shows this signal is generated by the Weapon class and received by the Target class.
State Machines and Signals

The Weapon class has a state machine, represented using this statechart:

1. **Weapon Idle**
   - *entry /*

   - `startEngagement(paramX:Integer){no=1, kl=W}`

   2. **Starting Engagement**
      - *entry /
      - `myTarget = this -> R1`

      - `generate TGT1:engagementStarted() to myTarget`

The Target class has a state machine, represented using this statechart:

1. **Being Tracked**
   - *entry /

   - `engagementStarted(){no=1, kl=TGT}`

   2. **Being Engaged**
      - *entry /

**Weapon**

- `determineRangeFromOwnShip`

**Target**

- `engagementStarted`
The Primary xUML Models

Domains
- Effector
  - AMRAAM
  - Sidewinder

Classes
- Weapon
  - Attributes:
    - weaponId (1), maxRange (1), newWeapon
  - Operations:
    - createWeapon(maxRange, newWeapon)
      - findPendingTargets()
  - States:
    1. Weapon Idle
       - entry /
       - startEngagement(paramX, no=1, kl=W)
    2. Starting Engagement
       - entry /
       - myTarget = this -> R1
       - generate TGT1:engagementStarted() to myTarget

- Target
  - Attributes:
    - targetId (1), currentWeaponId (R1), pendingWeaponId (R2), hostility: Text
  - Operations:
    - createTarget(hostility, newTarget)
      - determineRangeFromOwnShip(rangeFromOS)
xUML and AADL

Chris Raistrick

www.kc.com
A “Translation-Based” MDA Process

- Existing ALWI Docs
- Build Store API PIM
- PIM (xUML)
- Weapon experts build this
- PIM to PSM Mapping
- Apply the PIM to PSM Mapping
- PSM (AADL)
- Apply the PSM to PSI Mapping
- PSI (Ada)
- Software experts build this
- PSM to PSI Mapping
- Hardware and software architecture experts build this

Only the PIM is maintained... ...PSMs and PSIs are automatically generated
Aircraft Platform Characteristics

We could specify the hardware architecture, and existing software and bus loadings, of each aircraft type...

...and generate one or more Aircraft Platform Specific Models from each PIM
An xUML-AADL Process

1. Requirements
2. Build Platform Independent xUML API Model
3. Platform Independent xUML API Model
4. Automatically Generate AADL from xUML PIM
5. Analyse AADL Specification
6. Automatically Generate Target Code
7. Target Code
8. E2 JSF
9. E2 JSF
10. F16 Aircraft Platform Specification
11. Refine xUML API Model
12. AADL Problems
13. Identify Problems in AADL
14. AADL Problems
15. AADL Analysis Results
(Simplified) AADL Metamodel
(Simplified) AADL Metamodel Regions

Software Components

Hardware Components

Hardware-Software Allocations
Configure the AADL Model for the Hardware Components

The AADL model is configured with data to capture the aircraft platform hardware architecture.
Populate the AADL Metamodel with Software Components
PIM-PSM-PSI Mappings Are Defined at the Metamodel Level

- **(part of) PIM Metamodel (xUML)**
  - Class
  - Attribute
  - Signal

- **(part of) PSM Metamodel (AADL)**
  - SW Component
  - Data Port
  - Event Port

- **(part of) PSI Metamodel (Ada 83)**
  - Package
  - Data Type
  - Spec Subprogram

---

- **xUML Metamodel**
  - Define Mapping

- **PSM Metamodel**
  - Define Mapping

- **PSI Metamodel**
Apply Mappings to Generate AADL

(part of) xUML Metamodel
- Class
- Attribute
- SignalGeneration

The application experts instantiate the “xUML Metamodel” classes...

(part of) AADL Metamodel
- Software Component
- Data Port
- Event Port

The system experts define mappings onto AADL...

...the AADL mappings are applied to the xUML objects to generate the AADL

Aircraft System PIM

Generated Aircraft System AADL

6. Updates Complete
entry /[] = S3:processCompletedUpdates[]
generate S4:waitForEndOfGroup() to this

Scan
attribute

scanId: Integer {= (*1)}
cycleTime: Real
maxNumTracks: Integer
currentActiveGroup: Integer
noOfGroupsRemaining: Integer
associationCount: Integer
tracks...
The Ada experts define mappings onto Ada...

---

The application experts instantiate the “xUML Metamodel” classes...

---

The Ada mappings are applied to the xUML objects to generate the application Ada

---

Aircraft System PIM

6. Updates Complete

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

---

Generated Aircraft System Ada

---

K E N N E D Y  C A R T E R
Apply Mappings to Generate AADL and Code

--- Attribute access:

```plaintext
function READ_scanId(INSTANCE_PTR : in PTR_TYPE; return XNUM_INT is
begin
    return INSTANCE_PTR.all.scanId;
end READ_scanId;

procedure WRITE_scanId(INSTANCE_PTR : in PTR_TYPE; VALUE : in XNUM_INT is
begin
    INSTANCE_PTR.all.scanId := VALUE;
end WRITE_scanId;

function READ_cycleTime(INSTANCE_PTR : in PTR_TYPE; return XNUM_REAL is
begin
    return INSTANCE_PTR.all.cycleTime;
end READ_cycleTime;

procedure WRITE_cycleTime(INSTANCE_PTR : in PTR_TYPE; VALUE : in XNUM_REAL is
begin
    INSTANCE_PTR.all.cycleTime := VALUE;
end WRITE_cycleTime;
```

--- Aircraft System PIM

Scan

```plaintext
attributes

scanId: Integer {i="1"}
cycleTime: Real
maxNumTracks: Integer
currentActiveGroup: Integer
noOfGroupsRemaining: Integer
associationCount: Integer
track:
```

### 6. Updates Complete

entry /

\[
\text{generate S4.waitForEndOfGroup}() \text{to this}
\]
xUML and AADL

Summary
This approach ensures that all expertise is formalised, and made reusable, in xUML.
MDA Benefits

- **Captures intellectual property** - valuable subject matter expertise is captured in a standardised, reusable, readily shared and easily maintainable manner;

- **Executable and testable models** - early verification that requirements are understood and early demonstration of required system behaviour lowers through-life costs, reduces risk and increases confidence;

- **Resilience to change** - changes in functional requirement are isolated from changes in technology and platform, therefore giving the ability to migrate to the “next generation” platform without model rework;

- **100% code generation from models** - so models are maintained and code is a derived product, models never become obsolete and are guaranteed to always be up-to-date;

- **Reduced lifecycle costs** - through early defect identification, maintained separation of concerns and minimised impact of change;
The End