MDA at the MoD

Welcome

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
Agenda

Abstract Solutions and Military OSAs

The MoD Requirement

MDA for LOSA

Model Verification and Deployment

Summary
Abstract Solutions

Formerly Kennedy Carter

Since 1989, specialists in helping organisations adopt

MDD for Systems and Software

through

Training – Consulting - Mentoring
MDA with UML for Open System Architectures

MDA has been adopted for a number of military architectures...

**US OSD UCS Architecture**
- Mandated by DoD for all UAVs over 20 pounds
- Evolution of:
  - NIAG SG97 (ALWI)
  - NIAG SG125 (UAS Architecture)
  - www.ucsarchitecture.org
- Deployed in US

**Complex Weapons OSA**
- Standardised architecture for all stages of kill chain.

**Generic Vehicle Architecture / LOSA**
- For platform and equipment interoperability
- Deployed in UK. Adopted by NATO
- Now being extended for soldiers and bases.

**Weapon Integration UK**
- PnP Architecture for weaponised aircraft
- Evolution of NATO ALWI Architecture
- Deployed in UK and US
These military open system architectures embody the key principles of Model Driven Architecture (MDA):

- Domain based partitioning
- Platform independent models
- Automatically generated platform-specific artefacts
MDA at the MoD

The MoD Requirement

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
Requirements for LOSA

**Affordability**
- To minimise the cost of upgrades and UORs

**Flexibility & Portability**
- To allow deployment on different installations (base, vehicle, soldier)

**Managed Complexity and Open Standards**
- To facilitate universal understanding and reduce integration risk for systems with complex and rapidly evolving requirements

**Longevity and Scalability**
- In keeping with the long lifetimes of the deployed systems
The Requirement for Reusable Modules

Most modules are applicable to multiple installation types...

- **Vehicle**
  - Engine
  - Brakes
  - Fuel
  - LRF
  - LTD
  - Nav

- **Base**
  - DASS
  - Water distribution
  - Waste management
  - Base protection

- **Soldier**
  - Radar
  - Met
  - Mount
  - Weapon
  - Portable charger
  - Rifle

---

*Common Open Architecture (Land)*

- Data interfaces
- Power interfaces
- Data Model

---

*Ministry of Defence*

*Open Systems Architecture*

*Enabling Updates and Integration*

*Other Domains e.g.*

- Network
- ISTAR
- Training
- Land, Air, Maritime

*Defence Equipment & Support*
The Requirement for Portable Modules

Middleware and messaging technologies:
- evolve and are superseded over time
- vary from system to system

To enable easy adoption of different software technologies, the generic architecture model must be:
- independent of middleware and software architecture
- independent of messaging technologies and message passing strategies

IDL  XML  JSON  The Java Message Service

A Platform Independent Model
is a long-life reusable asset
MDA at the MoD

MDA for LOSA

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
The OMG Model Driven Architecture embeds three key principles:

- **Domain Partitioning of the System**
- **Platform Independent Modelling of each Domain**
- **Automated Generation of the Platform Specific artefacts**

These principles are designed to achieve specific goals:

- **Component Reuse** through pollution control
- **Model Longevity** through platform independence
- **Easier Upgrades** through extensible layered architecture and data driven models
The process for system development, known as **Model Driven Architecture (MDA)** 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)**.
A Platform Independent Model (PIM) is a technology agnostic model, which fully specifies the capabilities and data provided by the system, but suppresses detail concerning:
- Hardware architecture
- Software architecture
- Middleware
- Messaging technology

"Platform" in this context refers to the "execution platform"... ...although we aim to be vehicle platform independent too.

A PIM is therefore much simpler than a Platform-Specific Model (PSM)
The MDA System for Building Systems

- Use case analysis to specify capabilities
- Reusable domain-based platform independent models
- Translation of model onto multiple platforms

Manually Maintained Artefacts

Generic Architecture Design
- Use Cases to specify requirements
- Domains to specify components
- Interactions to specify interfaces

Data Model Design
- Classes to specify data
- Operations to specify capabilities
- States to specify behaviour

Automatically Generated Artefacts

- Platform Specific Models
  - Vehicle PSMs
  - Soldier PSM
  - Base PSM

- Open Interface Definitions
  - IDL Message Definitions for DDS
  - JSON Message Definitions For Lean Services

- System Documentation
  - Equipment ICDs
  - MoDAF Documents
Use cases specify the **capabilities** to be provided by the system, and **how they interact** with humans (e.g. Crew Member) and equipment (e.g. Monitored Entity).
The use case model provides the basis for building the PIM...

Note that configuration data shown in compartment 4 is not part of the model.
Domains embody the subject matters, or areas of expertise, in our system.

Layered data driven domains are easy to extend with new capabilities, and protect against change.

For each domain we build a Platform Independent Model (PIM) that is technology agnostic.

The PIMs can then be reused and deployed on any current or future execution platform.
Class models identify the things that exist within a domain, their characteristics, and the relationships between them. They establish the vocabulary of the domain, or area of expertise.
State models capture:
- Rules and policies defining when specific messages are valid
- System and equipment modes
- Behaviour (if desired)
Pitfall: Use of Complex Modelling Notations

UML & SysML have evolved into a set of complex notations... ...that can create a barrier to communication in multi-disciplinary teams...
Use a Simple Modelling Notation

The primary purpose of platform independent modelling is to formalise the knowledge in the heads of a variety of subject matter experts... complexity is the biggest threat to collaboration... and use of a simple subset of UML will facilitate universal understanding.
MDA at the MoD

Domain Partitioning

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
A Stores Management Computer

The stores management computer in a military aircraft performs a number of functions...

- Manage aircraft C of G as weapons are released
- Control clamps to release weapons
- Provide a display of current weapon loadout
- Ensure launch safety by monitoring safety settings
- Provide Power and Comms via MIL-STD-1760 and MIL-STD-1553 Interfaces
Pitfall: Pollution as a Cause of Component Complexity

Models polluted with multiple unrelated concerns become unnecessarily complex...

Weapon class without domain partitioning is:
- Complex
- Hard to understand
- Expensive to change & retest
- Hard to reuse
- Hard to port

Weapon class with domain partitioning is:
- Simple
- Easy to understand
- Cheap to change & retest
- Easy to reuse
- Easy to port

Separation of concerns gives highly cohesive, loosely coupled domains...
which are much simpler, and highly reusable...

I progress through a defined launch sequence

I check the aircraft is safely balanced after release

I release myself by opening electrically actuated clamps

I check all safety settings are correct before release

I render myself as an icon with colour to denote status

I communicate across a MIL-STD-1553 comms bus

I save my state in a DDS topic

I am running on Greenhills INTEGRITY-178B RTOS
Achieve Pollution Control through Domains

- **Strategic reuse through Pollution Control**
- **Highly cohesive, loosely coupled domains**
- **Separation of concerns** makes each domain much simpler…
  …and is the key to reuse
- **Platform Independent Models become long-life corporate assets**, making accumulated IP accessible and reusable

I affect the balance of the aircraft
- **Store_Configuration::Actual_Store**

I must check my safety settings before I am released
- **Station_Services::Actual_Unit**

I must progress through a defined launch sequence
- **Weapon_Control::Actual_Weapon**

I am a remote terminal with transmitted and received messages
- **MIL_STD_1553B::Remote_Terminal**
Example WIUK Counterpart Associations

Each real-world thing can be represented as different abstractions in various domains...the counterpart classes, linked via counterpart associations

I affect the balance of the aircraft

I must check my safety settings before I am released

I must progress through a defined launch sequence

I am a remote terminal with transmitted and received messages
The table below illustrates the possible F-16 Stores Configurations

<table>
<thead>
<tr>
<th>Rail ID</th>
<th>Right Wing</th>
<th>Center</th>
<th>Left Wing</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>AMRAAM</td>
<td>AMRAAM</td>
<td>AMRAAM</td>
</tr>
<tr>
<td>8</td>
<td>AMRAAM</td>
<td>GBU24</td>
<td>GBU24</td>
</tr>
<tr>
<td>7</td>
<td>Sidewinder</td>
<td>LANTIRN</td>
<td>Sidewinder</td>
</tr>
<tr>
<td>6</td>
<td>370g Tank</td>
<td>370g Tank</td>
<td>370g Tank</td>
</tr>
<tr>
<td>5R</td>
<td>370g Tank</td>
<td>370g Tank</td>
<td>AMRAAM</td>
</tr>
<tr>
<td>5L</td>
<td>370g Tank</td>
<td>370g Tank</td>
<td>AMRAAM</td>
</tr>
<tr>
<td>5</td>
<td>ECM Pod</td>
<td>ECM Pod</td>
<td>Sidewinder</td>
</tr>
<tr>
<td>3A</td>
<td>AGM65</td>
<td>AGM65</td>
<td>Sidewinder</td>
</tr>
<tr>
<td>3</td>
<td>370g Tank</td>
<td>370g Tank</td>
<td>Silweinder</td>
</tr>
<tr>
<td>2</td>
<td>Harm</td>
<td>LANTIRN</td>
<td>Harm</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Areas of change are:
- Aircraft type
- Station (rail) configuration
- Store types
- Station-store combinations
- Mission types

...so these must not be “hard-wired” into the PIM
Data Driven PIM

Aircraft Store Configuration Domain

Predefined Loadout Store Station Combination
- attributes
  - predefinedConfigurationName

Predefined Loadout
- attributes
  - predefinedConfigurationName

Type Of Aircraft
- attributes
  - aircraftTypeName
- is specified by
  - 0..*
  - 1 R11 specifies
  - 1 actualAircraftId
    actualAircraftId
    aircraftTypeName

Type Of Station
- attributes
  - aircraftTypeName
  - stationTypeName
- is specified by
  - 0..*
  - 1 R3 specifies
  - 1 actualStationId
    actualStationId
    stationTypeName

Type Of Store
- attributes
  - storeTypeName
- is specified by
  - 0..*
  - 1 R1 specifies
  - 1 actualStoreId
    actualStoreId
    actualStationId
    storeTypeName

Actual Aircraft
- attributes
  - actualAircraftId
  - aircraftTypeName

Actual Station On Aircraft
- attributes
  - actualAircraftId
  - actualStationId
  - stationTypeName

Actual Store
- attributes
  - actualStoreId
  - actualAircraftId
  - actualStationId
  - storeTypeName
Data Driven Store Configuration Domain

The model is instantiated with objects for a specific aircraft using data held in tables...

...allowing new weapon types and configurations to be added without changing any code...

...and allowing the model to be reused on any aircraft platform.
MDA at the MoD

Verification through Model Animation

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
Demonstrate the viability of the LOSA Modules

Understand in detail the requirements set out in DEF STAN 23-009 and explore issues for clarification
Executable Models

- **GVA Alerts Model**
  - **GVA Alerts Class Model** contains behaviour specified with state machines and action code

```
//isAcknowledged = false;
isActive = true;
```

```
 affirmationHasAcknowledgedAlert/
isAcknowledged = true;
notifyAlertAcknowledged();
```

```
 tm(ackTimeout)[hasAckTimeout]/
isAcknowledged = true;
notifyAlertAcknowledged();
```

```
 tm(repeatTimeout)/
isAcknowledged = false;
notifyReinstatedAlert();
```

```
 evAlertConditionCleared/
isActive = false;
notifyAlertCleared();
```

```
 evOperatorHasAcknowledgedAlert/
isAcknowledged = true;
notifyAlertAcknowledged();
```

- **Model remains uncontaminated with knowledge of User Interface policy**
Model Execution: Simulated User Interface

- Warning Tone is sounding
- Summary of Active Alerts
- Unacknowledged Alert shown in Popup
- Historic List of all Alerts

(U) W:Engine:Oil:High:Pressure
As well as the Panel Diagram, the model animation can be reviewed against expected behaviour during and after runtime...
GVA Alerts class model shown to be capable of representing the required data

18 Separate issues with the requirements have been identified
- Missing requirements
- Inconsistent terminology
- Unclear requirements
- Inconsistent requirements
- Undefined terms

In addition, a number of dynamic human factors problems identified

The model implements a number of (documented) decisions in order to deal with the gaps in requirements but now serves as an unambiguous statement of requirement
MDA at the MoD
Deployment through Model Translation

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
Before 1800

- Hand crafting by skilled practitioners
- Idiosyncratic design strategies
- Premium is on the practitioner

After 1800

- Automated production lines with no waste
- Consistent design strategies
- Premium is on the process
The PIM is independent of:
- Middleware and software architecture strategies
- Messaging technologies

The PIM specifies the data, operations and modes

We can generate multiple PSMs from a single PIM

The PSM embeds the chosen software architecture strategies

The PSI embeds the chosen middleware technologies
Translation for Architecture Component Reuse and Maintainability

Platform Independent Model

- Smaller, simpler and cheaper to maintain than PSM(s)
- Common service domains eliminate duplication of data items
- Independent of middleware and message definition languages
- Can be deployed on any execution platform in any installation (base, vehicle, soldier)
- Each PIM focusses on a single subject matter (domain)

PSM Generator

- Easy to change system-wide design patterns
- Easy to adapt for different distribution and communication strategies

Platform Specific Model

- Consistent use of design patterns across the system
- Coherent use of (one or more) software design strategies (SOA, Pub/Sub, Client/Server)
- Consistent inclusion of system-wide data items

PSI Generator

- Easy to adapt for different middleware and message definition languages
- DDS and IDL specific translation rules ensure consistency

Platform Specific Implementation

- Optimised for specific middleware technology (DDS, Web Services, Lean Services)
- Oriented towards specific message definition language (IDL, JSON, XML)
- Provides “on-the-wire” message interface definition

The smaller, simpler PIM becomes the maintained artefact. The PSM and PSI are automatically generated and consistent
PIM to PSM Examples – Classes, Attributes and Associations

PIM + translator properties = PSM
PIM to PSM Examples – Operations and States

Operation: `controlScan` in `Actual_Mount`
- **void controlScan(const ScanAction& set, const AxisType& startAxis, const AngleInDegreesType& interAxisOffset)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>ScanAction</td>
</tr>
<tr>
<td>startAxis</td>
<td>AxisType</td>
</tr>
<tr>
<td>interAxisOffset</td>
<td>AngleInDegreesType</td>
</tr>
</tbody>
</table>

Class: `Actual_Mount` in `Mount_PIM_PSM`

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipientID</td>
<td>IdentifierType in LDM_Common_New</td>
</tr>
<tr>
<td>sourceID</td>
<td>IdentifierType in LDM_Common_New</td>
</tr>
<tr>
<td>referenceNum</td>
<td>GVA_Long_Integer in LDM_Common_New</td>
</tr>
<tr>
<td>timeOfDataGeneration</td>
<td>DateTimeType in LDM_Common_New</td>
</tr>
<tr>
<td>set</td>
<td>ScanAction in Mount_PIM</td>
</tr>
<tr>
<td>startAxis</td>
<td>AxisType in Mount_PIM</td>
</tr>
<tr>
<td>interAxisOffset</td>
<td>AngleInDegreesType in LDM_Common_New</td>
</tr>
</tbody>
</table>

Maintenance

- Calibration
- Boresight
- Test
- Not_In_Maintenance

Type: `MaintenanceStateType` in `Mount_PIM_PSM`

- MaintenanceStateType_Maintenance_Calibration
- MaintenanceStateType_Maintenance_Boresight
- MaintenanceStateType_Maintenance_Test
- MaintenanceStateType_Maintenance_Not_In_Maintenance
  <New>
PSM to IDL Examples

```idl
// This reports the state of the mount.
struct C_Actual_Mount
{
    T_IdentifierType A_sourceID; // @key
    T_DateTimeType A_timeOfDataGeneration;
    T_CurrentScanState A_scan;
    T_MaintenanceStateType A_maintenanceMode;
    T_IdentifierType A_itsProtection_sourceID;
    T_IdentifierType A_itsUtility_sourceID;
};
```

```idl
// An enumeration of the Maintenance States that are possible.
enum T_MaintenanceStateType
{
    L_MaintenanceStateType_MaintenanceStateType_MaintenanceState_Type_MaintenanceState_Calibration,
    L_MaintenanceStateType_MaintenanceState_Type_MaintenanceState_BoreSight,
    L_MaintenanceStateType_MaintenanceState_Type_MaintenanceTest,
    L_MaintenanceStateType_MaintenanceState_Type_Maintenance_Not_In_Maintenance
};
```
Architecture Component Reuse

The PIM domains can be reused in multiple installations...

...and implemented on multiple deployment architectures.

Platform Independent Models

Platform Specific Model Translators

Generate Base PSM

Generate Vehicle PSM

Generate Soldier PSM

Base PSM

Vehicle PSM

Soldier PSM

Lean Services

JSON

DDDS

IDL

USB 2.0

XML
Inter-Platform Communication: LOSA MDA Artefacts

LOSA Reusable PIMs

LOSA Installation Configured PIMs

LOSA Installation PSMs

Base Application
- Base Internal Message Definitions
- Base External Message Definitions

Vehicle Application
- Vehicle Internal Message Definitions
- Vehicle External Message Definitions

Soldier Application
- Soldier Internal Message Definitions
- Soldier External Message Definitions

Air Inter-Platform Communication: LOSA MDA Artefacts

Sea Inter-Platform Communication: LOSA MDA Artefacts

Air
- HUMS
- 105 Gun

Sea
- HUMS
- Mortar

Base
- 105 Gun

Vehicle
- 30mm Rarden

Soldier
- Soldier Casualty Location Beacon

COI(L) Node Fabric

Air Message Definitions

Sea Message Definitions
MDA at the MoD

Summary

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk
Simplify Complexity

- Complexity and change are our enemies
- Abstraction is how we attack complexity... and defend against change... to reduce time and cost of system delivery
- Abstraction is used for:
  - Separation of concerns into domains to manage complexity and support extensibility
  - Platform Independent Domain Models to facilitate technology change
  - Simple modelling formalism to enable universal communication

Simplify the Process through executable model translation

Simplify the Model through domain based separation of concerns

Simplify the Formalism through platform independent xUML
Benefits of Model Driven Architecture Strategy

Model Driven Architecture for LOSA promotes:

- **Effective management of complexity**
- **Cost-effective maintenance of a single common model**
- **Large-scale reuse across base, vehicle and soldier architectures**
- **Compatibility with any present or future middleware**
- **Easier technology and capability insertion (including UORs)**
- **Automatic generation of consistent open interface definitions based on multiple languages and multiple messaging strategies**
MDA at the MoD

The End - Thank You

Chris Raistrick
chris.raistrick@abstractsolutions.co.uk