Introduction

- **Tutorial goals:**
  - Outline challenges in developing complex, high-performance systems
  - Demonstrate MDA PIM modeling, system deployment, and topology tuning
  - Highlight key characteristics of an effective MDA development environment
Complex System Development

• **Building RT/E Software is Difficult:**
  – Escalating Feature Complexity
  – Challenging System Execution Requirements
    • Reliability, Availability
    • Speed, Space, Environment
    • High Availability
  – Aggressive Competition
  – Diverse, Unique Platforms

Complex System Development

• **Common approaches have a poor record:**
  – 76% late or cancelled
  – 53% over budget
  – 87% delivered functionally incomplete
  
  *(source: Electronic Market Forecasters)*
Code-Driven Development

• Code-focused development approach
  – Architecture with implementation focus
  – Primary communication through code
  – All software maintained at implementation-level

• Single concept space
  – Mix of problem space and implementation detail
  – Hard to effectively decompose

Realities of Code-Driven Development

• More Complexity
  – Abstraction level quickly drops to code level
  – OOP over-generalization leads to inefficiency
  – Difficult to maintain
  – Achieving runtime performance is increasingly elusive
Realities of Code-Driven Development

• Brittle and Short Lived
  – Requirements Change
  – Platform Migration
  – Calcified by Optimizations and Patching

Realities of Code-Driven Development

• Difficult to Conceive, Implement and Follow
  – Coding conventions, rules and review can be critical to success
  – Architecture enforcement is manual

• Project must invent and implement
  – Interface mechanism layers
  – Portability and convenience mechanisms
  – Project-specific checking
Key Challenges

- Managing Complexity
- Communication
- Architectural Flexibility and Durability
- Achieving Runtime Performance

MDA for Real-Time Embedded

- Architecture-focused approach for developing high-complexity, high-performance systems:
  - Facilitate an architectural focus throughout system lifecycle
  - Manage overall system complexity with separation
  - Transform models to deployable code
MDA for Real-Time Embedded

• **MDA leverages modeling technology to gain independence and separation:**
  – Problem space from implementation space
  – One logical component (subject matter domain) from another

• **Partitioning inherent within MDA provides:**
  – Independence from any specific implementation.
  – Ease integration among heterogeneous components.
  – Large-grained reuse and system longevity.
MDA – PIM

• **Platform Independent Model (PIM):**
  – Complete, executable, verifiable UML model
  – Focused on problem-space solution analysis
  – Divorced from implementation space

• **PIM semantics facilitate**
  – Rapid development
  – Ease of adoption and understandability
  – Deployability
MDA - PSM

• **PSM - Platform Specific Model**
  – Combines PIM with details for a particular execution platform
  – Incorporates needed adaptations and optimizations
• For RT/E systems, the PSM is in a 3GL – implementation code:
  – Executable, deployable without manual edits
  – Rapidly re-generates for quick debug cycles

MDA - Transformation

• **Transforming a PIM to Implementation**
  – Convert PIM elements to implementation
  – Apply implementation patterns via code templates
  – Designer guides transformation via markings
  – Customized templates facilitate project-specific optimizations
MDA – Key Characteristics

• The PIM is the source for the system:
  – Models are executable and complete
  – Problem-space features are defined in analysis models
  – Execution-specific extensions such as porting and performance optimizations
    • Done in Design
    • Automatically leveraged across all models

• The architecture of the product is enforced by the process
  – Models, code, and doc are always synchronized
  – Reuse is independent of deployment environment and topology
  – High software quality and flexibility through product life
MDA Benefits

• Improved Software Quality
• Increased Productivity
• More Predictable Schedules
• Smoother Integration
• Reduced Maintenance Costs

MDA Summary

• MDA is an approach for engineering software where:
  – Complexity is managed by separation.
  – Communication is facilitated with raised abstractions, based on industry standards
  – Problem-space focus and automated transformation maintain architectural focus and longevity.
System Development

• **Architectural Analysis**
  - Use Cases
  - Domains

• **Platform Independent Modeling**
  - Classes
  - Scenarios
  - State Machines
  - Actions
  - Dynamic Verification

---

PIM - Use Cases

• **Use Cases** start modeling at the requirements level:
  - Focus on system interactions with user/operator and other external entities
  - Outline key capabilities and identify system boundaries
  - Provides a context for sets of system-level scenarios
PIM - Domains

• Partition a product or product family into logical components - subject matter *domains*:
  – Separate, conceptual universe
  – Inhabited by a related set of abstractions
  – Pertaining to a distinct subject matter
  – Interacts with other domains exclusively through published services

PIM - Domains

• Domain Chart is captured with Packages and Dependencies:
**PIM - Domains**

- Manages the subject matter complexity of the system
- Provides a durable foundation for system development
- Domain Modeling is not inherently object oriented
  - applies equally well to all system software elements

**PIM - Classes**

- The Class Model shows a static view of data for a domain:
  - Classes belonging to domain
  - Attributes - characteristics of a class
  - Associations - relationships between classes
  - Generalization - hierarchies
- Class Models are readily understandable by non-developers
- Facilitates communication between marketing and developers
PIM - Classes

- Class Model is shown with a UML Class Diagram:

PIM - Scenarios

- Scenario Models show the behavior of class instances participating in domain-level scenarios:
  - Establish the pattern of interaction between operations, states actions, and server domains
  - UML Sequence Chart shows interactions:

<table>
<thead>
<tr>
<th>External</th>
<th>LEC/PML Interface</th>
<th>exp 1:Experiment</th>
<th>exp 2:Experiment</th>
<th>exp 3:Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**PIM – State Machines**

- **State Modeling:**
  - Defines the lifecycle for a Class
  - Outlines the Actions that a Class instance performs in response to Events

**PIM - Actions**

- **Action Programming:**
  - Defines the processing performed by an operation or state action
  - Platform-independent Action Language (PAL):
    - OMG standard UML Action Semantics
    - PIM-level, not implementation code
    - Convenient and complete set of modeling processing primitives
    - Enables automated transformation to optimized implementations
PIM - Actions

- Each domain interface operation, class operation, and state/transition action is defined by PAL:

```java
String msg;
// Find my experiment spec
Ref<ExperimentSpecification> spec = FIND CLASS
ExperimentSpecification WHERE (name == exp_name);
IF (spec)
{
  msg = "Experiment " + exp_name;
  DataLogging:LogRecord(SoftwareMechanisms:
    DestinationHandleLocal(), msg);
  LINK this A2 spec;
  //# allocate appliances
  FOREACH required_appliance_type = spec->A5
  {
    Ref<LabAppliance> appl =
      LabAppliance:findAvailableAppliance(
        required_appliance_type.type);
    IF (appl != NULL)
    {
      LINK this A4 appl;
      (continued ...)
  }
```

PIM - Dynamic Verification

- Dynamic Verification
  - Execution of analysis models to verify behavior of analysis
  - Independent of target execution environment
  - Within the context of a single domain, or a tightly related set of domains
  - Unique to translation - only a rigorous and complete PIM-based approach affords this early opportunity to validate behavior

- Test scenarios are derived from Scenario Models
• Dynamic Verification
  – Spotlight provides visibility into the execution of analysis via "model-level debugging":

PIM Lifecycle

analysis

<table>
<thead>
<tr>
<th>Class Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Modeling - Scenarios and States</td>
</tr>
<tr>
<td>Action Modeling</td>
</tr>
<tr>
<td>Dynamic Verification</td>
</tr>
</tbody>
</table>
Tutorial Example Overview

- **ExperimentControl**
  - Sample system with moderate complexity
  - Multiple subject matters for some domain complexity
  - Some synchronous domains, some state machines

ExperimentControl – Overview

- **Controls experiment execution in a laboratory setting**
  - Secure required resources
  - Track budgets for supplies
  - Initiate experiment execution
  - Work area climate (temperature) control
  - Monitor environment contamination
• Core operational Use Cases:

- Experiment Execution
- Experiment supply dispenser
- Resource Allocation
- Central lab fiscal management system
- Lab Manager
- Emergency Containment
- Environment zone contamination sensor
- Lab Support Technician

• Support operations:

- Initialization
- Experiment supply dispenser
- Experiment appliance
- Worker in zone
- Environment Control
- Heater blower zone valve
- Lab Manager
- Environment zone temp sensors
- Supply Replenishment
- Lab Support Technician
- Central lab fiscal management system
- Lab subly ordering interface (to vendor)
ExperimentControl – New Requirements

- **Add Upsadaisium replenishment capability:**
  - Add executive capability to FacilitiesManagement domain
  - New hardware to be added – liquefied gas pump
  - Add new domain to implement gas transfer controls
Use Case

Perform Required Model Changes

• System Scenario Analysis
• Class Modeling
• Domain Scenario Analysis
• State Modeling
• Action Programming
• Dynamic Verification
The Eclipse Environment

- **What is Eclipse**
  - Software engineering data and control backplane
  - Extendable IDE
  - Open source
- **Originally developed by IBM**
  - Streamline development process
  - Configurable solution, easy to extend and adapt
  - Provide the mechanical infrastructure for an “ecosystem” of practitioners and providers.

General Eclipse IDE Layout

- **Workbench**
  - Perspectives
  - Views
- **Workspace**
  - Projects
  - Resources
MDA Modeling in Eclipse

- **Rational Software Modeler (RSM)**
  - UML 2.0 modeling

MDA Modeling in Eclipse

- **PathMATE transformation environment**
  - Powerful, open MDA environment
  - Template-based transformation
  - Extensive model checking
  - Behavior verification
  - Automatic report generation
  - Model-level debug on target
  - Highly optimized C, C++, Java
Initial Deployment

- Create running version of system with initial topology

Interface Instrumentation

- Use Spotlight to trace inter-domain traffic
Second Deployment

- Redeploy system with refined topology

open discussion
references

- MDA white papers, intro movie, blog, other resources: [www.pathfindermda.com](http://www.pathfindermda.com)
- Additional information about the PathMATE environment: [www.pathmate.com](http://www.pathmate.com)
- RSM is available from IBM Rational: [www.ibm.com](http://www.ibm.com)
- General information on UML and MDA: [www.omg.org](http://www.omg.org)
- Information on Eclipse: [www.eclipse.org](http://www.eclipse.org)