Presentation Overview

Why Model Driven Development?
- Speed Time to Market
- Improve communication, quality
- Manage complexity
- Raise productivity

• PI-MDD with PathMATE
• Keys to Achieving High Performance
Why Model Driven Development?

• Speed Time to Market
  – Immediate and continuous executable deployment
    • Test deployments
    • Target deployments
  – Standard techniques and technology
    • Avoid reinventing the wheel
    • Rapidly carry forward industry lessons learned
  – Highly automated
    • Rapid code production
    • Avoid human error in routine tasks
Why Model Driven Development?

- **Improve communication**
  - Among developers
  - With Systems Engineering
  - With all stakeholders

- **Models can be built with key characteristics that improve communications**
  - Graphical language
  - Precise semantics
  - Higher level of abstraction
Why Model Driven Development?

- **Manage complexity**
  - Address increasing function/feature content
  - Achieve deployment flexibility and performance
    - Topology
    - Environment
    - Complex optimizations
  - Apply Product Line architecture, development
Why Model Driven Development?

• **Improve system quality**
  – Apply a defined, enforceable process
  – Break problems down to facilitate resolution
  – Work more effectively through better communication
  – Reduce human error through automation
Why Model Driven Development?

- **Raise developer productivity**
  - Raise the level of abstraction
  - Follow a known, proven process
  - Ease maintenance/extension
Why Model Driven Development?

➤ Deliver systems
  • quality product
  • meet customer need
  • competitively priced
  • on time
• Why Model Driven Development?

➢ PI-MDD with PathMATE
  – PI-MDD Modeling
  – Methodology Summary – PI-MDD

• Keys to Achieving High Performance
Model-Driven Development

• In **Model Driven Development**, models – not code – are the central focus of system development
  – Models raise the level of abstraction - communication is more effective with all stakeholders
  – Complexity is addressed through separation
    • Problem space from implementation issues
    • Logical architecture and components
  – A durable system architecture is created and maintained
Platform-Independent MDD

• MDD includes a wide range of techniques and technologies

• **Platform-Independent MDD** is an advanced form of modeling
  – Modeling is focused on the problem space
  – Architecture remains a continuous focus of development
  – High degree of automation and optimization
  – Standards based and built upon open technology
Model Driven Architecture

• A set of Object Management Group (OMG) Standards
  – Open, vendor neutral
  – Facilitates structured, automated development approaches
  – Pivots on machine readable models and model transformation

• PI-MDD is built upon OMG MDA standards
• *Separate* platform independent solution from deployment specifics
  
  – Platform Independent Model (PIM)
    
    • Complete, executable UML model of system functional solution
    • Independent of target topology, environment
    • Transformed to
      
      – Simulation/verification executable
      – Deployment - platform-specific implementation
      – System documentation
      – Interchange format
    
  – Most complex systems can be largely platform independent
You are here:

• Why Model Driven Development?
• PI-MDD with PathMATE
  • PI-MDD Modeling
  ➢ Methodology Summary – PI-MDD
    • Analyze behavioral requirements with Use Cases
    • Develop a logical architecture with Domains
    • Construct PIMs for modeled domains
    • Construct the deployed system
• Keys to Achieving High Performance
• Use Case analysis
PI-MDD Method

- Develop a logical architecture with focus on problem-space Domains
  - Capture the logical architecture on a domain chart
    - Some domains have a PIM solution
    - Some domains interface to a non-modeled code-level solution
  - Specify inter-domain behavior with scenario Sequence Charts
PI-MDD Method

• Logical Architecture is problem-space focused
  – Highly durable
  – Deployable to varying topologies
PI-MDD Method

- Construct PIMs for modeled domains
  - Class diagram with classes and relationships
PI-MDD Method

- Construct PIMs for modeled domains
  - Specify inter-class collaborations with scenario Sequence Charts
• Construct PIMs for modeled domains
  – Capture state machines for active classes

```
C Subject
- threatLevel : Integer
- priority : Integer
- id : Integer
- location : gg_space_t
- getNextSubjectId ()
- findTraversedSpaces ()
- findAvailableFieldOfView ()
- findBestFieldOfView ()
- selectFieldOfView ()
```
• Construct PIMs for modeled domains
  – Specify actual behavior in UML platform independent action language
PI-MDD Method

- Execute PIMs
  - Dynamically verify PIM correctness through execution
PI-MDD Method

• Construct the deployed system
  – Establish implementation architecture
  – “Mark” PIM to
    • Indicate topological allocations
    • Specify optimization decisions
  – Transform PIMs to optimized platform specific implementation
  – Build in code-level solutions for non-modeled (realized) domains
  – Test
    • Features
    • Performance
• Why Model Driven Development?
• PI-MDD with PathMATE

- Keys to Achieving High Performance
  - Incorporating/Interfacing with External Components
    - Deploying to Multiple Platforms/Topologies
    - Performance Optimization
Integrating with Legacy Code

• Interface with legacy/handwritten code via domain
  – Only interface is abstracted – no painful/bulky “reverse engineering” of old code
  – Automated generation of interfaces from modeled components
  – Provides encapsulation/bridging

• Advanced performance optimizations
  – Extend mechanisms to use existing infrastructure
  – Customize code generation rules to use/interface with legacy primitives
/====================================================================
geo_distance_t LocationServices::UnpackX(geo_location_t location)
{
    // Mask out Y and shift down X, convert from tenths to miles
    int x_hundredths = (location & LS_UPPER_MASK) >> LS_SHIFT_HALF_WORD;
    geo_distance_t x = x_hundredths / 100.0;
    return x;
}
High Performance with PI-MDD

• Why Model Driven Development?
• PI-MDD with PathMATE

➢ Keys to Achieving High Performance
  – Incorporating/Interfacing with External Components
  ➢ Deploying to Multiple Platforms/Topologies
  – Performance Optimization
System Topology

• Define processing elements
  – Processors/Processes
  – Tasks/Threads

• Define communication paths
  – Network protocols
  – Shared memory
  – Target-specific data transport (MPI, Mercury PAS)
Example Allocation – Alternative 1

# MAIN processor
Domain, SAMP.CabinEnvironment, ProcessID, MAIN
Domain, SAMP.ChassisDynamics, ProcessID, MAIN
Domain, SAMP.DeviceInterface, ProcessID, MAIN
Domain, SAMP.GeoLocation, ProcessID, MAIN
Domain, SAMP.LocationServices, ProcessID, MAIN
Domain, SAMP.NavigationSupport, ProcessID, MAIN
Domain, SAMP.PowertrainControl, ProcessID, MAIN
Domain, SAMP.VehicleControl, ProcessID, MAIN

# COCKPIT processor
Domain, SAMP.DriverInterface, ProcessID, COCKPIT
Domain, SAMP.GUIToolkit, ProcessID, COCKPIT

# Run locally in ANY process/task
Domain, SAMP.SoftwareMechanisms, ProcessID, ANY
Example Allocation – Alternative 2

# COCKPIT processor
Domain,SAMP.DriverInterface,ProcessID,COCKPIT
Domain,SAMP.GUIToolkit,ProcessID,COCKPIT

# MAIN processor
Domain,SAMP.CabinEnvironment,ProcessID,MAIN
Domain,SAMP.GeoLocation,ProcessID,MAIN
Domain,SAMP.LocationServices,ProcessID,MAIN
Domain,SAMP.NavigationSupport,ProcessID,MAIN
Domain,SAMP.PowertrainControl,ProcessID,MAIN
Domain,SAMP.VehicleControl,ProcessID,MAIN

# CHASSIS_CONTROL processor
Domain,SAMP.ChassisDynamics,ProcessID,CHASSIS_CONTROL

# Run locally in ANY process/task
Domain,SAMP.DeviceInterface,ProcessID,ANY
Domain,SAMP.SoftwareMechanisms,ProcessID,ANY
• Why Model Driven Development?
• PI-MDD with PathMATE

➤ Keys to Achieving High Performance
  – Incorporating/Interfacing with External Components
  – Deploying to Multiple Platforms/Topologies

➤ Performance Optimization
Performance Optimizations

- Start with documented performance requirements
  - Start with functionally correct baseline
  - Review model for conciseness and “frugality”
- Initiate performance testing
- Address Performance Issues
  - Simplify/streamline mechanisms
  - Incorporate specialized infrastructure
  - Transform to alternative implementation pattern
Let's create 2 new markings to directly generate hardware register interface code — **Register** and **Bit**

```
# Set up status bits for digital control channel
Attribute,SAMP.DeviceInterface.DigitalControlChannel.busy/Register,DI_CTL_REG
Attribute,SAMP.DeviceInterface.DigitalControlChannel.busy/Bit, DI_STATUS_BIT_BUSY
Attribute,SAMP.DeviceInterface.DigitalControlChannel.online/Register,DI_CTL_REG
Attribute,SAMP.DeviceInterface.DigitalControlChannel.online/Bit, DI_STATUS_BIT_ONLINE
Attribute,SAMP.DeviceInterface.DigitalControlChannel.overloaded/Register,DI_CTL_REG
Attribute,SAMP.DeviceInterface.DigitalControlChannel.overloaded/Bit, DI_STATUS_BIT_OL
```
Create a custom templates to use **Register** and **Bit**

```c
// Generate code for an attribute read accessor
[ASSIGN reg_marking = PROPERTY(action.attribute, "Register", "")]
[IF (reg_marking != "")]
    [/* Use direct register access pattern */]
    [ASSIGN bit_marking = PROPERTY(action.attribute, "Bit", "")]
    MyHardwarePackage::AccessRegisterBit([reg_marking], [bit_marking]);
[ELSE /* Use default data member accessor */]
    [action.instanceReference]->[action.attribute.name];
[ENDIF]
```

Example generated code for online attribute:

```c
MyHardwarePackage::AccessRegisterBit(DI_CTL_REG, DI_STATUS_BIT_ONLINE);
```
Summary

- To manage complexity AND to achieve high performance
  - Use separation to manage complexity
  - Use open transformation and execution technology to optimize the implementation
• Mission:
  – Generate immediate, bottom line, business results in customer organizations through the adoption of model driven approaches and technology for systems engineering, software architecture and development.

• The leading provider of model-driven technology, training and expertise for high performance, embedded, and real-time systems development:
  – Formed in 1995 to bring more complete solutions to customers.
  – We provide training, mentoring, tools, and components for systems engineering, software architecture and development.
Contact Us

Peter Fontana: 508-568-0068 x111
peterf@pathfindersolns.com

Bart Shappee: 508-568-0068 x116
barts@pathfindersolns.com