Model-Based Integration of Reusable Component-Based Avionics Systems

Wendy Roll

The Boeing Company
Phantom Works
St. Louis, MO

This work was sponsored by the DARPA/IXO Model-Based Integration of Embedded Software program, under contract F33615-00-C-1704 with Air Force Research Laboratory Information Directorate, Wright-Patterson Air Force Base.
Outline

• DARPA Model-Based Integration of Embedded Software Program Introduction
• Boeing Open Experimental Platform Overview
• Model-Based Integration Vision
  – Context
  – Multi-view Modeling
  – Model-based Analysis
  – Model-based Composition
  – Resultant Process
• Experimentation
• Conclusion
MoBIES Objectives

“The objective of MoBIES is to develop the technology to flexibly integrate the physics of the underlying domain with the embedded software design tools in order to custom-tailor the software process to the application”

Impact if the Program is Successful:

<table>
<thead>
<tr>
<th>MoBIES Technology Advance</th>
<th>Impact for Department of Defense Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software design tools will be tailored to the application domain</td>
<td>• Physical constraints will be automatically integrated: less after-the-fact testing necessary</td>
</tr>
<tr>
<td></td>
<td>• Designers can program in their own languages: fewer potential errors</td>
</tr>
<tr>
<td></td>
<td>• Fewer bugs due to implementation issues: timing, resources, failures</td>
</tr>
<tr>
<td>Multiple design tools can be integrated in customized, application-specific suites</td>
<td>• Commercial tool vendors can provide modular, customized components: COTS tools will be more available and suitable</td>
</tr>
<tr>
<td></td>
<td>• Multiple design views can accommodate software collaboration with automated configuration control: fewer errors and reduced integration time.</td>
</tr>
<tr>
<td>Code can be automatically produced with correct-by-construction generators</td>
<td>• Larger, more complex programs can be written without the verification and validation (V&amp;V) roadblock</td>
</tr>
<tr>
<td></td>
<td>• Reduced time to produce executable code</td>
</tr>
</tbody>
</table>
Boeing OEP Project Summary

- Develop Open Experimental Platform (OEP)
- Define transitionable challenge problems
- Collaborate with integration technology researchers
- Experiment with and evaluate integration technologies for embedded weapon systems
- Demonstrate technology applicability and affordability
The Avionics Software Integration Challenge

- Reuse-based development approaches can dramatically improve cost, quality and cycle time.

- Cross-cutting extra-functional properties are endemic to embedded real-time systems and hinder reuse.

How do we compose systems from reusable components while satisfying large-scale embedded system requirements?

- Including
  - Hard and soft real-time deadlines
  - Fault tolerance
  - Distribution…
Context

• Successful transition requires insertion of technology into existing process
  – Extend existing tools
  – Add new tools where needed

• Boeing Bold Stroke initiative
  – Existing open systems architecture based product line for avionics systems
  – Reusable components

• UML/Rational Rose
• Leverage existing work by engineers

Reuse Library

Essential Common Components

Desired Pluggable Common Components

OFP Configurator

Project Specific Components

Essential Project Specific Plug-ins

Project Library
Product Line Development

Components

Common

HelperClass
FacadeClass
CommonClass
RefClass
HelperClass
RefHelperClass

Create common components

Project Specific

P1 Configurator
P2 Configurator

Functional

Non-Functional

Create project specific components

Extend common/create plugs for project specific req'ts

Create common components
Model-Based Component Integration Approach

Focus Area

Model Composition

Analyze

Generate Configuration

Build

Test

Model Reusable Components

OMG RTWS 2003
Wendy Roll - 8
Challenges for Model-Based Component Integration

• Multi-view modeling
  – Represent system features that impact cross cutting constraints in feature-appropriate models
    • Process view models
    • Deployment view models
  – Integrate multiple views

• Model-based analysis
  – Apply analytic methods to the design models to ensure satisfaction of cross cutting embedded constraints

• Model-based system configuration
  – Use system models to generate integration code needed to assemble a system from components
Process Related Views

- Logical fault management
  - Operational and backup modes and components
  - Components that need replicated backups

- Execution dependencies
  - Triggers and trigger types
  - Trigger based dependency graphs
  - Execution rates for the roots of dependencies

- Threads
  - Threads and their associated rates and priorities
**Deployment Related Views**

- **Physical fault management**
  - Relationships between fault modes and physical resources

- **Component quality of service**
  - Execution rates
  - Importance
  - Resources requirements

- **Process**
  - System physical resources
  - Allocation of threads to processes

- **Component allocation**
  - Components that are strongly coupled
  - Allocation of components to processors and processes
  - Parameters for automatic generation of integration code
    - Identify and generate CORBA stubs and skeletons as needed
Model-Based Analysis

Having models that capture cross-cutting aspects of a system is the basis for analysis

- **Fault-tolerance**
  - Determine status of components in various fault scenarios
  - Support allocation of backup components to processors to meet fault-tolerance goals

- **Execution dependencies**
  - Identifying cyclic dependencies
  - Ensuring consistency of dependency graphs
  - Using dependency graphs to identify execution requirements for timing analysis

- **Timing analysis**
  - Schedulability
  - Utilization
Model-Based Configuration

- Automatic generation of configuration code based on models can yield increased speed and quality and reduced cost
  - Manual creation of integration code is time consuming, tedious and error prone
  - Much integration code is fully determined by a model of the system configuration
  - Tools already exist that generate much similar code
    - CORBA IDL compilers, etc.
Resultant Process

Input Translation
- Parse Rose
- Filter/Translate To XML
- Model Importer

Inter-View Translation
- Logical Fault Mgmt
- Threading
- Event Dependency
- Invocation Dependency
- Component Thread Map.

Analysis Translation
- Build
- Test
- Fault Tolerance
- Event Dependencies
- Timing
- Instrumentation

Well Defined XML Interfaces Enable Tool Integration

Configuration Translation
- Generate Configuration

OEP Configuration
- Build
- Test

Fault Tolerance
- Event Dependencies
- Timing
- Instrumentation

Application Component Library (ACL)
- Model Components
- Model Editor

Logical Fault Mgmt
- Process
- Component Allocation
- Physical Fault Mgmt

Component Thread Map.
**Experimentation Approach**

**Development Scenarios and ...**

<table>
<thead>
<tr>
<th>Logical Fault Mgmt</th>
<th>Thread Mgmt</th>
<th>Invocation Dependency</th>
<th>Event Dependency</th>
<th>Component Thread Map.</th>
<th>Process</th>
<th>Component Allocation</th>
<th>Physical Fault Mgmt</th>
<th>Analysis</th>
<th>Translational</th>
</tr>
</thead>
</table>

**Product Scenarios...**

...Comprise Experiments

<table>
<thead>
<tr>
<th>PS 1</th>
<th>PS 2</th>
<th>PS 3</th>
<th>PS 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Single</td>
<td>Exp 1-1</td>
<td>Basic Dist.</td>
<td>Exp 2-2</td>
</tr>
<tr>
<td>Rep. Single</td>
<td>Exp 2-3</td>
<td>Exp 2-4</td>
<td></td>
</tr>
<tr>
<td>Basic Dist.</td>
<td>Exp 3-1</td>
<td>Exp 3-2</td>
<td></td>
</tr>
<tr>
<td>Exp 3-3</td>
<td>Exp 3-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp 4-3</td>
<td>Exp 4-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp 5-3</td>
<td>Exp 5-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Development Scenarios**

- PS 1
  - Basic Single
  - Exp 1-1

- PS 2
  - Rep. Single
  - Exp 2-1
  - Exp 2-2

- PS 3
  - Basic Dist.
  - Exp 2-3
  - Exp 2-4

- PS 4
  - Exp 3-1
  - Exp 3-2
  - Exp 3-3
  - Exp 3-4
  - Exp 4-3
  - Exp 4-4
  - Exp 5-3
  - Exp 5-4

**Development Scenarios and Product Scenarios**

- PS 1: Basic Single
- PS 2: Rep. Single
- PS 3: Basic Dist.

**Experiments**

- DS 1: Basic
  - Exp 1-1
- DS 2: Evt Analysis
  - Exp 2-1
  - Exp 2-2
- DS 3: Scheduling
  - Exp 3-1
  - Exp 3-2
- DS 4: Fault Toler.
  - Exp 4-3
  - Exp 4-4
- DS 5: Full
  - Exp 5-3
  - Exp 5-4
Experimentation Results

- Demonstrated capability to:
  - Model multiple views
  - Perform timing analysis
  - Generate configuration code
  - Initialize and run configured system

- ... Using an integrated set of tools from multiple researchers
Current Status

- Heavy Focus Now On Preparing For Transition Of Component-Oriented Programming
  - Filling in capability gaps
  - Increasing scalability, usability…
  - Optimizing run-time performance
- Realistic evaluation of
  - Overall approach
  - Integrated tool sets
  - Individual tools
Conclusions

• Model-based integration technologies promise dramatic advances in component-based system quality, affordability, and timeliness
  – Integrated tool support
  – …for component-based product line development
  – …satisfying cross-cutting constraints
• …And address unmet needs of product integrators
  – Automates many manual steps
  – Predicts system correctness prior to construction