Model Driven Development for Distributed Real-time & Embedded Systems

or

“Why I’d Rather Write Code That Writes Code Than Write Code”

MODELS 2005 Conference, Wednesday, October 12, 2005

Dr. Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee
The Promise

- Develop standardize technologies that:
  1. **Model**
  2. **Analyze**
  3. **Synthesize** &
  4. **Provision**

complex software systems
The Reality

- Architects (sometimes) use UML to express software designs at a high-level
- Developers write & evolve code manually

We need/ought to be able to do much better than this!
Sources of the Problems

Inherent & Accidental Complexities

- More automated specification & synthesis of
  - Broader range of target domain capabilities
  - Model interpreters & transformations
  - Static & dynamic quality of service (QoS) properties
- Round-trip engineering from models $\leftrightarrow$ source
- Poor support for debugging at the model level
- Version control of models at the model level

Impediments of human nature

- Organizational, economic, administrative, political, & psychological barriers

Ineffective technology transition strategies

- Disconnects between methodologies & production software development realities
- Lack of incremental, integrated, & triaged transitions
New Demands on Distributed Real-time & Embedded (DRE) Systems

Key challenges in the **problem space**

- Network-centric, dynamic, very large-scale “systems of systems”
- Stringent simultaneous quality of service (QoS) demands
- Highly diverse, complex, & increasingly integrated/autonomous application domains

Key challenges in the **solution space**

- Vast accidental & inherent complexities
- Continuous evolution & change
- Highly heterogeneous (& legacy constrained) platform, language, & tool environments

Mapping & integrating *problem artifacts* & *solution artifacts* is hard
Mission-critical DRE systems have historically been built directly atop hardware
- Tedious
- Error-prone
- Costly over lifecycles

Consequence: Small changes to legacy software often have big (negative) impact on DRE system QoS & maintenance
Evolution of DRE Systems Development

Mission-critical DRE systems historically have been built directly atop hardware
- Tedious
- Error-prone
- Costly over lifecycles

Middleware has effectively factored out many reusable services from traditional DRE application responsibility
- Essential for *product-line architectures*
- Middleware is no longer the primary DRE system performance bottleneck

Middleware alone is insufficient to solve key large-scale DRE system challenges!

Technology Problems
- Legacy DRE systems often tend to be:
  - Stovepiped
  - Proprietary
  - Brittle & non-adaptive
  - Expensive
  - Vulnerable
DRE Systems: The Challenges Ahead

- Limit to how much application functionality can be refactored into reusable COTS middleware
- Middleware itself has become very hard to use & provision statically & dynamically

- Component-based DRE systems are also very hard to deploy & configure
- There are many middleware platform technologies to choose from
DRE Systems: The Challenges Ahead

It’s enough to make you scream!
Promising Solution: **Model Driven Development (MDD)**

- Develop, validate, & standardize generative software technologies that:
  1. **Model**
  2. **Analyze**
  3. **Synthesize &**
  4. **Provision**

multiple layers of middleware & application components that require simultaneous control of multiple QoS properties end-to-end

- Partial specialization is essential for inter-/intra-layer optimization & advanced product-line architectures

**Goal is to enhance developer productivity & software quality** by providing **higher-level languages & tools** for middleware/application developers & users
Technology Evolution (1/4)

Programming Languages & Platforms

- Operating Systems
- Hardware
- Assembly
- C/Fortran
- Machine code

Model-Driven Development (MDD)

- State chart
- Data & process flow
- Petri Nets

Level of Abstraction

- Model
- Generated Code
- Platform

Large Semantic Gap

Translation

- Code
- Code
- Code
- Code
- Code
- Code
- Code
• New languages & platforms have raised abstraction level significantly
  • “Horizontal” platform reuse alleviates the need to redevelop common services

• There are two problems, however:
  • Platform complexity evolved faster than 3rd-generation languages
  • Much application/platform code still (unnecessarily) written manually
    • Particularly for D&C aspects
Technology Evolution (3/4)

Programming Languages & Platforms

- Level of Abstraction
- Saturation!!!!
- Components
  - Frameworks
  - Class Libraries
  - Operating Systems
  - Hardware
- Generated Code
- Framework
- Pattern Language
- Platform

Model-Driven Development (MDD)

Domain-specific modeling languages
- ESML
- PICML
- Mathematic
- Excel
- Metamodels

Domain-independent modeling languages
- State Charts
- Interaction Diagrams
- Activity Diagrams

Manual translation

Semi-automated

- OMG is evaluating MDD via MIC PSIG
  - mic.omg.org
Technology Evolution (4/4)

Programming Languages & Platforms

Model-Driven Development (MDD)

Domain-specific modeling languages
- ESML
- PICML
- Mathematic
- Excel
- Metamodels

Domain-independent modeling languages
- State Charts
- Interaction Diagrams
- Activity Diagrams

Research is needed to automate DSMLs & model translators

Level of Abstraction

Saturation!!!!
Generic Modeling Environment (GME)

“Write Code That Writes Code That Writes Code!”

GME Architecture

Application Developers (Modelers)

MDD Tool Developers (Metamodelers)

Supports “correct-by-construction” of software systems

GME is open-source: www.isis.vanderbilt.edu/Projects/gme/default.htm
MDD Application Development with GME

• Application developers use modeling environments created w/MetaGME to build applications
  • Capture elements & dependencies visually
MDD Application Development with GME

• Application developers use modeling environments created w/MetaGME to build applications
  • Capture elements & dependencies visually
**MDD Application Development with GME**

- **Application developers** use modeling environments created w/MetaGME to build applications.
- Capture elements & dependencies visually.
- Model interpreter produces something useful from the models.
  - e.g., 3rd generation code, simulations, deployment descriptions & configurations.
MDD Tool Development in GME

- Tool developers use MetaGME to develop a domain-specific graphical modeling environment
- Define syntax & visualization of the environment via metamodeling
Tool developers use MetaGME to develop a domain-specific graphical modeling environment.

- Define syntax & visualization of the environment via metamodeling.
- Define static semantics via Object Constraint Language (OCL).
MDD Tool Development in GME

- Tool developers use MetaGME to develop a domain-specific graphical modeling environment
- Define syntax & visualization of the environment via metamodeling
- Define static semantics via Object Constraint Language (OCL)
- Dynamic semantics implemented via model interpreters
Goals of D&C Phase

- Promote component reuse
- Build complex applications by assembling existing components
- Automate common services configuration
- Declaratively inject QoS policies into applications
- Dynamically deploy components to target heterogeneous domains
- Optimize systems based on component configuration & deployment settings
OMG Component Deployment & Configuration

OMG D & C Spec (PIM & PSMs)

XML Schema Generation

D & C Profile

IDL Generation

Interchange Formats

Deployment Interfaces

Deployment Tools (generic)

Deployment Interfaces

Shipping

SW Deployer

SW Creator

SW Creator

A1

A2

Implementations

Deployment requirements

Interchange Formats

OMG Deployment & Configuration (D&C) specification (ptc/05-01-07)
MDD Example: OMG Deployment & Configuration

**Specification & Implementation**
- Defining, partitioning, & implementing app functionality as standalone components

**Packaging**
- Bundling a suite of software binary modules & metadata representing app components

**Installation**
- Populating a repository with packages required by app

**Configuration**
- Configuring packages with appropriate parameters to satisfy functional & systemic requirements of an application without constraining to physical resources

**Planning**
- Making deployment decisions to identify nodes in target environment where packages will be deployed

**Preparation**
- Moving binaries to identified entities of target environment

**Launching**
- Triggering installed binaries & bringing app to ready state

**QoS Assurance & Adaptation**
- Runtime (re)configuration & resource management to maintain end-to-end QoS

OMG Deployment & Configuration (D&C) specification (ptc/05-01-07)
Challenge 1: The Packaging Aspect

- Application components are bundled together into assemblies.
- Several different assemblies tailored towards delivering different end-to-end QoS and/or using different algorithms can be part of the package.
  - e.g., large-scale DRE systems require 100s-1,000s of components.
- Packages describing the components & assemblies can be scripted via XML descriptors.
Challenge 1: The Packaging Aspect

- Application components are bundled together into assemblies
- Assemblies convey component interconnections & implementation alternatives

- Several different assemblies tailored to deliver different end-to-end QoS behaviors and/or algorithms can be part of the package
  - e.g., large-scale DRE systems require 100s-1,000s of components
- Packages describing the components & assemblies can be scripted via XML descriptors
Packaging Aspect Problems (1/2)

Ad hoc techniques for ensuring component syntactic & semantic compatibility

Inherent Complexities

Ad hoc means to determine pub/sub support

Distribution & deployment done in ad hoc manner
Packaging Aspect Problems (2/2)

Accidental Complexities

<!– Associate components with impls -->
<assemblyImpl>
  <instance xmi:id="RateGen">
    <name>RateGen Subcomponent</name>
    <package href="RateGen.cpd"/>
  </instance>
  <instance xmi:id="GPS">
    <name>GPS Subcomponent</name>
    <package href="GPS.cpd"/>
  </instance>
  <instance xmi:id="NavDisplay">
    <name>NavDisplay Subcomponent</name>
    <package href="NavDisplay.cpd"/>
  </instance>
</assemblyImpl>

Existing practices involve handcrafting XML descriptors.

XML file in excess of 3,000 lines, even for medium sized scenarios.

Modifications to the assemblies requires modifying XML file.
MDD Solution for Packaging Aspect

Approach:
• Develop a *Platform-Independent Component Modeling Language (PICML)* to address inherent & accidental complexities of packaging
  • Capture dependencies visually
  • Define semantic constraints using Object Constraint Language (OCL)
  • Generate domain-specific metadata from models
  • Correct-by-construction
• PICML is developed using Generic Modeling Environment (GME)

www.cs.wustl.edu/~schmidt/PDF/RTAS-PICML.pdf
Example Metadata Generated by PICML

- **Component Interface Descriptor (.ccd)**
  - Describes the interface, ports, properties of a single component

- **Implementation Artifact Descriptor (.iad)**
  - Describes the implementation artifacts (e.g., DLLs, OS, etc.) of one component

- **Component Package Descriptor (.cpd)**
  - Describes multiple alternative implementations of a single component

- **Package Configuration Descriptor (.pcd)**
  - Describes a configuration of a component package

- **Top-level Package Descriptor (package.tpd)**
  - Describes the top-level component package in a package (.cpk)

- **Component Implementation Descriptor (.cid)**
  - Describes a specific implementation of a component interface
  - Implementation can be either monolithic- or assembly-based
  - Contains sub-component instantiations in case of assembly based implementations
  - Contains inter-connection information between components

- **Component Packages (.cpk)**
  - A component package can contain a single component
  - A component package can also contain an assembly

Based on OMG (D&C) specification (ptc/05-01-07)
A Component Implementation Descriptor (*.cid) file

- Describes a specific implementation of a component interface
- Describes component interconnections

Example Output from PICML Model

```xml
<monolithicImpl> [...] </monolithicImpl>
<deployRequirement>
    <name>GPS</name>
    <resourceType>GPS Device</resourceType>
    <property>
        <name>vendor</name>
        <value>
            <type><kind>tk_string</kind></type>
            <value><string>My GPS Vendor</string></value>
        </value>
    </property>
</deployRequirement>

[... Requires Windows OS ...]

<connection>
    <name>GPS Trigger</name>
    <internalEndpoint>
        <portName>Pulse</portName>
        <instance href="#RateGen"/>
    </internalEndpoint>
    <internalEndpoint>
        <portName>Refresh</portName>
        <instance href="#GPS"/>
    </internalEndpoint>
</connection>

<connection>
    <name>NavDisplay Trigger</name>
    <internalEndpoint>
        <portName>Ready</portName>
        <instance href="#GPS"/>
    </internalEndpoint>
    <internalEndpoint>
        <portName>Refresh</portName>
        <instance href="#NavDisplay"/>
    </internalEndpoint>
</connection>
```
Challenge 2: The Configuration Aspect

Component middleware is characterized by a large configuration space that maps known variations in the application requirements space to known variations in the middleware solution space.

- Hook for marshaling strategy
- Hook for the request demuxing strategy
- Hook for the concurrency strategy
- Hook for the underlying transport strategy
- Hook for the connection management strategy
- Hook for the event demuxing strategy
- Hook for the concurrency strategy
- Hook for the underlying transport strategy
Configuration Aspect Problems

Middleware developers

- Documentation & capability synchronization
- Semantic constraints & QoS evaluation of specific configurations

Application developers

- Must understand middleware constraints & semantics
  - Increases accidental complexity
- Different middleware uses different configuration mechanisms

XML Configuration Files

XML Property Files

CIAO/CCM provides ~500 configuration options
MDD Solutions for Configuration Aspect

Approach:

• Develop an *Options Configuration Modeling Language (OCML)* w/GME to ensure semantic consistency of option configurations

• OCML is used by

  • **Middleware developers** to design the *configuration model*

  • **Application developers** to configure the middleware for a specific application

• OCML *metamodel* is platform-independent

• OCML *models* are platform-specific

www.cs.wustl.edu/~schmidt/PDF/RTAS-process.pdf
Applying OCML to CIAO+TAO

• Middleware developers specify
  • Configuration space
  • Constraints
  • OCML generates config model
Applying OCML to CIAO+TAO

• Middleware developers specify
  • Configuration space
  • Constraints
• OCML generates config model
  • Application developers provide a model of desired options & their values, e.g.,
    • Network resources
  • Concurrency & connection management strategies
Applying OCML to CIAO+TAO

- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model
- Application developers provide a model of desired options & their values, e.g.,
  - Network resources
  - Concurrency & connection management strategies
- OCML constraint checker flags incompatible options & then
  - Synthesizes XML descriptors for middleware configuration
- Generates documentation for middleware configuration
- Validates the configurations
Challenge 3: Planning Aspect

Component integrators must make appropriate deployment decisions, identifying nodes in target environment where packages will be deployed.

Select the appropriate package to deploy on selected target.

Select appropriate target platform to deploy packages.

Determine current resource allocations on target platforms.

Diagram:
- **Component Repository**: Configures and Installs Package, Gets the Configured Package.
- **Target Manager**: Manages.
- **Repository Administrator**: Accesses Via URL.
- **Planner**: Creates the Deployment Plan.
- **Domain Administrator**: Creates.
Planning Aspect Problems

How to ensure deployment plans meet DRE system QoS requirements

How do you determine current resource allocations?

How do you ensure that the selected targets will deliver required QoS?

How do you correlate QoS requirements of packages to resource needs?

How do you evaluate the performance of the infrastructure before the applications are built?
MDD Solution for Planning Aspect

Approach

- Develop **Component Workload Emulator (CoWorkEr)** w/GME to allow architects to detect, diagnose, & resolve system performance & stability problems stemming from decisions during design phase

**CoWorkEr Workflow for Architects**

1. Compose scenarios to exercise critical system paths
2. Associate QoS properties with scenarios (e.g., latency, jitter, or thruput) & assign properties to components specific to paths
3. Configure workload generators to run experiments, generate path-specific deployment plans, & measure QoS along critical paths
4. Feedback results into models to verify if deployment plan meets appropriate QoS *at design time*
Integrating MDD & Middleware for Planning

CoWorkEr models system components, requirements, & constraints

Deployment Manager

Resource Allocation & Control Engine (RACE) middleware provides deployment planners

- Deployment And Configuration Engine (DAnCE) maps plans to computing nodes
- RACE controls reallocations

Gigabit Ethernet

www.dre.vanderbilt.edu/~schmidt/CoWorkEr-paper.doc
Concluding Remarks

• To realize the promise of model-driven technologies, we need to augment model-driven methodologies with a solid (ideally standard) tool infrastructure.

• Model-driven tools need to coexist with & enhance existing middleware platform technologies.

• We need to validate model-driven technologies on (increasingly) large-scale, real-world systems.

Although hard problems with model-driven technologies remain, we’re reaching critical mass after decades of R&D & commercial progress.

• Open-source CoSMIC MDD tools use Generic Modeling Environment (GME)
  • CoSMIC is available from www.dre.vanderbilt.edu/cosmic
  • GME is available from www.isis.vanderbilt.edu/Projects/gme/default.htm