Meeting the Challenges of Ultra-Large-Scale Systems via Model-Driven Engineering

February 2, 2007

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee
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
Evolution of DRE Systems Development

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

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

Technology Problems
• Legacy DRE systems often tend to be:
  • Stovepiped
  • Proprietary
  • Brittle & non-adaptive
  • Expensive
  • Vulnerable

• 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
Overview of Component Middleware

“Write Code That Reuses Code”

- **Components** encapsulate application “business” logic
- Components interact via **ports**
  - Provided interfaces, e.g., facets
  - Required connection points, e.g., receptacles
  - Event sinks & sources
  - Attributes
- **Containers** provide portable execution environment for components that have common operating requirements
- Components/containers can also
  - Communicate via a middleware bus and
  - Reuse common middleware services
**DOC Middleware for DRE Systems (1/2)**

- **CORBA** is standard middleware
- **Real-time CORBA** adds QoS to classic CORBA to control:

1. **Processor Resources**
   - Thread pools
   - Priority models
   - Portable priorities
   - Standard synchronizers
   - Static scheduling service

2. **Network Resources**
   - Protocol policies
   - Explicit binding

3. **Memory Resources**
   - Request buffering

- These capabilities address key DRE application development & QoS-enforcement challenges

---

**www.omg.org**
TAO is an open-source version of Real-time CORBA
- >> 1,000,000 SLOC
- 100+ person years of effort

Pioneered R&D on DRE middleware design & optimizations

TAO is basis for many middleware R&D efforts

Example of good synergy between researchers & practitioners

www.dre.vanderbilt.edu/TAO/
### Applying TAO in Mission-Critical DRE Systems

<table>
<thead>
<tr>
<th>Organization</th>
<th>Application Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing</td>
<td>Aircraft mission &amp; flight control computers</td>
</tr>
<tr>
<td>SAIC</td>
<td>Distributed interactive simulation (HLA/RTI)</td>
</tr>
<tr>
<td>ATDesk</td>
<td>Automated stock trading</td>
</tr>
<tr>
<td>Raytheon</td>
<td>Aircraft carrier &amp; destroyer computing systems</td>
</tr>
<tr>
<td>Cisco &amp; Qualcomm</td>
<td>Wireless/wireline network management</td>
</tr>
<tr>
<td>Raytheon &amp; Army</td>
<td>Joint Tactical Terminal (JTT)</td>
</tr>
<tr>
<td>Contact Systems</td>
<td>Surface mounted “pick-and-place” systems</td>
</tr>
<tr>
<td>Turkish Navy</td>
<td>Shipboard resource management</td>
</tr>
<tr>
<td>Krones</td>
<td>Process automation &amp; quality control</td>
</tr>
<tr>
<td>Siemens</td>
<td>Hot rolling mill control systems</td>
</tr>
<tr>
<td>LMCO &amp; Raytheon</td>
<td>Dynamic shipboard resource management (DDG)</td>
</tr>
<tr>
<td>CUSeeMe</td>
<td>Monitor H.323 Servers</td>
</tr>
<tr>
<td>Northrup-Grumman</td>
<td>Airborne early warning &amp; control (AWACS)</td>
</tr>
<tr>
<td>JPL/NASA</td>
<td>SOFIA telescope, Cassini space probe</td>
</tr>
<tr>
<td>BAE Systems</td>
<td>Joint Tactical Radio System (JTRS)</td>
</tr>
</tbody>
</table>

www.dre.vanderbilt.edu/users.html
Component Middleware for DRE Systems

- Event Notifications
- Dynamic & Static Scheduling
- Multimedia Streaming
- Security
- Component Implementation Definition Language

- Fault Tolerance & Load Balancing
- Real-time Policies & Mechanisms
- Time/space Optimizations

- Component Implementation Definition Language
- Component Deployment & Configuration

- Real-Time ORB Core
- GIOP/IIOP/ESIOPs
- IOP

- www.dre.vanderbilt.edu/
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

Middleware alone is insufficient to solve key large-scale DRE system challenges!
DRE Systems: The Challenges Ahead

It’s enough to make you scream!

Middleware
Services
DRE Applications
Operating System & Protocols
Hardware & Networks

RT-CORBAs: Apps
J2ME Apps
DRTSJ Apps

RT-CORBAs: Services
J2ME Services
DRTSJ Services

Gigabit Ethernet

E. Munch
Technology Evolution (1/4)

Programming Languages & Platforms

Model-Driven Engineering (MDE)

- State chart
- Data & process flow
- Petri Nets

Operating Systems
Assembly
Hardware
Machine code

Level of Abstraction
Translation
Translation
Translation

Large Semantic Gap

Translation
Translation
Translation

Model-Driven Engineering (MDE)

- Model
- Model
- Model
- Model
- Model
- Model
- Model

Generated Code
Platform
Technology Evolution (2/4)

- Newer 3rd-generation 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
Technology Evolution (3/4)

Programming Languages & Platforms

- Components
- Frameworks
- Class Libraries
- Operating Systems
- Hardware

Manual translation

Semi-automated

Model-Driven Engineering (MDE)

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

- Domain-independent modeling languages
  - State Charts
  - Interaction Diagrams
  - Activity Diagrams
Technology Evolution (3/4)

Programming Languages & Platforms

Model-Driven Engineering (MDE)

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

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

OMG is evaluating MDE via MIC PSIG
- mic.omg.org

Model Integrated Computing PSIG
Technology Evolution (3/4)

Programming Languages & Platforms

- Model
- Generated Code
- Framework
- Pattern Language
- Platform

Model-Driven Engineering (MDE)

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

Manual translation

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

Semi-automated

- OMG is evaluating MDE via MIC PSIG
  - mic.omg.org

Components

- Frameworks
- Class Libraries
- Operating Systems
- Hardware

C++/Java
C/Fortran
Assembly
Machine code

Level of Abstraction
Research is needed to automate DSMLs & model translators

See February 2006 IEEE Computer special issue on MDE techniques & tools
Pattern, Framework, & MDD Synergies

- Frameworks codify expertise in the form of reusable algorithms, component implementations, & extensible architectures.
- Patterns codify expertise in the form of reusable architecture design themes & styles, which can be reused event when algorithms, components implementations, or frameworks cannot.
- MDE tools codify expertise by automating key aspects of pattern languages & providing developers with domain-specific modeling languages to access the powerful (& complex) capabilities of frameworks.

There are now powerful feedback loops advancing these technologies.
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).
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.
Tool developers use MetaGME to develop a domain-specific graphical modeling environment.

- Define syntax & visualization of the environment via *metamodelling*.
- Define static semantics via *Object Constraint Language (OCL)*.
- Dynamic semantics implemented via *model interpreters*. 
MDD Application Development with GME

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

Example DSL is the “Platform-Independent Component Modeling Language” (PICML) tool
MDD Application Development with GME

- Application developers use modeling environments created w/MetaGME to build applications
- Capture elements & dependencies visually

Example DSL is the “Platform-Independent Component Modeling Language” (PICML) tool
Application developers use modeling environments created with 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

PICML generates XML descriptors corresponding to OMG Deployment & Configuration (D&C) specification.
OMG Component Deployment & Configuration

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 Deployment & Configuration (D&C) specification (ptc/05-01-07)
OMG Component Deployment & Configuration

OMG D & C Spec (PIM & PSMs)

XMLSchema Generation

IDL Generation

Interchange Formats

Deployment Interfaces

Deployment Infrastructure

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
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.
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

```xml
<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> </property> [...] Requires Windows OS [...] </monolithicImpl> <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 connection management strategy
- Hook for the concurrency strategy
- Hook for the request demuxing strategy
- Hook for the event demuxing 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

```
ACE_Svc_Conf
<!--  -->
<!--  -->
<static id="Advanced_Resource_Factory"
  params="-ORBReactorType select_mt -ORBReactorMaskSignals 0 -ORBflushingStrategy blocking" />
<static id="Client_Strategy_Factory"
  params="-ORBTransportMuxStrategy EXCLUSIVE -ORBClientConnectionFactory RW" />
<static id="Server_Strategy_Factory"
  params="-ORBConcurrency thread-per-connection" />
/ACE_Svc_Conf
```
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.
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 selected targets will deliver required QoS?

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

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

Approach

• Develop **Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)** w/GME to allow architects to detect, diagnose, & resolve system QoS problems before system integration phase

  - CoWorkEr is an component assembly of monolithic components responsible for generating respective workload
  - CoWorkEr ports can be connected to define operational strings
  - Workload Modeling Language (WML) is used to define CoWorkEr behavior
  - WML is translated to XML metadata descriptors that configure CoWorkErs

*www.cs.wustl.edu/~schmidt/PDF/QoSPML-WML.pdf*
MDD Solution for Planning Aspect

CUTS Workflow for Architects

1. Compose scenarios to exercise critical system paths
2. Associate performance properties with scenarios & assign properties to components specific to paths
3. Configure CoWorkers to run experiments, generate deployment plans, & measure performance along critical paths
4. Analyze results to verify if deployment plan & configurations meet performance requirements

www.cs.wustl.edu/~schmidt/PDF/CUTS.pdf
Integrating MDD & Middleware for Planning

CoWorkEr models system components, requirements, & constraints

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

Deployment And Configuration Engine (DAnCE) maps plans to computing nodes

RACE controls reallocations

www.cs.wustl.edu/~schmidt/PDF/DAnCE.pdf
Commercial Related Work

- Software Factories go beyond “models as documentation” by
  - Using highly-tuned DSL & XML as source artifacts &
  - Capturing life cycle metadata to support high-fidelity model transformation, code generation & other forms of automation

  [www.softwarefactories.com](http://www.softwarefactories.com)

- The Graphical Modeling Framework (GMF) forms a generative bridge between EMF & GEF, which linkes diagram definitions to domain models as input to generation of visual editors

- GMF provides this framework, in addition to tools for select domain models that illustrate its capabilities

  [www.eclipse.org/gmf/](http://www.eclipse.org/gmf/)

- openArchitectureWare (oAW) is a modular MDA/MDE generator framework implemented in Java

- It supports parsing of arbitrary models & a language family to check & transform models, as well as generate code based on them

  [www.openarchitectureware.org](http://www.openarchitectureware.org)
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