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

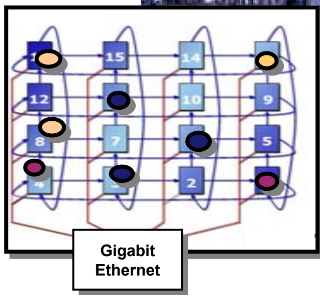
Vanderbilt University

Nashville, Tennessee

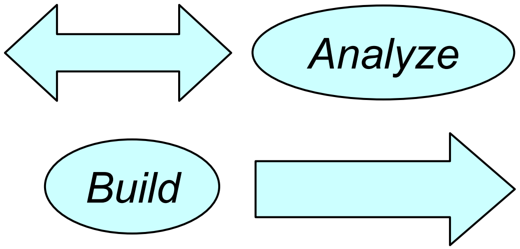
The Promise

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

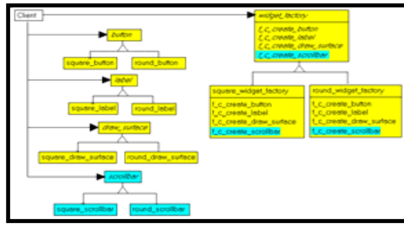
  complex software systems

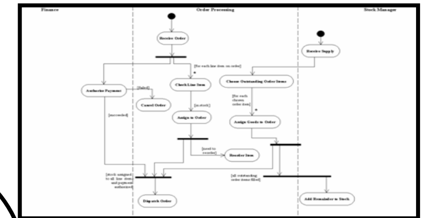


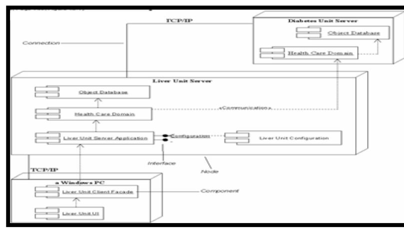
<CONFIGURATION_PASS>
<HOME>
<…>
<COMPONENT>
<ID> <…></ID>
<EVENT_SUPPLIER>
<…events this component supplies…>
</EVENT_SUPPLIER>
</COMPONENT>
</HOME>
</CONFIGURATION_PASS>

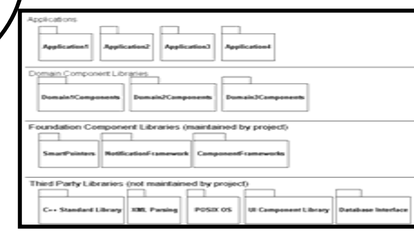


The Reality

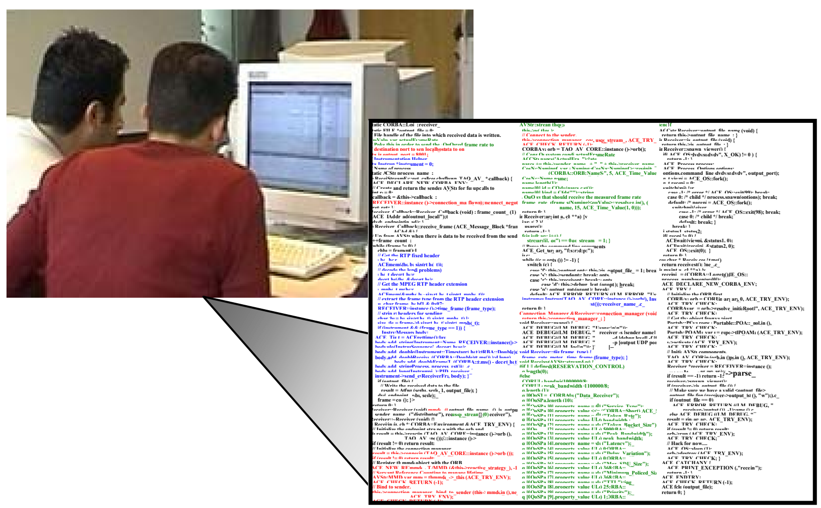








- Architects (sometimes) use UML to express software designs at a high-level

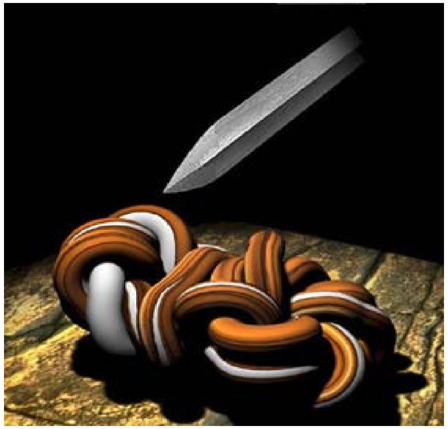


- Developers write & evolve code manually

We need/ought to be able to do much better than this!

Sources of the Problems

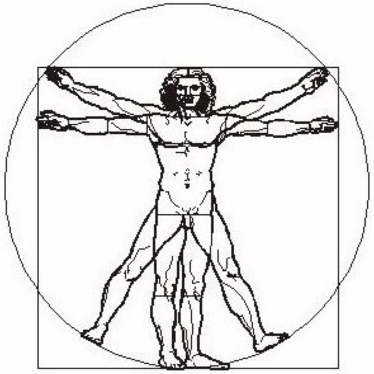
Technical Challenges



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 ↔ source
- Poor support for debugging *at the model level*
- Version control of models *at the model level*

Non-Technical Challenges



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

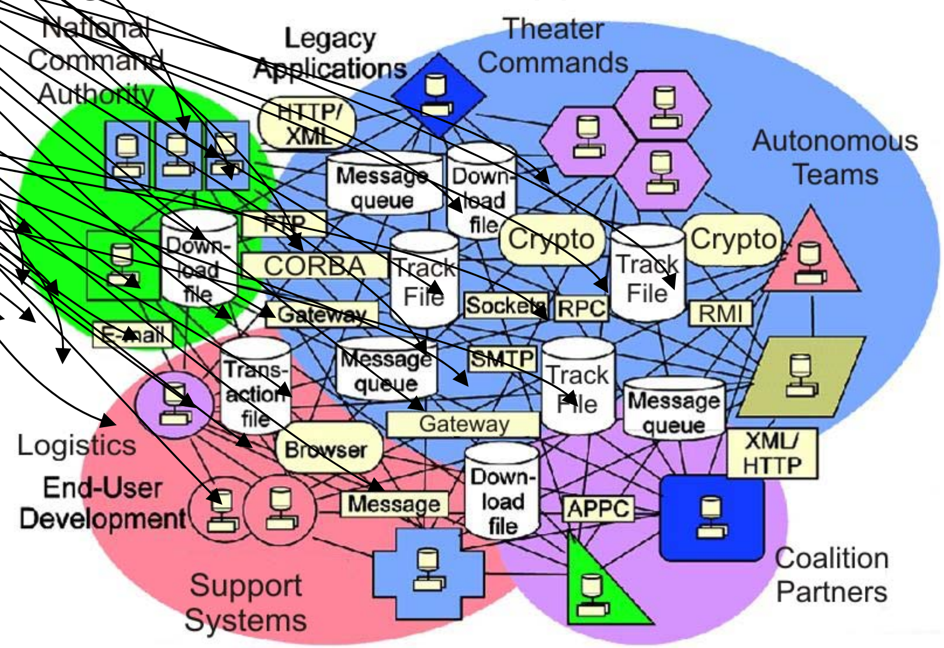
New Demands on Distributed Real-time & Embedded (DRE) Systems

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

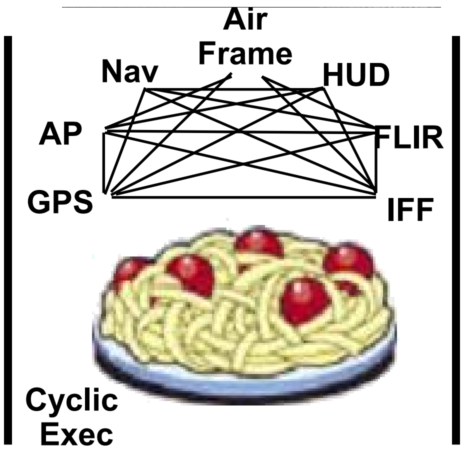
Key challenges in the ***solution space***

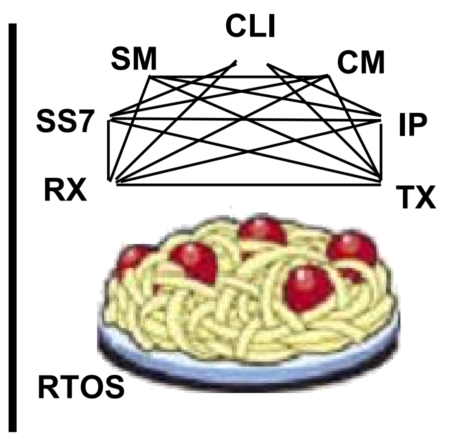
- Vast accidental & inherent complexities
- Continuous evolution & change
- Highly heterogeneous (& legacy constrained) platform, language, & tool environments



Mapping & integrating *problem artifacts* & *solution artifacts* is hard

Evolution of DRE Systems Development





Technology Problems

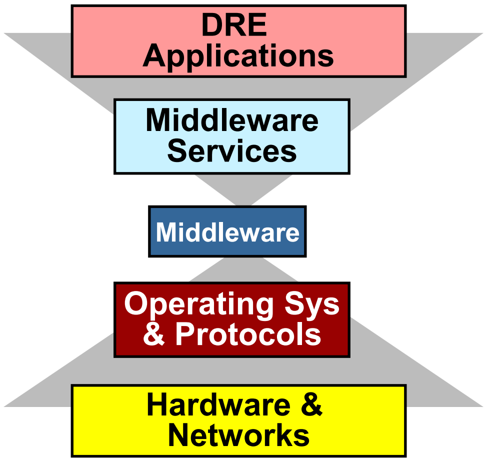
- Legacy DRE systems often tend to be:
  - Stovepiped
  - Proprietary
  - Brittle & non-adaptive
  - Expensive
  - Vulnerable

Mission-critical DRE systems have historically been built directly atop hardware

- Tedious
- Error-prone
- Costly over lifecycles

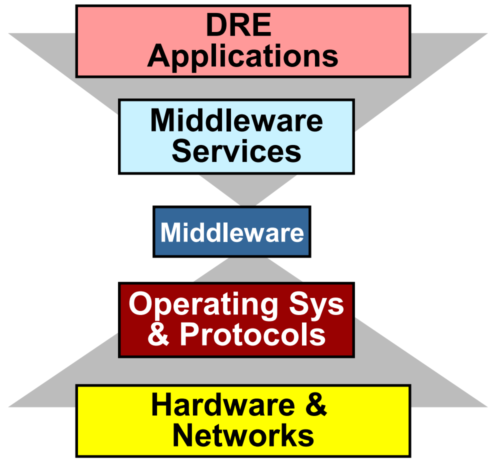
Consequence: Small changes to legacy software often have big (negative) impact on DRE system QoS & maintenance

Evolution of DRE Systems Development



Mission-critical DRE systems historically have been built directly atop hardware

- Tedious
- Error-prone
- Costly over lifecycles



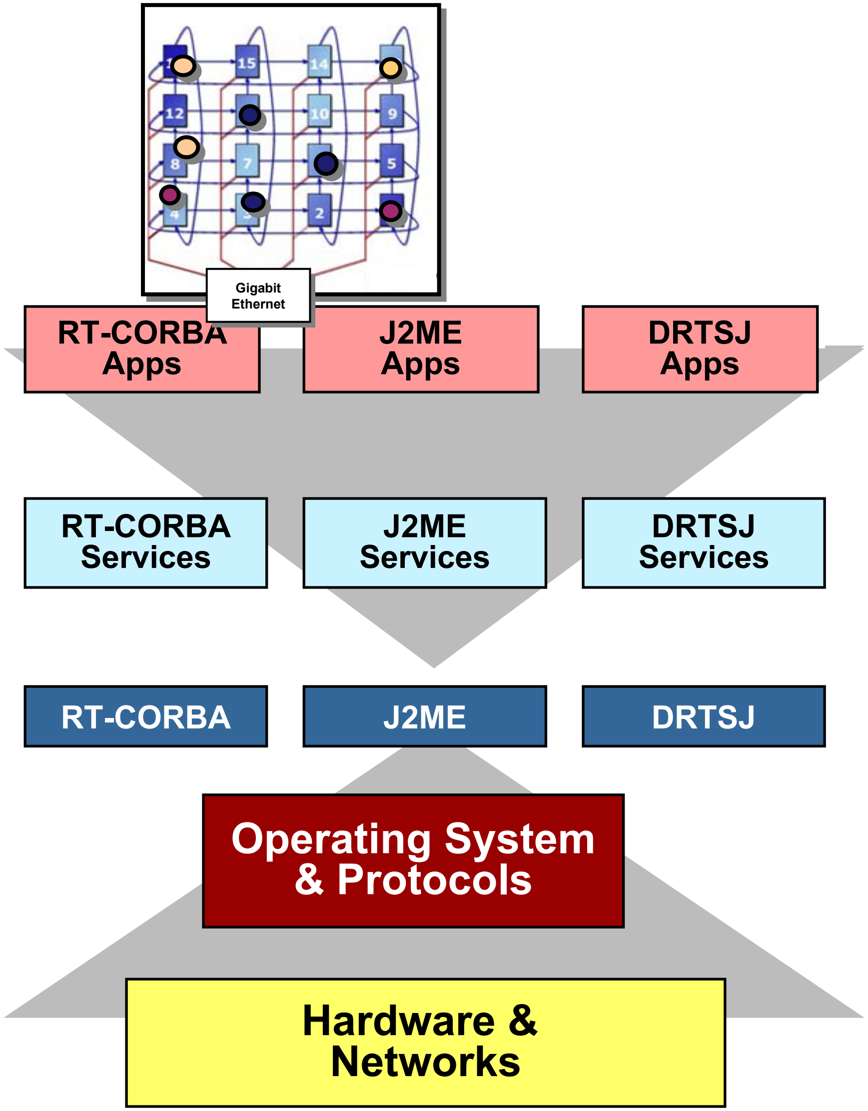
Technology Problems

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

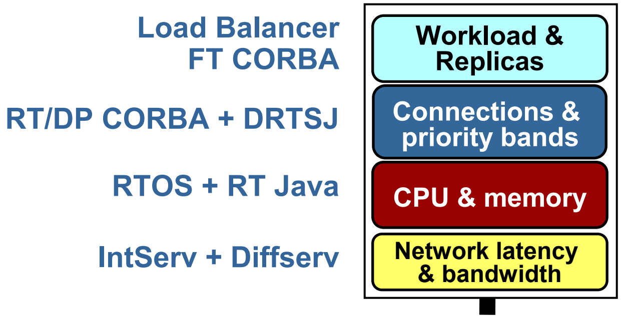
- Middleware has effectively factored out many reusable services from traditional DRE application responsibility
  - Essential for ***product-line architectures***
- Middleware is no longer the primary DRE system performance bottleneck

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

DRE Systems: The Challenges Ahead

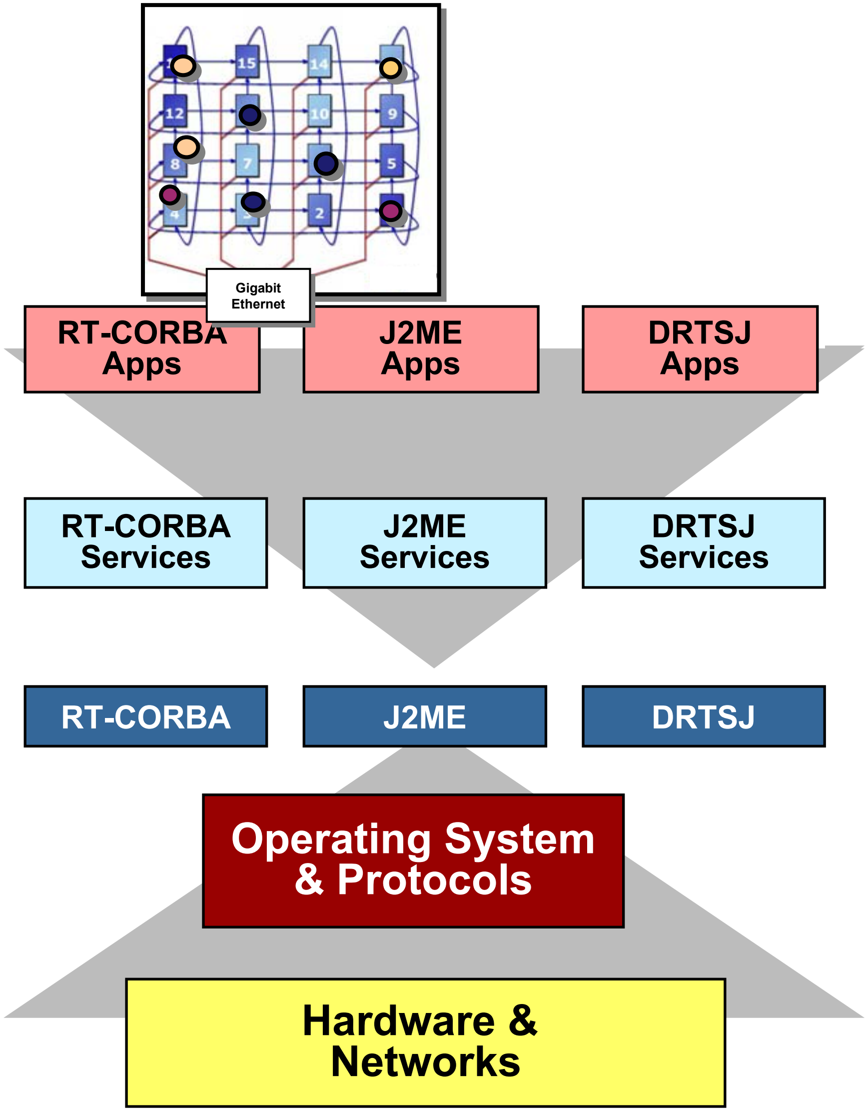


- Limit to how much application functionality can be refactored into reusable COTS middleware
- Middleware itself has become very hard to use & provision statically & dynamically



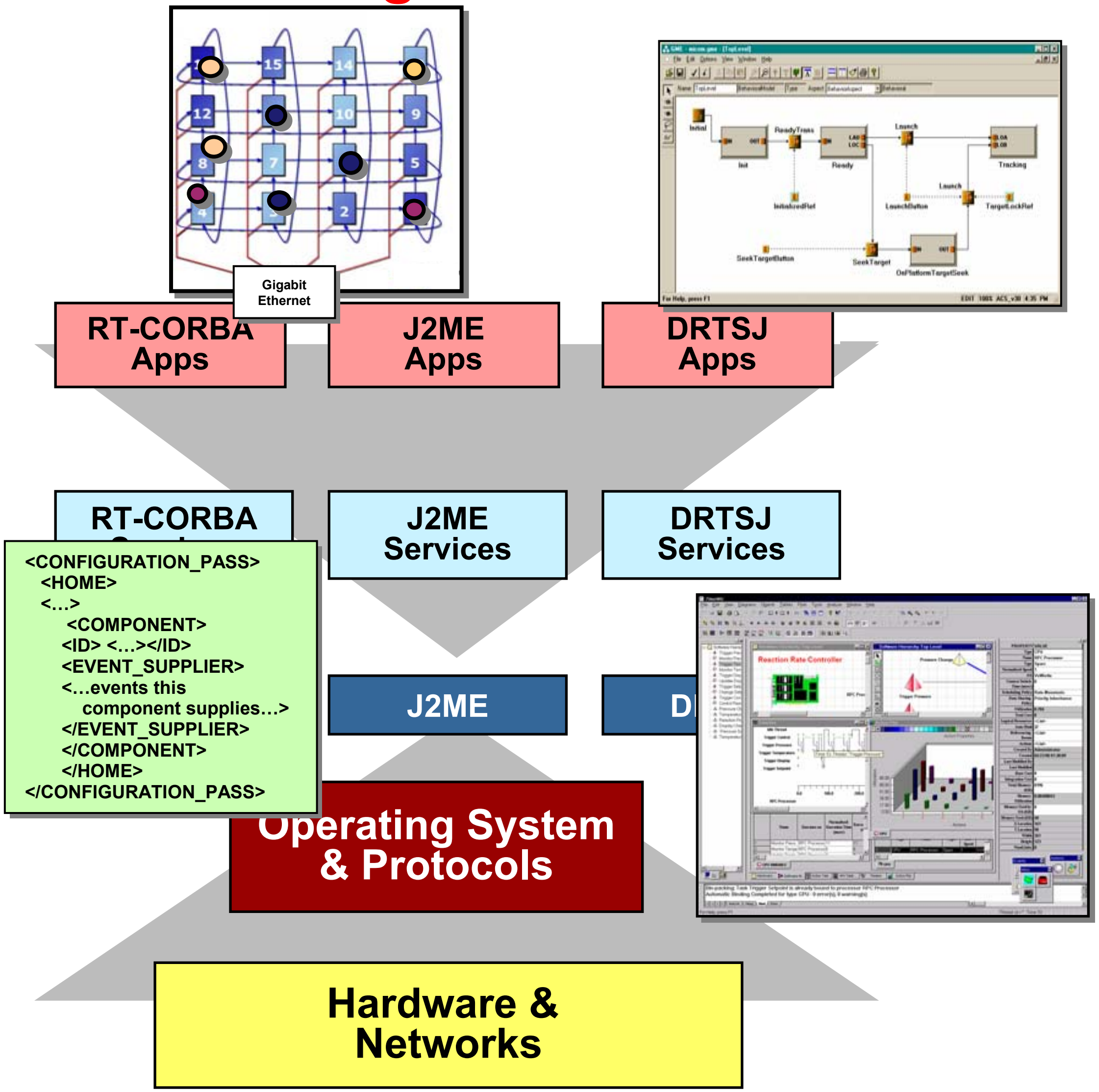
- Component-based DRE systems are also very hard to deploy & configure
- There are many middleware platform technologies to choose from

DRE Systems: The Challenges Ahead



It's enough to make you scream!

Promising Solution: *Model Driven Development (MDD)*

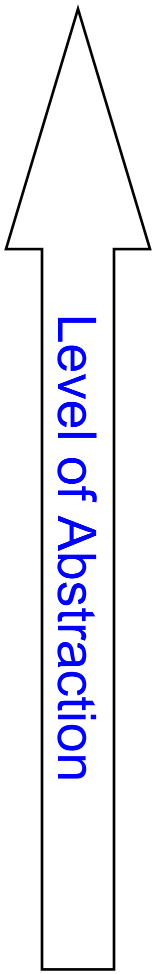


- Develop, validate, & standardize generative software technologies that:
  1. ***Model***
  2. ***Analyze***
  3. ***Synthesize &***
  4. ***Provision***

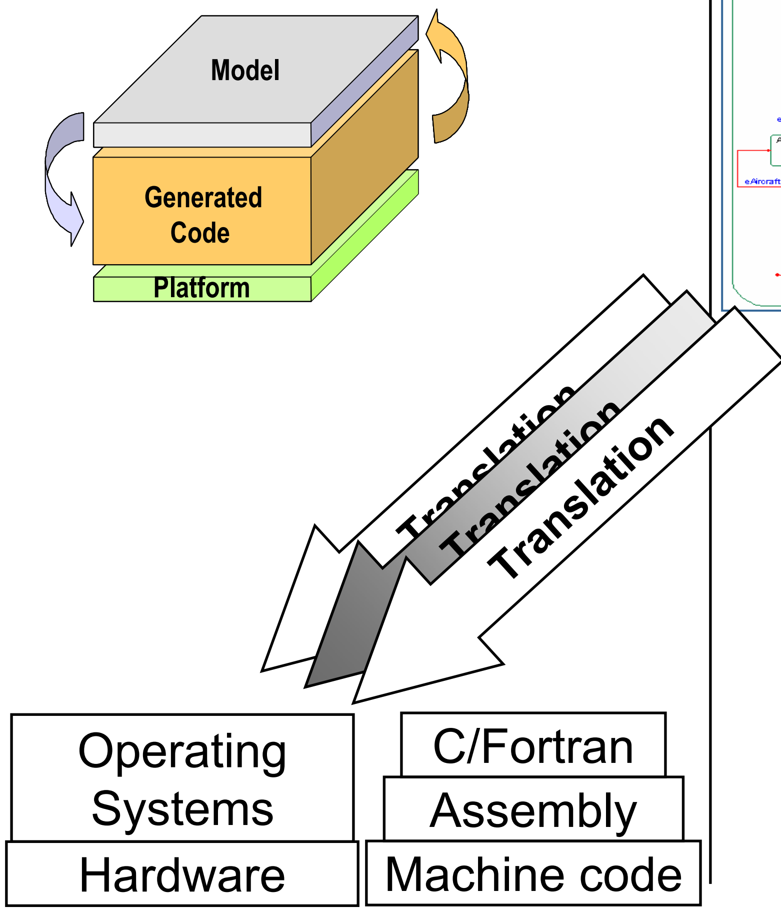
  multiple layers of middleware & application components that require simultaneous control of multiple QoS properties end-to-end
- Partial specialization is essential for inter-/intra-layer optimization & advanced product-line architectures

Goal is to ***enhance developer productivity & software quality*** by providing ***higher-level languages & tools*** for middleware/application developers & users

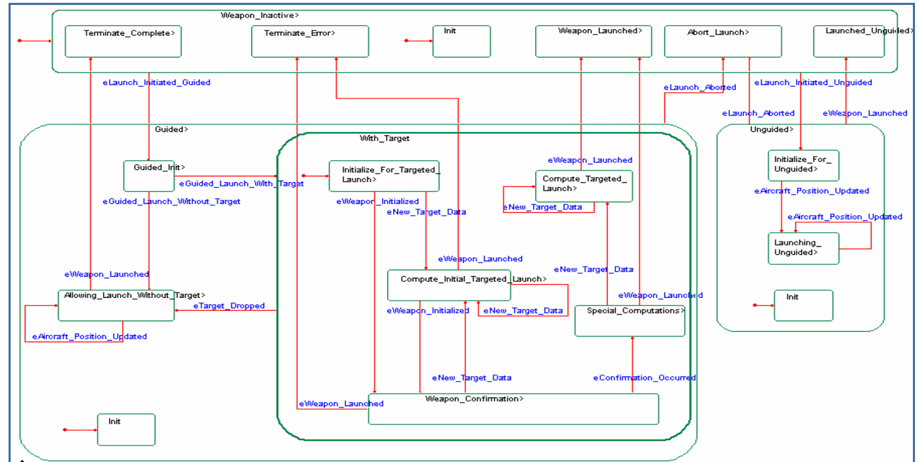
Technology Evolution (1/4)



Programming Languages & Platforms



Model-Driven Development (MDD)



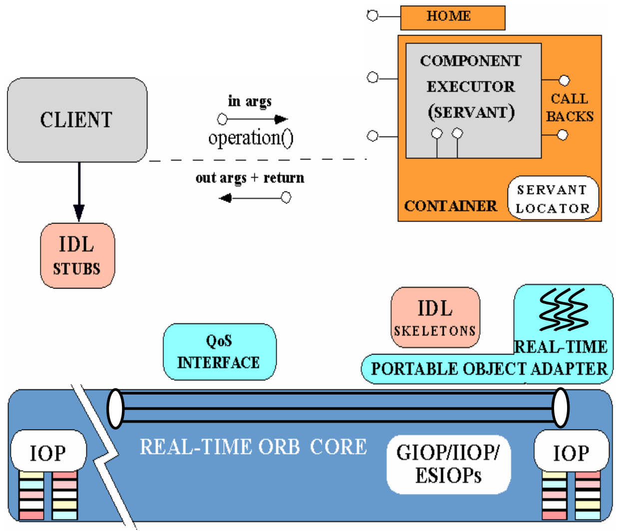
Large Semantic Gap

- State chart
- Data & process flow
- Petri Nets

Technology Evolution (2/4)

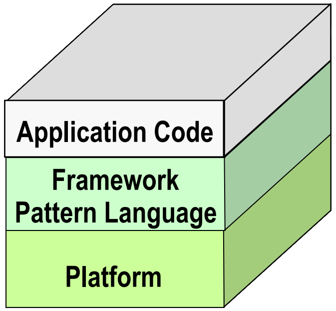
Level of Abstraction

Programming Languages & Platforms



Components	
Frameworks	
Class Libraries	C++/Java
Operating Systems	C/Fortran
	Assembly
Hardware	Machine code

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

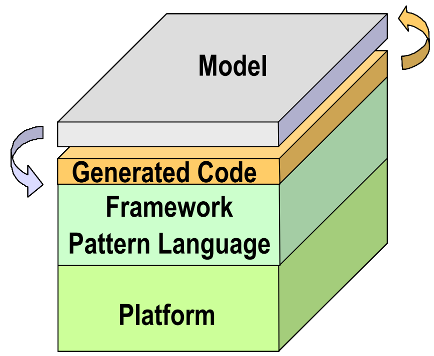


- There are two problems, however:
  - Platform complexity evolved faster than 3rd-generation languages
  - Much application/platform code still (unnecessarily) written manually
    - Particularly for D&C aspects

Technology Evolution (3/4)

Level of Abstraction

Programming Languages & Platforms

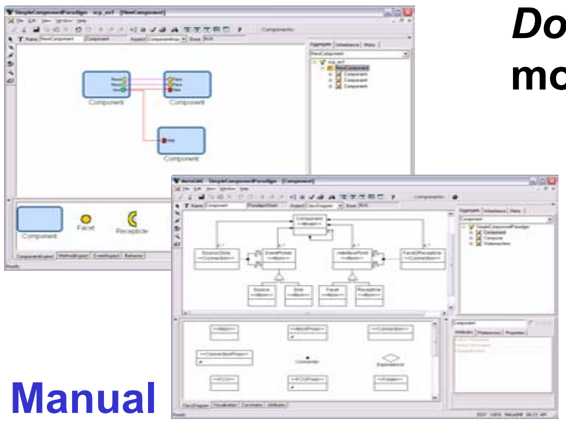


Saturation!!!!

Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

Model-Driven Development (MDD)



Manual translation

***Domain-specific* modeling languages**

- ESML
- PICML
- Mathematic
- Excel
- *Metamodels*

Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

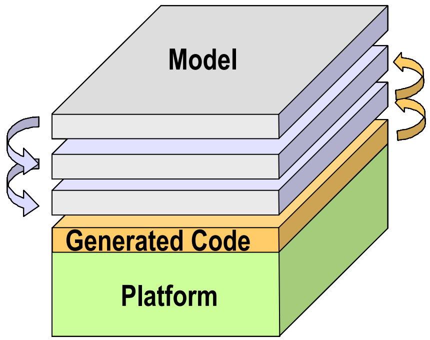
Semi-automated

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

Technology Evolution (4/4)

Level of Abstraction

Programming Languages & Platforms



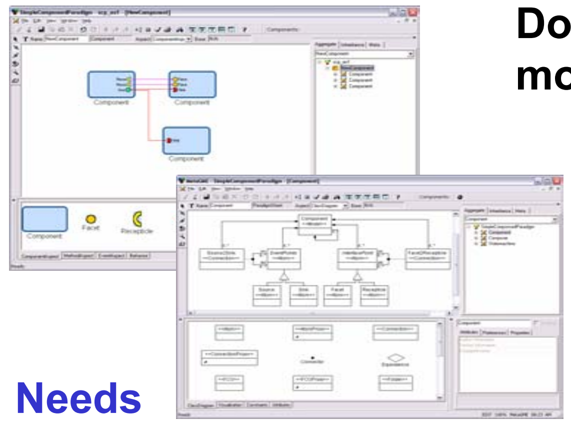
Needs Automation

Saturation!!!!

Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

Model-Driven Development (MDD)



Domain-specific modeling languages

- ESML
- PICML
- Mathematic
- Excel
- *Metamodels*

Needs Automation

Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

Needs Automation

Research is needed to automate DSMLs & model translators

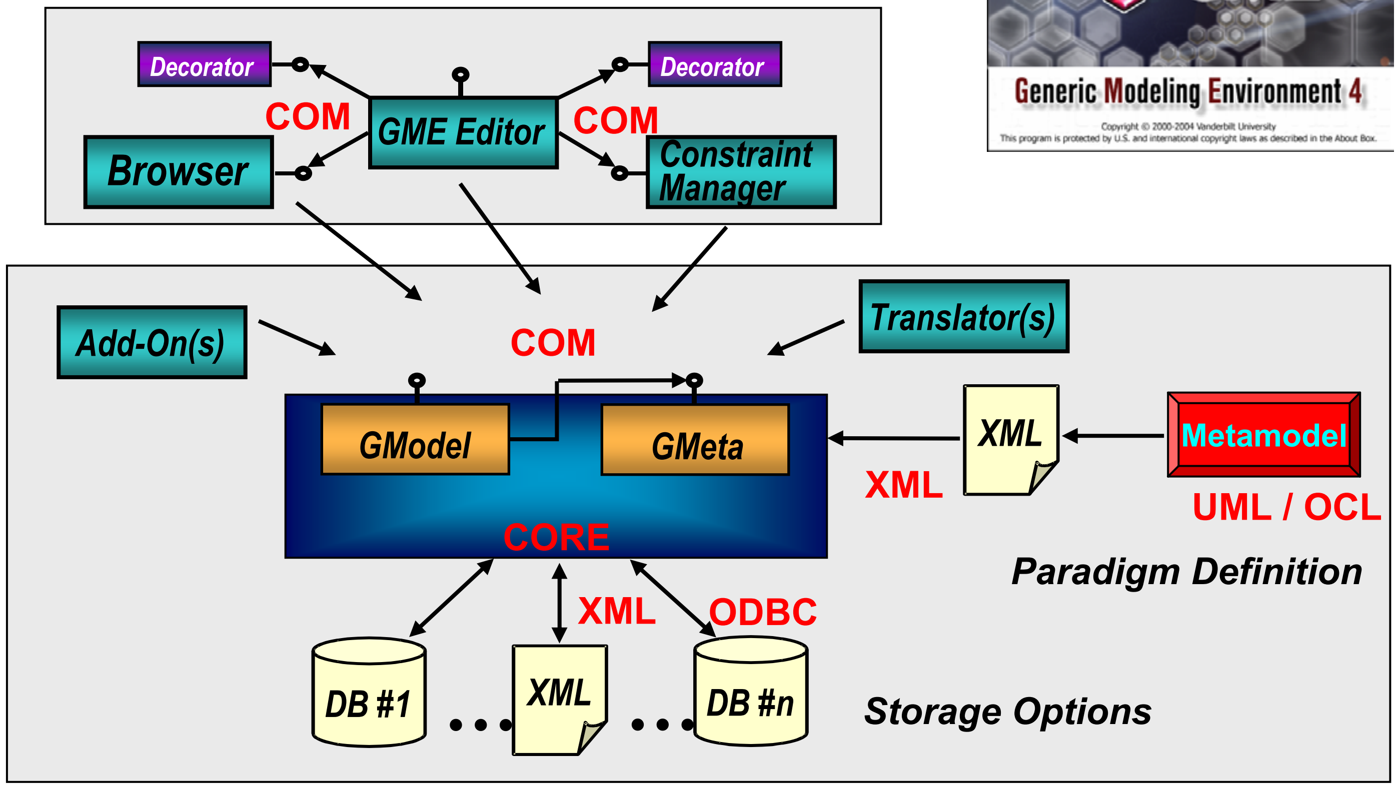
Generic Modeling Environment (GME)

"Write Code That Writes Code That Writes Code!"

GME Architecture

Application Developers (Modelers)

MDD Tool Developers (Metamodelers)

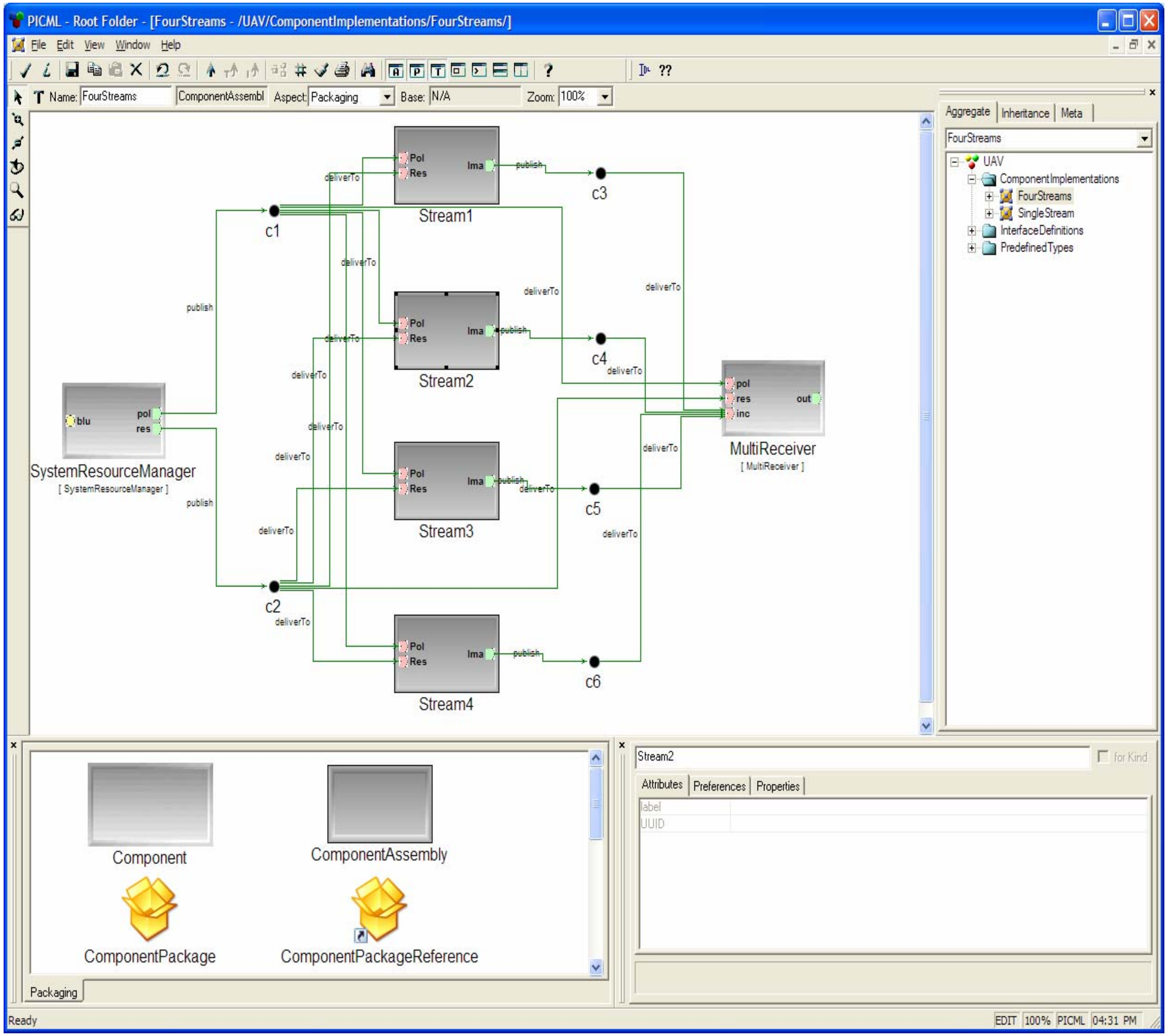


Supports "correct-by-construction" of software systems

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

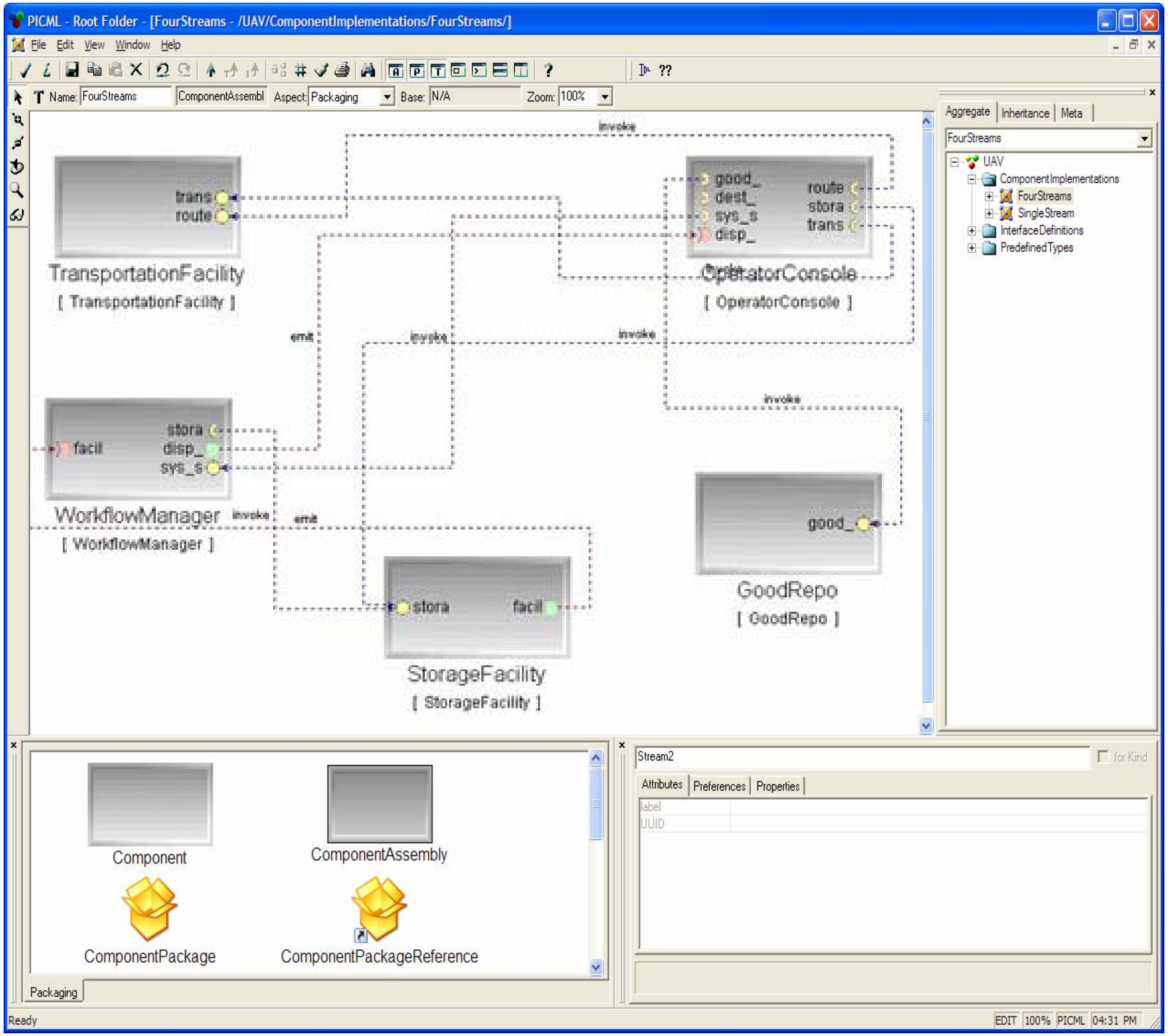
MDD Application Development with GME

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



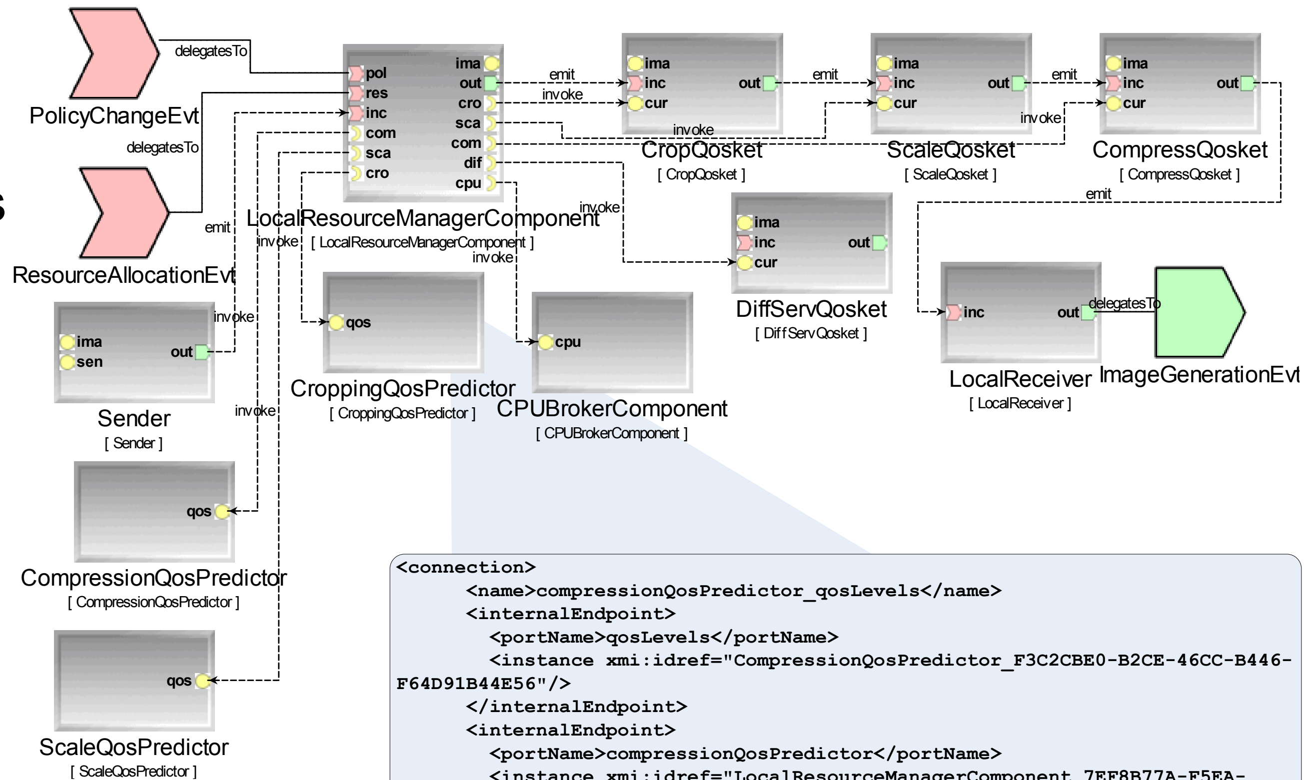
MDD Application Development with GME

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



MDD Application Development with GME

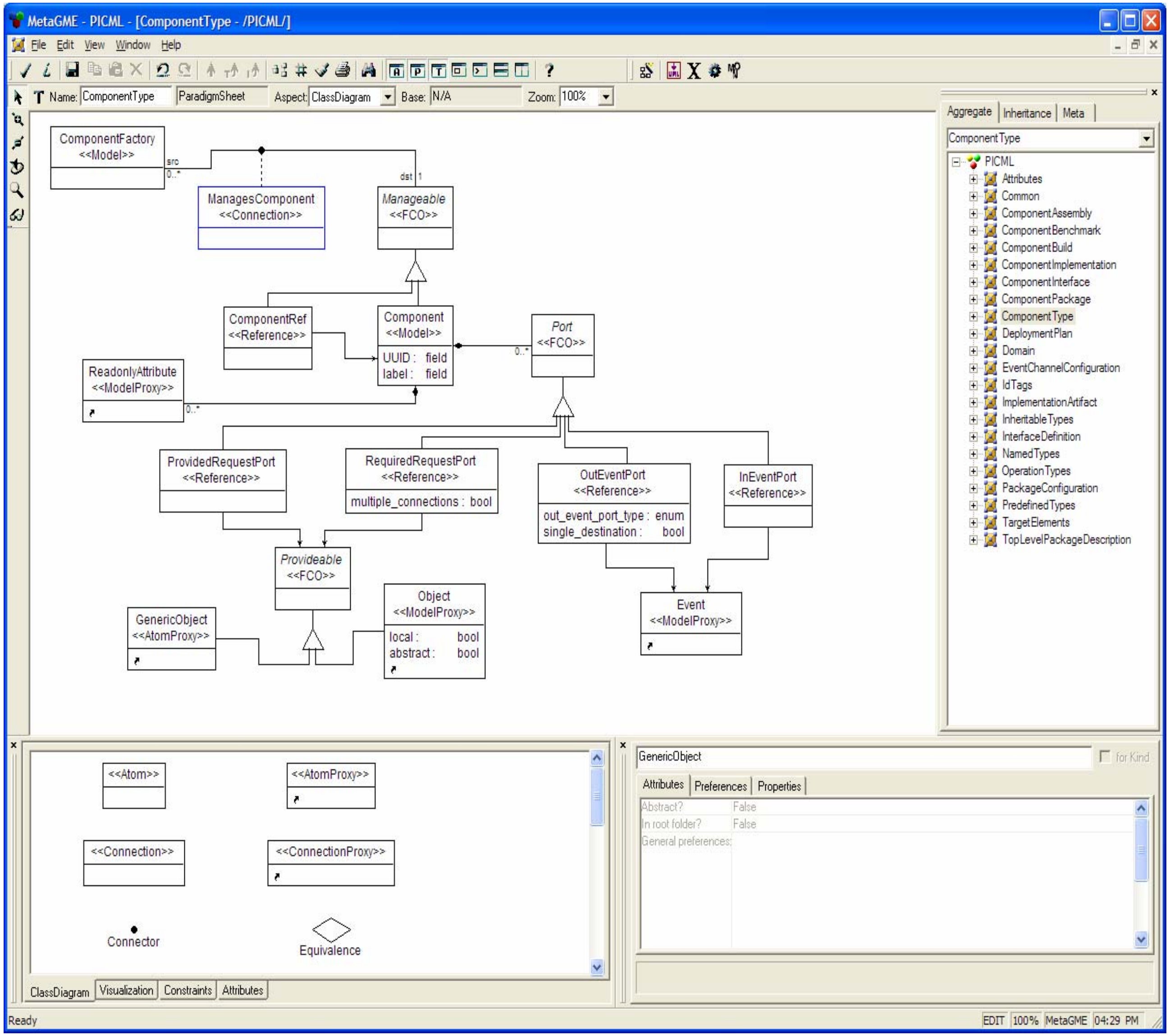
- **Application developers** use modeling environments created w/MetaGME to build *applications*
  - Capture elements & dependencies visually
  - Model interpreter produces something useful from the models
    - e.g., 3rd generation code, simulations, deployment descriptions & configurations




```
<connection>
      <name>compressionQosPredictor_qosLevels</name>
      <internalEndpoint>
        <portName>qosLevels</portName>
        <instance xmi:idref="CompressionQosPredictor_F3C2CBE0-B2CE-46CC-B446-
F64D91B44E56"/>
      </internalEndpoint>
      <internalEndpoint>
        <portName>compressionQosPredictor</portName>
        <instance xmi:idref="LocalResourceManagerComponent_7EF8B77A-F5EA-
4D1A-942E-13AE7CFED30A"/>
      </internalEndpoint>
    </connection>
    <connection>
      <name>scalingQosPredictor_qosLevels</name>
      <internalEndpoint>
        <portName>qosLevels</portName>
        <instance xmi:idref="ScaleQosPredictor_F3024A4F-F6E8-4B9A-BD56-
A2E802C33E32"/>
      </internalEndpoint>
      <internalEndpoint>
        <portName>scalingQosPredictor</portName>
        <instance xmi:idref="LocalResourceManagerComponent_7EF8B77A-F5EA-
4D1A-942E-13AE7CFED30A"/>
      </internalEndpoint>
    </connection>
```

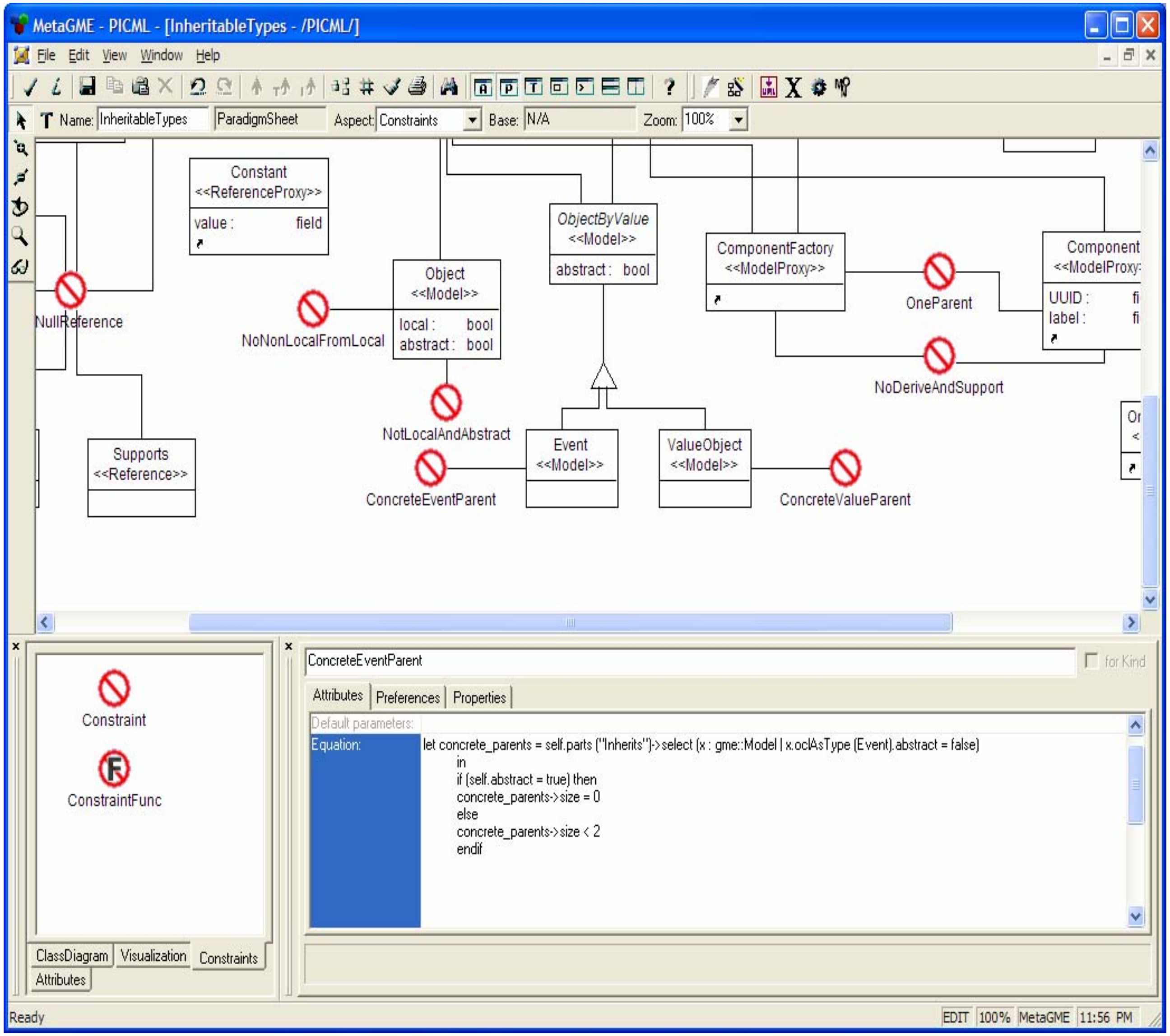
MDD Tool Development in GME

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



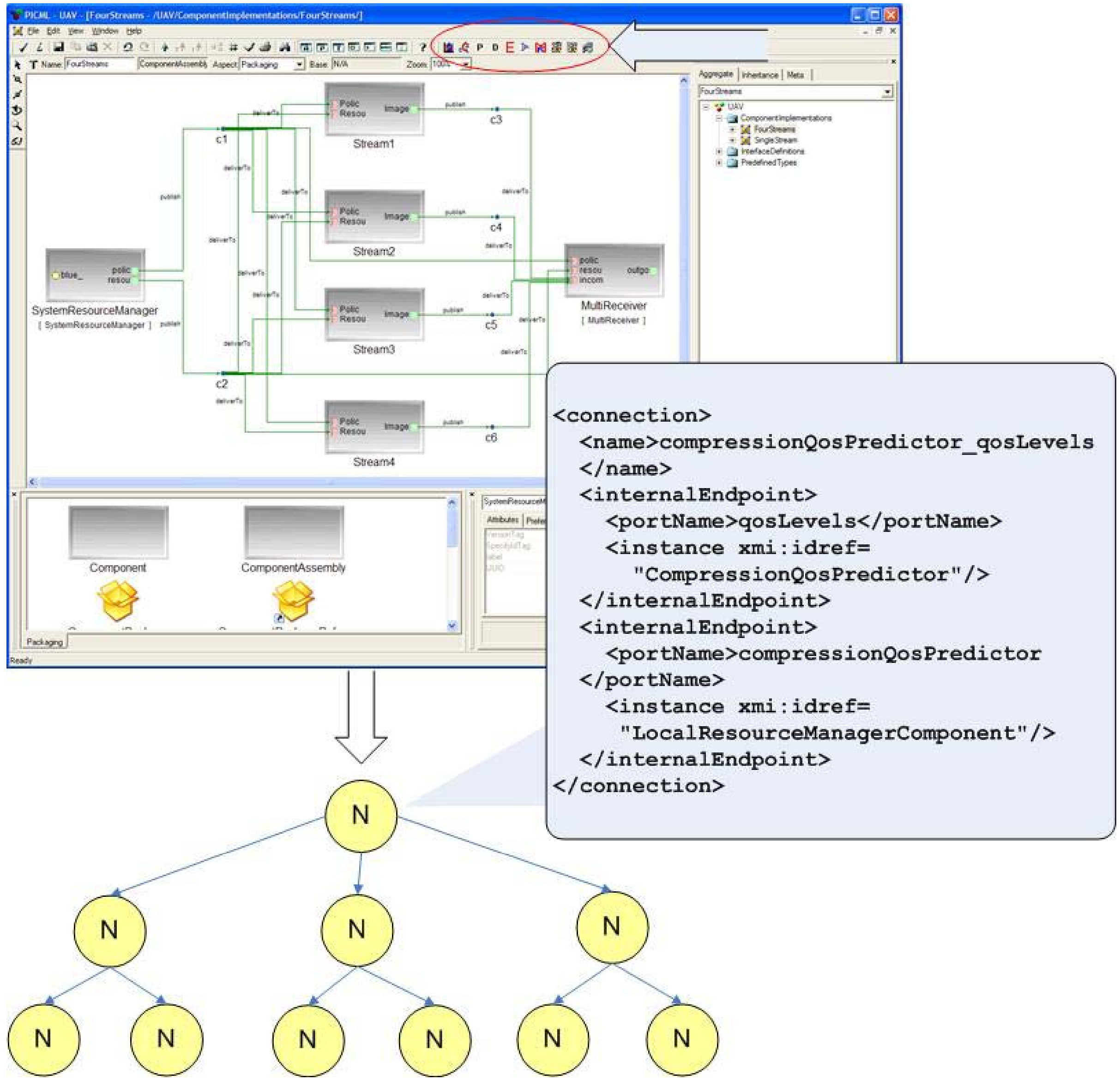
MDD Tool Development in GME

- **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)*

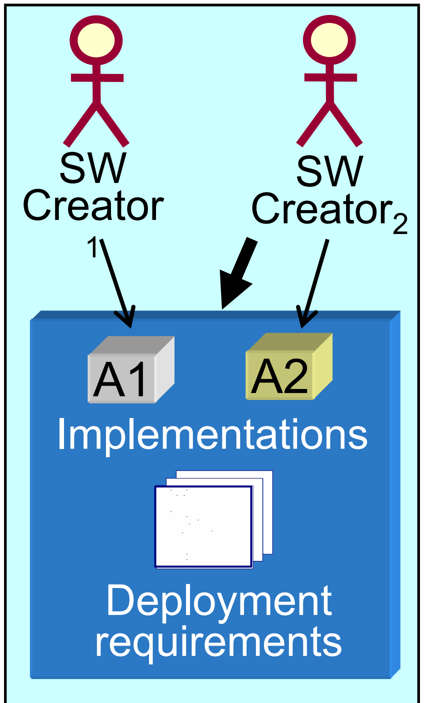


MDD Tool Development in GME

- **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*

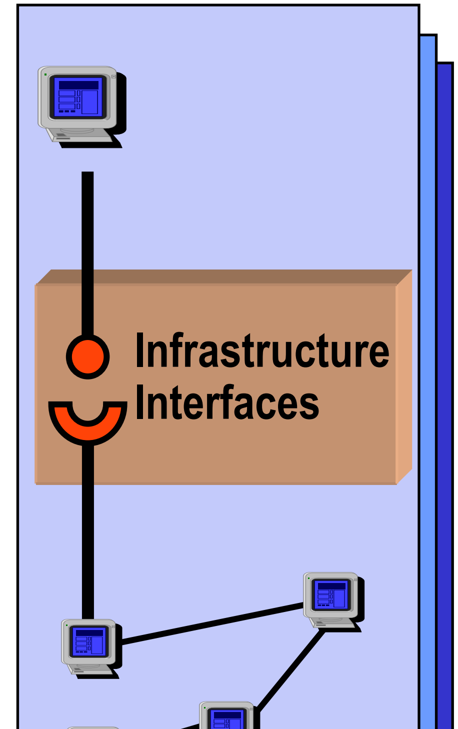


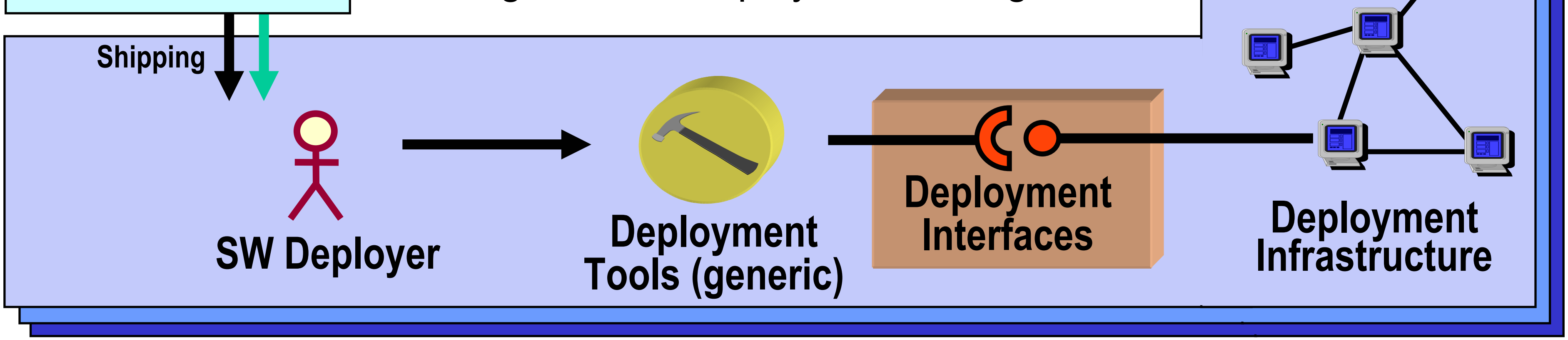
OMG Component Deployment & Configuration



Goals of D&C Phase

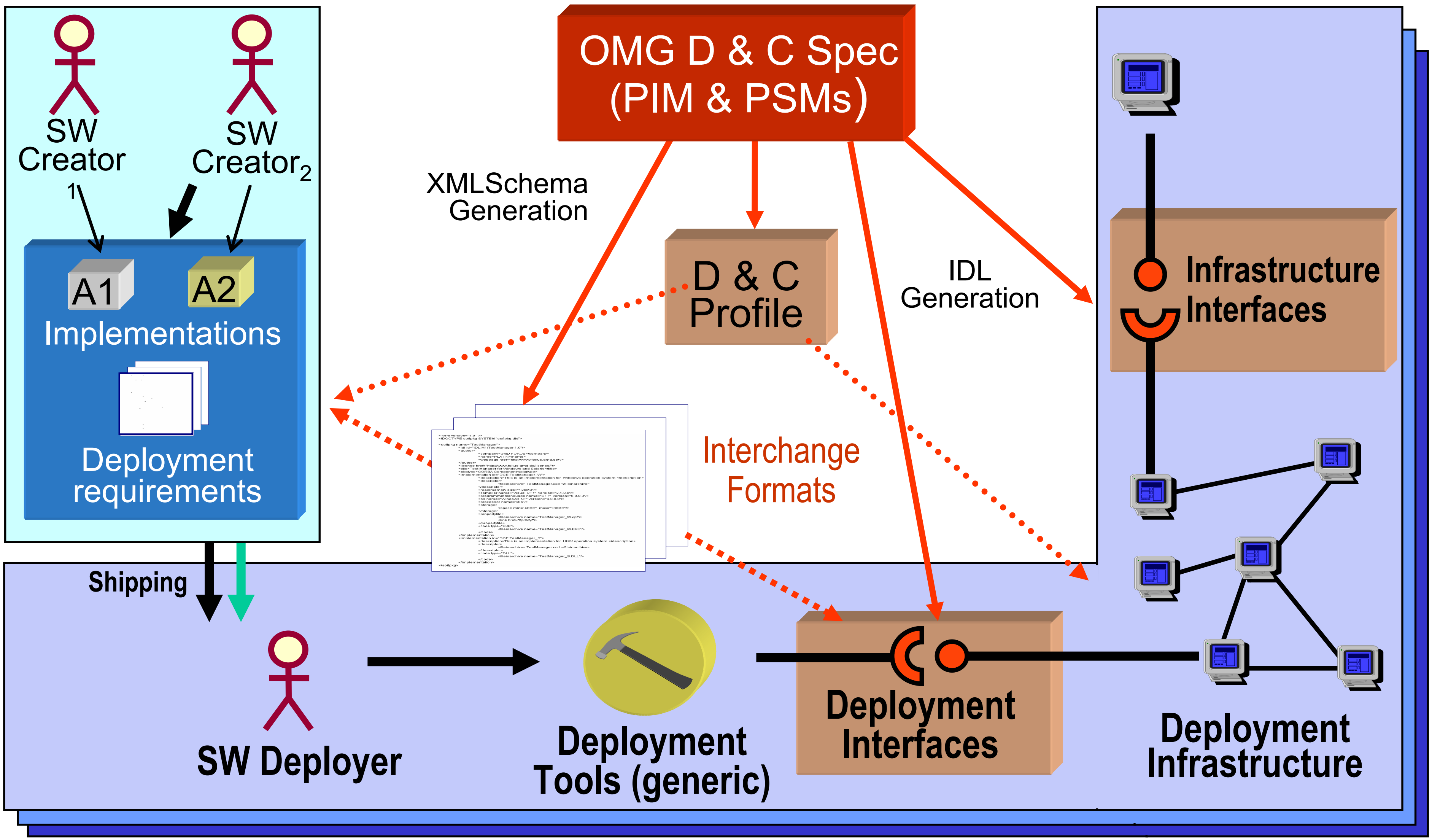
- Promote component reuse
- Build complex applications by assembling existing components
- Automate common services configuration
- Declaratively inject QoS policies into applications
- Dynamically deploy components to target heterogeneous domains
- Optimize systems based on component configuration &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





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

OMG Component Deployment & Configuration



OMG 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

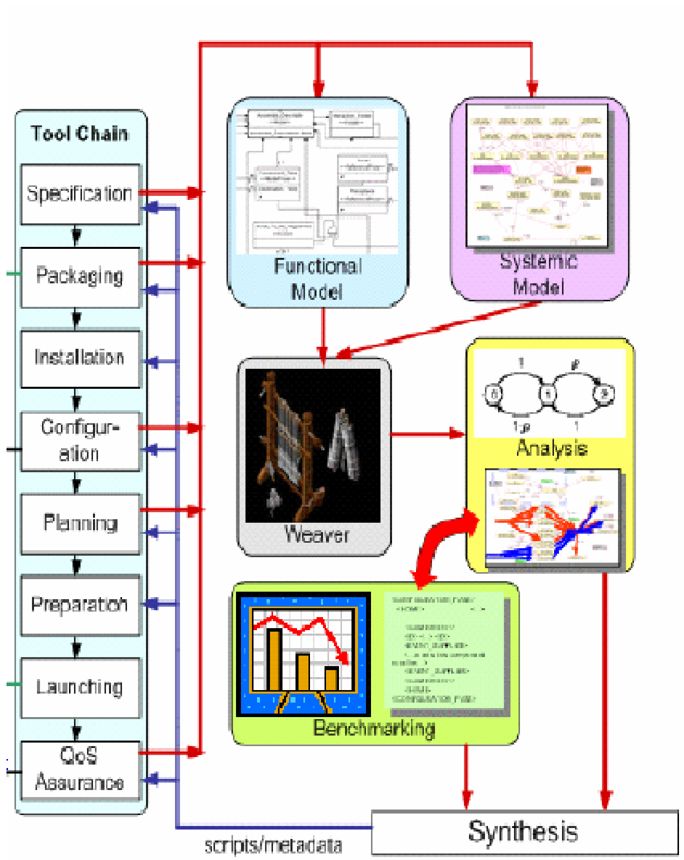
- Moving binaries to identified entities of target environment

Launching

- Triggering installed binaries & bringing app to ready state

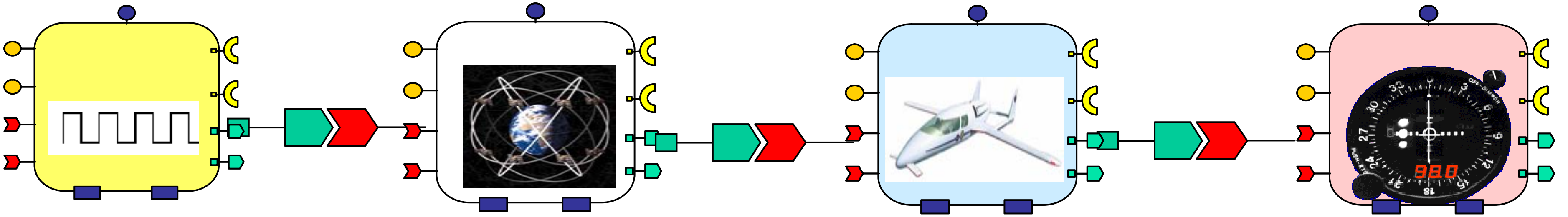
QoS Assurance & Adaptation

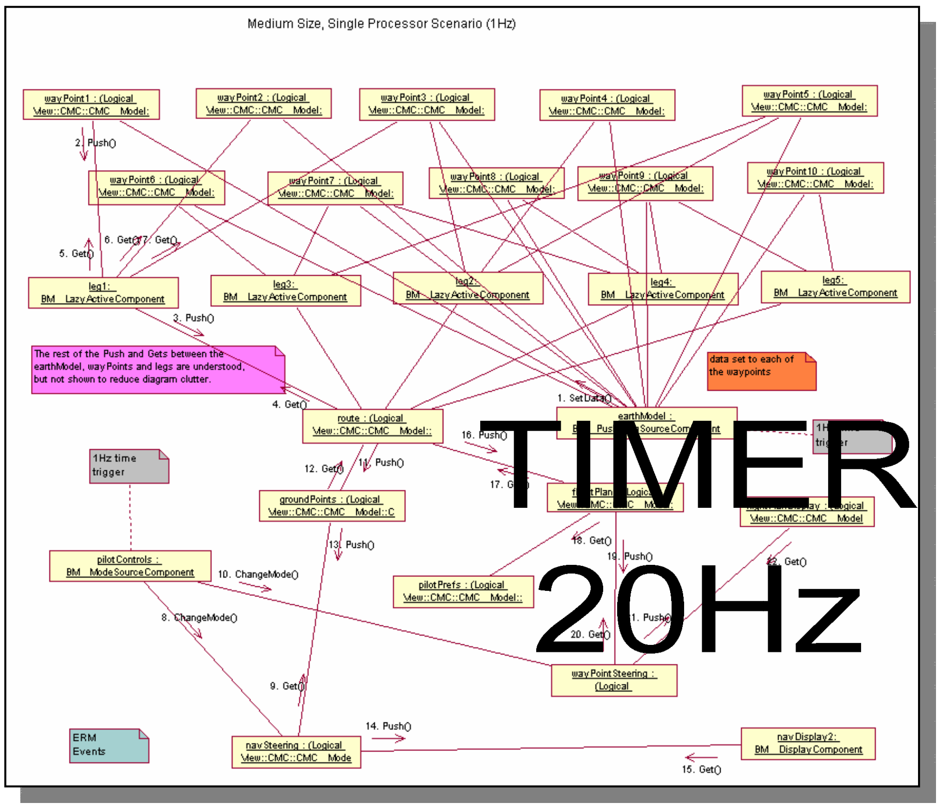
- Runtime (re)configuration & resource management to maintain end-to-end QoS



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

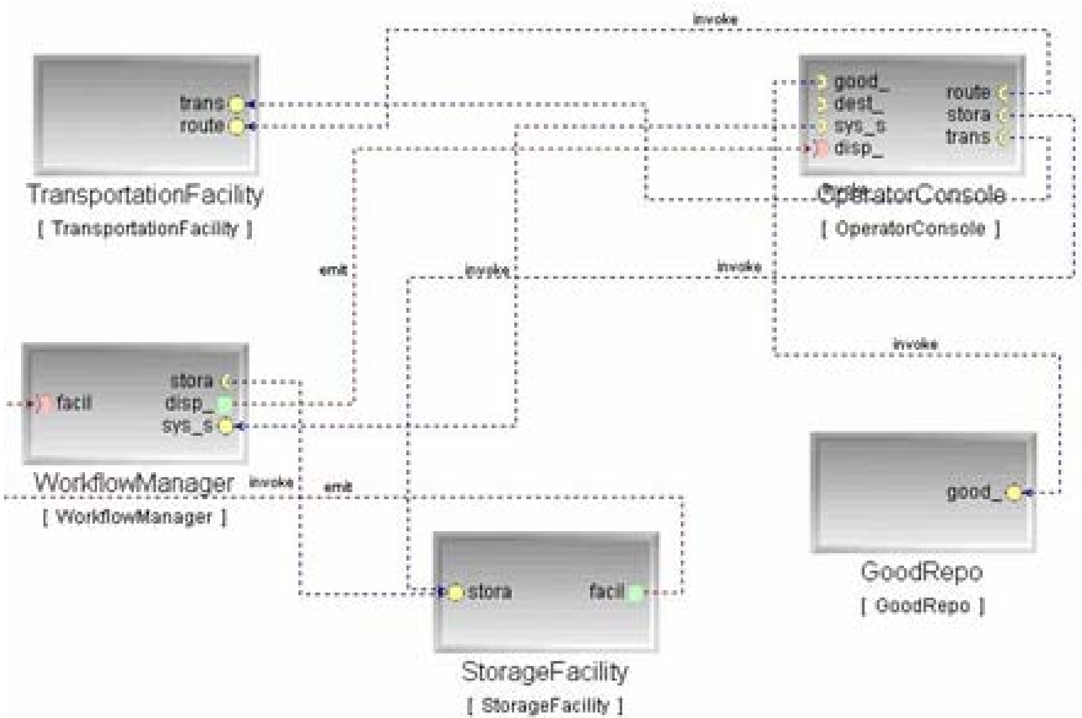
Challenge 1: The Packaging Aspect





- 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



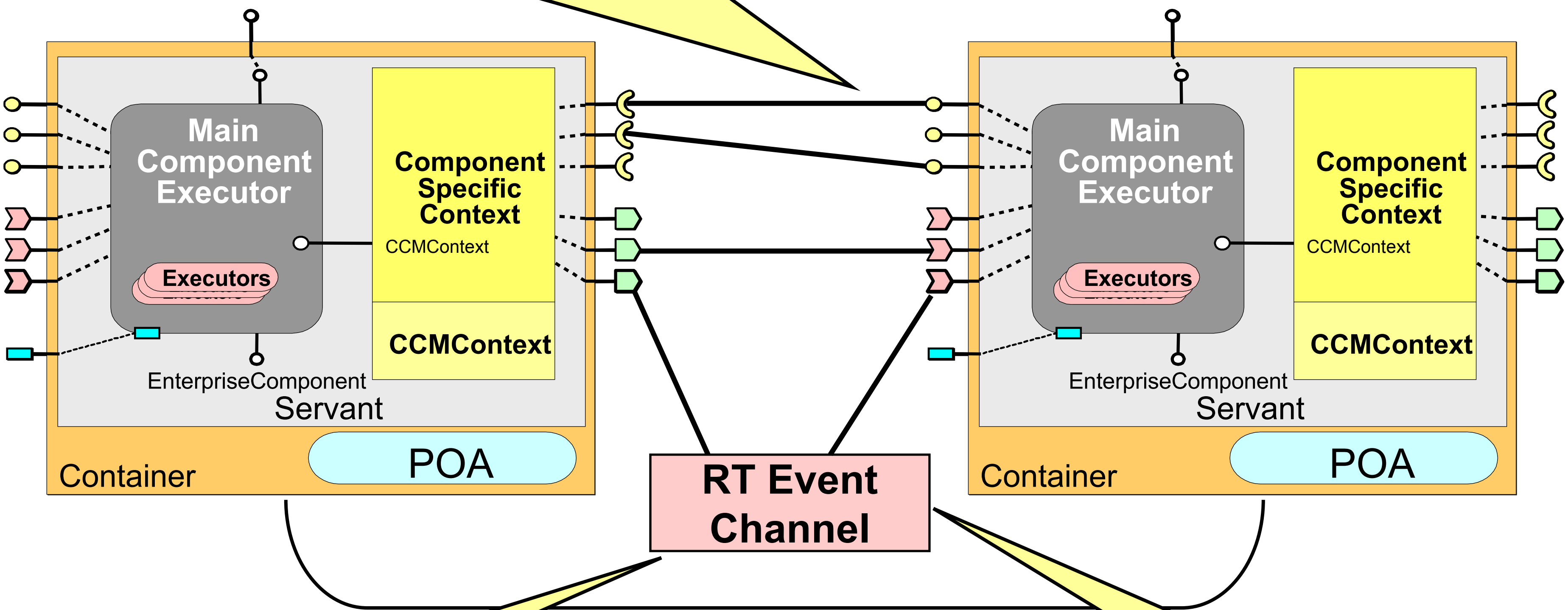
- Application components are bundled together into *assemblies*
- Assemblies convey component interconnections & implementation alternatives

- 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

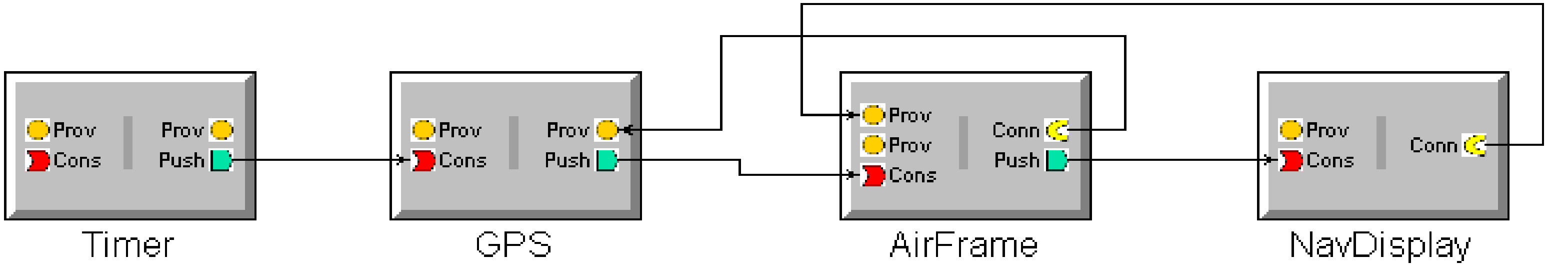
```
<!– Associate components with impls -->
<assemblyImpl>
   <instance xmi:id="RateGen">
     <name>RateGen Subcomponent</name>
     <package href="RateGen.cpd"/>
   </instance>
   <instance xmi:id="GPS">
     <name>GPS Subcomponent</name>
     <package href="GPS.cpd"/>
   </instance>
   <instance xmi:id="NavDisplay">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

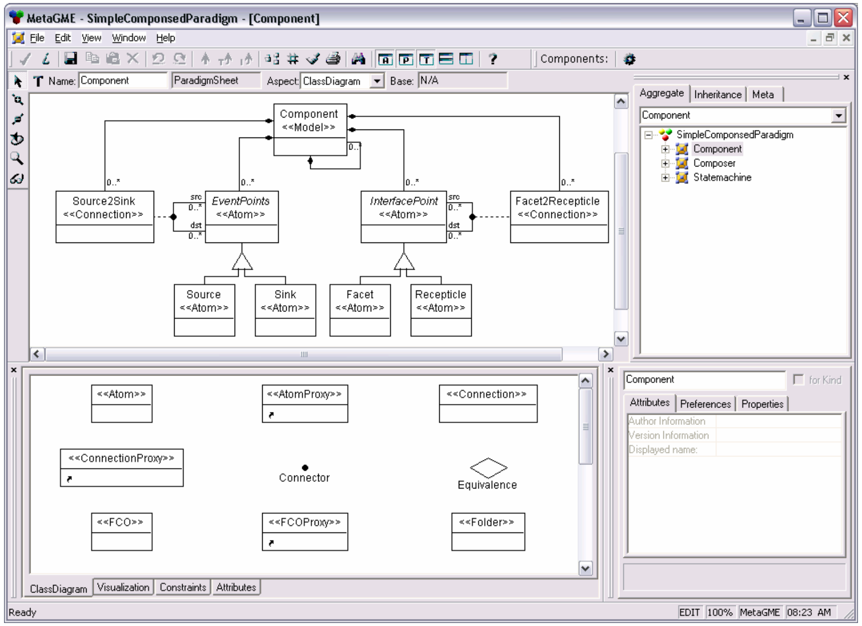
Modifications to the assemblies requires modifying XML file

MDD Solution for Packaging Aspect



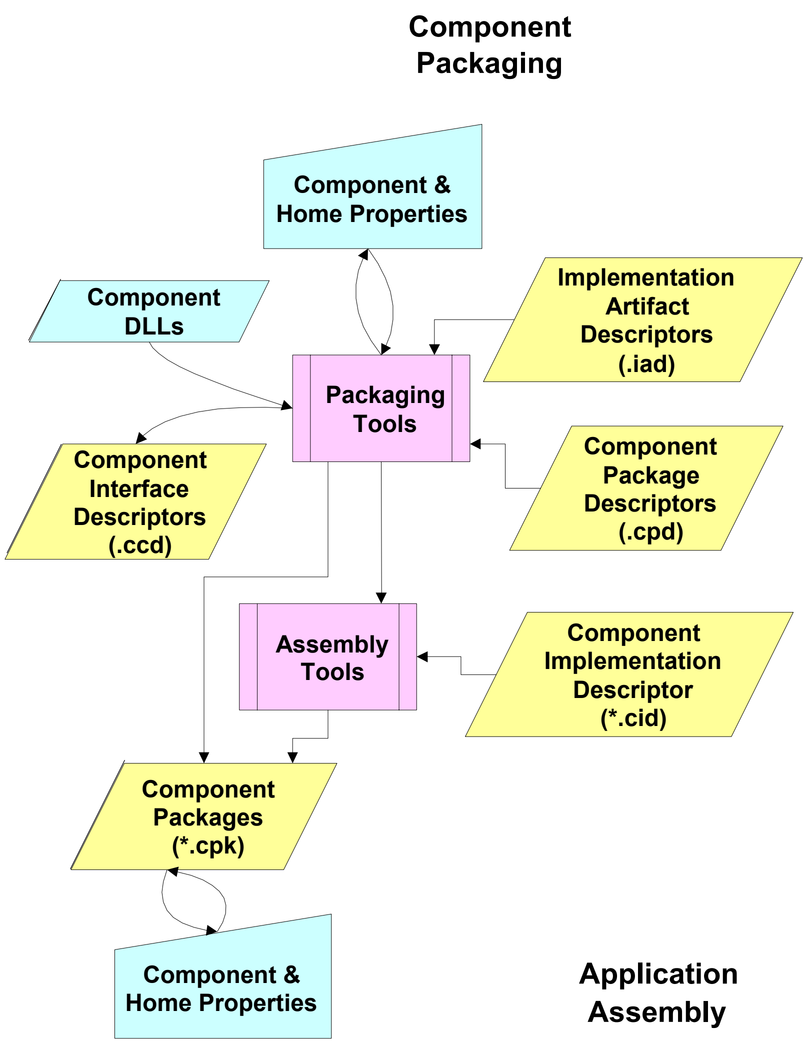
Approach:

- 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)



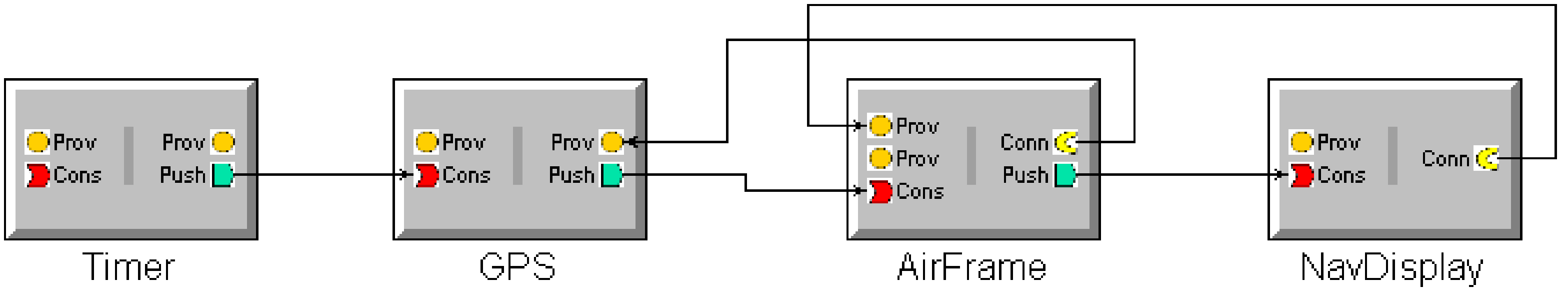
Example Metadata Generated by PICML

- **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)

Example Output from PICML Model



A Component Implementation Descriptor (*.cid) file

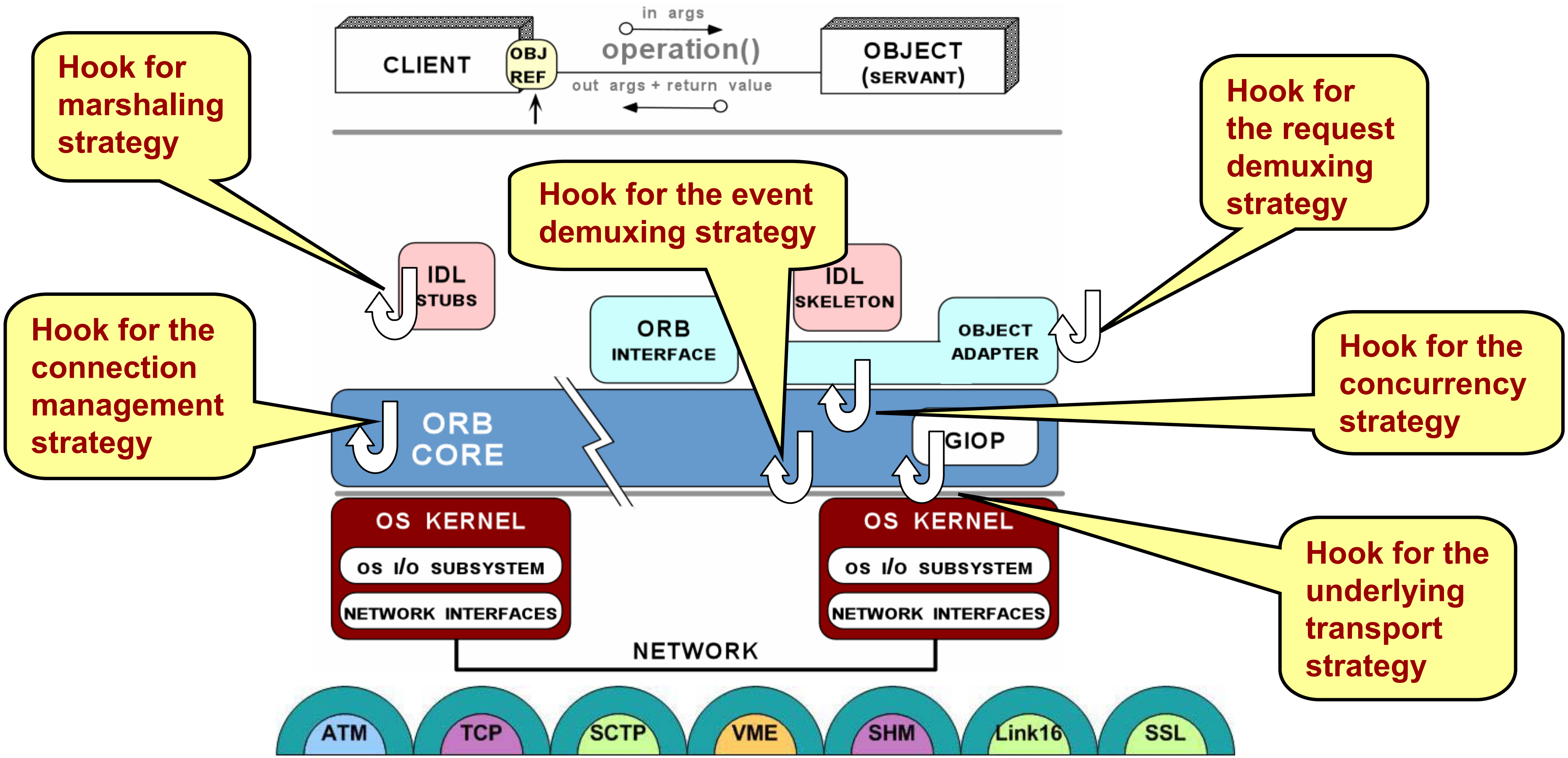
- Describes a specific implementation of a component interface
- Describes component interconnections

```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>
   </deployRequirement>
   [... Requires Windows OS ...]
 </monolithicImpl>
```

```xml
<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

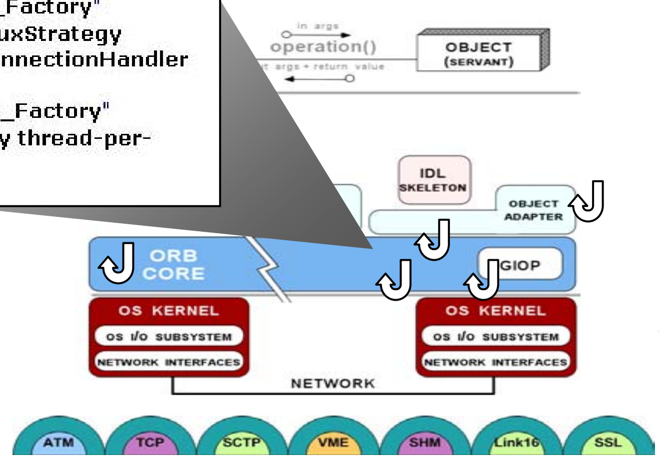


Configuration Aspect Problems

Middleware developers

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

```
ests/Latency/Thread_Per_Connection/svc.conf
ACE_Svc_Conf>
 <!--      -->
 <!--   $Id: svc.conf.xml,v 1.1 2002/08/23
 22:23:04 nanbor Exp $   -->
 <!--      -->
 <static id="Advanced_Resource_Factory"
   params="-ORBReactorType select_mt -
   ORBReactorMaskSignals 0 -
   ORBFlushingStrategy blocking" />
 <static id="Client_Strategy_Factory"
   params="-ORBTransportMuxStrategy
   EXCLUSIVE -ORBClientConnectionHandler
   RW" />
 <static id="Server_Strategy_Factory"
   params="-ORBConcurrency thread-per-
   connection" />
/ACE_Svc_Conf>
```



Application developers

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

XML Configuration Files

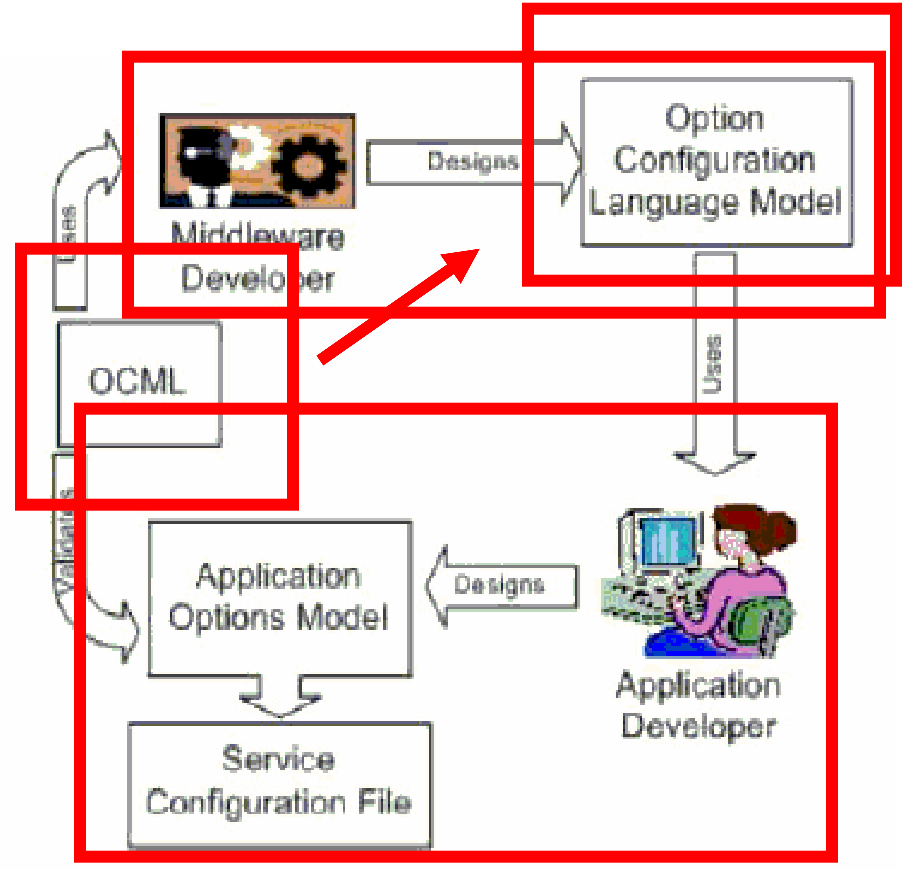
XML Property Files

CIAO/CCM provides ~500 configuration options

MDD Solutions for Configuration Aspect

Approach:

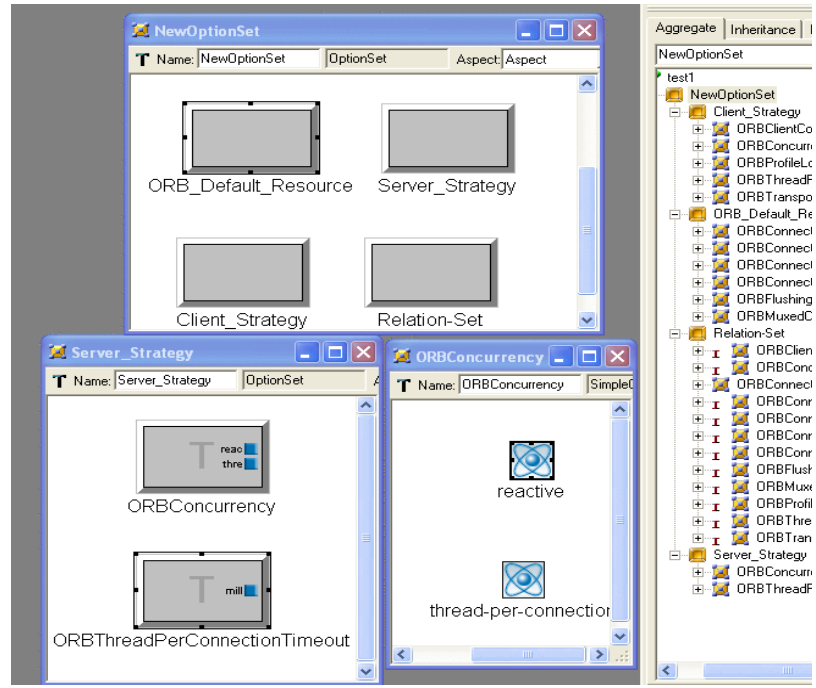
- Develop an ***Options Configuration Modeling Language (OCML)*** w/GME to ensure semantic consistency of option configurations
- OCML is used by
  - **Middleware developers** to design the *configuration model*
  - **Application developers** to configure the middleware for a specific application
- OCML *metamodel* is platform-independent
- OCML *models* are platform-specific

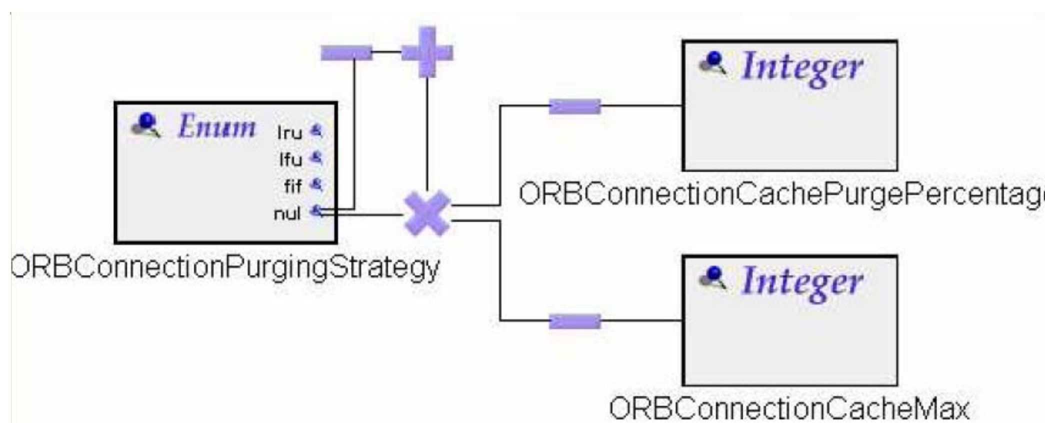


www.cs.wustl.edu/~schmidt/PDF/RTAS-process.pdf

Applying OCML to CIAO+TAO

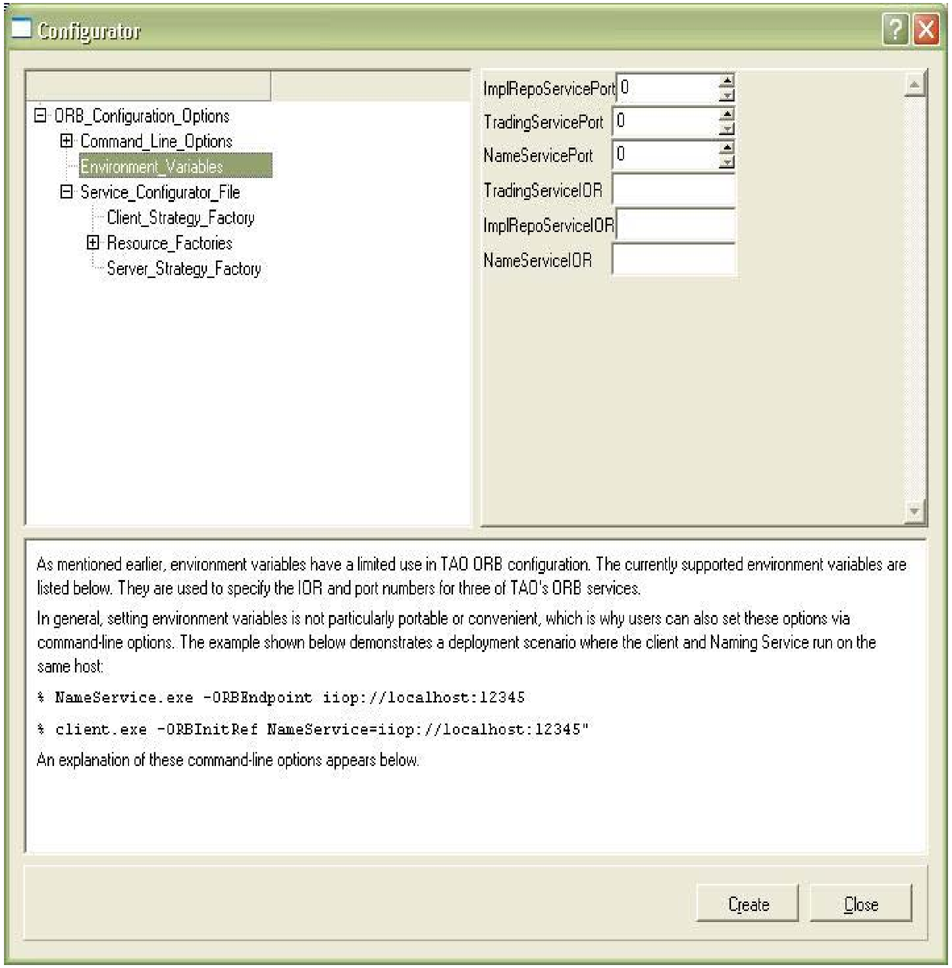
- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model





Applying OCML to CIAO+TAO

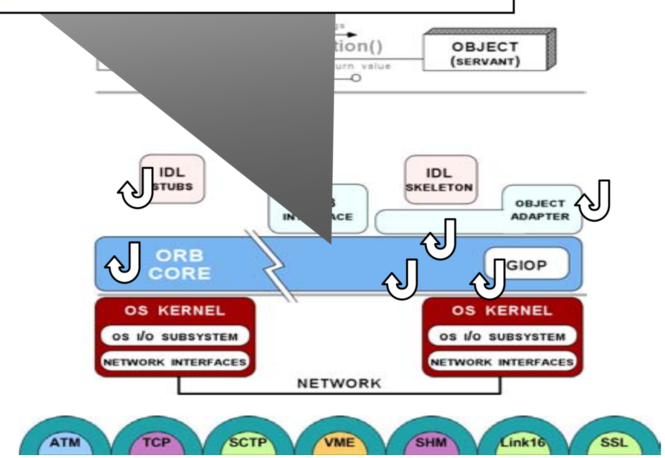
- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model
- Application developers provide a model of desired options & their values, e.g.,
  - Network resources
  - Concurrency & connection management strategies



Applying OCML to CIAO+TAO

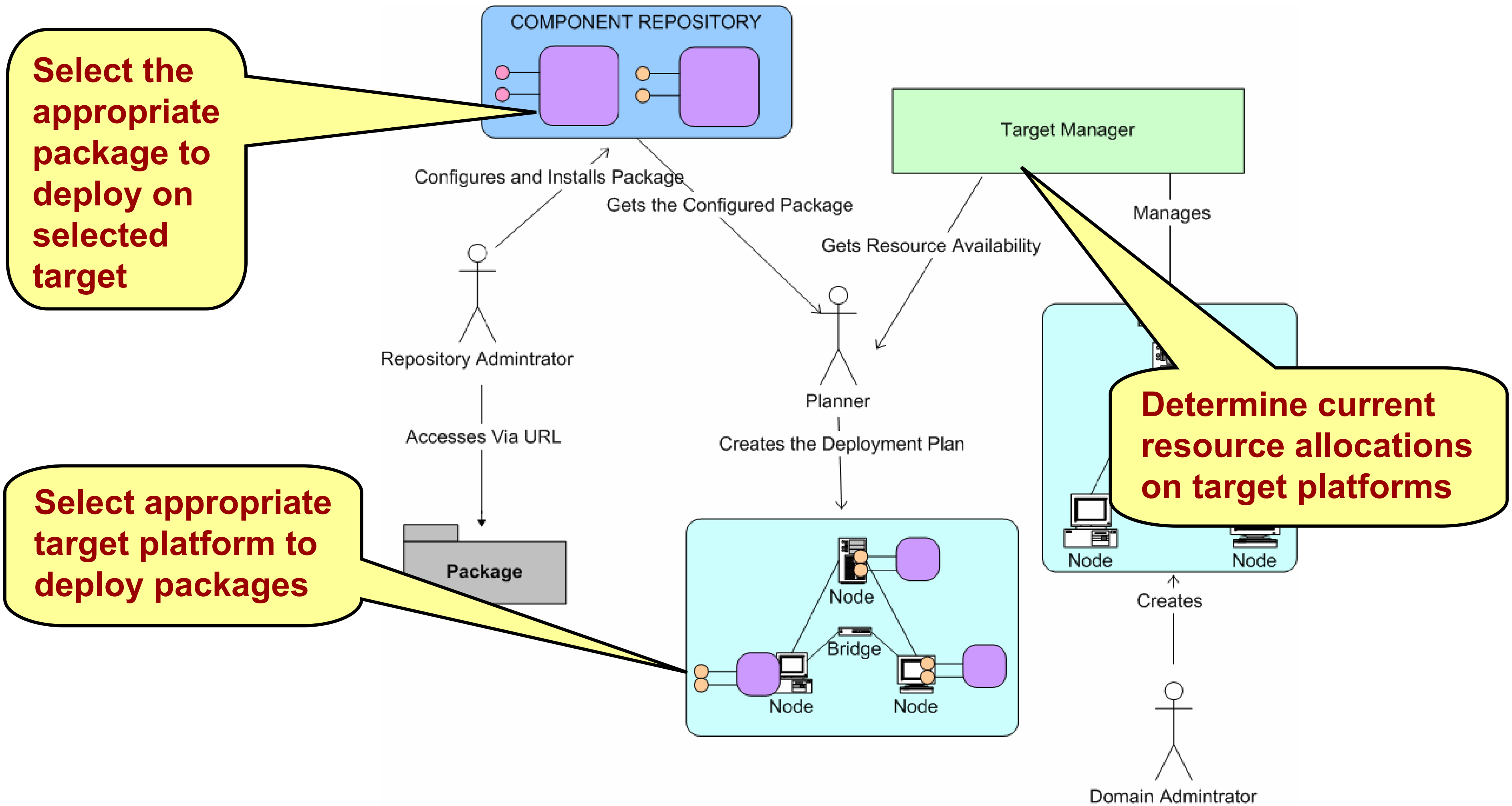
- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model
- Application developers provide a model of desired options & their values, e.g.,
  - Network resources
  - Concurrency & connection management strategies
- OCML constraint checker flags incompatible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

```
ests/Latency/Thread_Per_Connection/svc.conf
ACE_Svc_Conf>
 <!--        -->
 <!--    $Id: svc.conf.xml,v 1.1 2002/08/23
 22:23:04 nanbor Exp $    -->
 <!--        -->
 <static id="Advanced_Resource_Factory"
   params="-ORBReactorType select_mt -
   ORBReactorMaskSignals 0 -
   ORBFlushingStrategy blocking" />
 <static id="Client_Strategy_Factory"
   params="-ORBTransportMuxStrategy
   EXCLUSIVE -ORBClientConnectionHandler
   RW" />
 <static id="Server_Strategy_Factory"
   params="-ORBConcurrency thread-per-
   connection" />
/ACE_Svc_Conf>
```



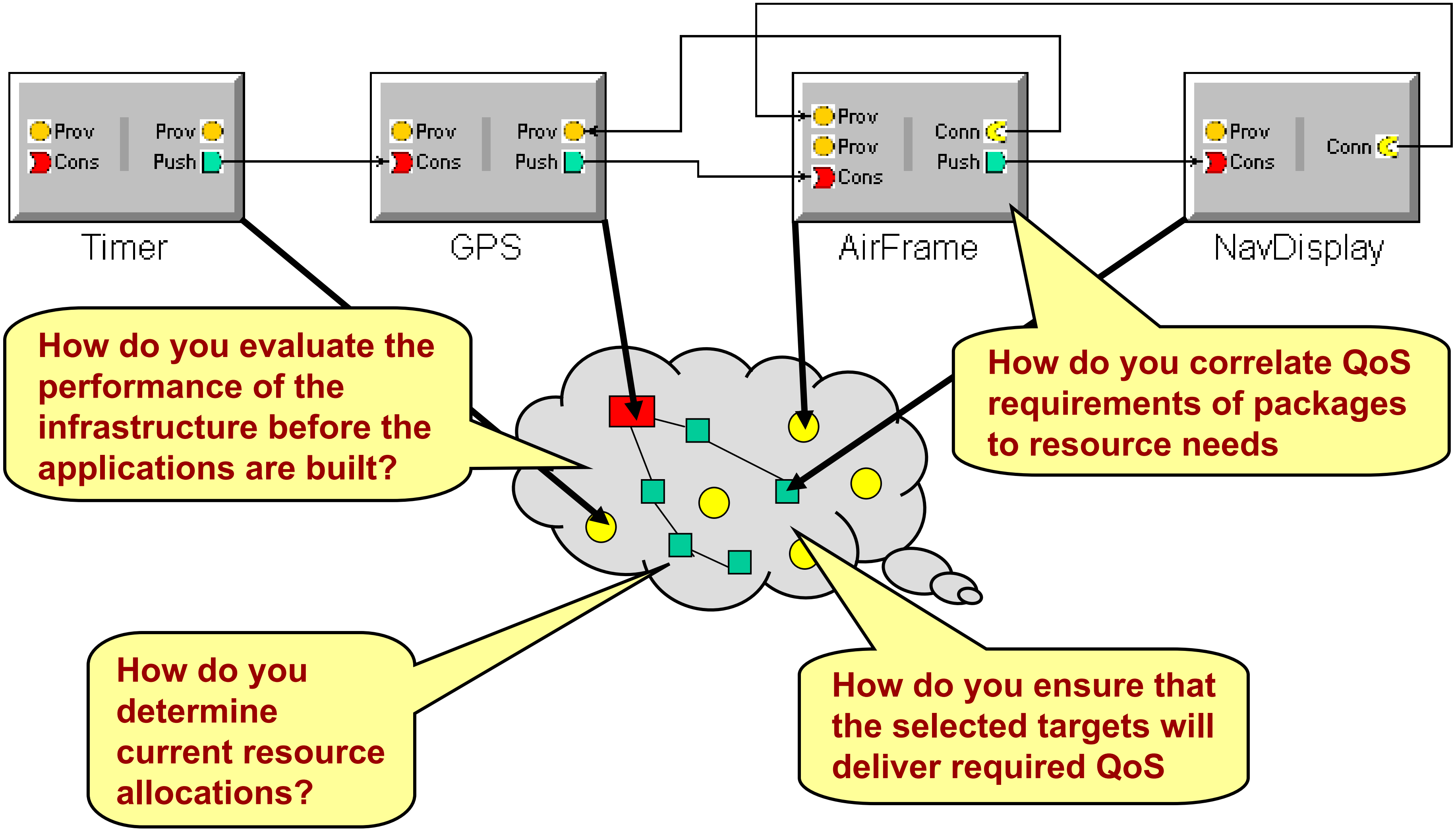
Challenge 3: Planning Aspect

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



Planning Aspect Problems

How to ensure deployment plans meet DRE system QoS requirements



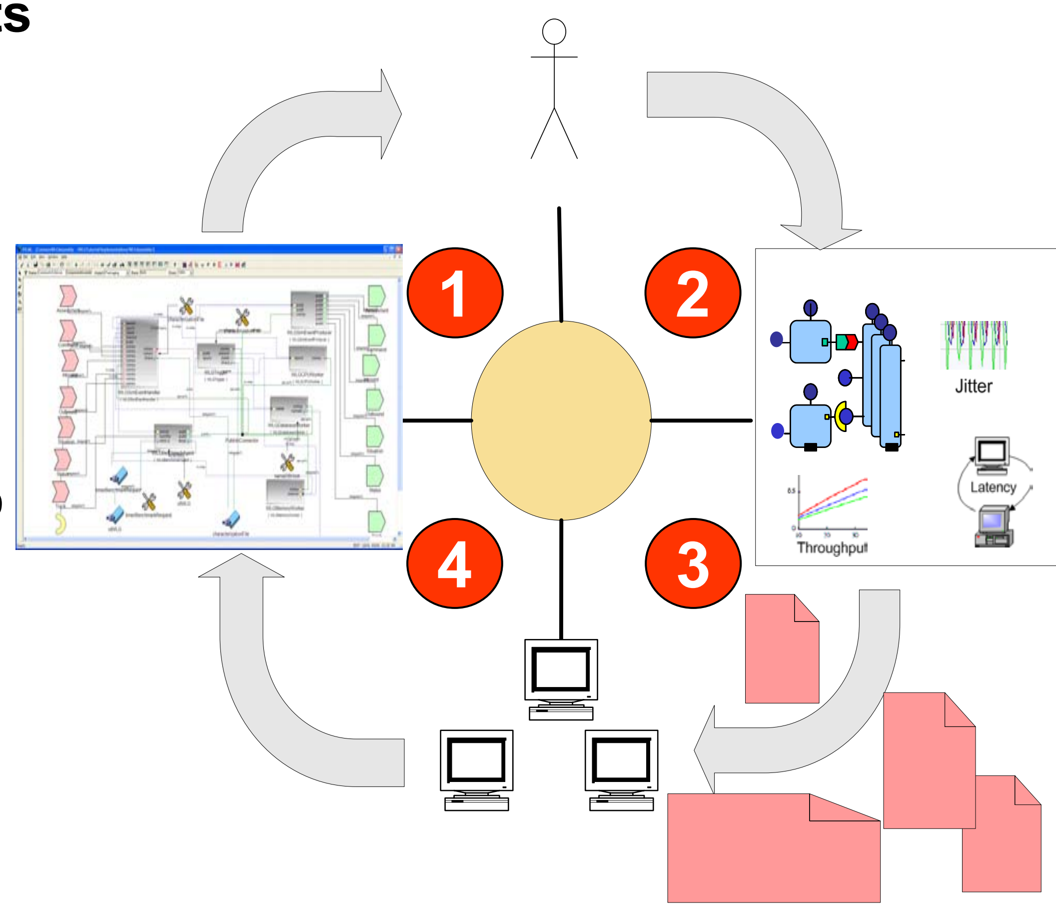
MDD Solution for Planning Aspect

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

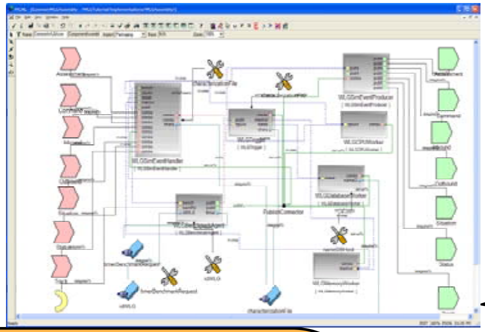
CoWorkEr Workflow for Architects

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*

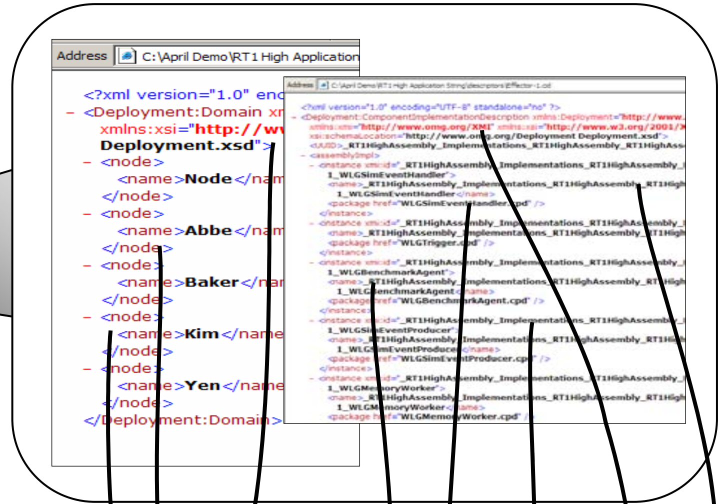


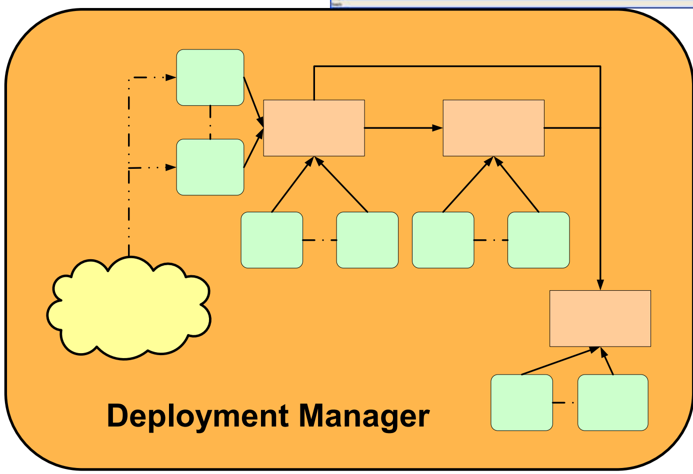
Integrating MDD & Middleware for Planning

CoWorkEr models system components, requirements, & constraints



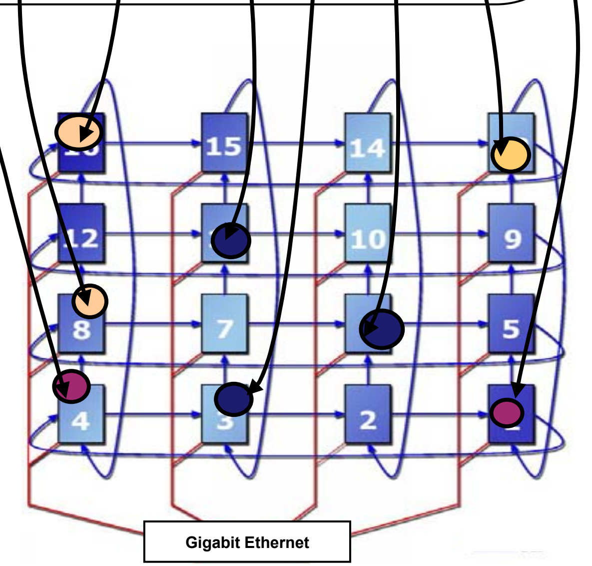
Deployment Plan





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

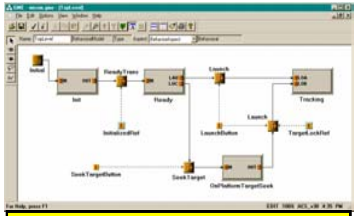
- *Deployment And Configuration Engine (DAnCE) maps plans to computing nodes*
- *RACE controls reallocations*



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

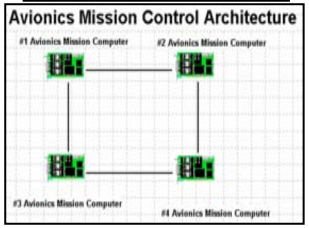
Concluding Remarks

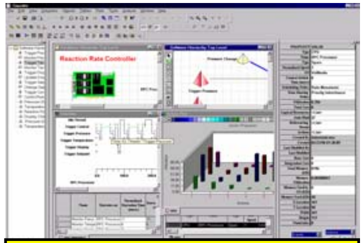
- To realize the promise of model-driven technologies, we need to augment model-driven methodologies with a solid (ideally standard) tool infrastructure
- Model-driven tools need to coexist with & enhance existing middleware platform technologies
- We need to validate model-driven technologies on (increasingly) large-scale, real-world systems



APPLICATION MDD TOOLS

PLATFORM MDD TOOLS





ANALYSIS MDD TOOLS

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