Applying Domain-Specific Modeling Languages to Develop DRE Systems

Krishnakumar Balasubramanian, Jaiganesh Balasubramanian,
Jeff Parsons, Aniruddha Gokhale,
Douglas C. Schmidt
kitty@dre.vanderbilt.edu
Institute for Software Integrated Systems,
Vanderbilt University

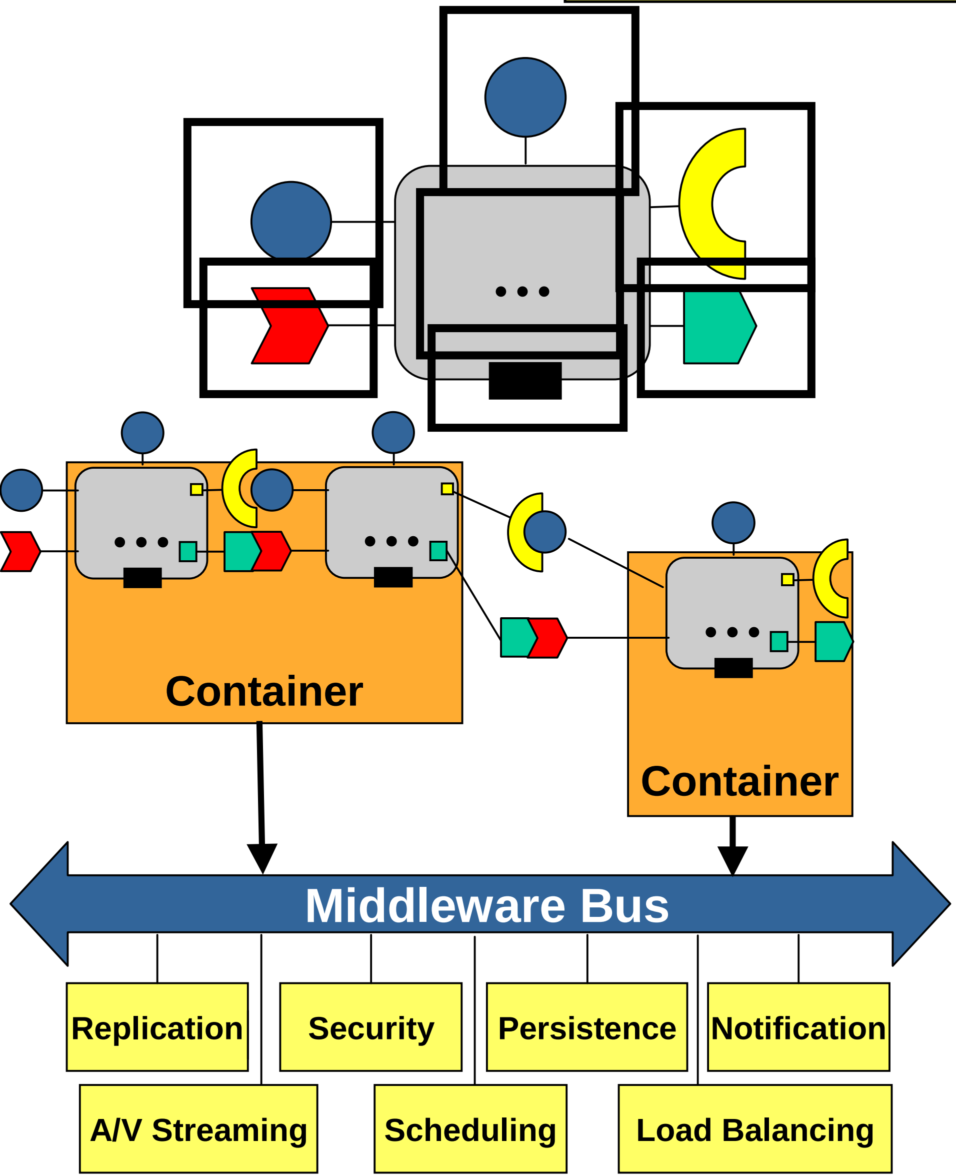
October 6, 2004

Overview

- Deployment & Configuration of Component-based systems
  - Introduction
  - Challenges
- Platform-Independent Component Modeling Language (PICML)
- Future work

Overview of Component Middleware

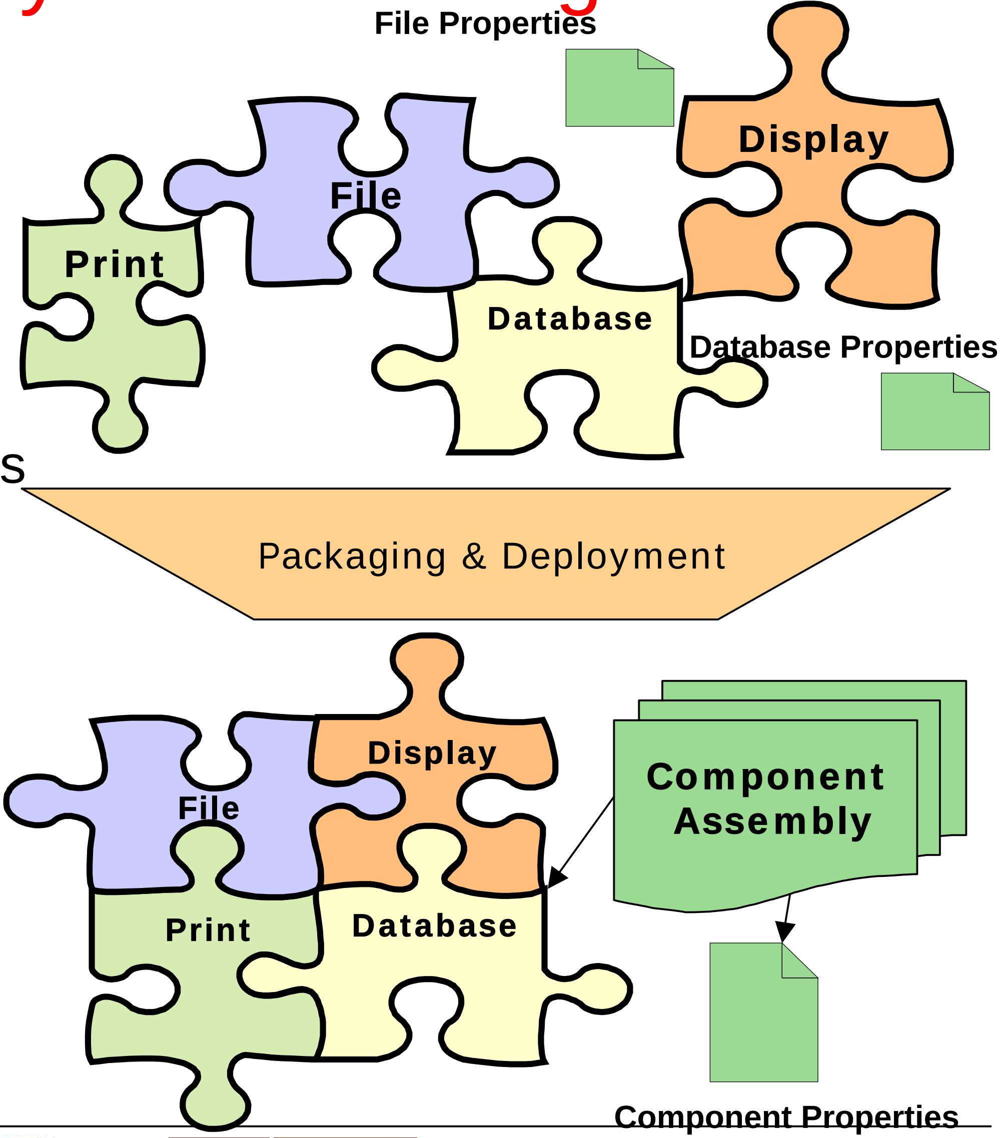
"Write Code That Reuses Code"



- *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 execution environment for components with common operating requirements
- Components/containers can also
  - Communicate via a *middleware bus* and
  - Reuse *common middleware services*

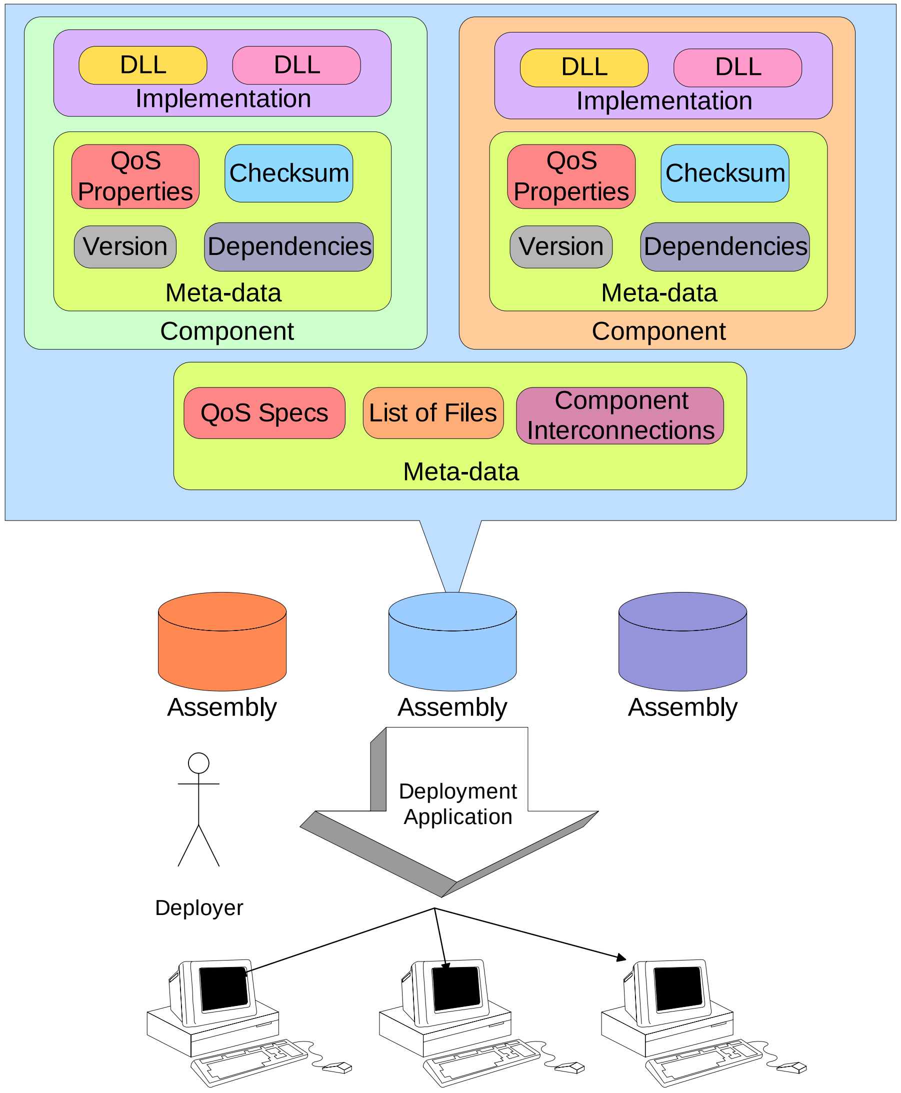
Motivation for Deployment & Configuration

- Goals
  - Ease component reuse
  - Build complex applications by assembling existing components
  - Standardize deployment of applications into heterogeneous domains
- Separation of concerns
  - Component development
  - Application assembly
  - Application deployment
  - Application configuration
  - Middleware configuration



OMG Deployment & Configuration Spec

- Specification defines deployment of component-based applications
- Intended to replace *Packaging & Deployment* chapter of CCM specification
- Meta-information is captured using XML descriptors
- Platform Independent Model (PIM)
- Defined in two dimensions
  - Data models vs. management (run-time) models
  - Component software vs. target vs. execution



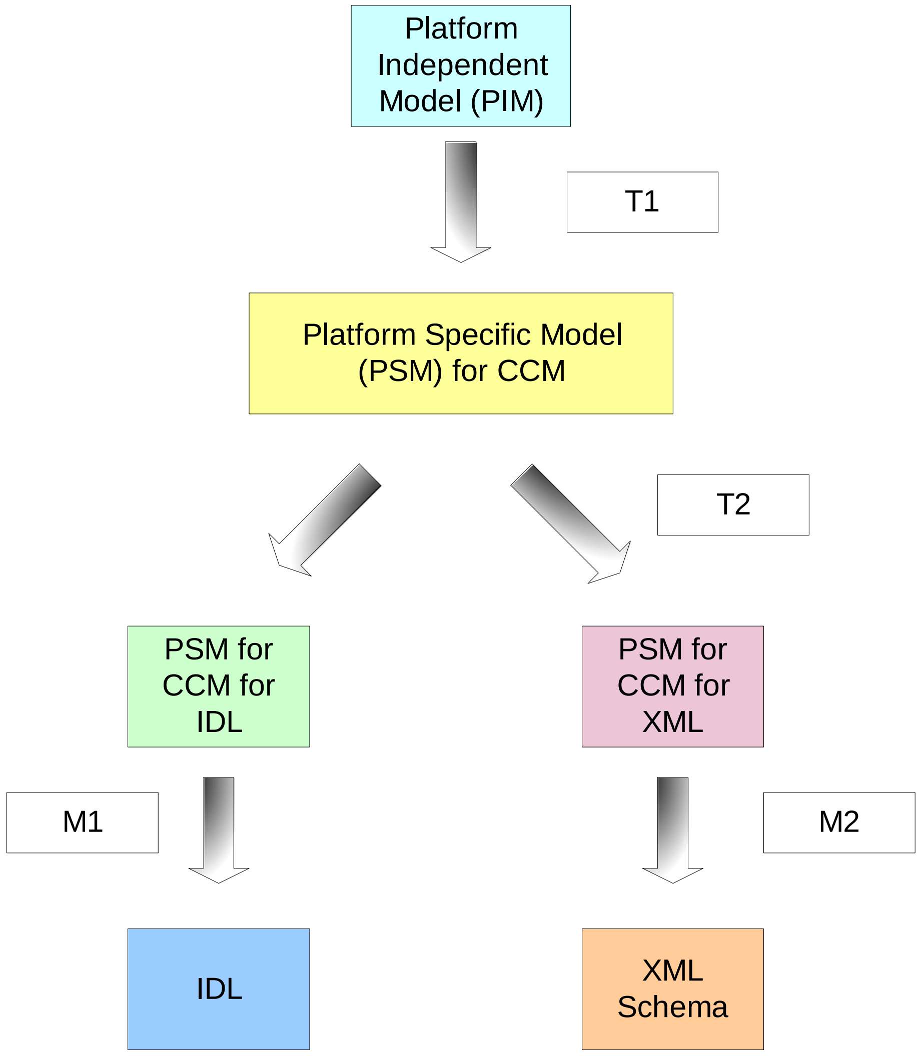
Platform-independent Model (PIM) Dimensions

- Modeling view-points
  - Conceptual, logical, & physical view-point
- Platform-independent model
  - Conceptual & logical viewpoint of deployment & configuration
- Defined in two-dimensions

PIM	Data Model	Run-time Model
Component Software	Meta-data to describe component based applications and their requirements	Interfaces to browse, store and retrieve such meta-data
Target	Meta-data to describe heterogeneous distributed systems & their capabilities	Interfaces to collect & retrieve such meta-data and commit resources
Execution	Meta-data to describe a specific deployment of an application into a distributed system	Prepare environment, Execute on target to Deployment plan, manage lifecycle

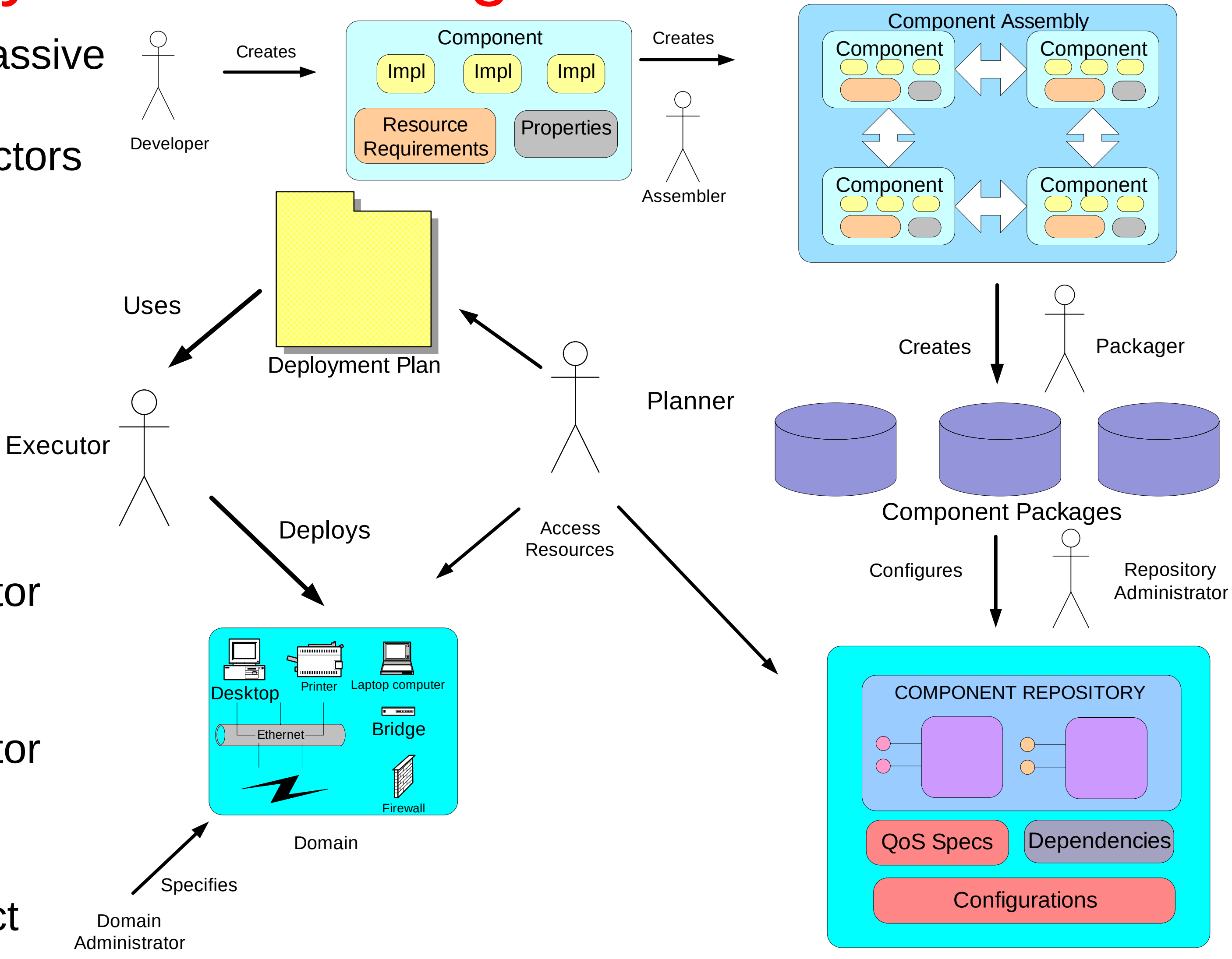
PIM Mapping to CCM

- Physical viewpoint
  - Mapping from PIM to platform specific model (PSM) for CCM
- Set of transformations
  - T1 → PIM to PSM for CCM
  - T2 → PSM to
    - PSM for IDL
    - PSM for XML
- Set of mapping rules
  - M1 → PSM to IDL
  - M2 → PSM to XML schema


Platform Independent Model (PIM)
T1
Platform Specific Model (PSM) for CCM
T2
PSM for CCM for IDL
PSM for CCM for XML
M1
M2
IDL
XML Schema

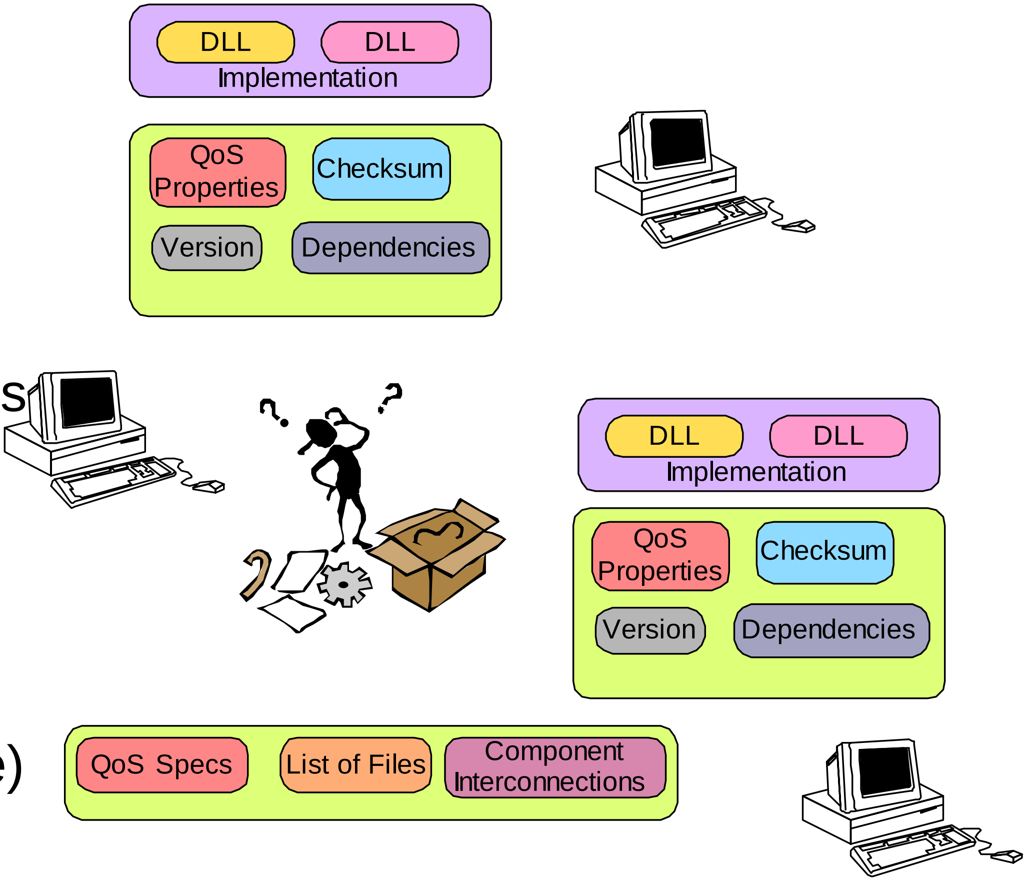
Deployment & Configuration Activities

- Descriptors are passive entities
- Manipulated by Actors
- Different Stages
  - *Development*
    - Developer
    - Assembler
    - Packager
  - *Target*
    - Domain Administrator
  - *Deployment*
    - Repository Administrator
    - Planner
    - Executor
- Actors are abstract



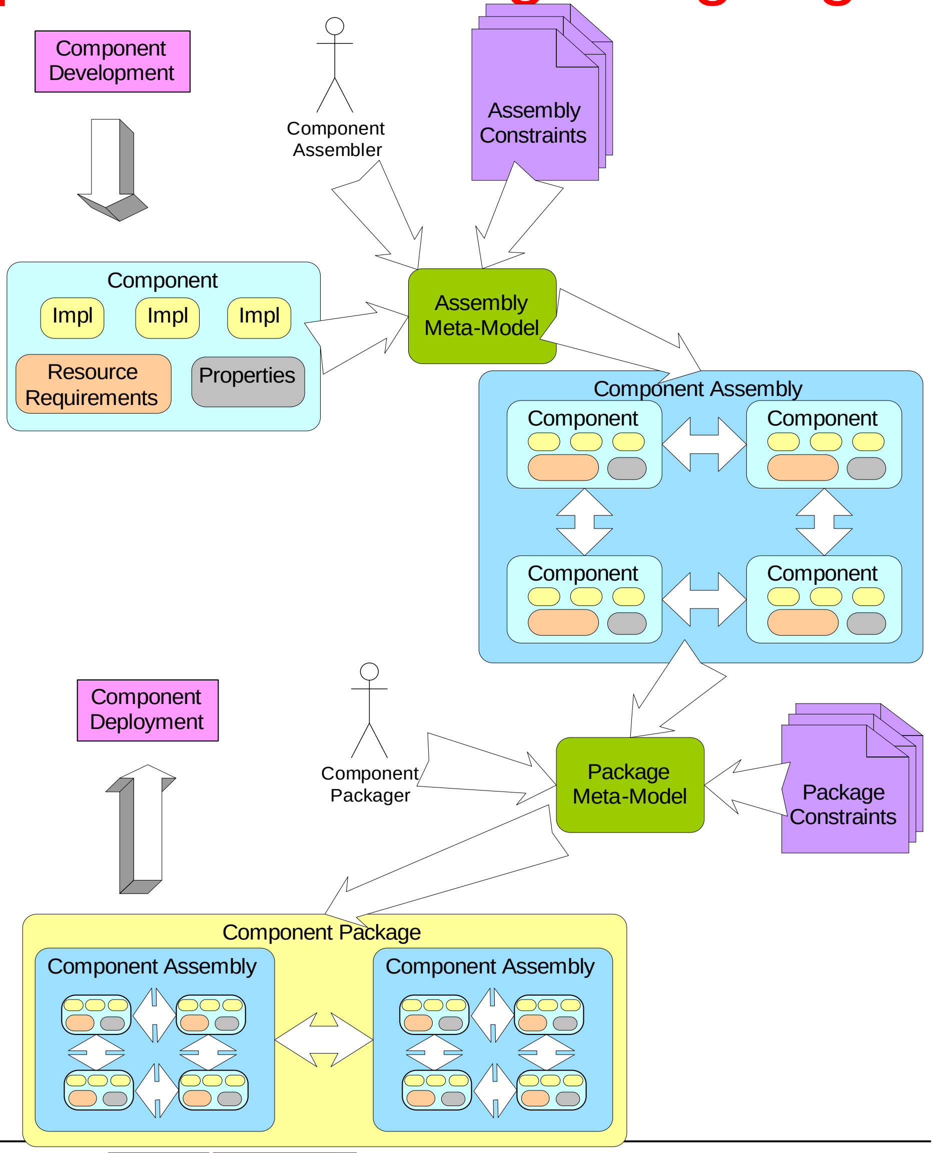
Configuration Challenges

- Context
  - Configuring & composing component-based applications using XML meta-data
- Problem
  - Meta-data split across multiple XML descriptors
  - Complex inter-dependencies between descriptors
  - XML is error-prone to read/write manually
  - No guarantees about semantic validity (only syntactic validation possible)
  - If meta-data is wrong, what about the application?



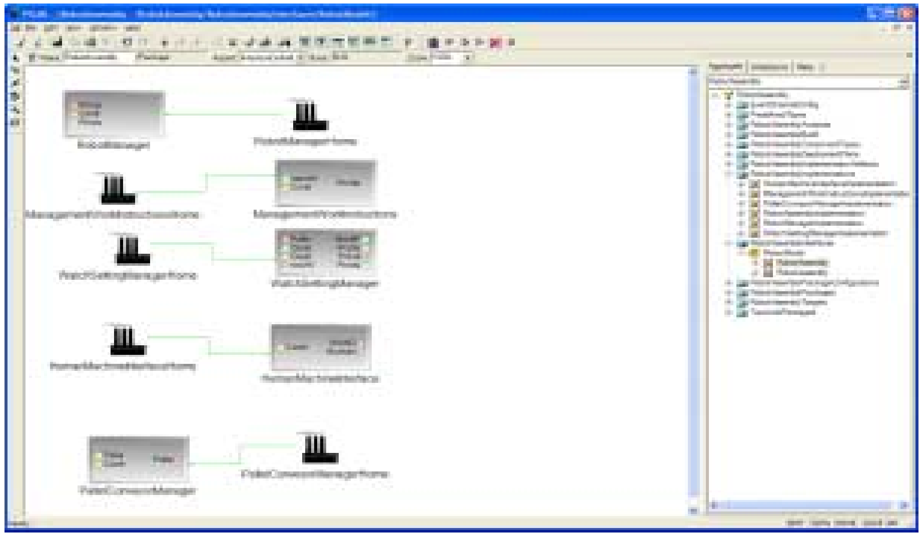
Platform-Independent Component Modeling Language

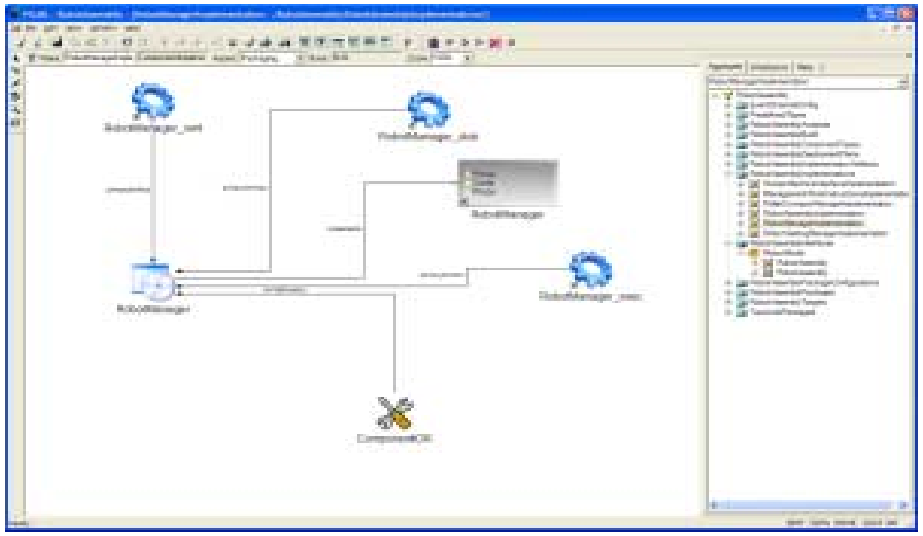
- Solution
  - PICML
  - Developed in Generic Modeling Environment (GME)
  - Core of Component Synthesis using Model-Integrated Computing (CoSMIC) toolchain
  - Capture elements & dependencies visually
  - Define "static semantics" using Object Constraint Language (OCL)
  - Define "dynamic semantics" via model interpreters
    - Also used for generating domain specific meta-data
  - "Correct-by-construction"



Steps in a typical usage of PICML (1/2)

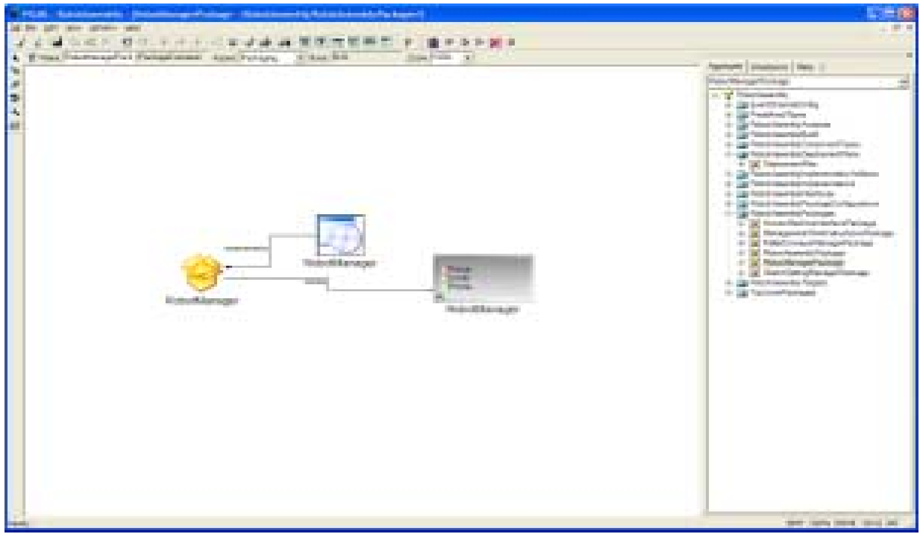
- Define component types
  - Import/Export IDL
- Define implementation artifacts
  - External libraries, individual component libraries
- Define component implementations
  - Monolithic components
  - Assembly-based components
  - Assembly of assemblies

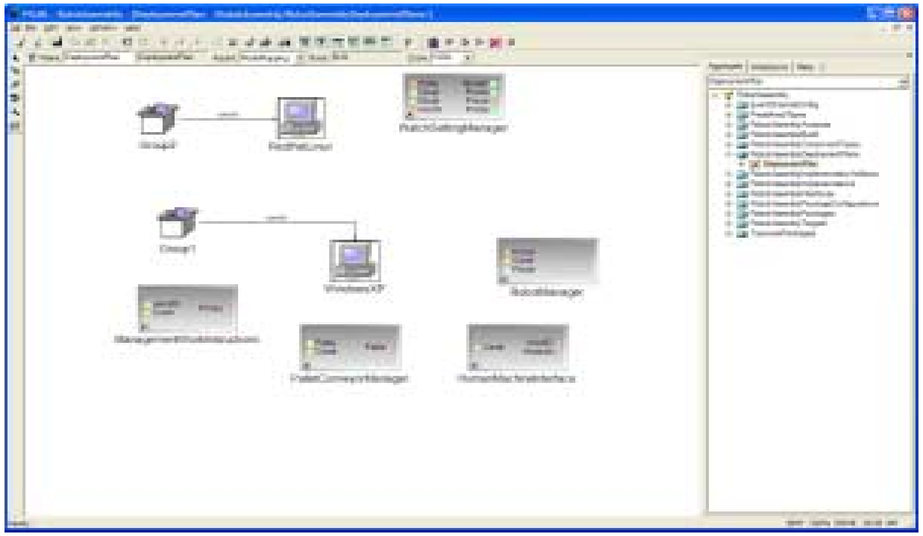




Steps in a typical usage of PICML (2/2)

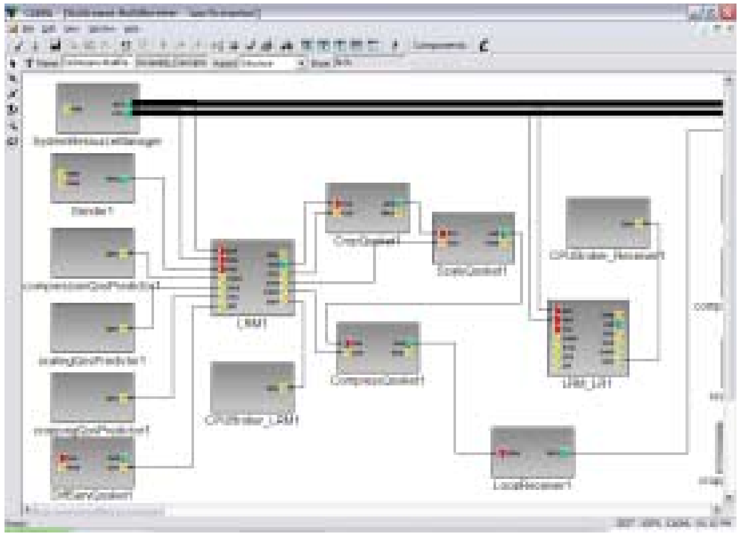
- Define component packages
  - Configure (previously defined) component packages
- Define deployment target
  - Can be done as a decoupled activity
- Define deployment plan
  - Place components on different nodes of the target

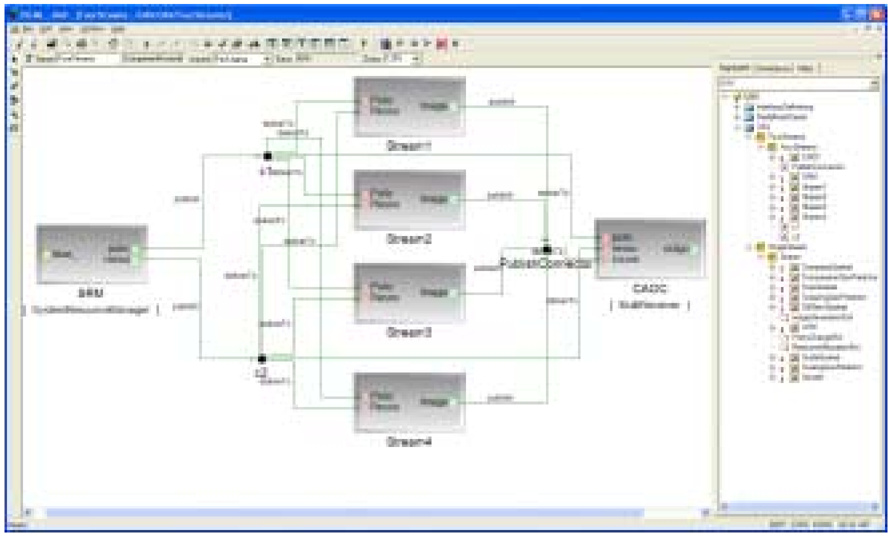


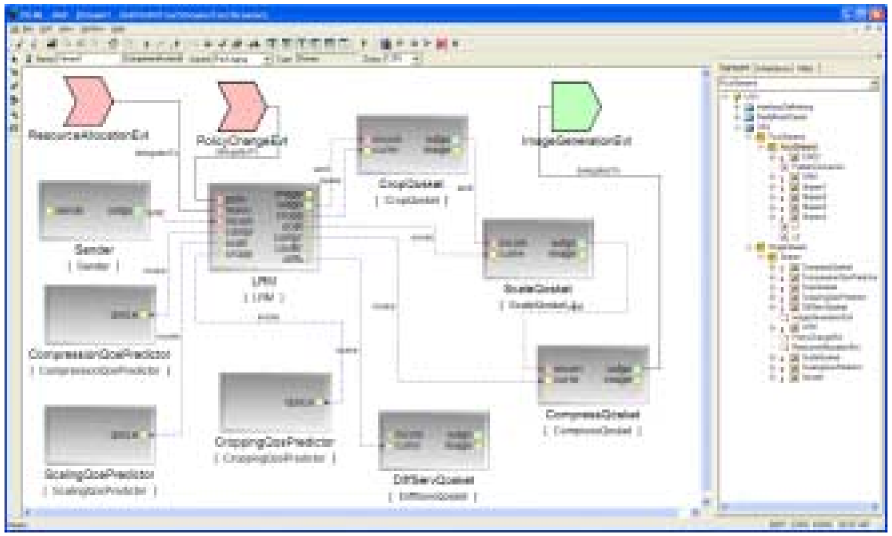


Hierarchical Composition

- Six streams of image data
  - System incomprehensible
- Hierarchical composition
  - Reason about system at multiple levels of abstraction

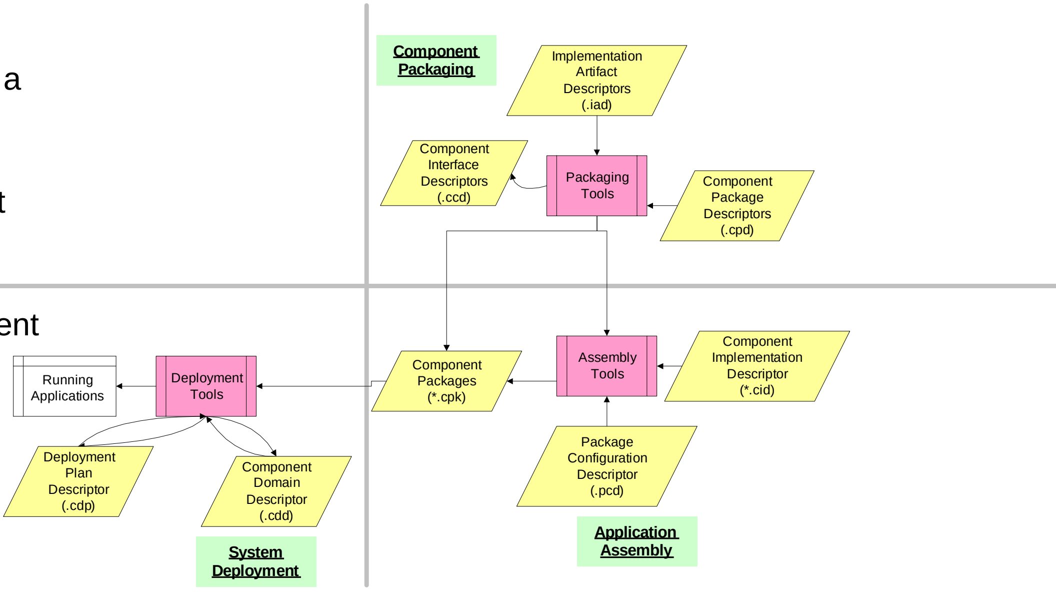




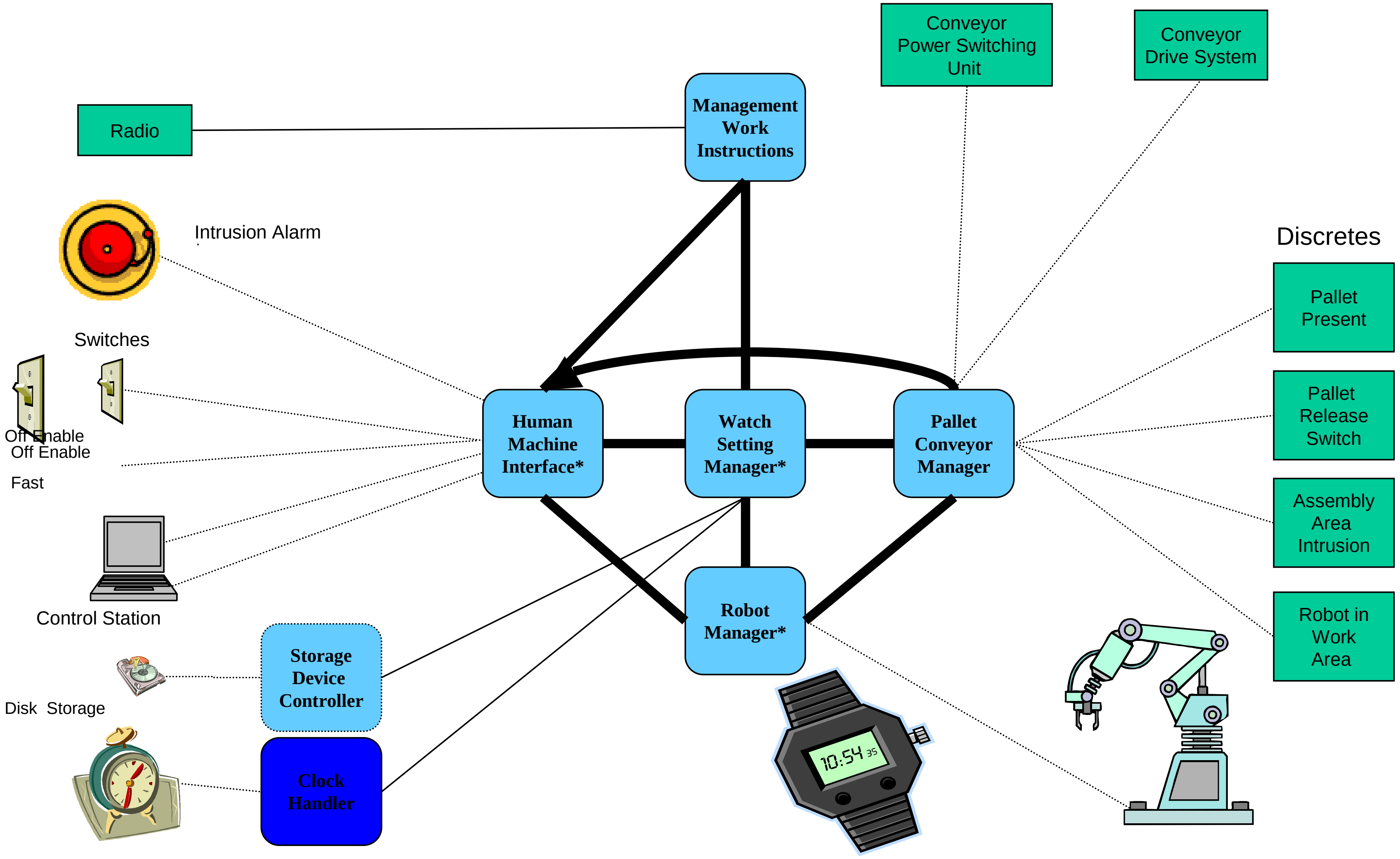


Types Of Meta-data generated by PICML

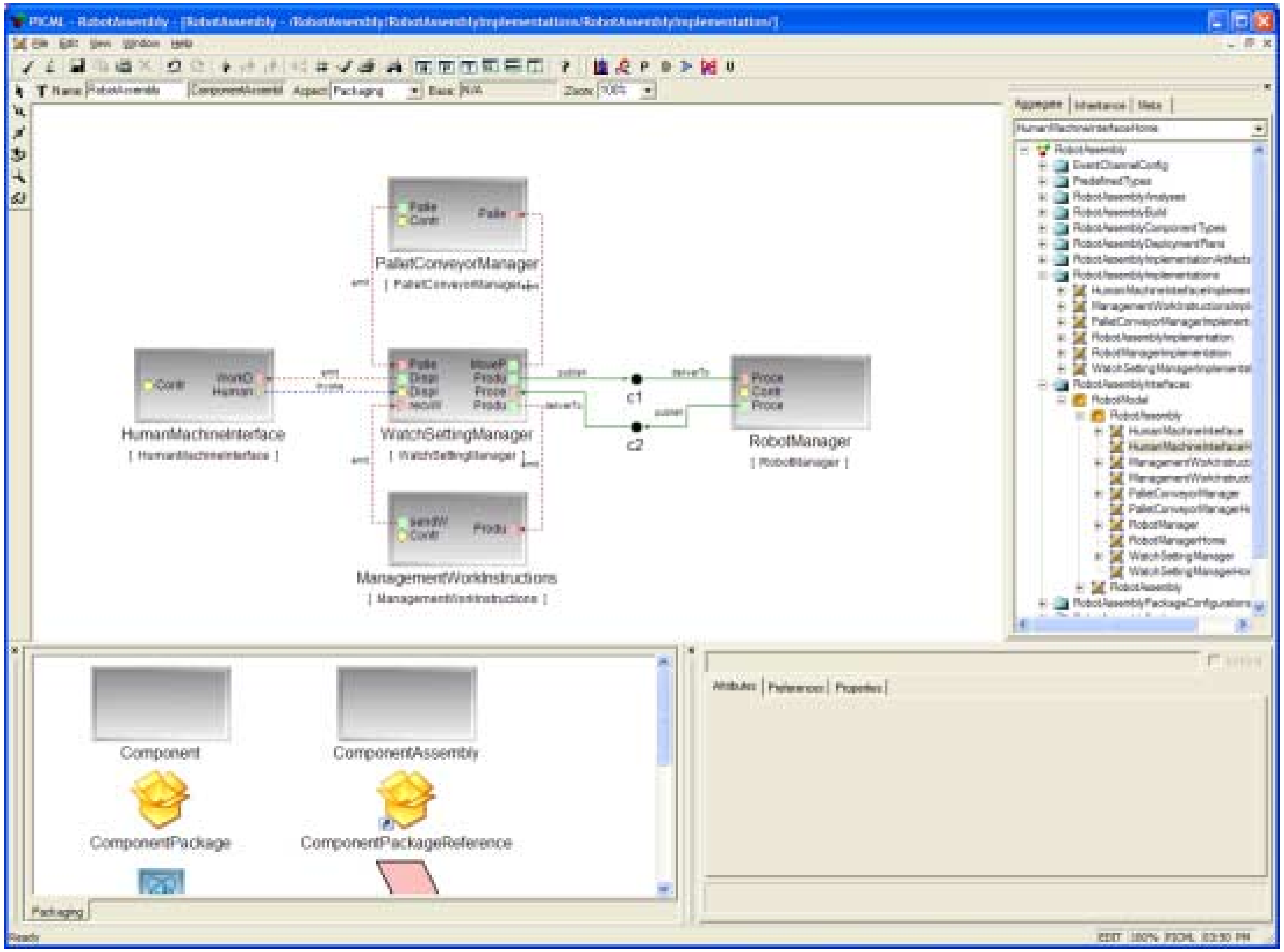
- **Component Interface Descriptor (.ccd)**
  - Describes the interface, ports, properties of a single component
- **Implementation Artifact Descriptor (.iad)**
  - Describes the implementation artifacts (e.g., DLLs, OS, etc.) of a single component
- **Component Package Descriptor (.cpd)**
  - Describes multiple alternative implementations of a single component
- **Package Configuration Descriptor (.pcd)**
  - Describes a specific configuration of a component package
- **Component Implementation Descriptor (.cid)**
  - Describes a specific implementation of a component interface
  - Contains component inter-connection information
- **Component Deployment Plan (.cdp)**
  - Plan which guides the actual deployment
- **Component Domain Descriptor (.cdd)**
  - Describes the target domain of deployment
- **Component Packages (.cpk)**
  - Aggregation of all of the above


Component Packaging
Implementation Artifact Descriptors (.iad)
Component Interface Descriptors (.ccd)
Packaging Tools
Component Package Descriptors (.cpd)
Running Applications
Deployment Tools
Component Packages (*.cpk)
Assembly Tools
Component Implementation Descriptor (*.cid)
Deployment Plan Descriptor (.cdp)
Component Domain Descriptor (.cdd)
Package Configuration Descriptor (.pcd)
System Deployment
Application Assembly

Example Application: RobotAssembly



RobotAssembly in PICML

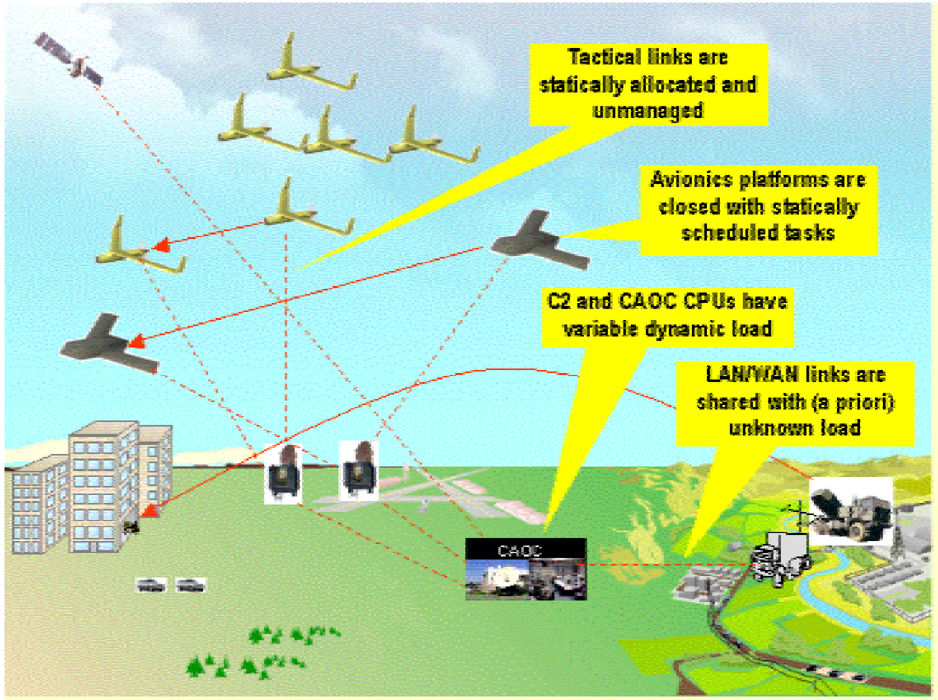

PICML - RobotAssembly - [RobotAssembly - /RobotAssembly/RobotAssemblyImplementations/RobotAssemblyImplementation/]
File Edit View Window Help
Aspect: Packaging
Zoom: 100%
PalletConveyorManager
WatchSettingManager
HumanMachineInterface
ManagementWorkInstructions
RobotManager
c1
c2
Aggregate Inheritance Meta
RobotAssembly
EventChannelConfig
PredefinedTypes
RobotAssemblyImplementations
RobotAssemblyInterfaces
RobotModel
Component
ComponentAssembly
ComponentPackage
ComponentPackageReference
Packaging
Attributes Preferences Properties
Ready

Example output for RobotAssembly

```
<!--Component Implementation Descriptor(.cid) associates components with impl. artifacts-->
<Deployment:ComponentImplementationDescription>
  <UUID>FB9D7161-1765-4784-BC1D-EA9EAAB3ED2A</UUID>
  <implements href="RobotManager.ccd" />
  <monolithicImpl>
    <primaryArtifact>
      <name>RobotManager_exec</name>
      <referencedArtifact href="RobotManager_exec.iad" />
   </primaryArtifact>
   <primaryArtifact>
      <name>RobotManager_stub</name>
      <referencedArtifact href="RobotManager_stub.iad" />
   </primaryArtifact>
   <primaryArtifact>
     <name>RobotManager_svnt</name>
     <referencedArtifact href="RobotManager_svnt.iad“ />
   </primaryArtifact>
  </monolithicImpl>
</Deployment:ComponentImplementationDescription>
```

Example Application: UAV

UAV-OEP-the Problem: Multi-UAV Surveillance and Target Tracking Requires Dynamic End-to-End QoS Management


Tactical links are statically allocated and unmanaged
Avionics platforms are closed with statically scheduled tasks
C2 and CAOC CPUs have variable dynamic load
LAN/WAN links are shared with (a priori) unknown load
CAOC

End-to-End Mission-Driven QoS Management

Surveillance Mode

- Maximize surveillance area
- Sufficient resolution in delivered imagery to determine items of interest

Target Acquisition and Engagement

- UAV observing target provides high resolution imagery so that target or threat identification is possible

Battle Damage Assessment

- UCAV must provide high resolution imagery until a human operator has determined that it is sufficient
- UAV over target area must continue to provide target acquisition and engagement mission

Mission-defined requirements and tradeoffs (e.g., rate, image size, fidelity)

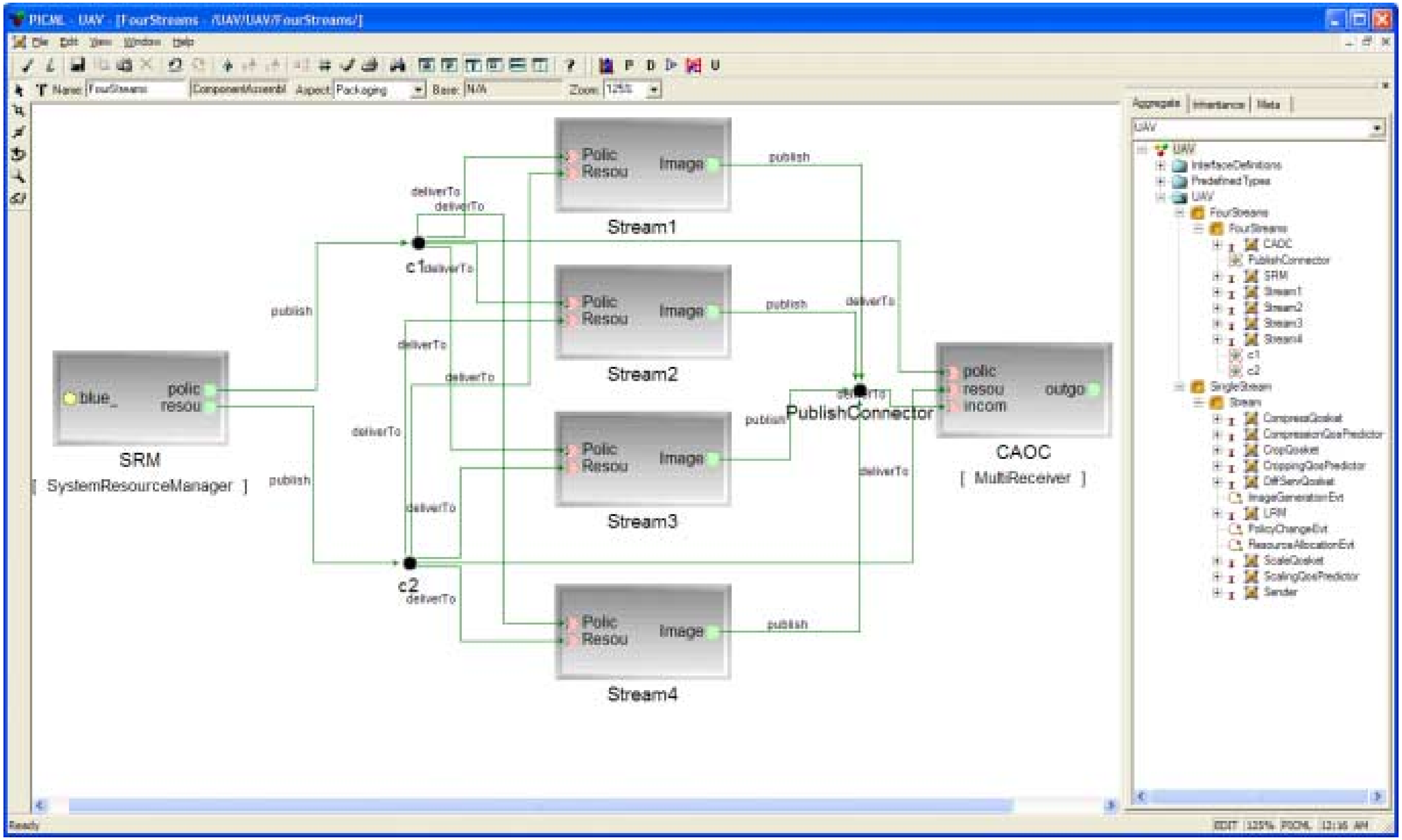
Changing modes, participants, and environmental conditions

Heterogeneous, shared, and constrained resources

Multi-layer points of view: System-view, mission-view, application-string view, local resource view

The challenge is to program the dynamic control and adaptation to manage and enforce end-to-end QoS

UAV in PICML


PICML - UAV - [FourStreams - /UAV/UAV/FourStreams/]
File Edit View Window Help
Name: FourStreams
ComponentAssembl
Aspect: Packaging
Base: N/A
Zoom: 125%
Polic
Resou
Image
Stream1
Stream2
Stream3
Stream4
publish
deliverTo
c1
c2
blue_
polic
resou
SRM
[SystemResourceManager]
PublishConnector
polic
resou
incom
outgo
CAOC
[MultiReceiver]
Aggregate
Inheritance
Meta
UAV
InterfaceDefinitions
Predefined Types
FourStreams
CAOC
PublishConnector
SRM
Stream1
Stream2
Stream3
Stream4
c1
c2
SingleStream
Stream
CompressQosket
CompressionQosPredictor
CropQosket
CroppingQosPredictor
DiffServQosket
ImageGenerationEvt
LRM
PolicyChangeEvt
ResourceAllocationEvt
ScaleQosket
ScalingQosPredictor
Sender
Ready
EDIT 125% PICML 12:16 AM

Concluding Remarks

- PICML
  - Model component-based systems
  - Allows design-time validation of systems
  - Generates component meta-data
- Future work
  - Scalability issues
  - Traditional system analysis
  - Complete system generation *aka* “Middleware Compiler”
  - Generate optimized systems *aka* “Optimizing Middleware Compiler”
- Available as open-source
  - http://cvs.dre.vanderbilt.edu (CoSMIC)