Model-Driven Optimizations of Component Systems

OMG Real-time Workshop
July 12, 2006

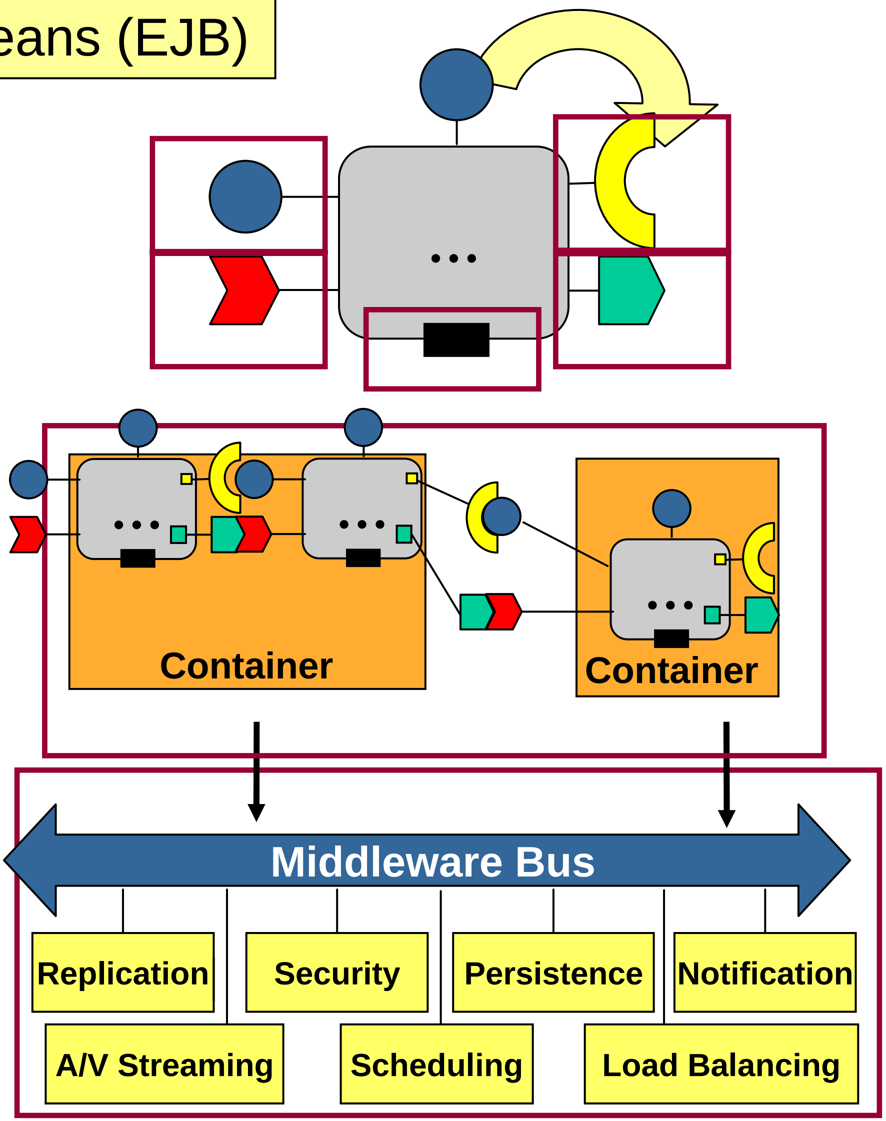
Krishnakumar Balasubramanian
Dr. Douglas C. Schmidt
{kitty,schmidt}@dre.vanderbilt.edu

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee

Component Middleware

e.g., CORBA Component Model (CCM), Microsoft .NET Web Services, Enterprise Java Beans (EJB)

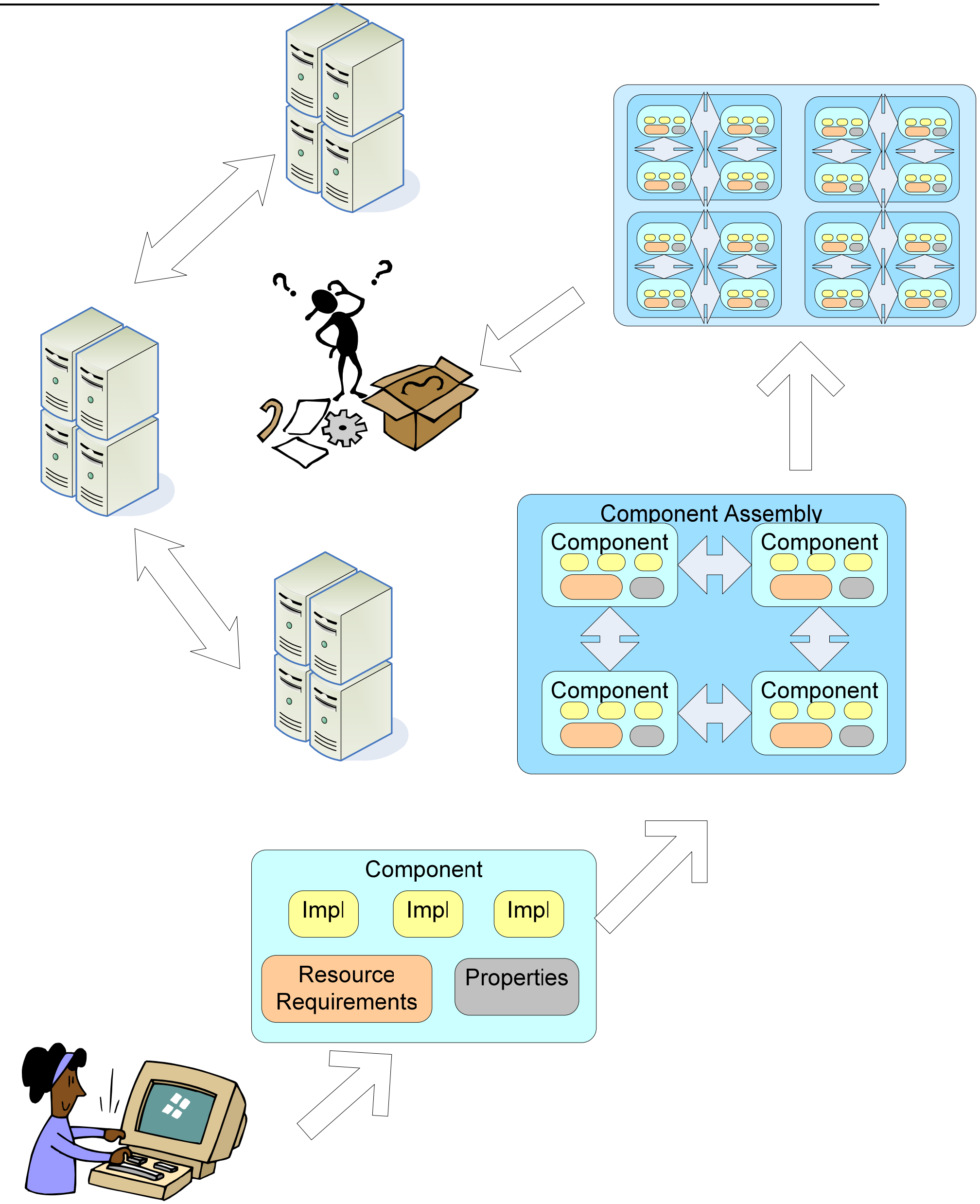
- *Components* encapsulate "business" logic
- Components interact via *ports*
  - *Provided interfaces*
  - *Required interfaces*
  - *Event sinks & sources*
  - *Attributes*
- Allow navigation between ports
- *Containers* provide execution environment for components
- Components/containers can also
  - Communicate via a *middleware bus* & reuse *common middleware services*



Components allow reasoning about systems at higher level of abstraction

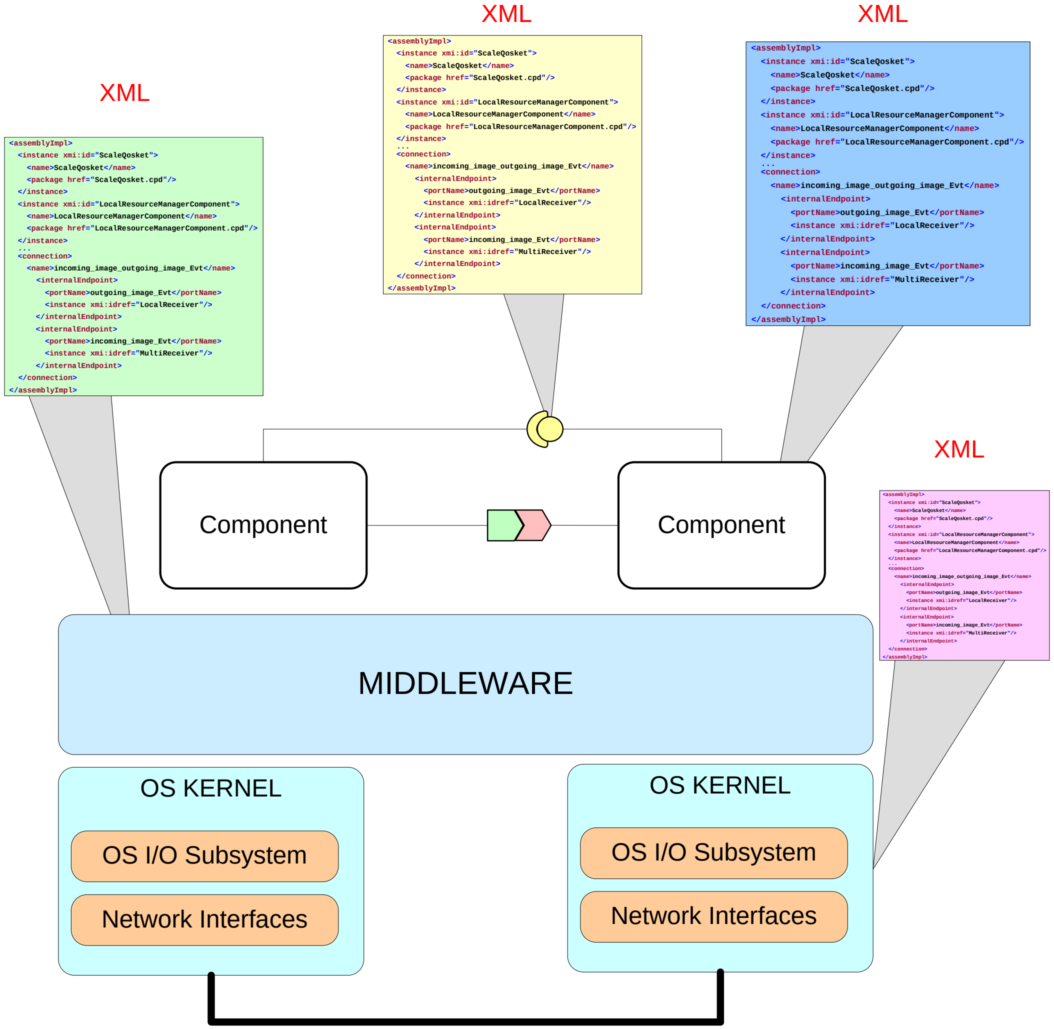
Component System Development Challenges

- Lack of system composition tools



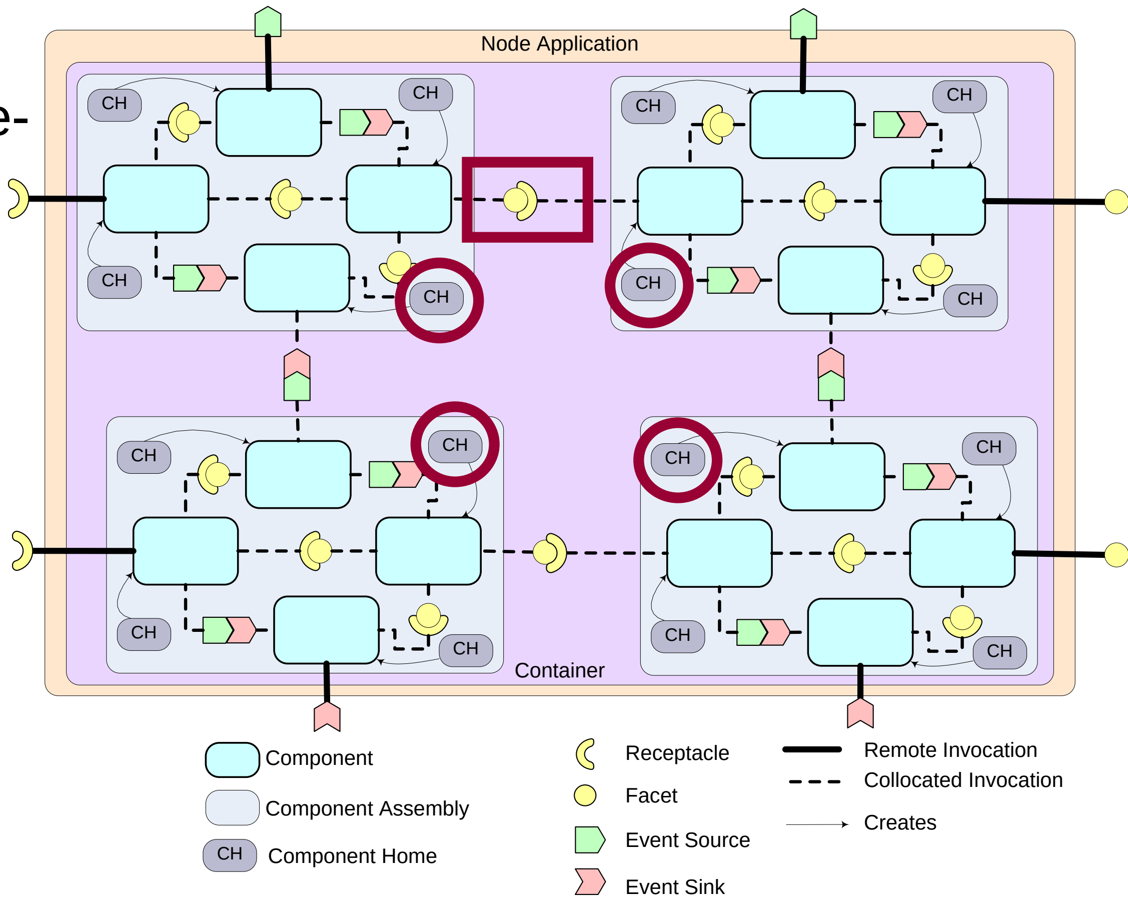
Component System Development Challenges

- Lack of system composition tools
- Complexity of declarative platform API & notations



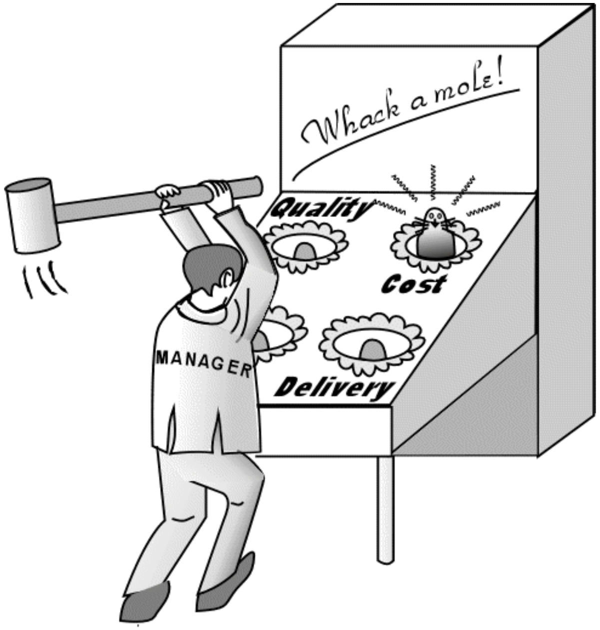
Component System Development Challenges

- Lack of system composition tools
- Complexity of declarative platform API & notations
- Composition overhead in large-scale systems



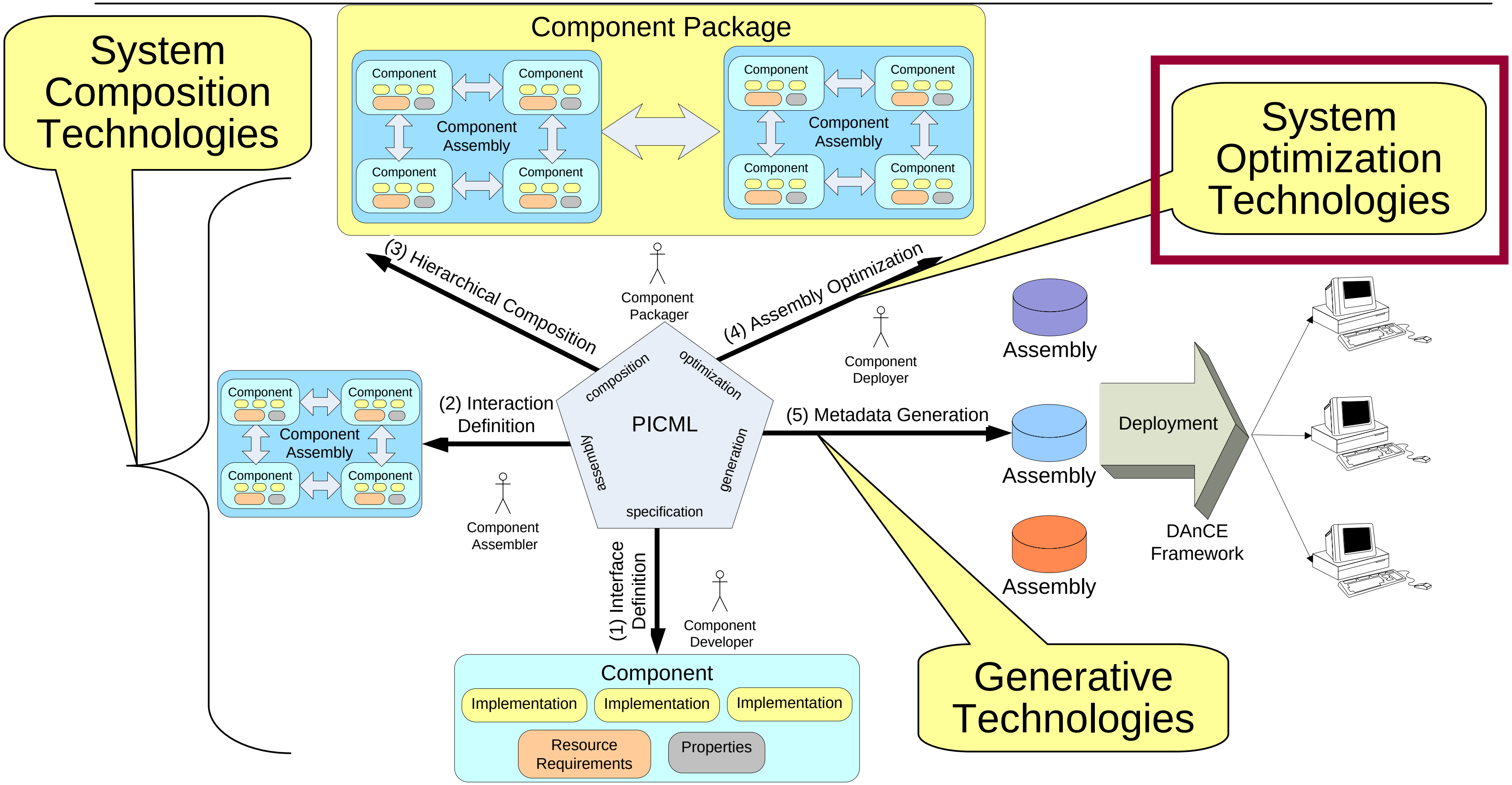
Component System Development Challenges

- Lack of system composition tools
- Complexity of declarative platform API & notations
- Composition overhead in large-scale systems
- Emphasis still on *programming-in-the-small*
  - *Whack-a-mole* approach to system development
  - Violation of *Don't Repeat Yourself (DRY)* principle
- Lack of abstractions for expressing system level design intent



Need for tools to *design* & *optimize* "systems-in-the-large"

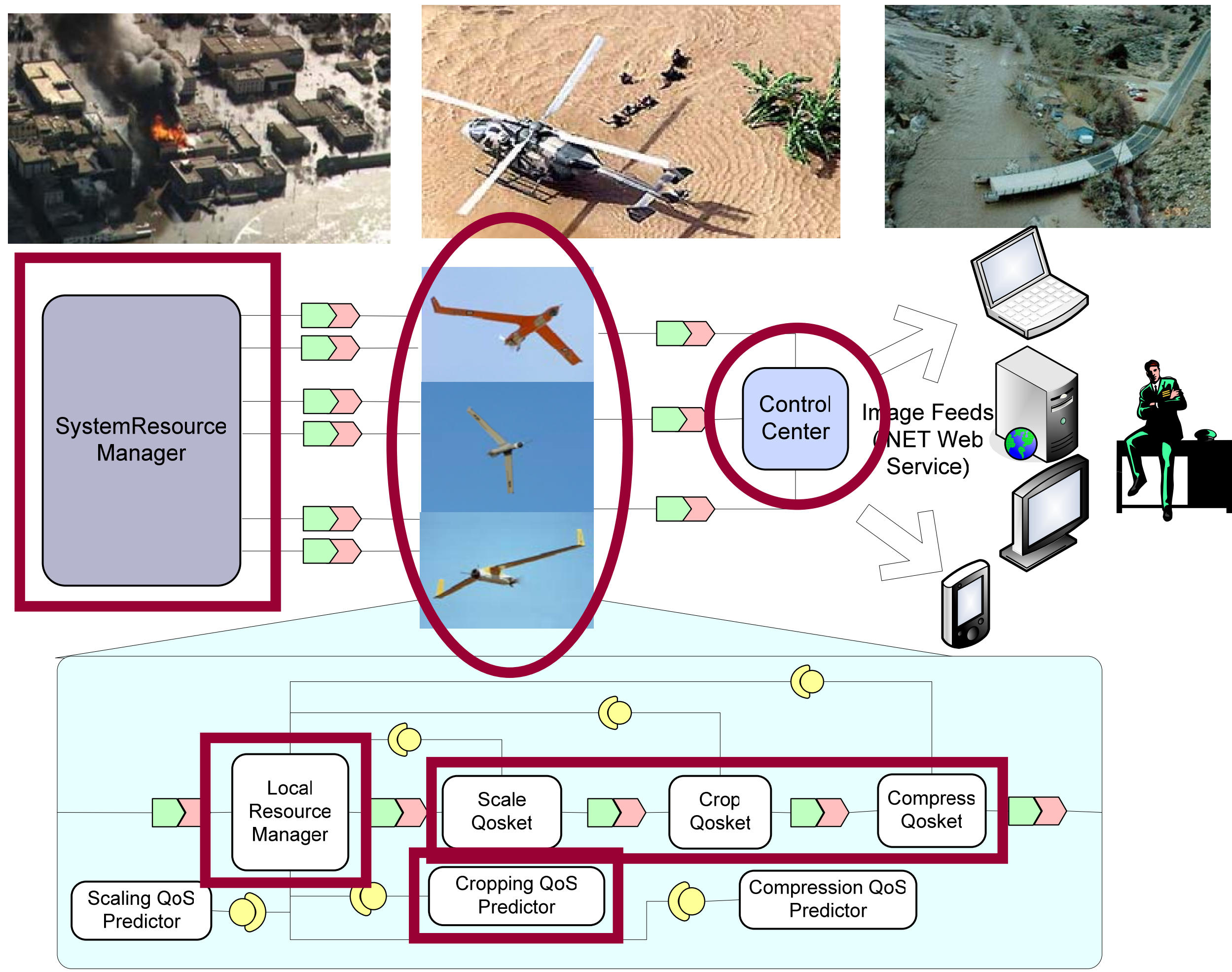
Solution Approach: Model-Driven Engineering



- System Composition Technologies - A Domain-Specific Modeling Language (DSML) to allow component *specification & composition*
- System Optimization Technologies – System *optimization* infrastructure
- Generative Technologies – Metadata *generation* infrastructure

Example Scenario: Emergency Response System

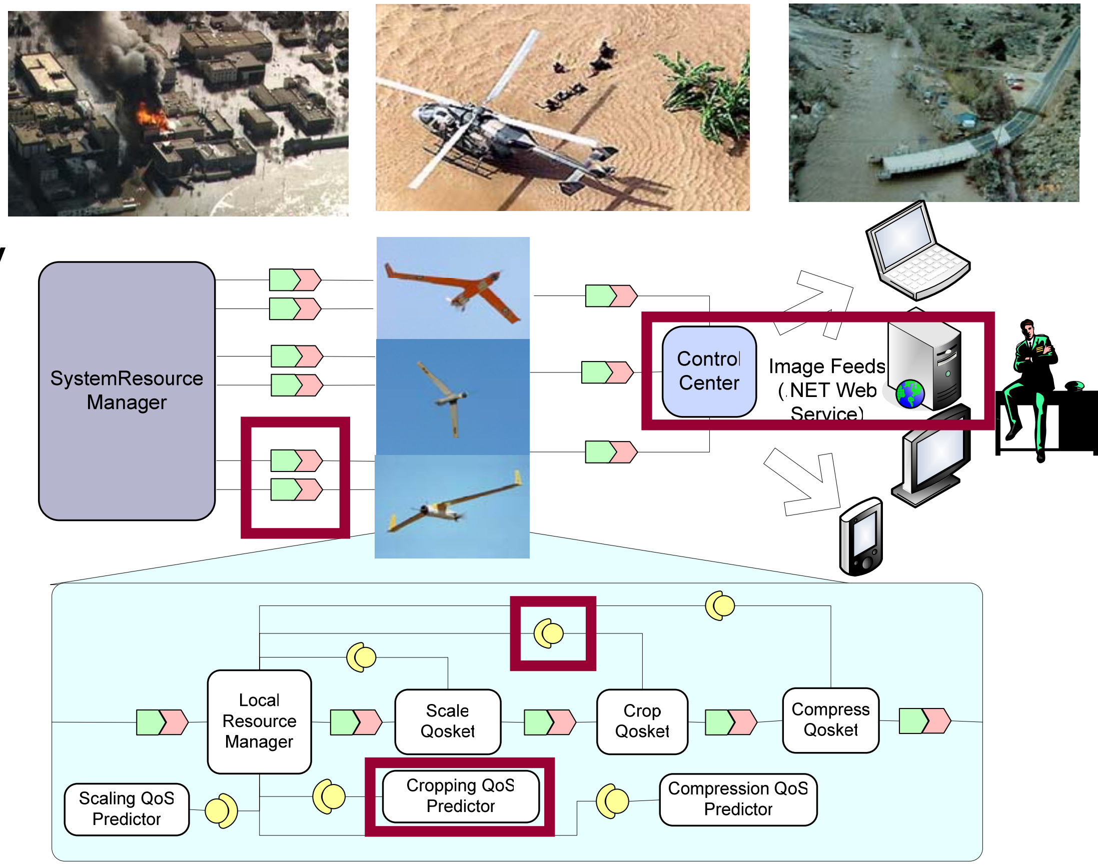
- System Resource Manager
  - Global QoS manager
- Control Center
  - User interaction
- Image Stream(s)
  - Local Resource Manager
    - Local QoS manager
  - Qoskets
    - QoS enforcer
  - QoS predictors
    - QoS estimators
- Built using the Component-Integrated ACE ORB (CIAO)



Developed for DARPA PCES program (dist-systems.bbn.com/papers/)

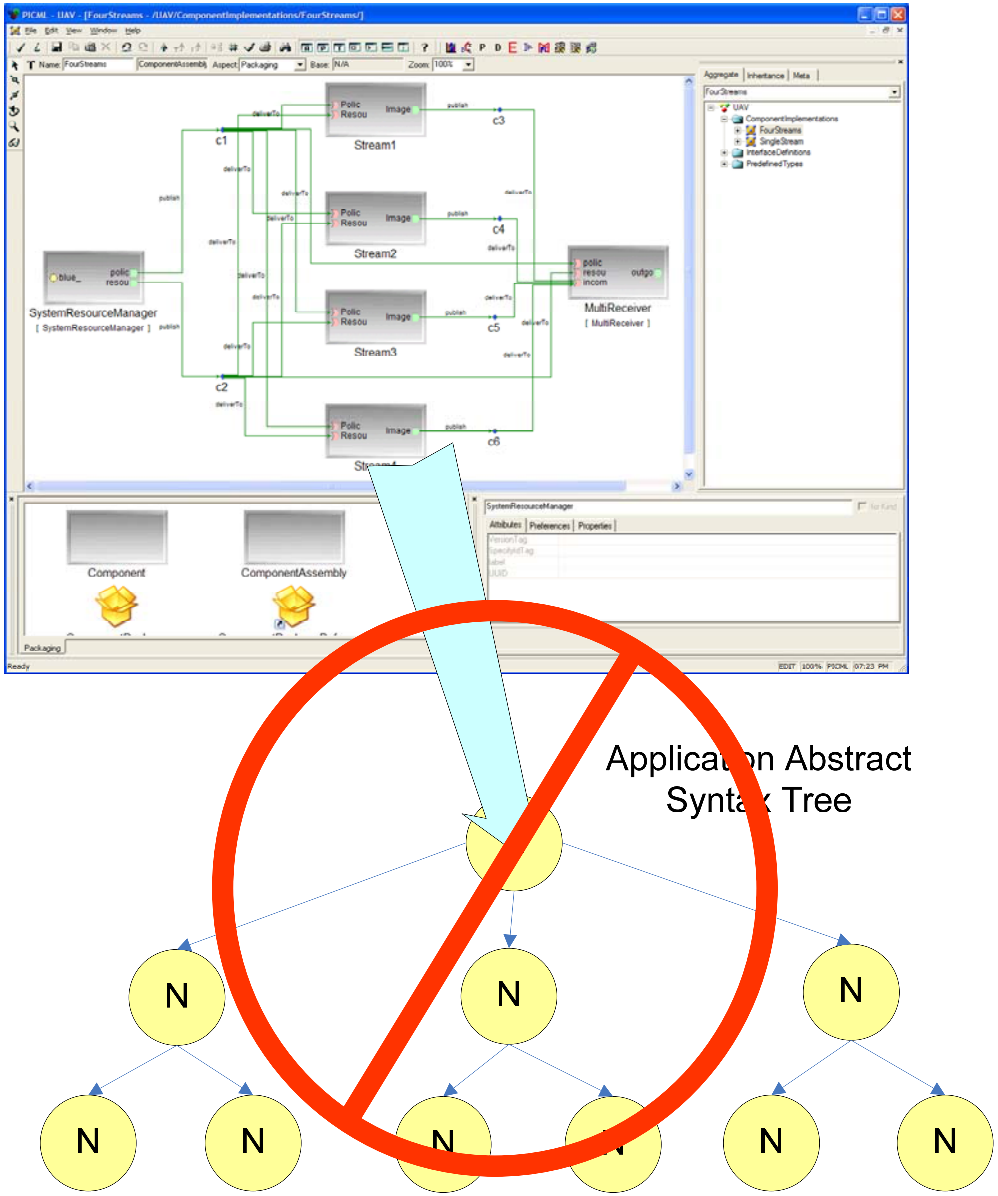
Application Specific Optimizations

- Middleware tries to optimize execution for every application
- Collocated method invocations
  - Optimize the (de-)marshaling costs by exploiting locality
- Specialization of request path by exploiting protocol properties
  - Caching, Compression, Various Encoding schemes, e.g. FOCUS tool-chain
- Reducing communication costs
  - Moving data closer to the consumers by replication
- Reflection-based approaches
  - Choosing appropriate alternate implementations



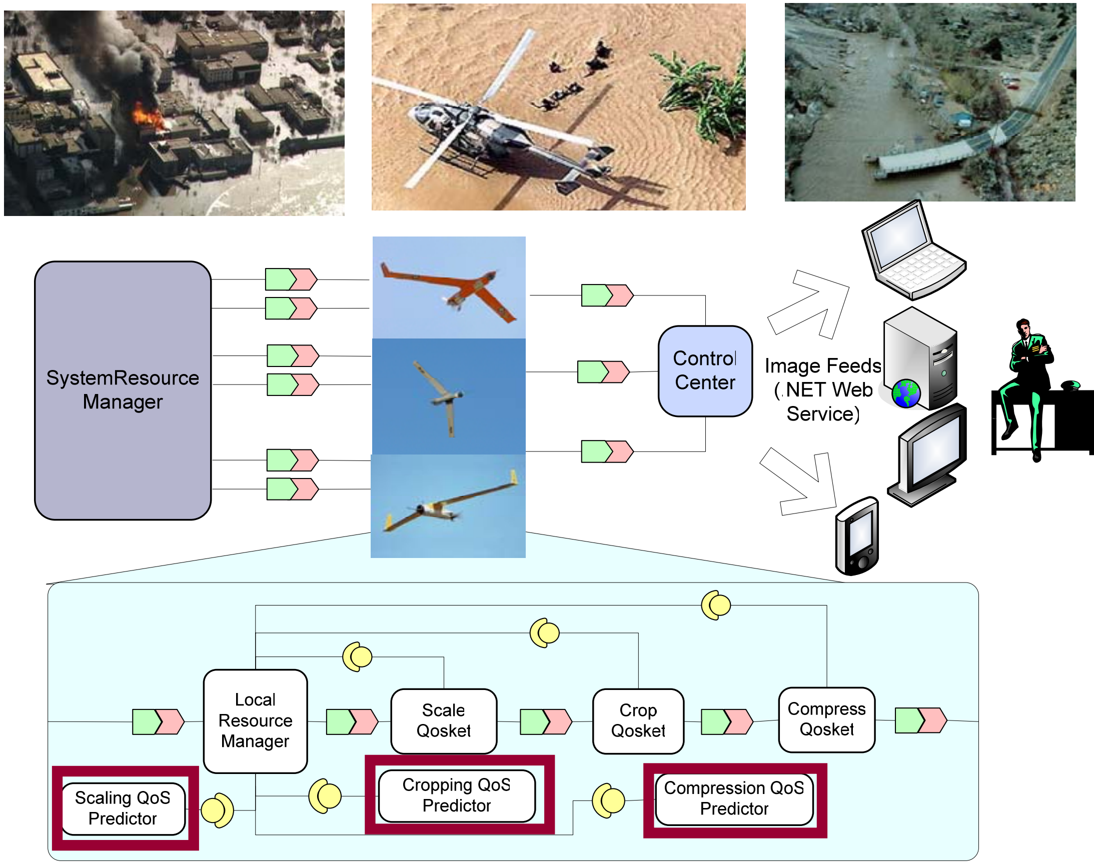
Application Specific Optimizations: What's missing?

- Lack of high-level notation to guide optimization frameworks
  - Missing AST of application



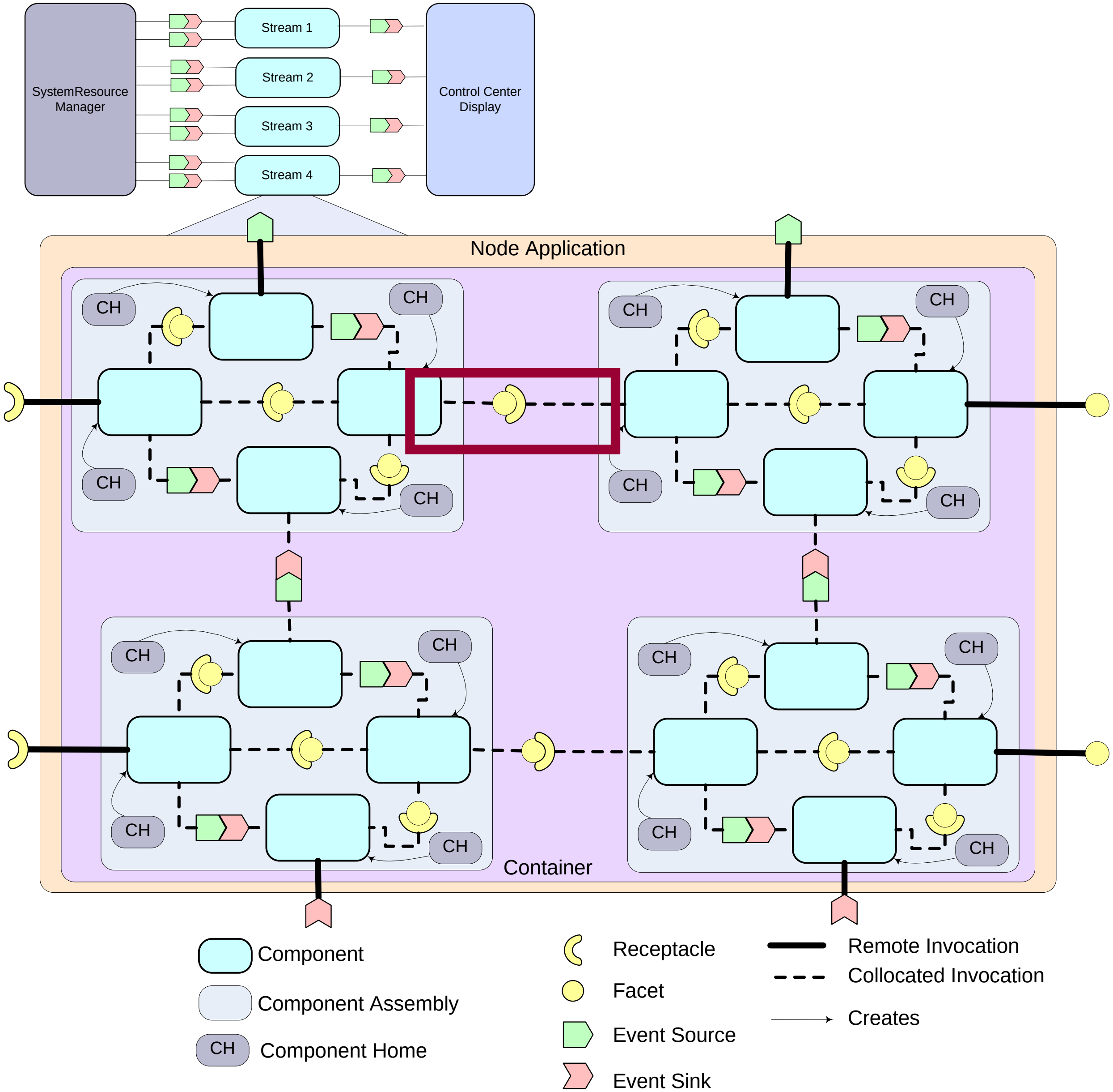
Application Specific Optimizations: What's missing?

- Lack of high-level notation to guide optimization frameworks
  - Missing AST of application
- Emphasis on detection at run-time (reflection)
  - Additional overhead in the fast path
  - Not suitable for all systems
- Not completely application transparent
  - Requires providing multiple implementations
- Optimization performed either
  - Too early, or too late



Application Specific Optimizations: Unresolved Challenges

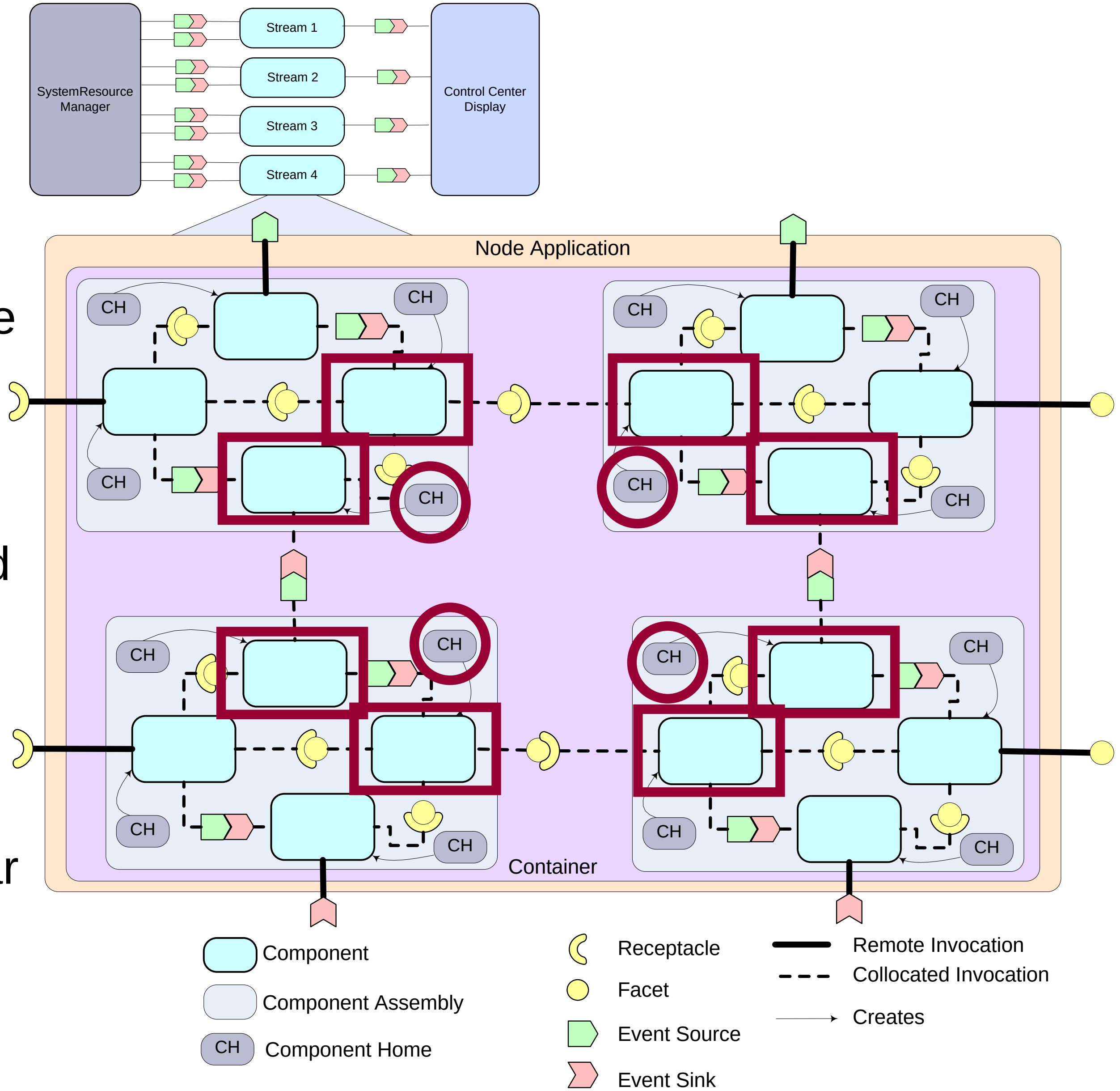
1. Lack of application context
   - Missed middleware optimization opportunities
     - E.g., every invocation performs check for locality
   - Optimization decisions relegated to run-time
   - Impossible for middleware (alone) to predict application usage
   - Settle for near-optimal solutions



Cannot be solved efficiently at middleware level alone!

Application Specific Optimizations: Unresolved Challenges

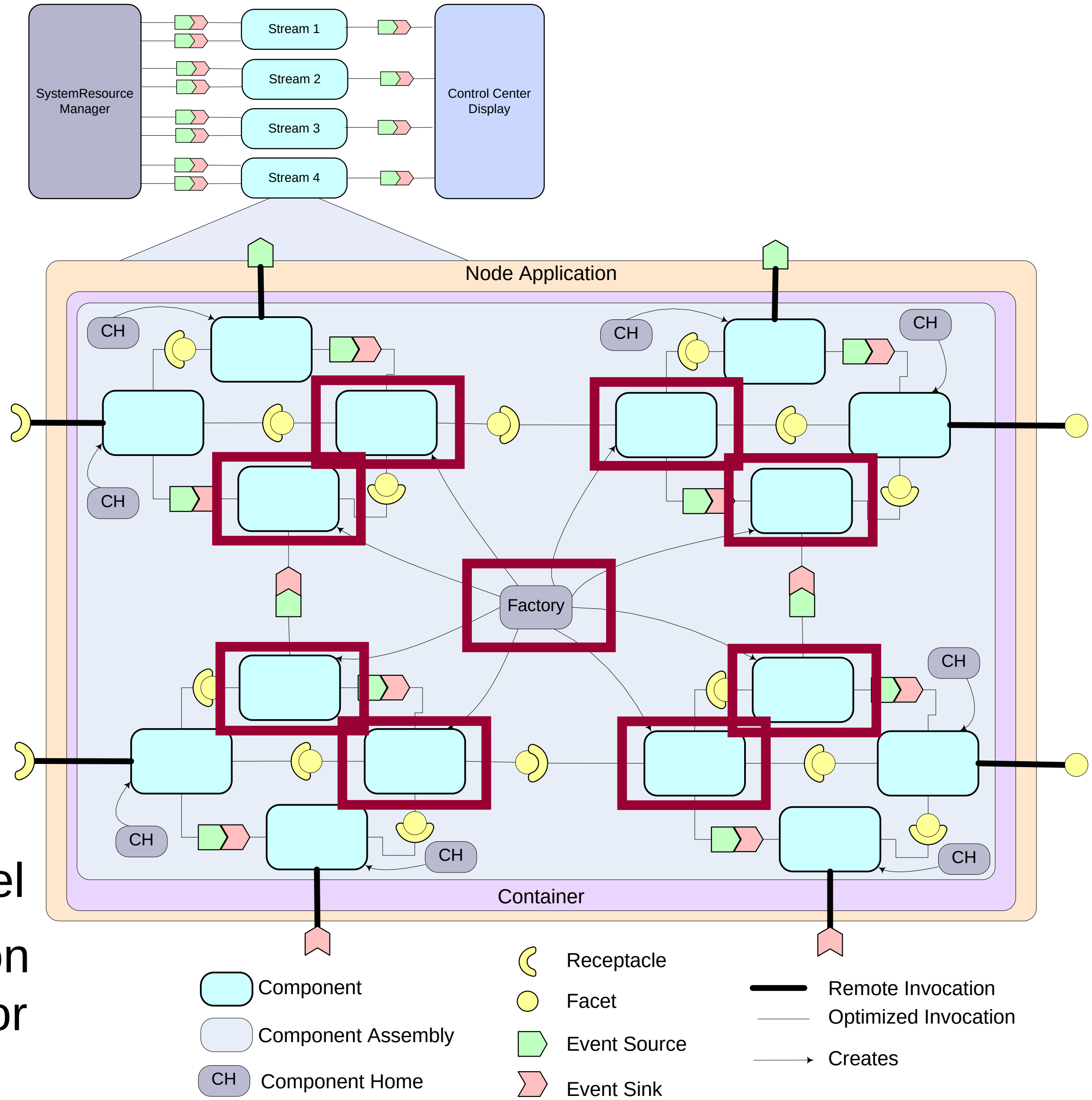
2. Overhead of platform mappings
    - Blind adherence to platform semantics
    - Inefficient middleware glue code generation per component
    - Example: Every component is created using a Factory Object
        - Overhead of external components similar to internal ones
3. Standard component models define only



Need optimization techniques to build large-scale component systems!

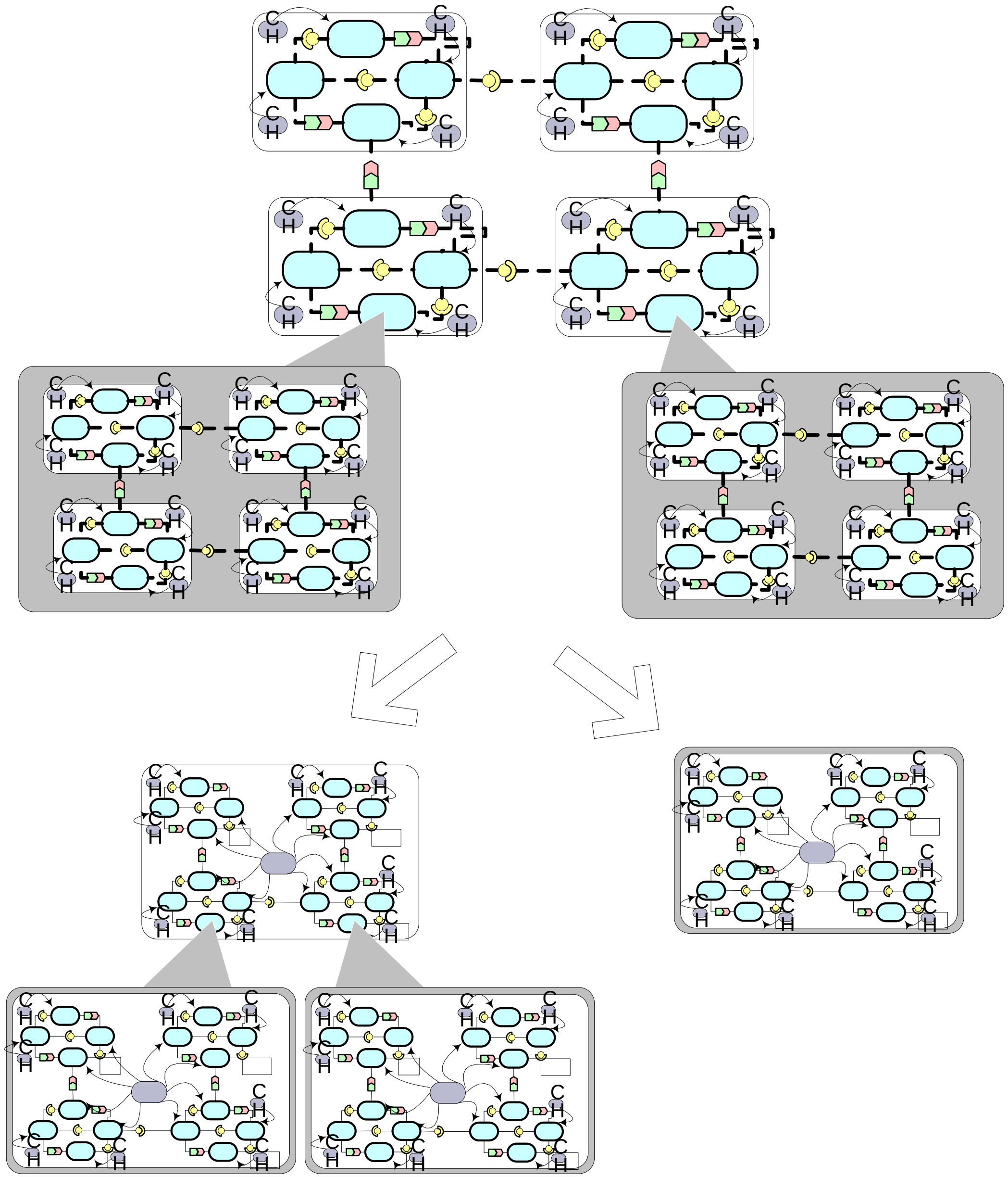
Proposed Approach: Supply Application Context w/Models

1. Use models to capture & derive application context
   - Explicit, e.g., sensor & monitor are collocated (user-specified)
   - Implicit, e.g., sensor & monitor deployed onto same node
   - Detect components internal to an assembly
2. Optimize platform mappings
   - Eliminate space overhead at system level
     - e.g., eliminate creation overhead of homes for internal components



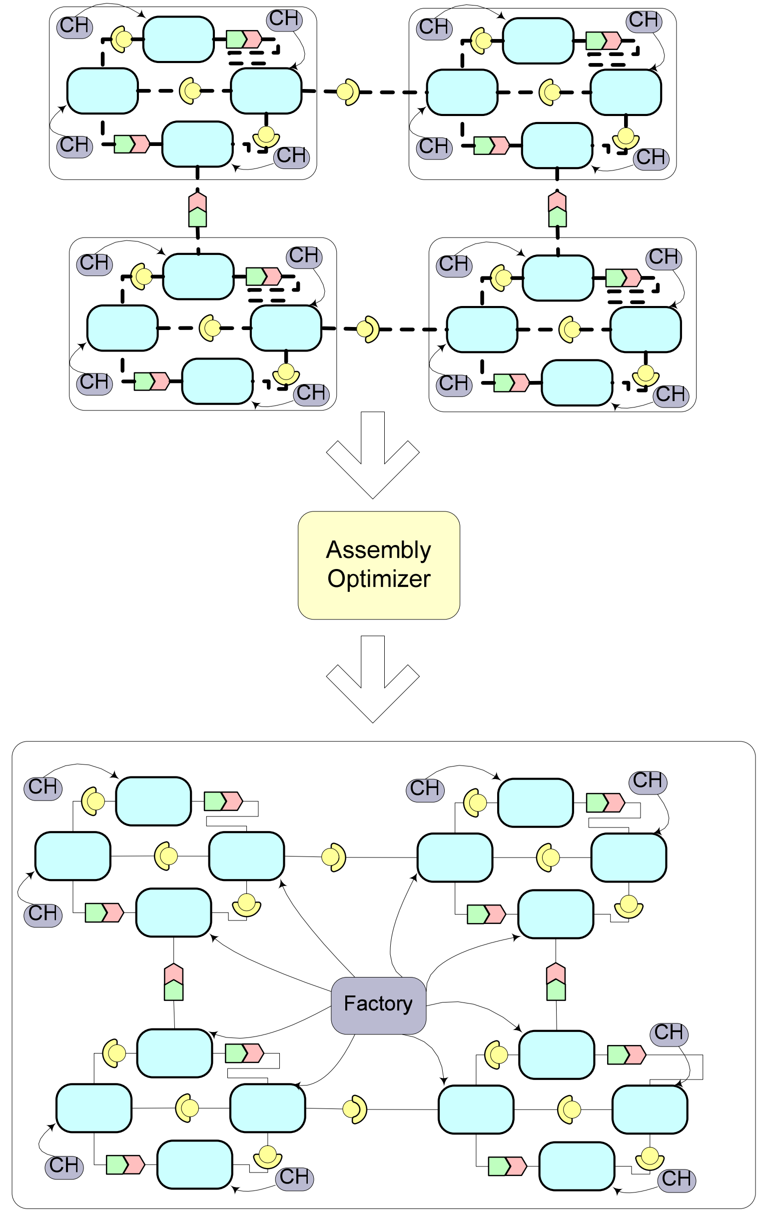
Proposed Approach: Physical Assembly Mapping

3. Devise mapping for *physical* component assembly
   - Exploit hierarchy of application structure to *fuse* (make a component internal) at multiple levels in hierarchy
   - Experimentally validate right depth of hierarchy to stop fusion
     - Too deep – Single giant blob
     - Too shallow – Potentially lower benefits



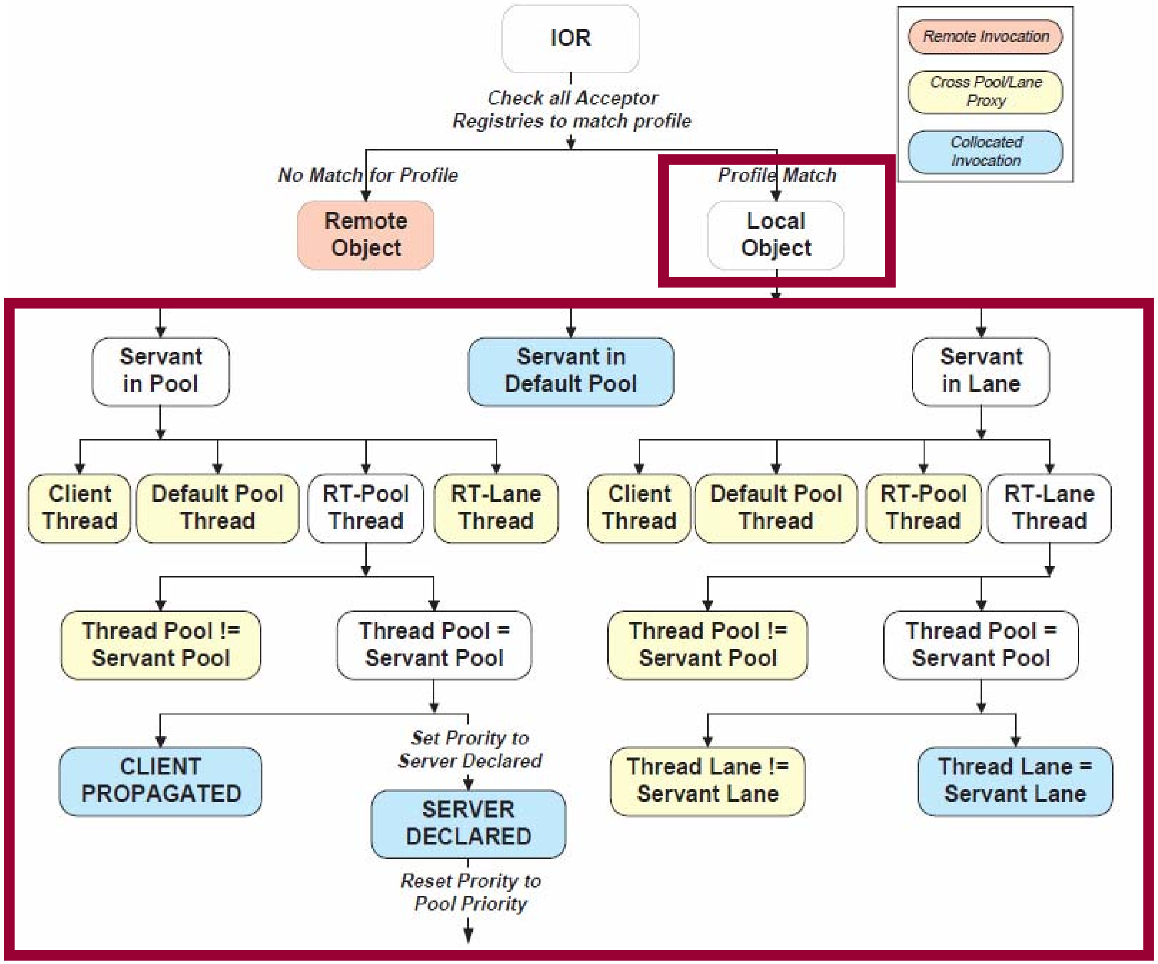
Proposed Approach: Evaluation Criteria

- Baseline for comparison
  - Performance & footprint (with vanilla CIAO)
    - Emergency Response System (30+ components)
    - ARMS GateTest scenarios (1,800+ components)
    - Scenario with & without inherent hierarchy
- Reduce static & dynamic footprint
  - *n = no. of internal components, x = total no. of components in the assembly*
  - Reduce no. of homes by (n-1)/x
  - Reduce no. of objects registered with POA by (n-1)/x



Proposed Approach: Evaluation Criteria

- Improve performance
  - *t = no. of interactions between components within an assembly*
  - Transform *t checked collocation calls* to *t unchecked calls*
- Eliminate mis-optimizations
  - Check incompatible POA policies
  - Incompatible invocation semantics (oneway vs. twoway)
- No changes to individual component implementations
  - Eliminate need for a local vs. remote version



- Customizable & application transparent

Concluding Remarks

- Component middleware is an emerging paradigm
  - Crucial to realizing the vision of Software Factories
- Problems with component middleware
  - Significant gaps in the development & integration toolchain
    - Potential to negate benefits of using component middleware
  - Direct application to DRE systems not always feasible
    - Might not meet the stringent QoS requirements of DRE systems
- Our research
  - Proposes to perform optimizations on component middleware that were previously infeasible
    - Exploit application context made available by MDE tool-chain

Tools can be downloaded from www.dre.vanderbilt.edu/CoSMIC/