Safe Specialization of the LwCCM Container for Simultaneous Provisioning of Multiple QoS

Akshay Dabholkar & Aniruddha Gokhale

Institute of Software Integrated Systems (ISIS),

Vanderbilt University, Nashville, TN, USA

Contact : aky@dre.vanderbilt.edu

Object Management Group Real-Time Workshop

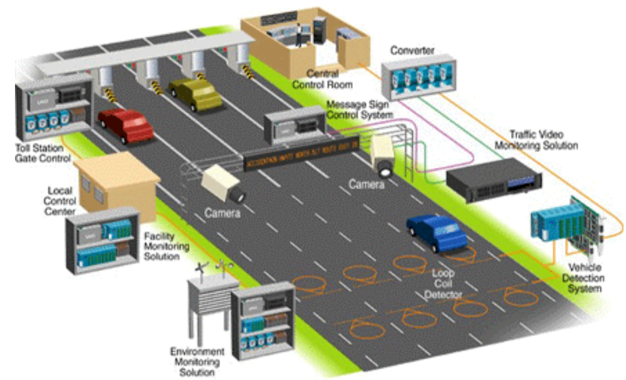
(OMG RTWS 2011)

March 22-24, 2011, Washington DC, USA

Research supported by NSF CAREER CNS# 0845789, Vanderbilt Discovery

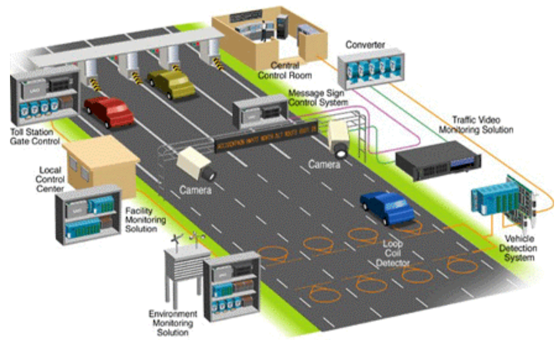
Context: Distributed Real-time Embedded (DRE) Systems

- Large-scale, systems-of-systems
- Operation in resource-constrained environments
  - Memory-constrained
  - Low processor speeds
  - Low power availability
- Stringent and simultaneous QoS demands
  - High availability
  - Timeliness
  - Efficient resource utilization
- Examples
  - Intelligent Transportation Systems (ITS)
  - Inventory Tracking Systems
  - NASA's Magnetospheric Multi-scale mission (MMS)



(Images courtesy: Google)

Overcoming Variability in DRE System Domain Concerns



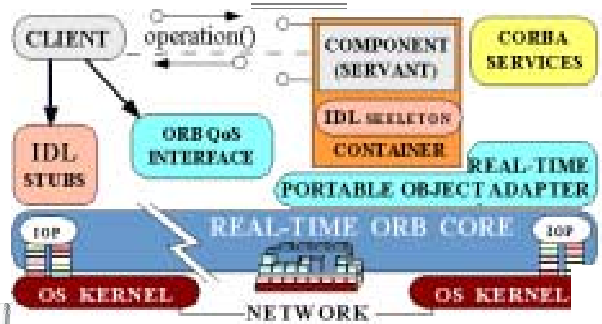
Direct Application

Generalization

Maximize Throughput

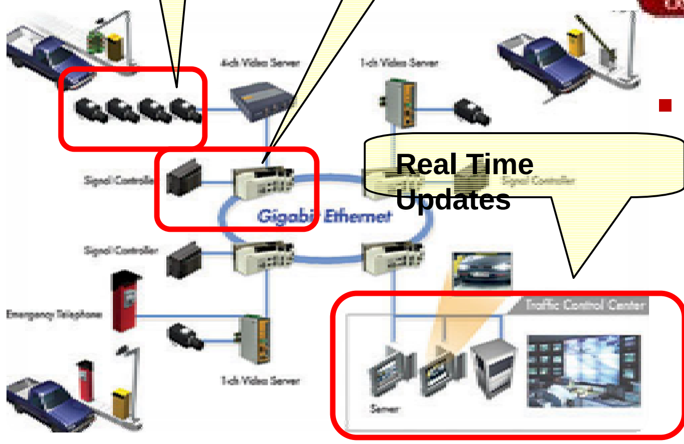
Resource Constrained

Intelligent Transportation System



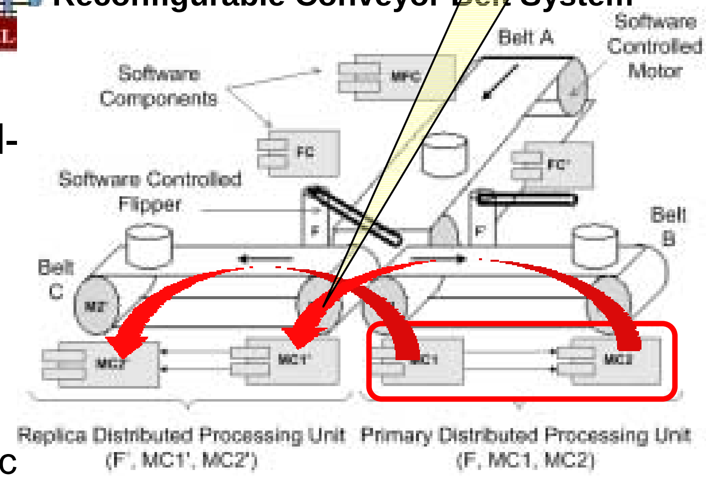
Group Failover Semantics

Reconfigurable Conveyor Belt System



Real Time Updates

- Development via General-purpose Middleware
  - Feature-rich
  - Satisfies wide range of DRE systems
  - Uses extensible frameworks
  - CORBA, .NET, J2EE, etc



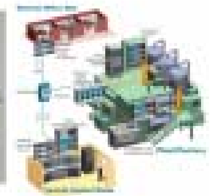
Performance Requirements

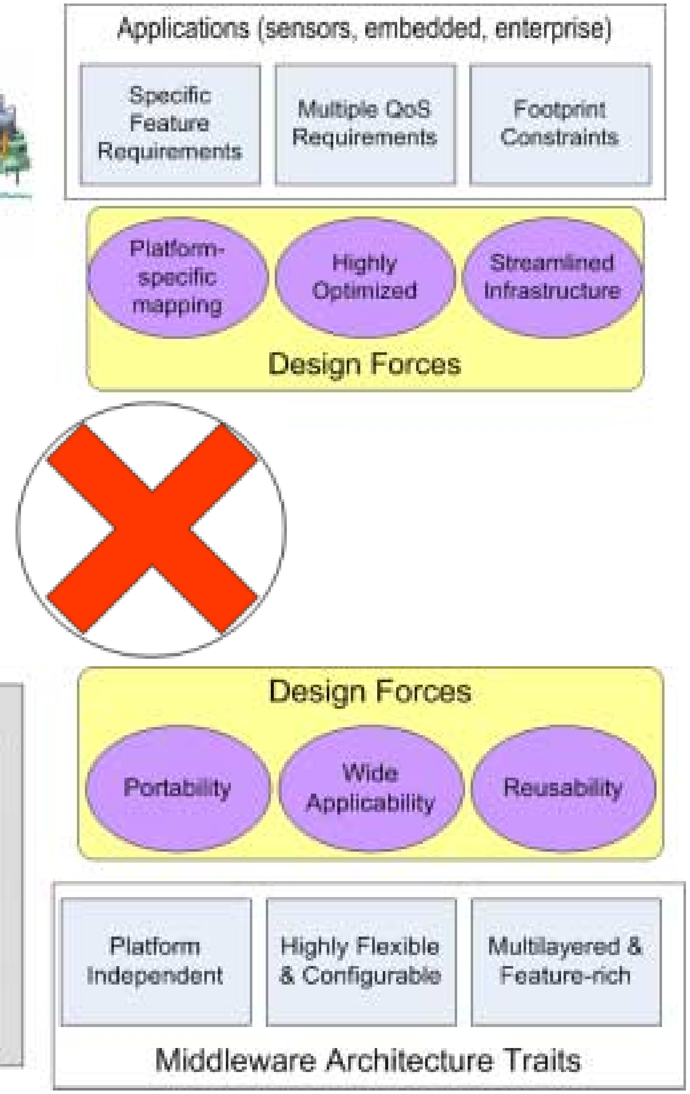
Dependability Requirements

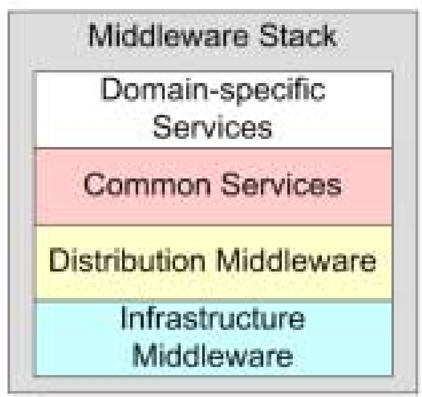
Impediments to Using General-purpose Middleware

- General-purpose middleware supports a wide range of DRE systems
- However, individual DRE systems have streamlined requirements
- **Antagonistic** Design Forces
  - ***Excessive features*** due to wide applicability
  - Unnecessary ***overhead*** due to high flexibility and configurability
  - Moreover, focus is on ***horizontal decomposition*** into ***layers***
  - Incurs ***time*** and ***space*** *overhead* due to rigid layered processing
  - Application concerns are ***tangled*** across middleware modularization boundaries

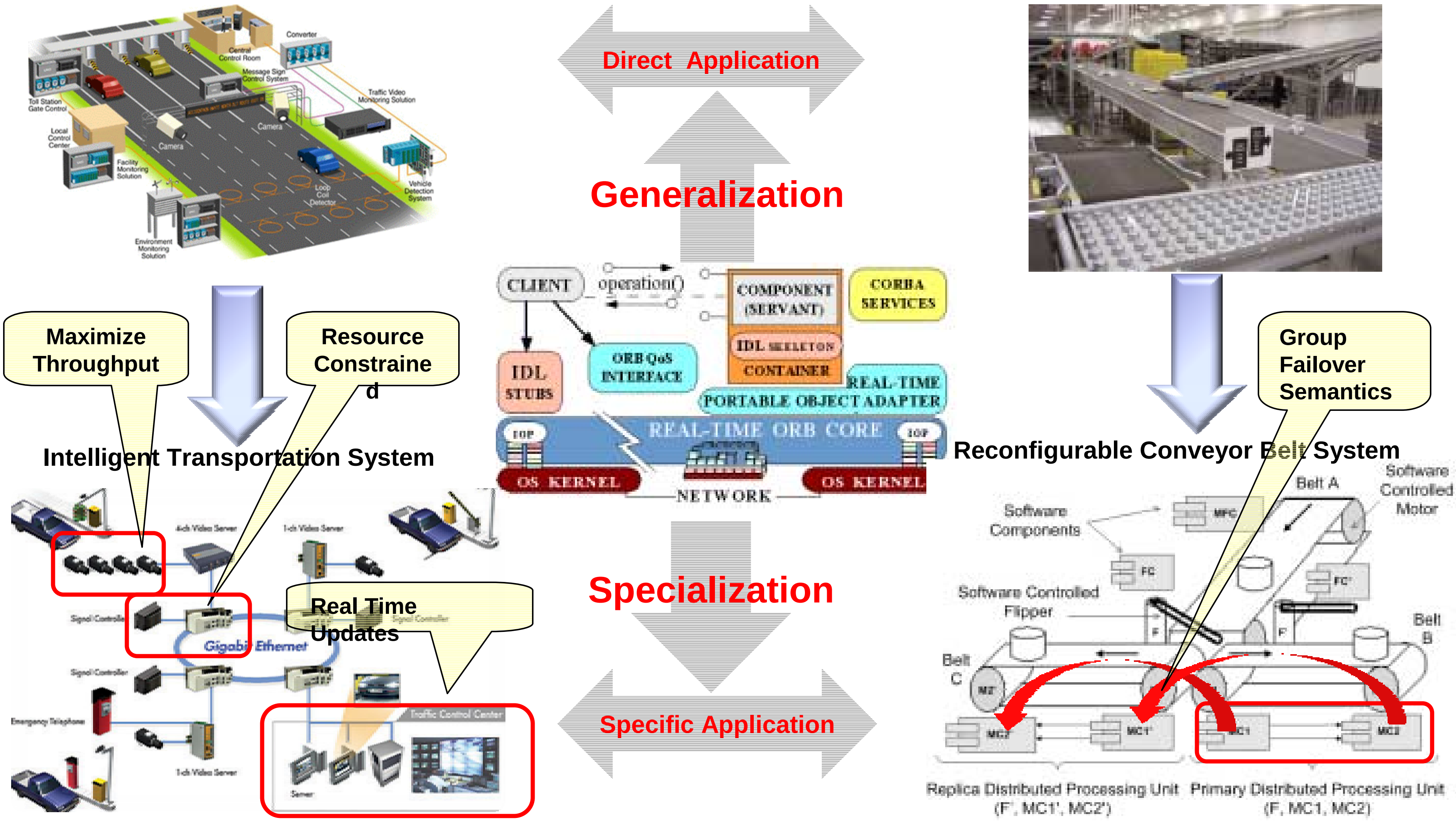
Need Specialization of General-purpose Middleware







Preferred Approach to Overcome Variability in DRE Systems

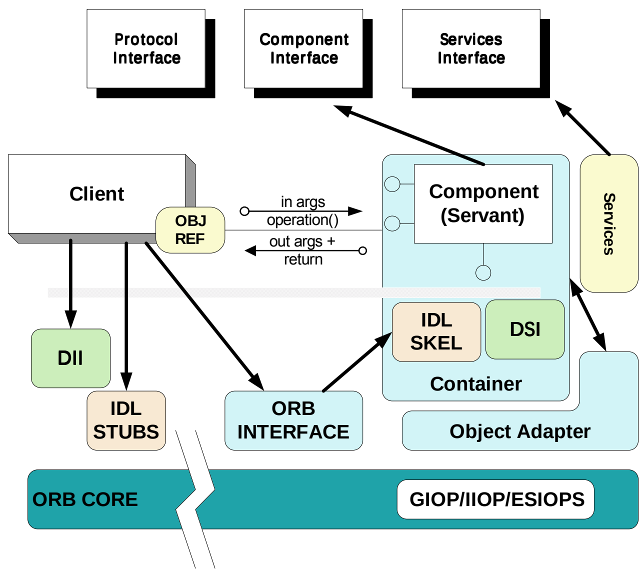


Performance Requirements

Dependability Requirements

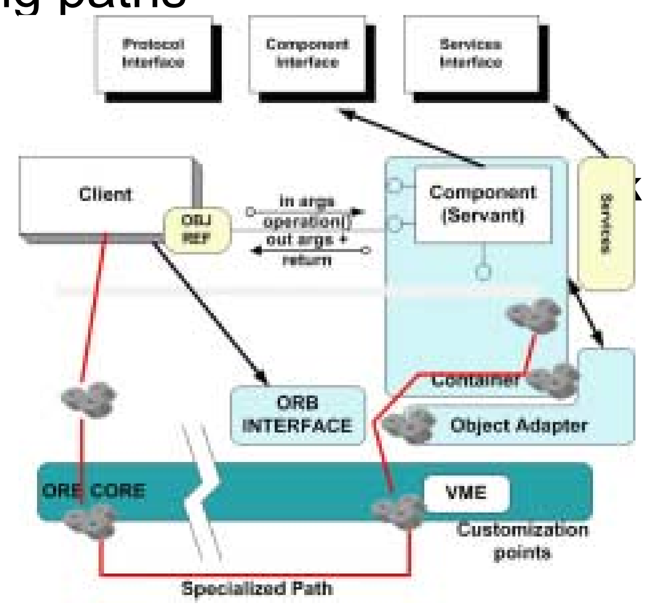
What is Middleware Specialization?

- Resolves the tension between ***Generality*** *and* ***Specificity***
- Creates specialized ***forms*** of middleware for each system by
  - ***Pruning*** away unnecessary features based on system concerns
  - ***Augmenting*** application-specificity by embedding their semantics
  - ***Optimizing*** performance by moving away from the rigid layered processing by creating specialized processing paths



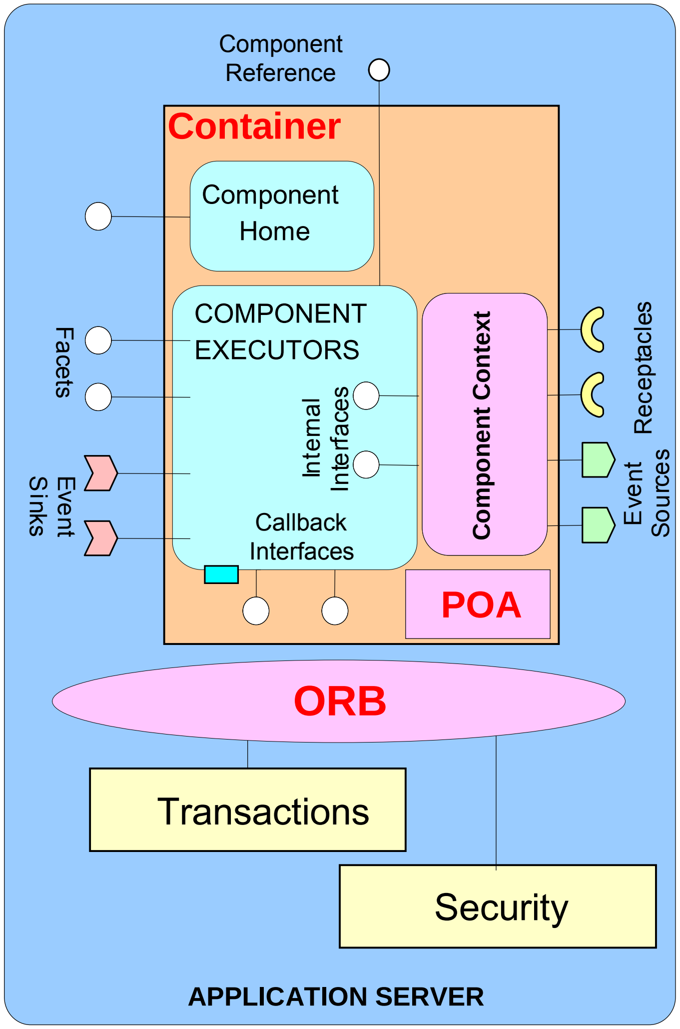
Standards-based, General-purpose, Layered Middleware Stack

Specialization



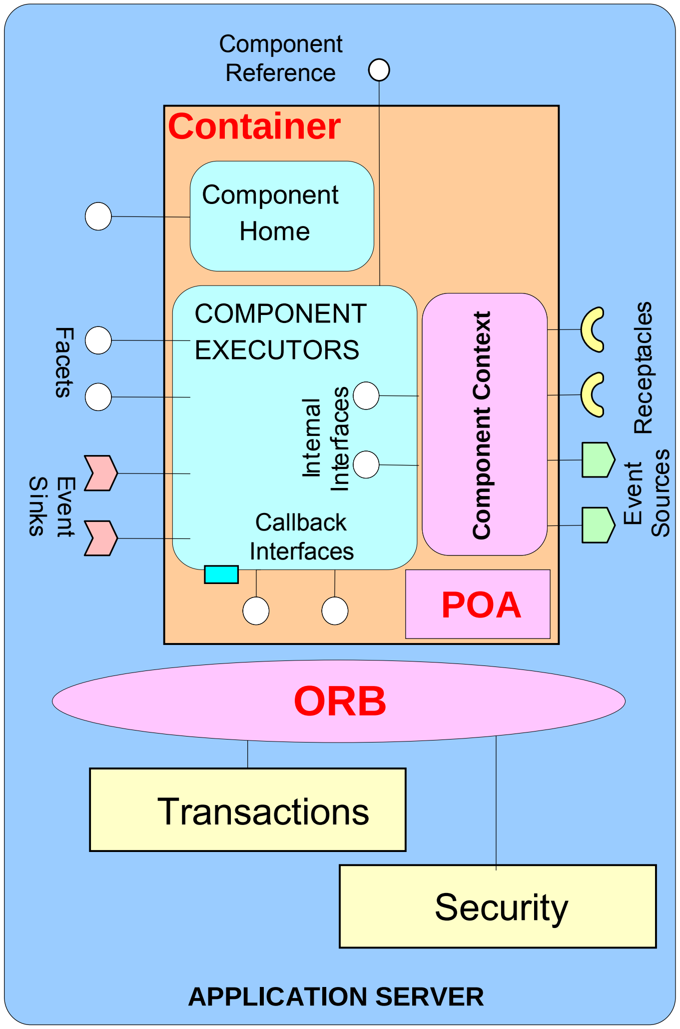
Specialized Middleware Stack

State of Art: Component Middleware QoS Provisioning



- Focus is on provisioning only one QoS at a time in the middleware container results in -
  - Too ***proprietary, ad-hoc*** and ***custom*** implementations
  - ***Redundant, repetitive*** efforts requiring ***reinvention*** of existing solutions
  - Expensive to ***develop*** and ***maintain***
  - Hard to ***evolve*** over application lifetime
  - Difficult to ***test*** for correctness and QoS
  - Lack ***openness*** and ***interoperability***

State of Art: Component Middleware QoS Provisioning



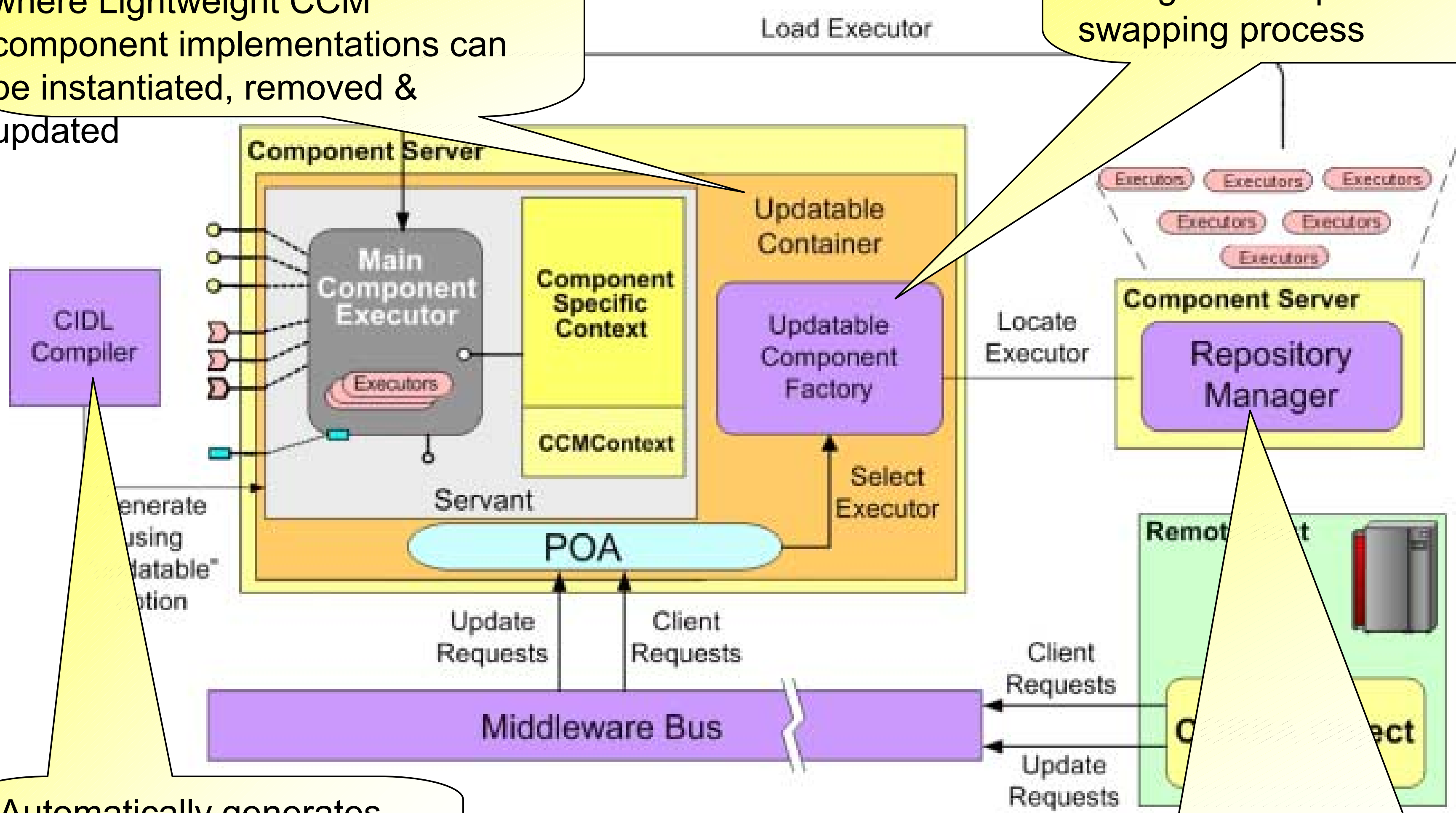
- Container architectures lack the flexibility necessary for -
  - Simultaneous QoS provisioning and configuration while minimizing footprint and runtime overhead
  - Ensuring the correct configuration and functioning of multiple QoS mechanisms in unison
- Resulting Consequences -
  - Performance degradation and increase in memory footprint if required to combine QoS mechanisms
  - Difficult to program and configure multiple QoS mechanisms

Case Study 1: Provisioning Dynamic Component Swapping

Extends SessionContainer to provide an execution environment where Lightweight CCM component implementations can be instantiated, removed & updated

SwapCIAO Architecture

Dynamically opens & loads component implementations during the component swapping process



Automatically generates glue code for updating component

Stores component implementations that are retrieved by the updatable component factory at

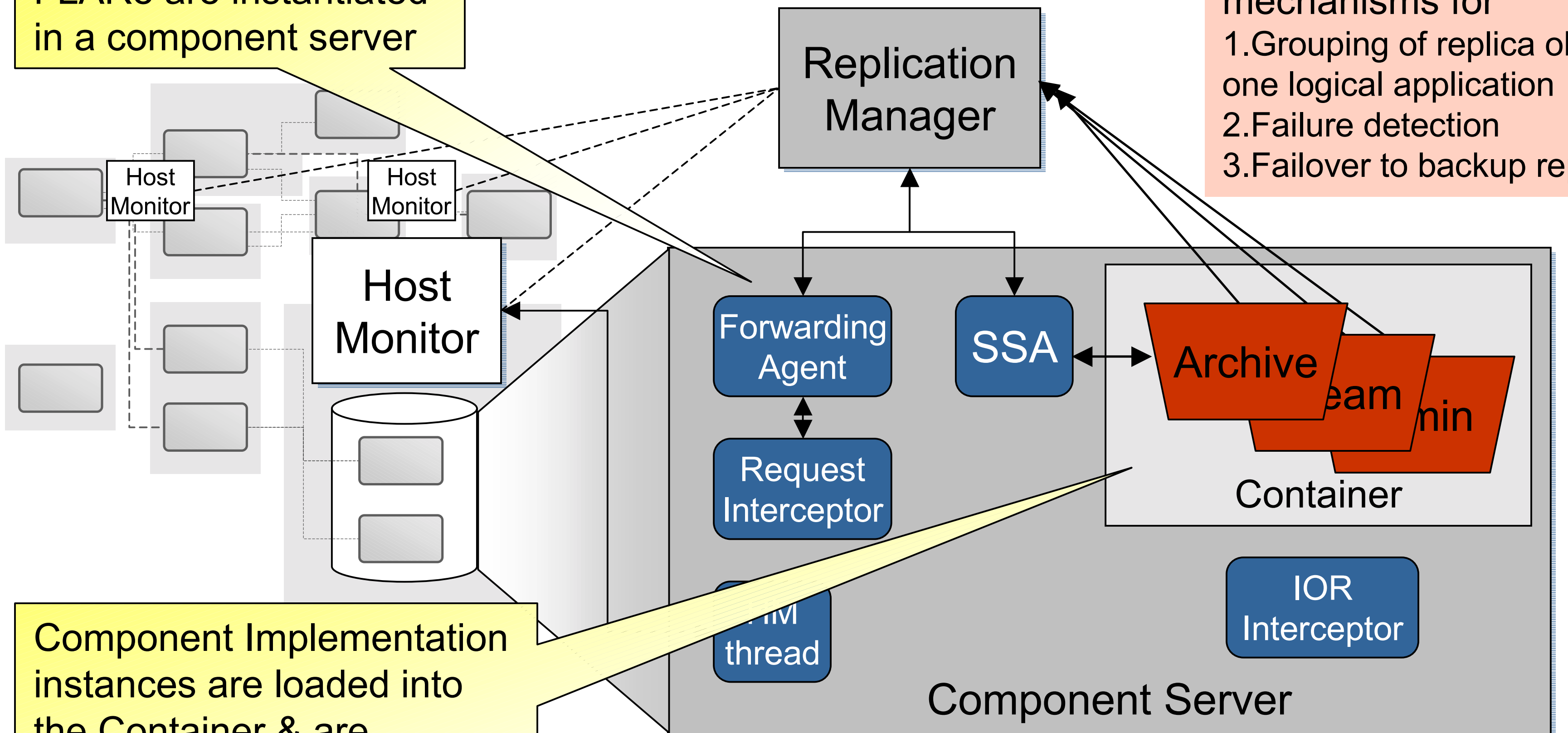
Case Study 2: Provisioning Reliability QoS

All client & server side entities related to FLARe are instantiated in a component server

CORFU Architecture

FLARe is a lightweight approach based on passive replication that provides mechanisms for

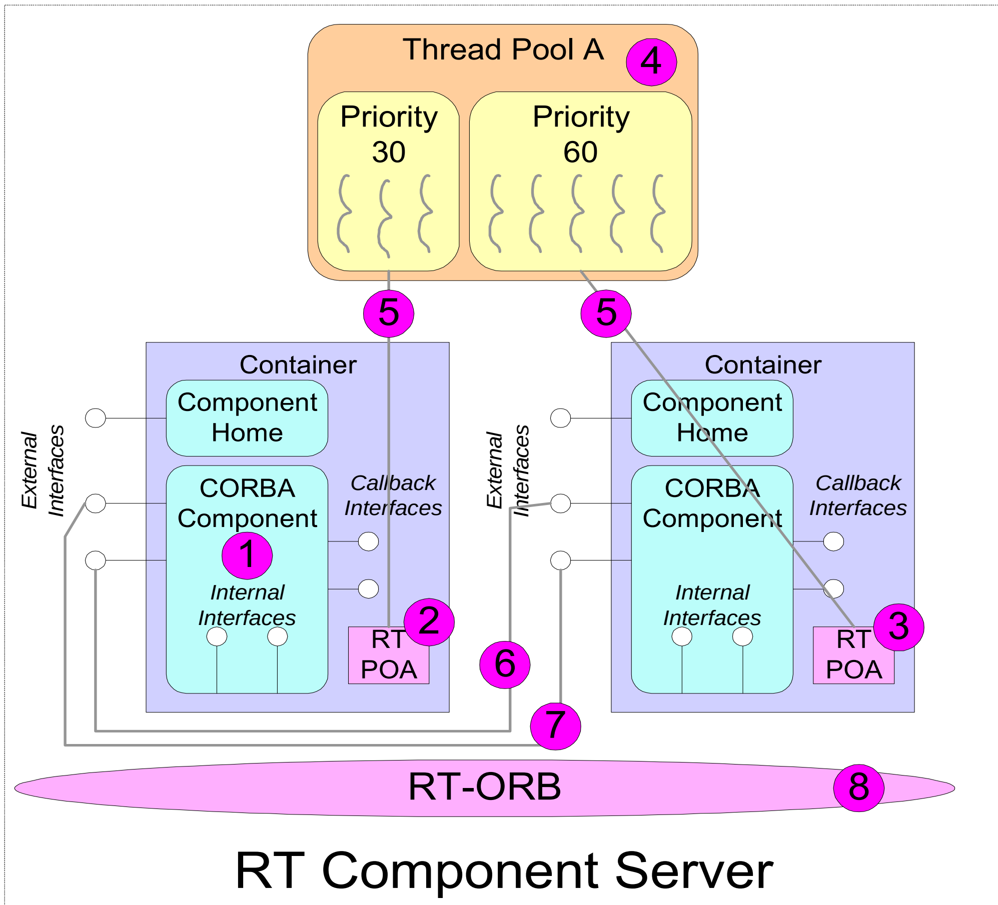
1. Grouping of replica objects as one logical application
2. Failure detection
3. Failover to backup replica



Component Implementation instances are loaded into the Container & are automatically integrated into FLARe

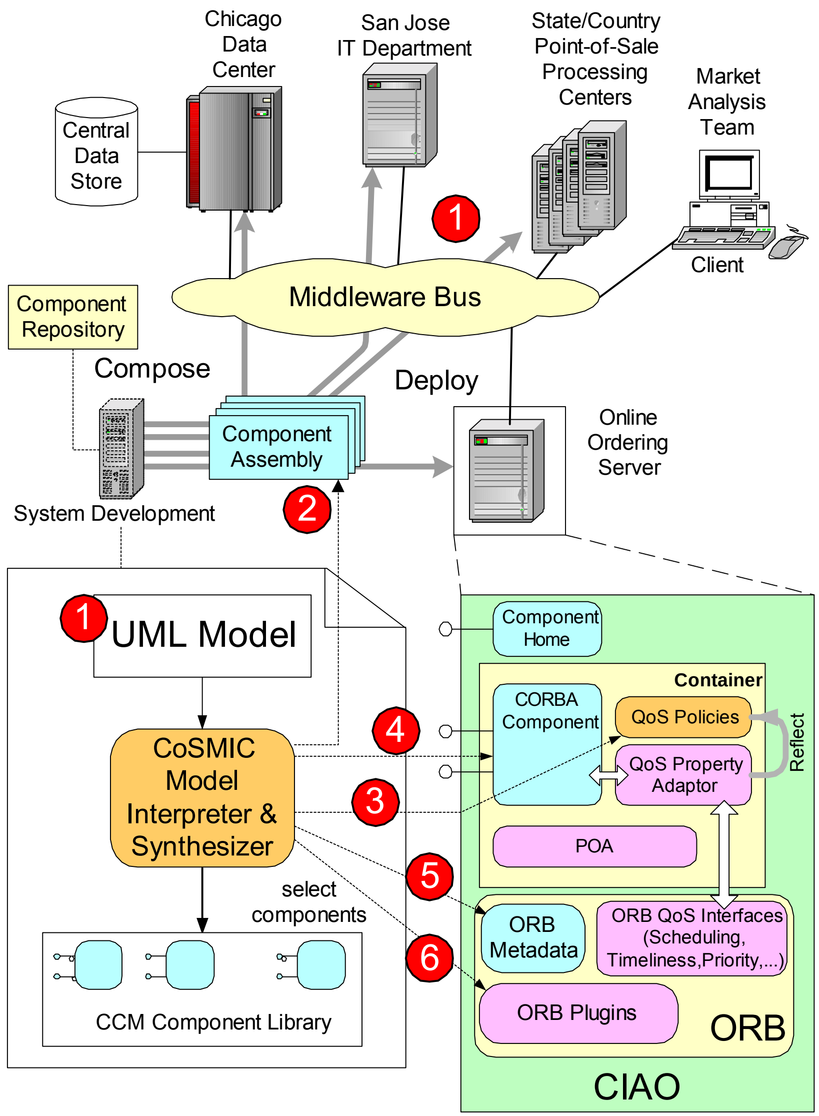
Case Study 3: Provisioning Real-Time QoS

Extends CIAO's meta-model to make RT policies an integral part of CCM



1. Component default priority model
2. Override component priority model
3. Priority level of a component instance
4. Defining thread pools
5. Associate thread pools with containers
6. Specify queuing policies
7. Specify pre-connections and private connections, banded-connections
8. Configure ORB components
   - Custom protocols
   - Priority mappings

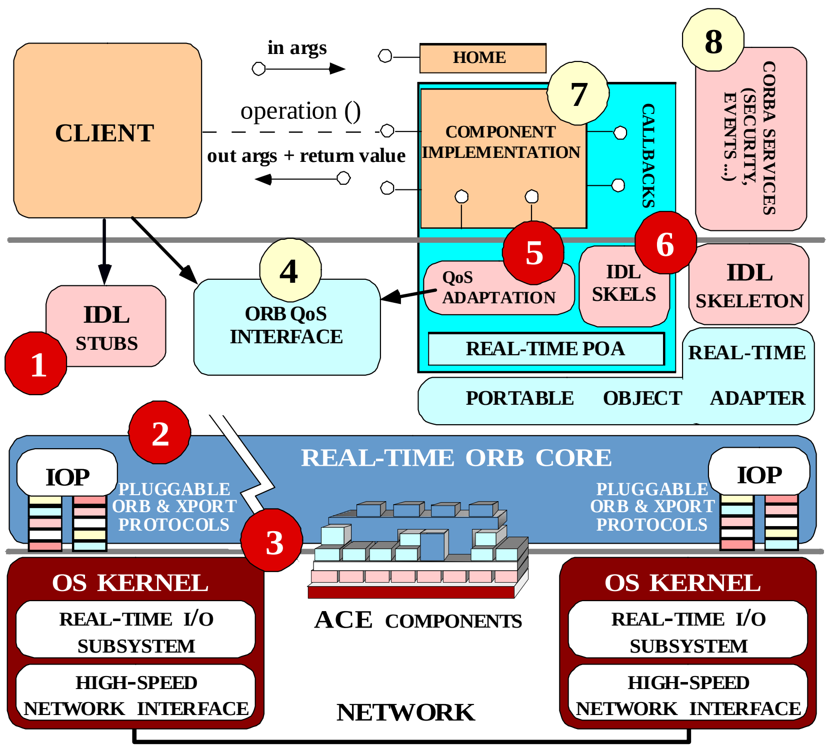
Shortcomings of current CCM QoS implementations



- Integrating individual QoS mechanisms directly within the container doesn't automatically ensure multiple QoS support e.g., RTCCM ≠ CCM + RTCORBA
- CCM only enforces the specified QoS policies, it does not ensure they are correctly composed together
- Doesn't take QoS4CCM specification into account for the runtime QoS management
- Does not enable selective composability of the necessary QoS mechanisms

Efficient Design and Safe Specialization of the CCM Container for provisioning multiple QoS

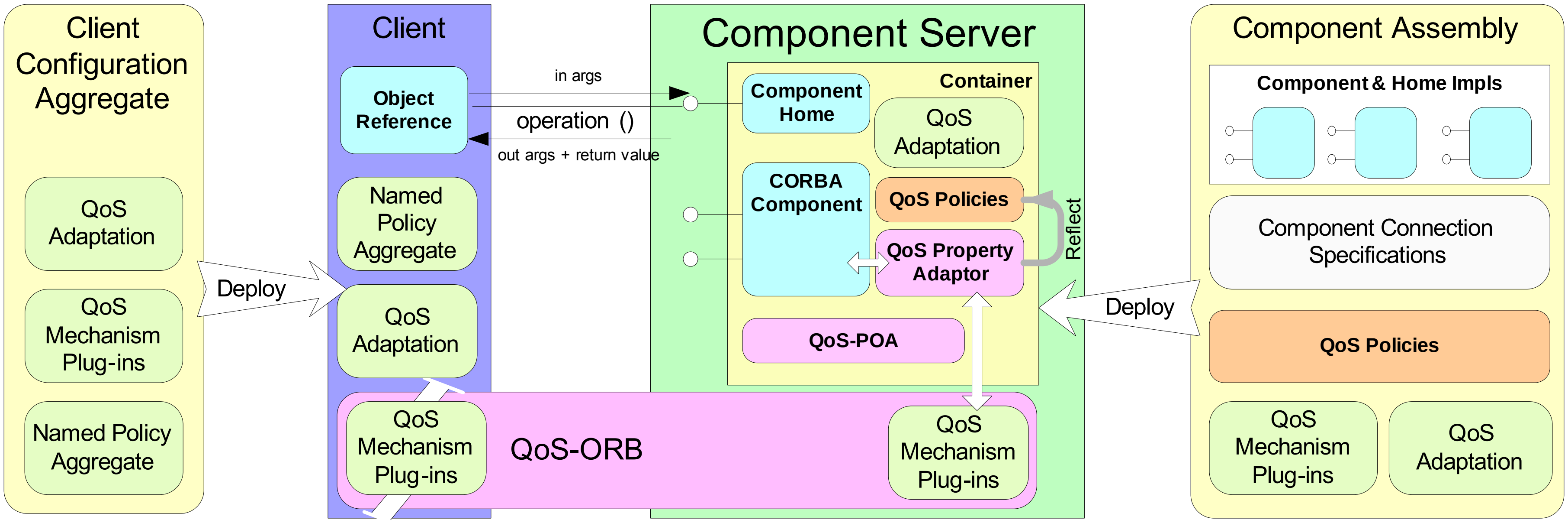
Current Status of CIAO



1) COLLOCATION STUBS
2) COLLOCATION XPORT SELECTION
3) SHARED MEMORY XPORT
4) ORB LEVEL QOS INTERFACE
5) COMPONENT QOS ADAPTATION
6) DYNAMIC LINKING OF COMPONENT SERVANTS
7) DYNAMIC CONFIGURATION OF COMPONENTS
8) INTEGRATION OF SERVICES

- *CIAO* : A LwCCM implementation based on *the ACE ORB* (TAO)
- Extends the (component and assembly) descriptors for configuring RT policies
- Applies reflective middleware techniques to support other non-functional aspects with CCM metadata
  - Bandwidth reservation
  - Memory management
  - Transport selection

Work In Progress: A Generic Pluggable Container Architecture



- Pluggable and highly Composable Container Infrastructure and Framework (IaFe) that provides:
  1. A ***Composition Framework*** for correctly plugging in the required QoS mechanisms within the Containers based on deployment and QoS policies
  2. A ***Scheduling Framework*** for the correct order of the instantiation of multiple QoS mechanisms
  3. A ***Runtime Framework*** for enabling dynamic configuration and adaptation of QoS mechanisms

Questions?

Backup Slides

Provisioning Other QoS