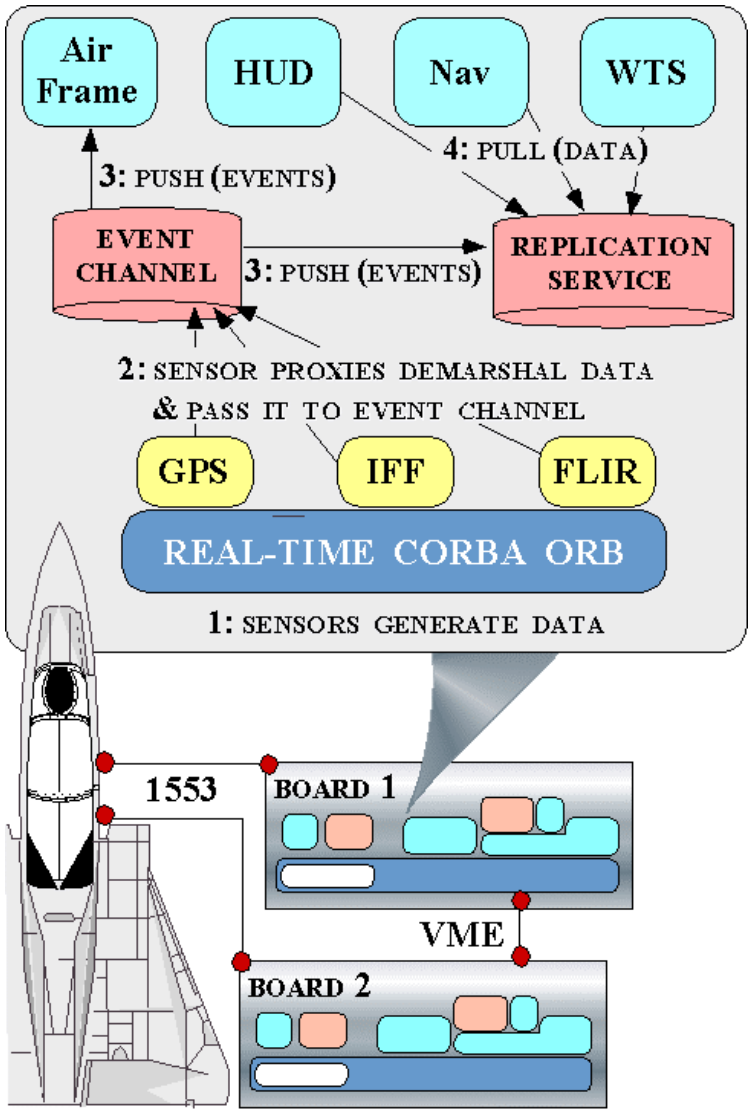
A QoS-aware CORBA Component Model for Distributed Real-time and Embedded System Development

Nanbor Wang and Chris Gill
{nanbor,cdgill}@cse.wustl.edu
Department of Computer Science and Engineering
Washington University in St. Louis

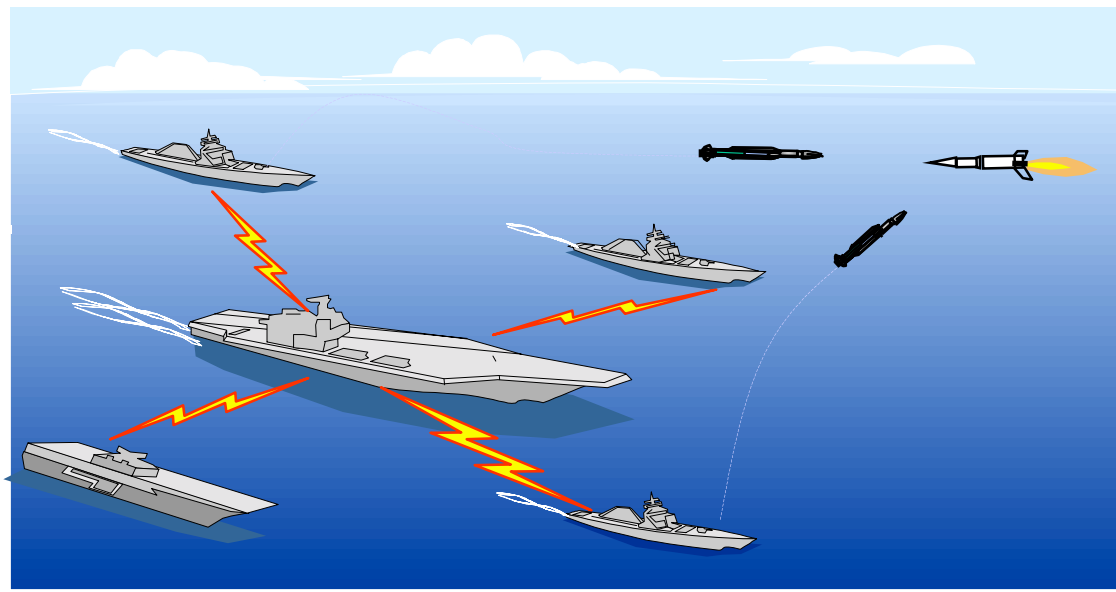
This research is sponsored by DARPA PCES program under Contract to Boeing (F33615-00-C-3048) & Washington University in St.Louis (F33615-00-C-1697), partially in collaboration with BBN Technology.

July 7, 2003

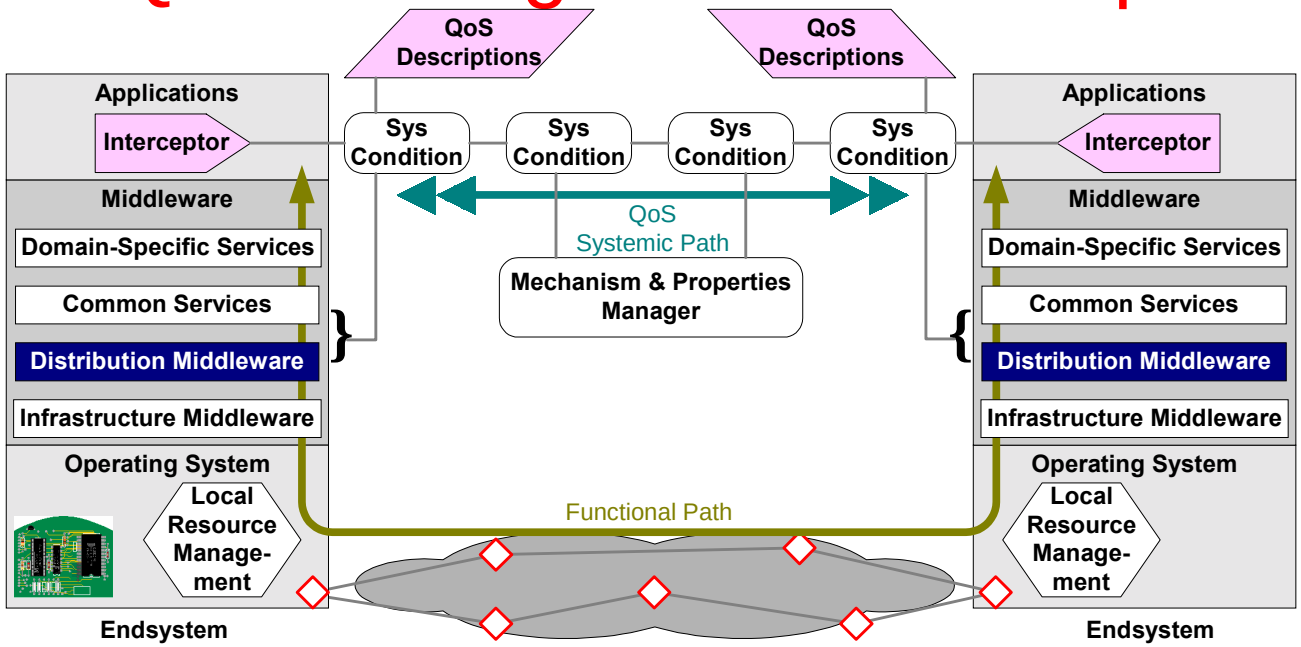
Characteristics of DRE Applications



- Representative hard real-time applications
  - Avionic mission/control systems
  - Theater missile defense
  - Command and control
- QoS resources which must be managed
  - Computation resources
  - Communication resources
  - Power resources

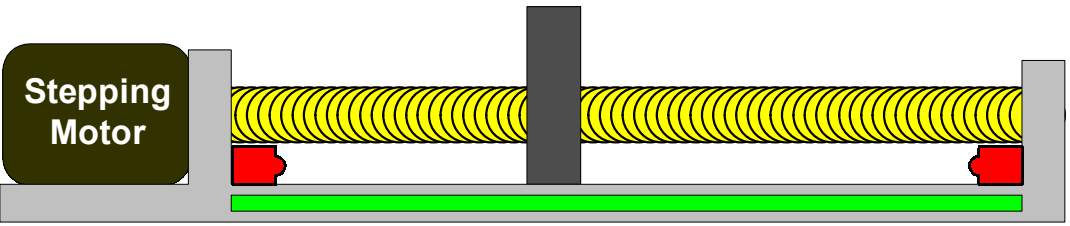


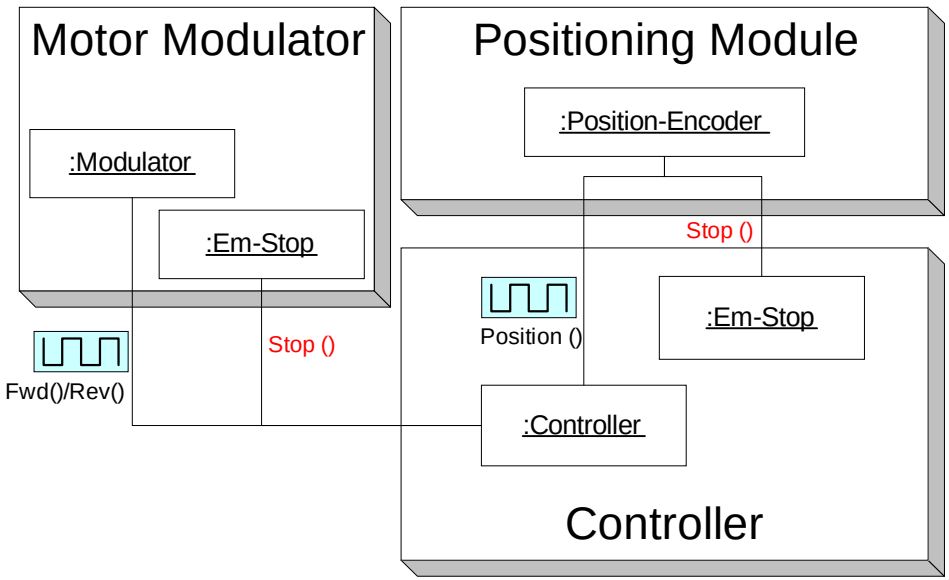
QoS Management Techniques



- Static QoS management
  - Done at system design time
  - Resources are provisioned before system runs
- Dynamic QoS management
  - Done at system runtime
  - Adapts to changing environmental conditions

A Motivating Real-Time Application

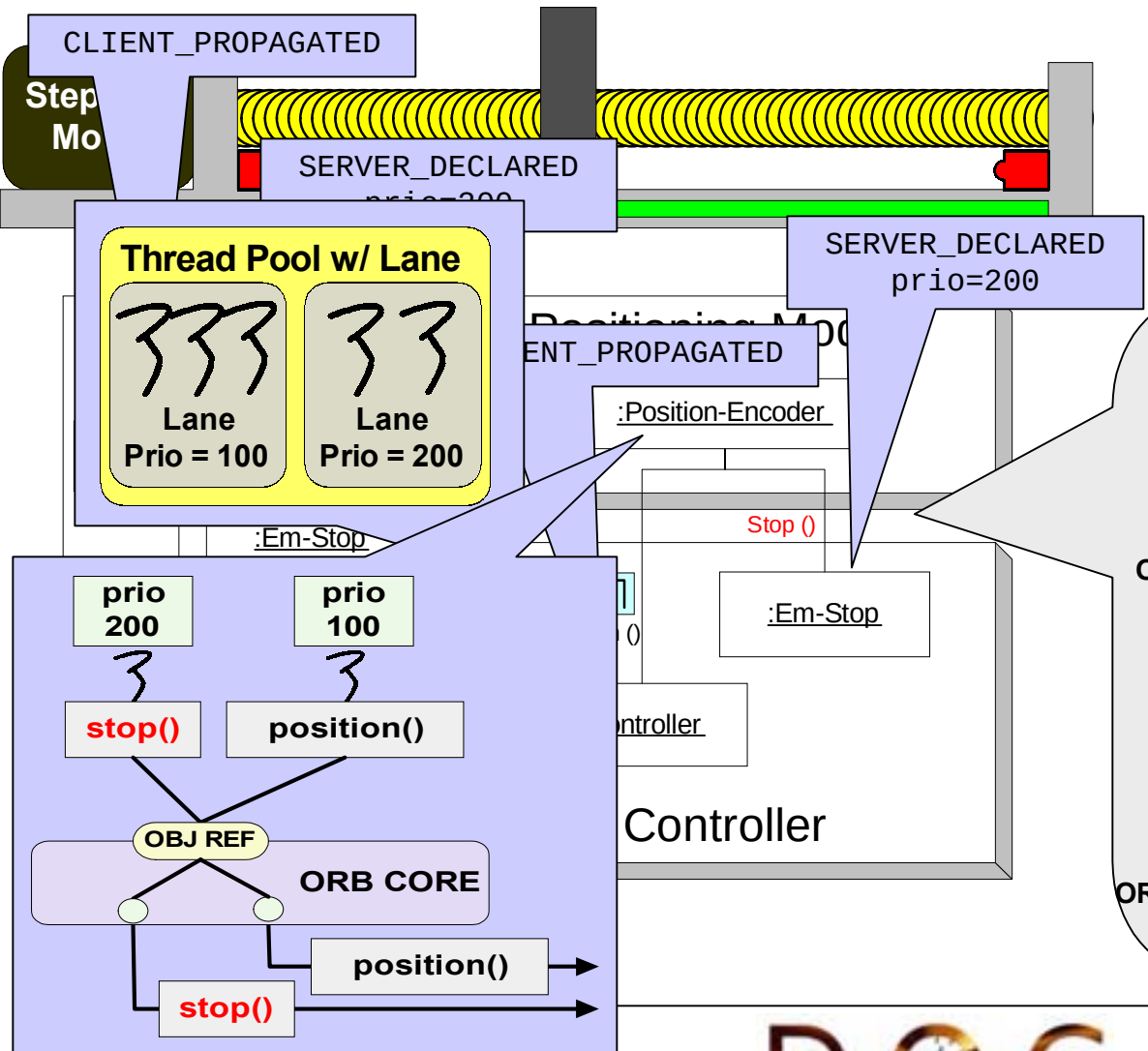


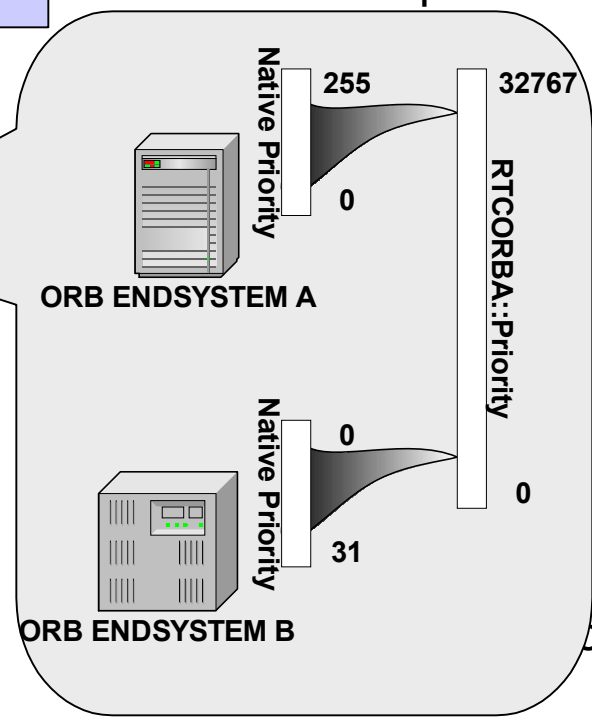


- An one-axis robot arm controller application
  - 3 separate processes connected via ethernet
- Motor Modulator
  - Advances stepping motor fixed angle for every Fwd/Rev command
  - Activate mechanical brake when “stop”
- Controller
  - Updates current location
  - Accelerates and decelerates motor
  - Stops the motor
- Positioning Module
  - Sends differential positioning information
  - Programming proximity limits

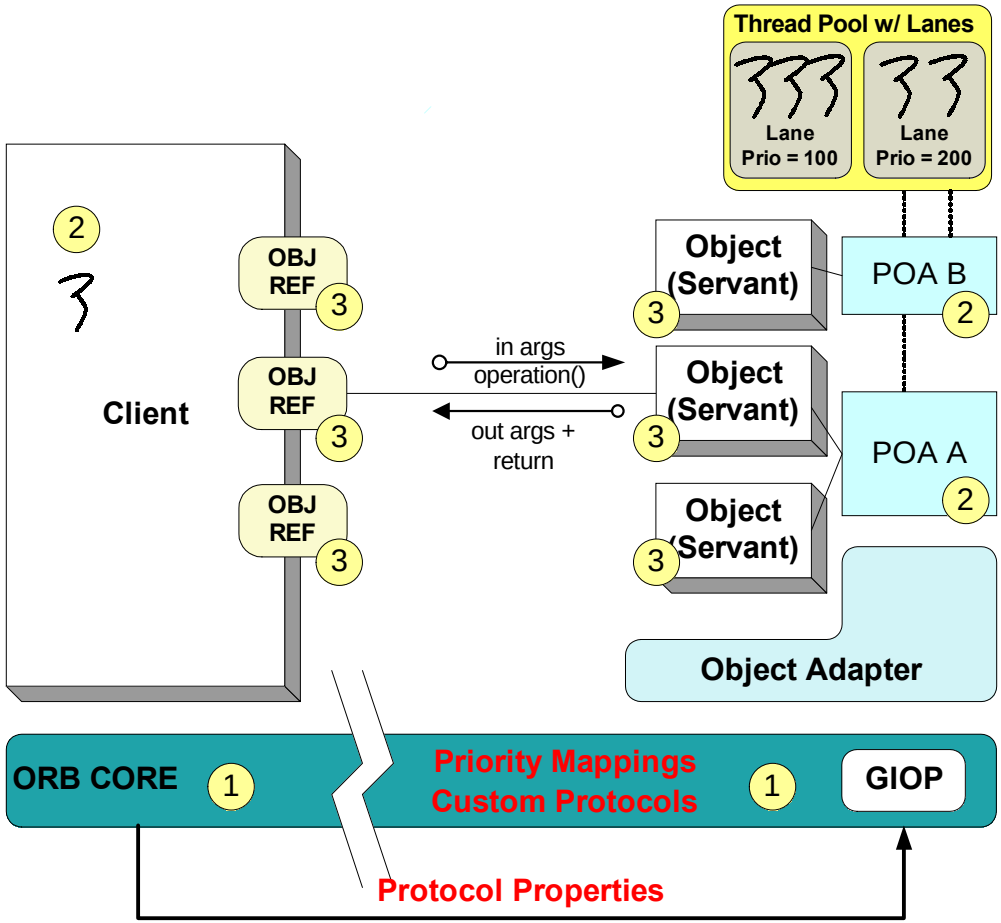
Types of Real-time Resources

- Ensure timely response of "important" tasks
  - Define unified view of "importance"
    - Portable Priority Mapping
- Handle "important" tasks





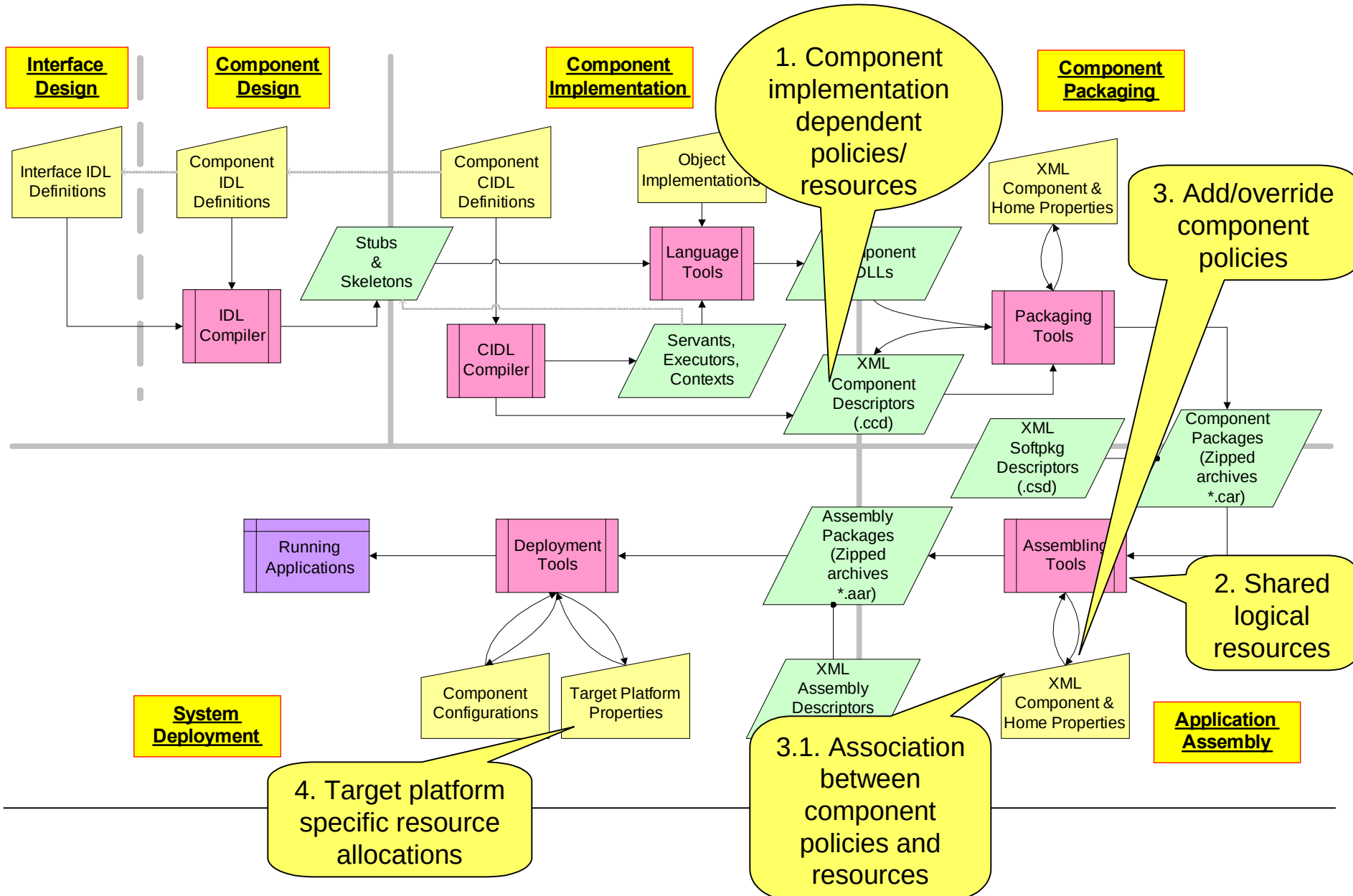
Review - RT Policies/Resources



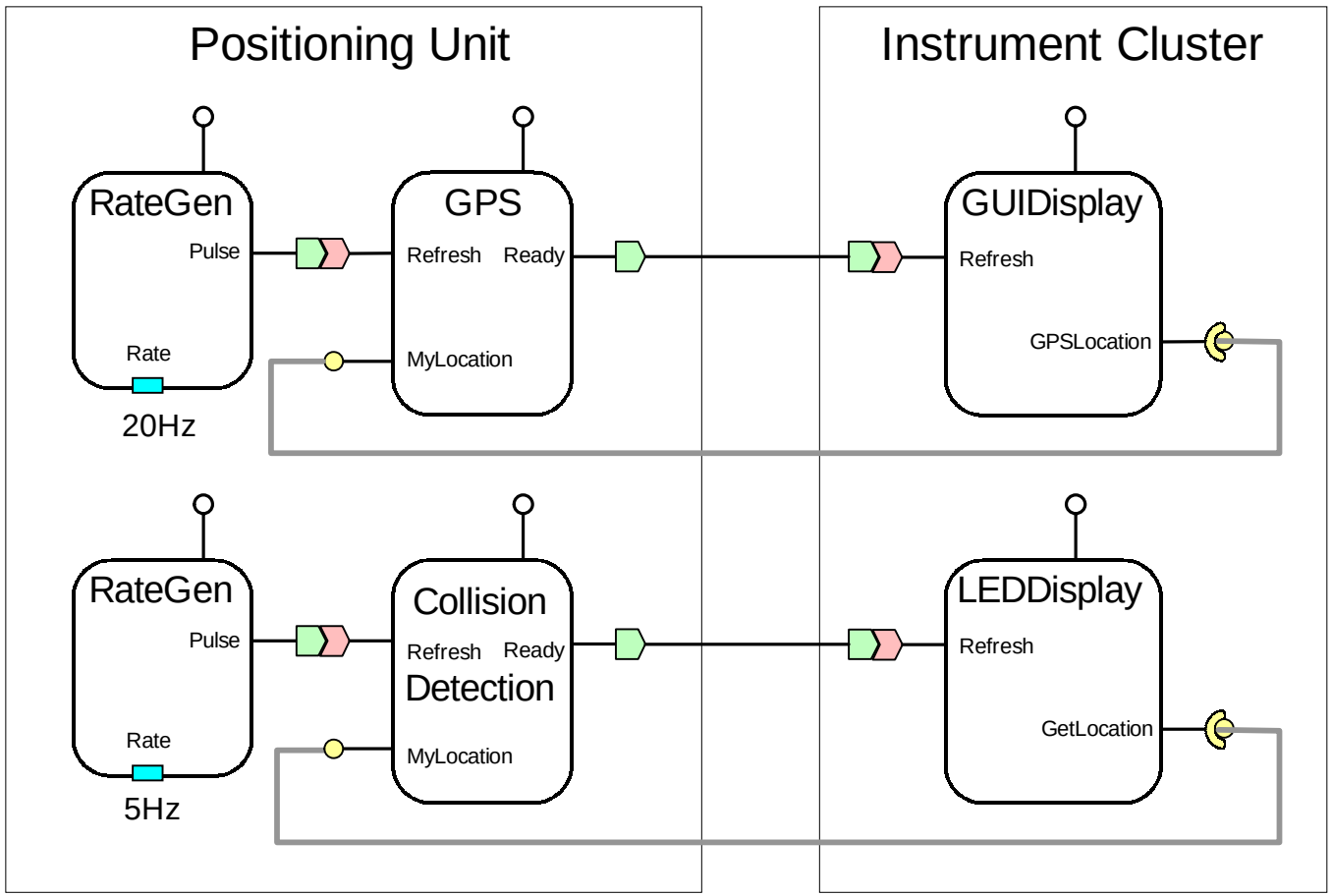
- RT policies can associate with objects of various granularities
  - Client-side
    1. ORB level
    2. Thread level (RTCurrent)
    3. Object level
  - Server-side
    1. ORB level
    2. POA level
    3. Object level
- Shared RT resources
  - Shared by several POAs, objects (Thread Pooling)
  - ORB: certain protocol policies
  - Priority-mapping
- Requires end-to-end enforcement
- Results tightly coupled code

Real-time CORBA leverages the CORBA Messaging QoS Policy framework

Insertion Points for RT Systemic Properties into CCM



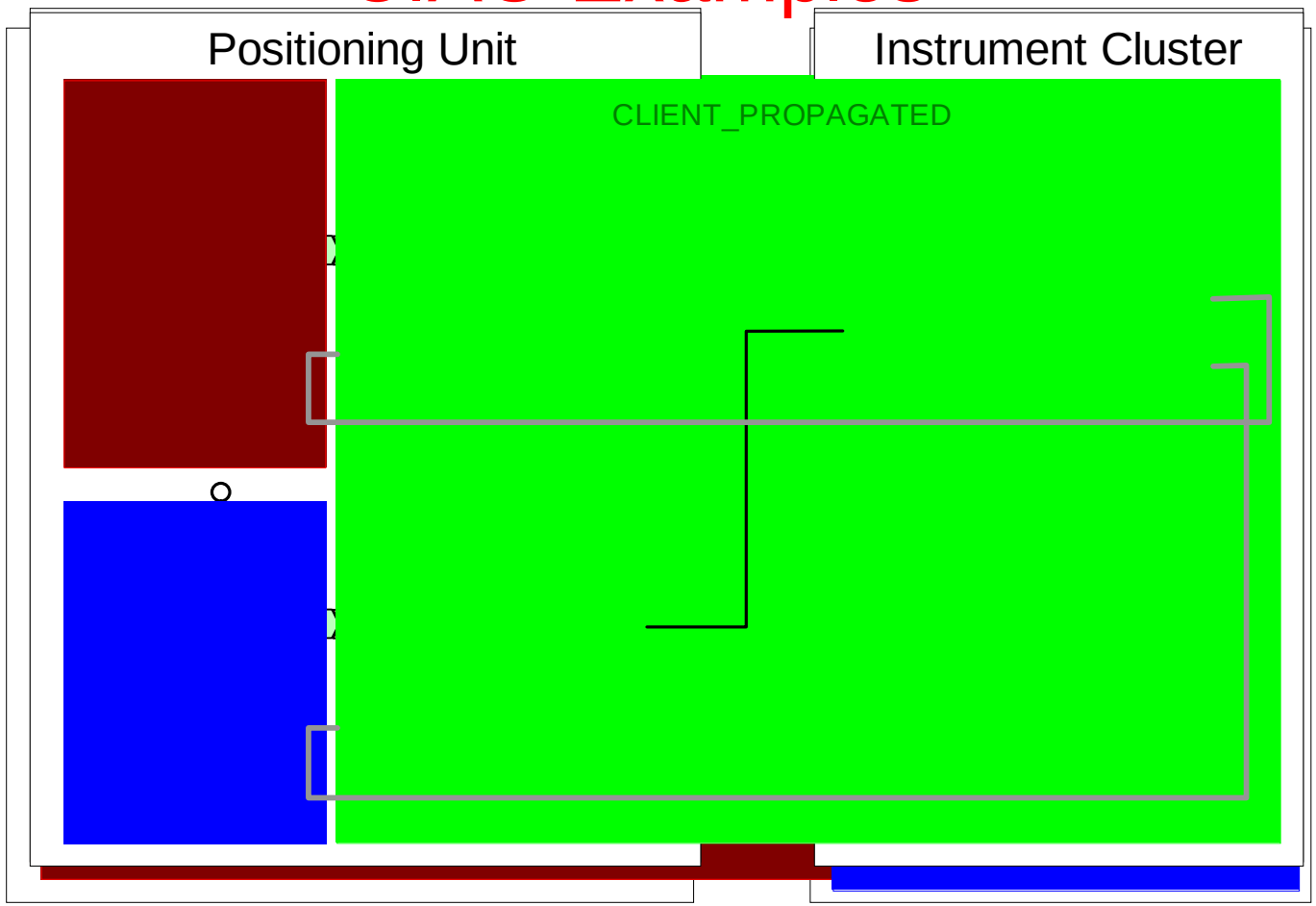
Application Development Revisited



- ***Two parallel applications***
  - GPS display runs at higher rates
  - Collision detection runs at lower rates
- ***Collision detection requires immediate attention***

Can we now program this with CIAO?

CIAO Examples



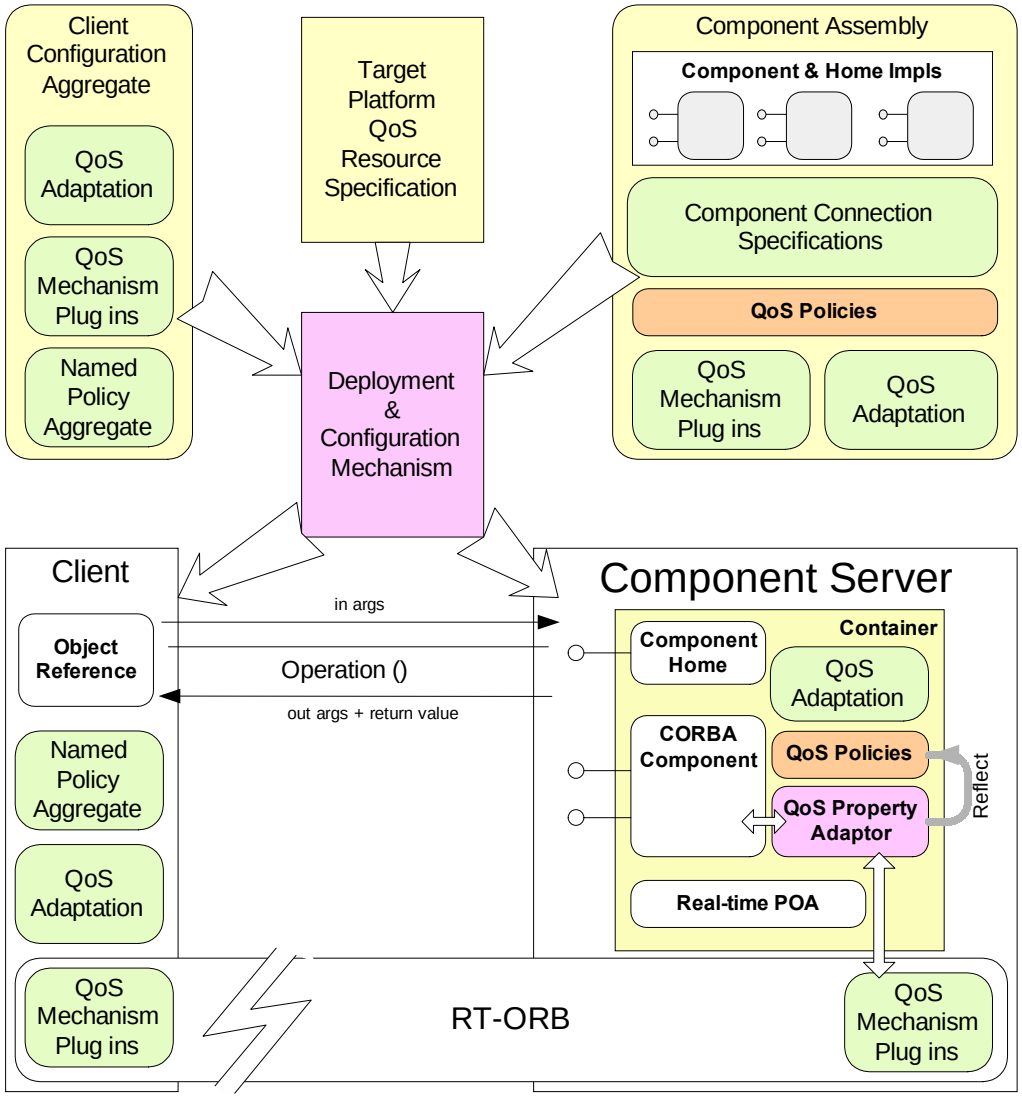
CIAO Example - BoldStroke Configuration

Config Phase	Available Rates (in Hz)	WCET (in msec)
Packaging	RateGenerator: {1,5,10,20,40} HiResGPS: {x | x < 40} cockpitDisplay: {y | y ≥ 5}	RateGenerator:1@400MHz HiResGPS: 40@400MHz cockpitDisplay: 5@400MHz
Assembly	RateGenerator⟶HiResGPS⟶ cockpitDisplay: {10,20}	RateGenerator:1@400MHz HiResGPS: 40@400MHz cockpitDisplay: 5@400MHz
Deployment (400MHz CPU)	RateGenerator⟶HiResGPS⟶ cockpitDisplay: 20 (not feasible on 200MHz CPU)	RateGenerator:1@400MHz HiResGPS: 40@400MHz cockpitDisplay: 5@400MHz
Deployment (200MHz CPU)	RateGenerator⟶HiResGPS⟶ cockpitDisplay: 10 (not optimal on 400MHz CPU)	RateGenerator:2@200MHz HiResGPS: 80@200MHz cockpitDisplay: 10@200MHz

Summary of Meta-data for RT Policies

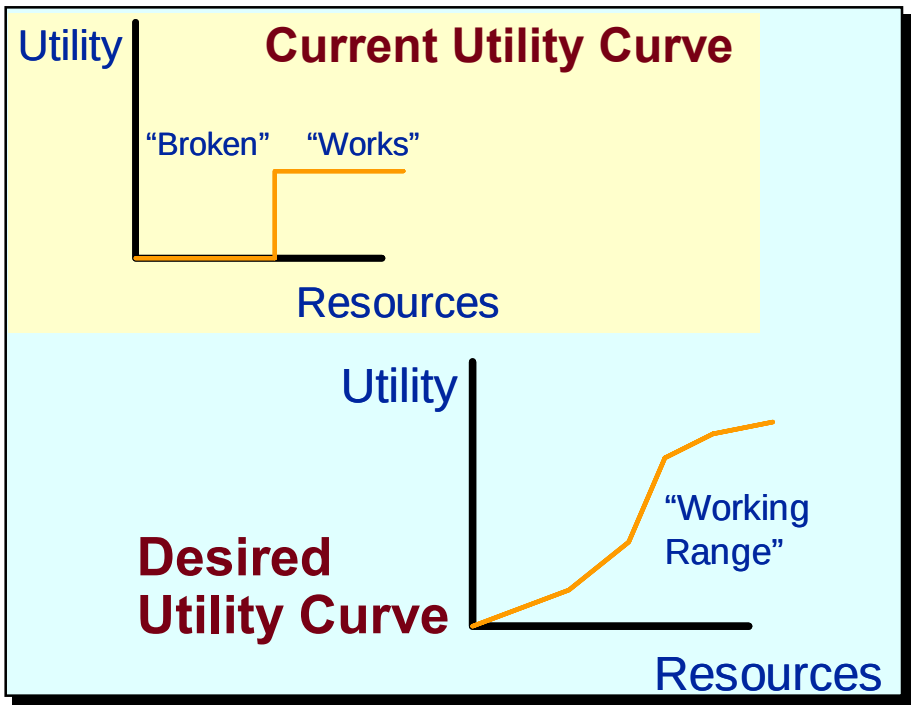
1. Component dependent
   - Require RT ORB
   - Priority model/default priority level
2. Logical resources
   - Thread pooling
   - Priority-banded
   - Custom protocol policy
3. Application assembly
   - Container policy (`<homeplacement>`)
     - Priority model/priority level
     - Association with (2)
   - Component policy:
     - Override priority (SERVER_DECLARED)
     - Priority-banded
   - Connection policy:
     - Priority-banded
     - Request `_validate_connection ()`
4. Application deployment
   - Priority mapping
   - Server protocol policy
   - Client protocol policy

Static QoS Provisioning in Component-Integrated ACE ORB (CIAO)



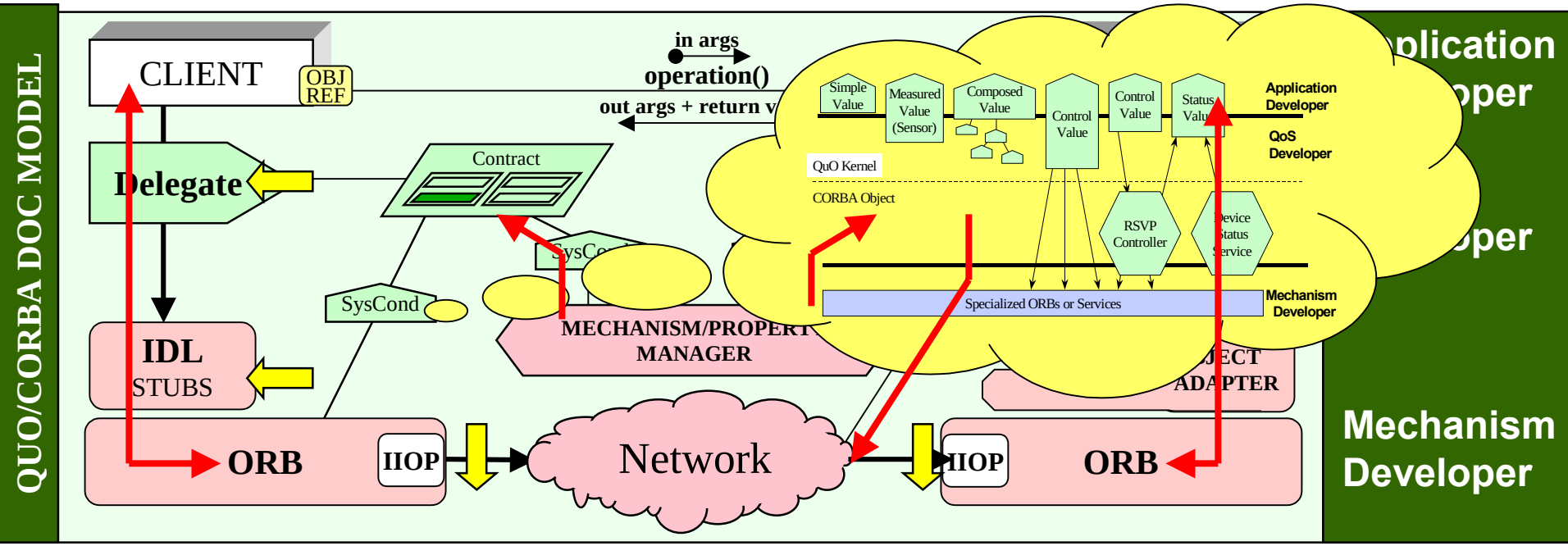
- Extension to CCM descriptors
  - Component and connection QoS specifications
  - ORB modules
  - Adaptation modules
- QoS-enabled containers
- Policy-based adaptation insertion
- Client-side policy aggregates
- Integrating RT-CORBA

What is Dynamic QoS Management?



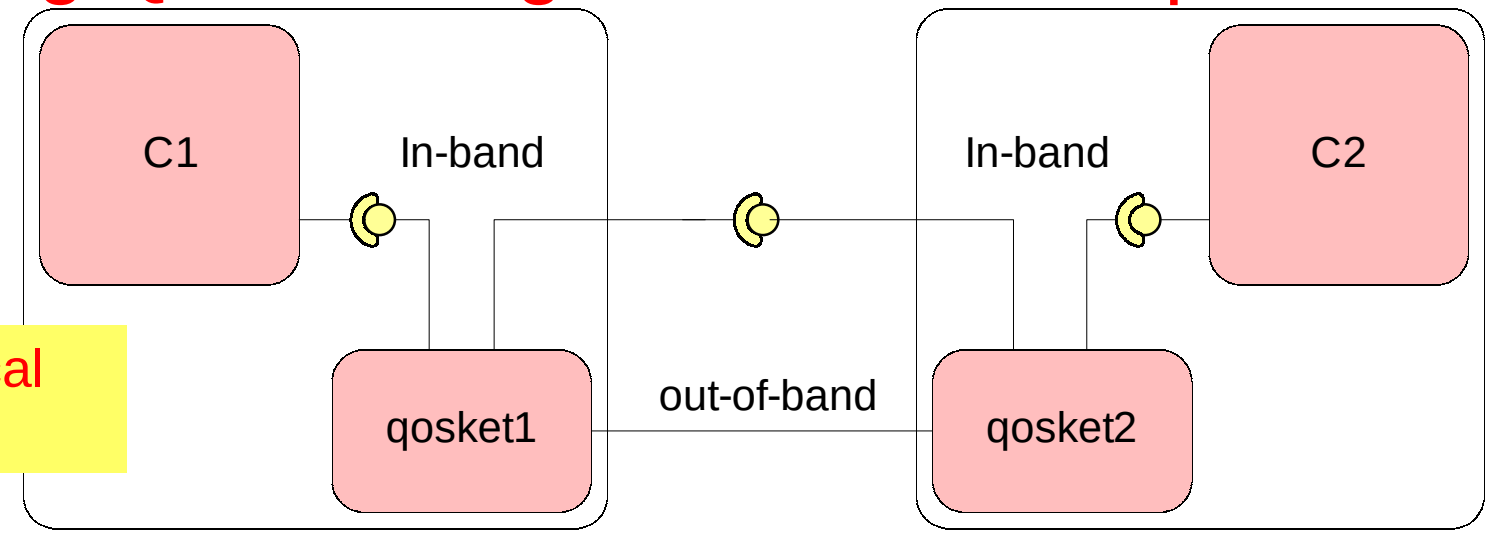
- *Measure* (sensors) system resource properties and environmental conditions
- *Evaluate* performance based on specified QoS requirements for the system
- *Adapt* application behavior to meet QoS requirements
  - Uses actuators to *control* behavior

BBN's QuO Add QoS Management to CORBA Middleware



- Plain CORBA addresses only application's functional aspects
- QuO injects QoS management
  - Measurement
    - In-band: via Instrumentation
    - Out-of-band: provided by syscond objects
  - Adaptation
    - In-band: via delegates and gateways
    - Out-of-band: triggered by transitions in contract regions

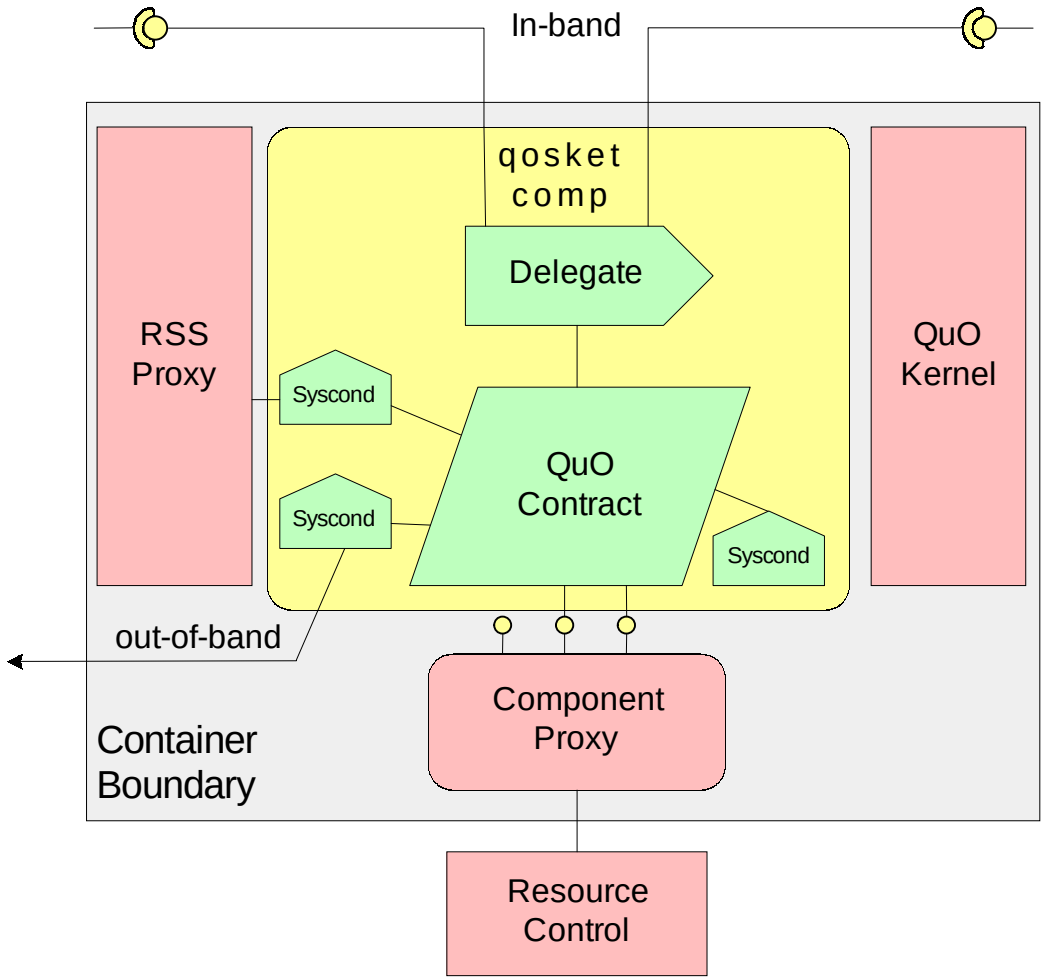
Packaging QoS Management into Components



A Prototypical Approach

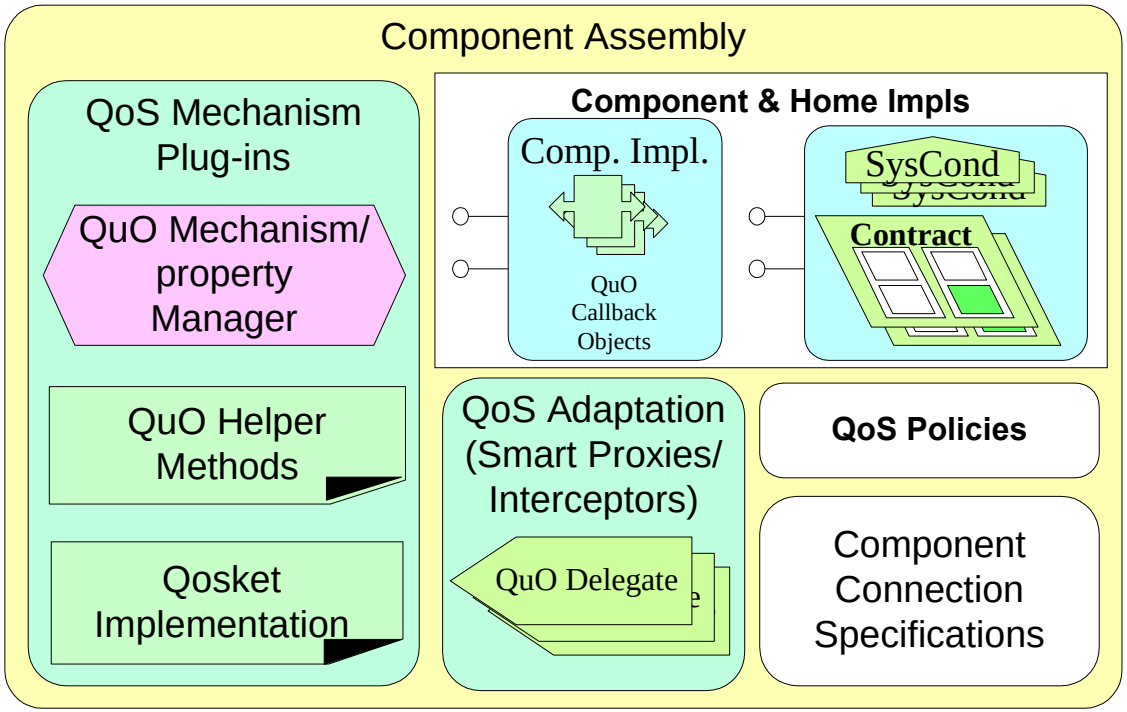
- In-band QoS components
  - QoS components are inserted between two application components, e.g. C1 and C2
  - QoS components expose delegate interfaces which *intercept* method invocations for C1 and C2 and adds QoS adaptation behaviors for C1 and C2
- Out-of-band QoS component
  - QoS components contain system condition (syscond) objects which measure system and application performance, and callbacks (actuators) which trigger adaptive behaviors

Example of a Componentized Qosket



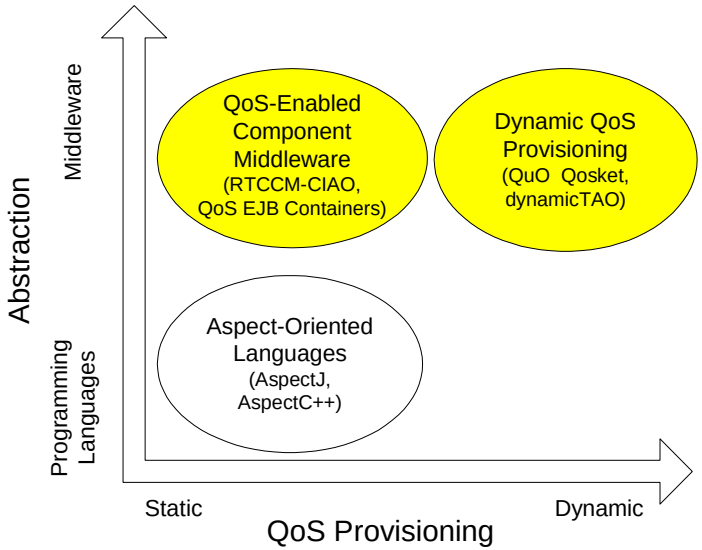
- A qosket component encapsulates
  - Delegate interception interfaces
  - Contract objects
  - Syscond objects
- A qosket component interacts internally with
  - Other qosket components for out-of-band control
  - Component proxies for accessing resource control mechanisms
- Open questions
  - Integration with event delivery mechanisms
  - Installation/connecting of resource control mechanisms

Composing Dynamic QoS Provisioning into CCM



- Container aspect hooks provide fine grained control for inserting QuO's delegates
- ORB configuration mechanism can install Qosket specific mechanisms and implementations
- Extend CIAO to insert Qosket modules into applications transparently
- Customized CCM components can implement QuO's contracts, SysConds, and callbacks objects

CIAO's Contributions: Total QoS Provisioning and Enforcement



- Statically provision QoS resources end-to-end
- Monitor and manage QoS of the end-to-end functional application interaction
- Enable the adaptive and reflective decision-making for dynamic QoS provisioning
- Integration with MDA tools such as CoSMIC and Cadena

- Integrating CIAO and Qosket covers the QoS provisioning at the middleware level
- Separation of functional and systemic paths