

# Real-time CORBA Tutorial

Bill Beckwith  
Objective Interface Systems, Inc.  
+1 703 295 6500  
[bill.beckwith@ois.com](mailto:bill.beckwith@ois.com)  
<http://www.ois.com>  
OMG Real-time and Embedded Workshop  
July 2002

1

## Agenda



- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

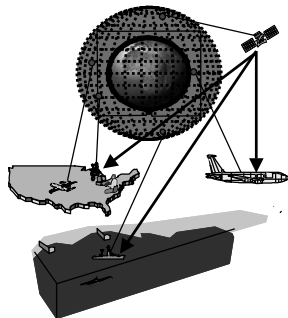
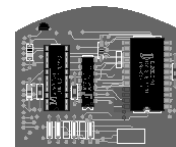
## Agenda

- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## Motivation for Real-time Middleware

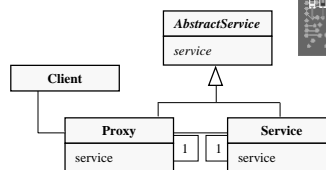
### Trends

- Hardware keeps getting smaller, faster, & cheaper
- Software keeps getting larger, slower, & more expensive
- Building distributed systems is hard
- Building them on-time & under budget is even harder



### New Challenges

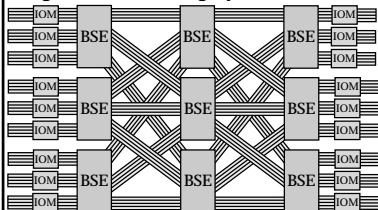
- Many mission-critical distributed applications require real-time guarantees
  - e.g., combat systems, online trading, telecom
- Building real-time applications manually is tedious, error-prone, & expensive
- Conventional middleware does not support real-time requirements effectively



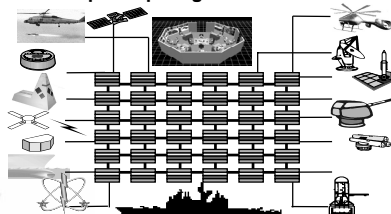
# Motivating Mission-Critical Applications



## Large-scale Switching Systems



## Total Ship Computing Environments

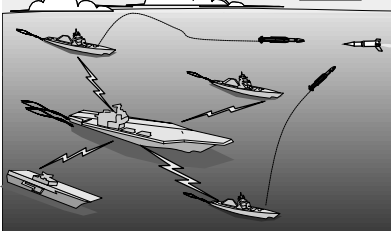


Avionics

## Industrial Process Control



## Theater Missile Defense



# Problems with Current Approaches

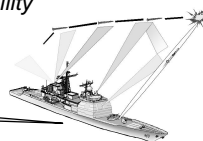


- Mission-critical system QoS requirements historically not supported by COTS
  - i.e., COTS is too big, slow, buggy, incapable, & inflexible

- Likewise, the proprietary **multiple technology bases** in systems today limit effectiveness by impeding
  - Assurability (of QoS),
  - Adaptability, &
  - Affordability

Today, each combat system brings its own:

- networks
- computers
- displays
- software
- people



## Applications

**Sensor Systems**

Technology base: Proprietary MW  
Mercury Link16/11/4

**Command & Control System**

Technology base: DII-COE  
POSIX  
ATM/Ethernet

**Engagement System**

Technology base: Proprietary MW  
POSIX  
NTDS

**Weapon Control Systems**

Technology base: Proprietary MW  
VxWorks  
FDD/LANS

**Weapon Systems**

Technology base: Proprietary MW  
POSIX  
VME/1553

## Problems

- **Non-scalable** tactical performance
- **Inadequate real-time** control for joint operations
  - e.g., distributed weapons control
- **High** software lifecycle costs
  - e.g., many "accidental complexities" & low-level platform dependencies



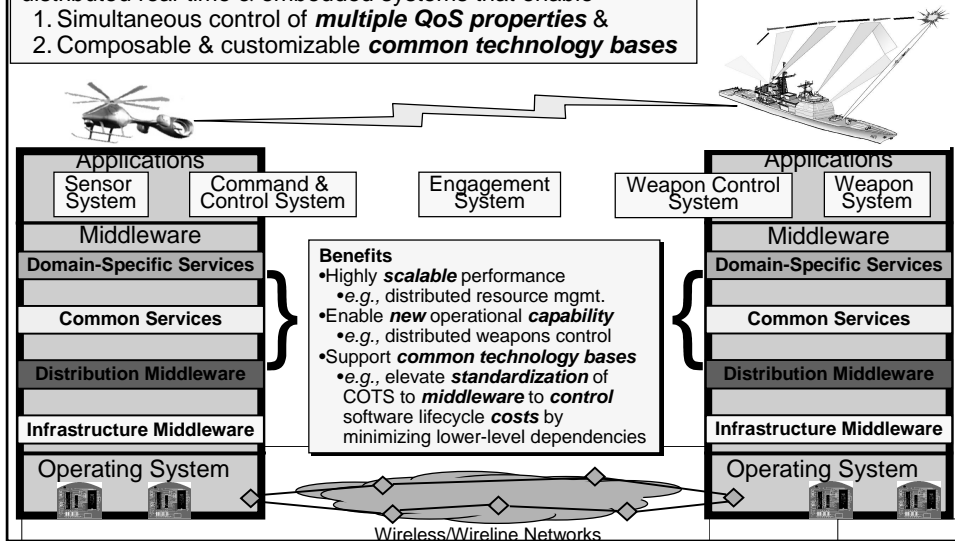
Wireless/Wireline Networks

## A More Effective Approach



Create the new generation of **middleware** technologies for distributed real-time & embedded systems that enable

1. Simultaneous control of **multiple QoS properties** &
2. Composable & customizable **common technology bases**



## Agenda

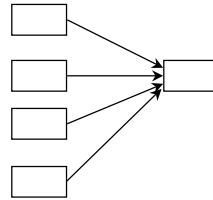


- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## What is Real-Time All About?



- Real-Time is **Not** About
  - Speed
  - Efficiency
- Real-Time **is** About Time Constraints
  - Objects contend for system resources (e.g., CPU, LAN, I/O)
  - Rules for contention resolution follow different strategies
    - These strategies are **NOT** equivalent in terms of resource utilization or engineering viability
    - There is a wide range of approaches (e.g., frames, round-robin, priority)

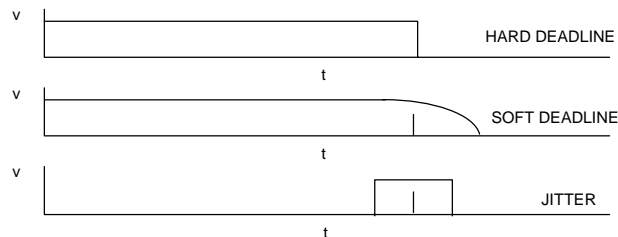


## Introduction to Real-Time



- Definition of a Real-Time System
- A real-time system is one in which correctness depends on meeting time constraints.
  - Correctness arguments must reason about response time requirements as well as functional requirements
- A real-time system produces a value to the user which is a function of time

Sample real-time program values



## Hard Real-Time, Soft Real-Time

- Hard Real-Time
  - Resources must be managed to guarantee all hard real-time constraints are met, all the time
  - No unbounded priority inversions are permitted
  - Missing a time constraint is considered a failure to meet requirements
- Soft Real-Time
  - At least three kinds, possibly in combination:
    - Time constraints may be missed by only small amounts (usually expressed as a percentage of the time constraint)
    - Time constraints may be missed infrequently (usually expressed as a percentage of instances)
    - Occasional events or periodic instances may be skipped (usually expressed as a probability)
  - Resources must be managed to meet the stated requirements
    - Same mechanisms as for hard real-time, but guarantees harder to define
    - Soft real-time is not the same as non-real-time, or meeting an average response time
- Hard Real-Time is hard, but Soft Real-Time is harder!

## Timing Requirement Sources

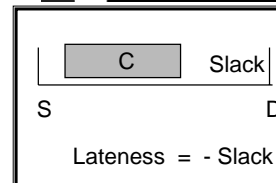
- Time Constraint Requirements Have Only Two Sources:
  - Explicit Top Level Requirements, e.g.,
    - Display a video frame within 33 milliseconds.
    - This is not the most common source of Timing Requirements.
  - Derived Requirements, e.g.,
    - Accuracy – “Maintain aircraft position to within 1 meter => Periodicity”
    - Fault Tolerance – “Recover from message loss within 500 ms.”
    - Human Computer Interface Requirements –
      - Process switch depressions within 250 ms.
      - Refresh display within 68 ms.

## Real-Time Scheduling

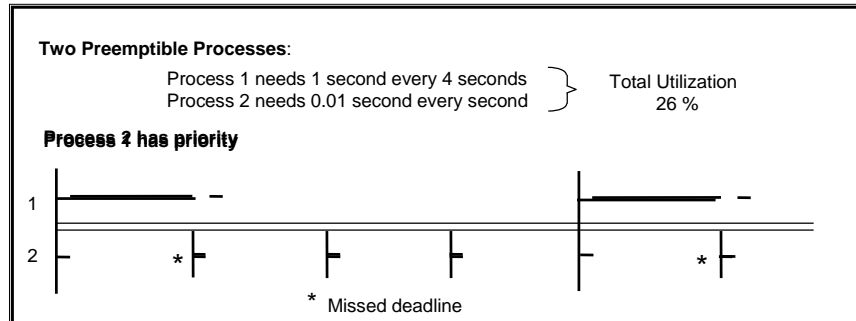
- System Resource Scheduling: The Principal Difference Between a Real-Time and Non-Real-Time System.
  - Definition of Real-Time Scheduling:
    - Real-Time scheduling is the process of sequencing shared resource allocations to meet user's time constraints.
  - The Principal Three Goals:
    - Meet all application time constraints, if feasible.
    - Meet all important time constraints if meeting all time constraints is not feasible.
    - Be able to accurately predict how well goals 1 and 2 are met for any given process load.
  - Real-Time  $\neq$  Real-Fast

## Known Real-Time Scheduling Algorithms

- Shortest Processing Time First (SPT)
  - Minimizes mean lateness
  - Optimal for Goals 1 and 3, stochastically
- Earliest Deadline First (EDD or Deadline)
  - Minimizes maximum lateness
  - Optimal for Goals 1 and 3 (fails disastrously on overload)
- Smallest Slack Time First (Minimum Slack or Minimum Laxity)
  - Maximizes minimum lateness
  - Optimal for Goals 1 and 3 (fails disastrously on overload)
- Rate Monotonic Scheduling (RMS)
  - Approximates EDD with reduced utilization bound
  - Non-optimal, but fulfills all real-time goals for mostly periodic processing domains
  - Optimal for fixed priority scheduling



## Scheduling Example



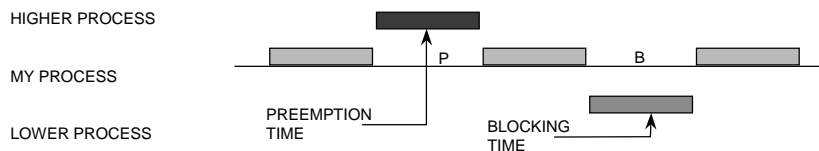
### World's Lowest Budget Nuclear Power Plant

- Process 1 => Reactor Control Rods
- Process 2 => Dishwasher Water Valve

## Shared Resource Concepts



- Preemption.
  - Execution delayed by higher priority tasks
- Blocking (Priority inversion)
  - Execution delayed by lower priority tasks
- Mutual Exclusion (Mutex)
  - Sequenced access to a shared resource, typically implemented by locking and waiting for locks.
- Critical Section
  - Execution while holding a lock.





## Dynamic Real-Time Systems



- Dynamic
  - Super-set of static systems
  - May not have sufficiently predictable workload to allow static approach
  - Set of applications is may be too large or not known in advance
  - Variable application processing requirements may impair pre-planning
  - The arrival time of the inputs may be too variable
    - other sources of variability
    - the underlying infrastructure must be still able satisfy real-time requirements in a changing environment

## Real-Time Principles Summary



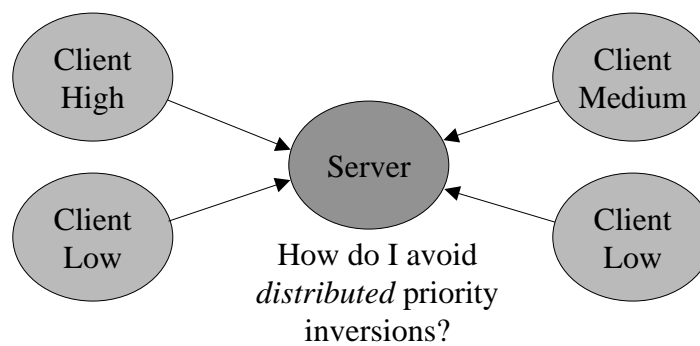
- Real-Time  $\neq$  Real-Fast
- Scheduling concepts are frequently counter-intuitive
- Resource management is the fundamental requirement to deal with response time issues, whether hard real-time or soft real-time

## Agenda

- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## Real-time Distribution Challenges – 1

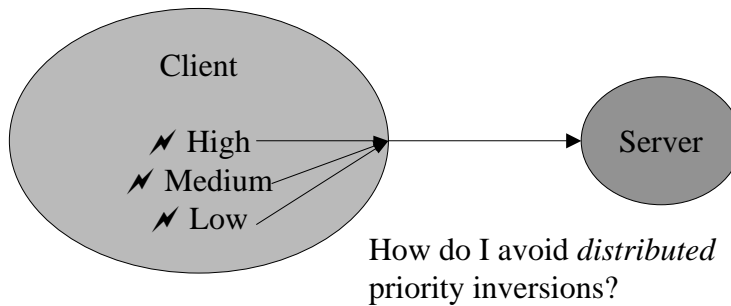
- How does a server resolve contention by multiple clients?



## Real-time Distribution Challenges – 2



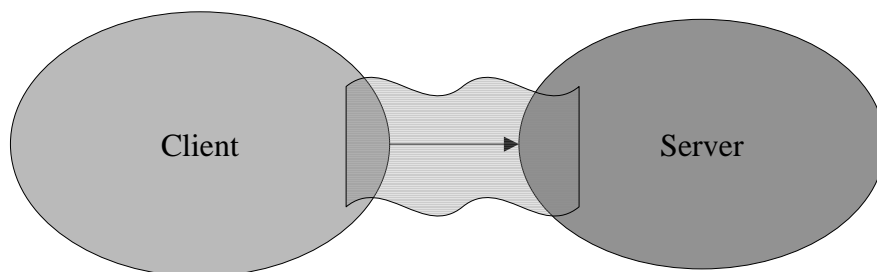
- How does a server resolve contention by multiple *threads* within each clients?



## Real-time Distribution Challenges – 3



- How do I use a real-time transport but remain *independent of the transport API*?



## Real-time Distribution is Hard



- Most engineers don't understand complexities of real-time distribution *with software*
- Real-time communication theory is hard
- Implementing correct semantics in a software application is *harder*

## Simple, but Expensive Solutions Commonplace



- Result:
  - Engineers are not confident that their communications software will meet deadlines
  - No trust of real-time communication software
  - Time constraints solved with dedicated hardware
  - Software architecture designed around hardware
  - Software oversimplified at the expense of functionality
  - Expensive to build and maintain
  - Software investment is thrown away in future re-designs

## Real-time Communication Solutions



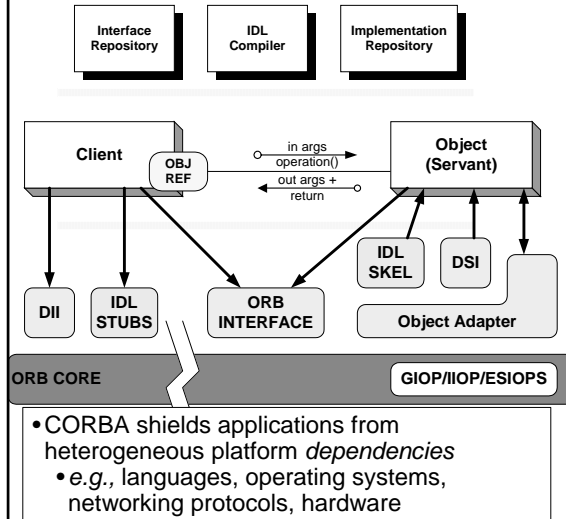
- What if there was a real-time communication software infrastructure that:
  - Makes communication easy
  - Can meet application time constraints
  - Can save software investment over product generations
  - Can reduce product time-to-market
  - Is based on an open standard
  - Has alternative, competing products available
  - Is proven reliable in many existing systems

## Agenda



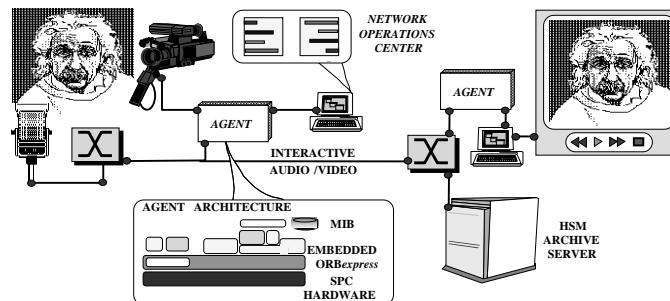
- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## Overview of CORBA



- Common Object Request Broker Architecture (CORBA)
  - A family of specifications
  - OMG is the standards body
  - Over 800 companies
- CORBA defines *interfaces*, not *implementations*
- It simplifies development of distributed applications by automating/encapsulating
  - Object location
  - Connection & memory mgmt.
  - Parameter (de)marshaling
  - Event & request demultiplexing
  - Error handling & fault tolerance
  - Object/server activation
  - Concurrency
  - Security

## Caveat: Historical Limitations of CORBA for Real-time Systems



### Requirements

- Location transparency
- Performance transparency
- Predictability transparency
- Reliability transparency

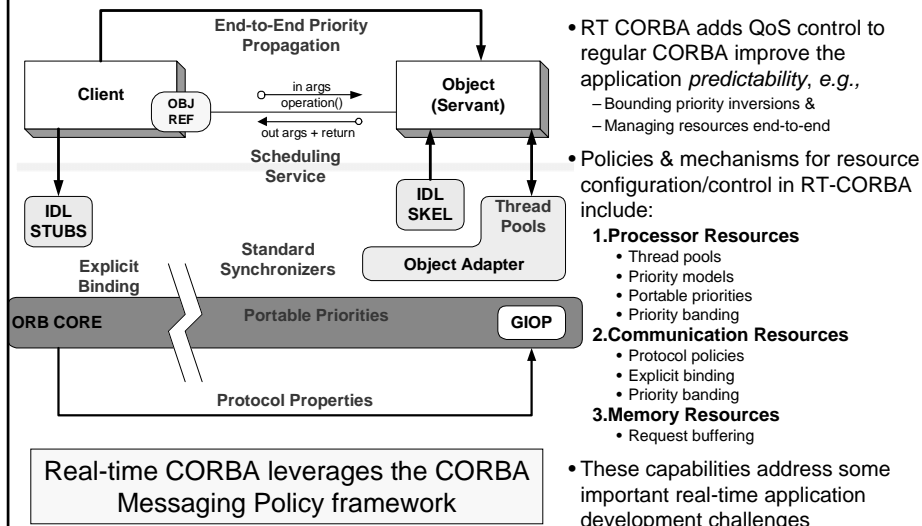
### Historical Limitations

- Lack of timeliness specifications
- Lack of timeliness enforcement
- Lack of real-time programming features
- Lack of performance optimizations

## Do I Need Real-Time CORBA?

- You need Real-Time CORBA if:
  - You use priorities in your distributed system
  - You want application-level control of the resources used by the ORB, including connections and connection-related resources
  - You need a foundation for building predictable systems

## Real-Time CORBA Overview





## History

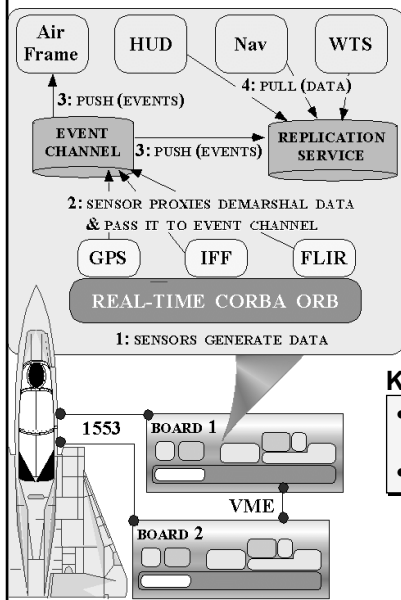
- Real-Time CORBA 1.0
  - RFP issued 9/1997
  - Final submissions 7/1999
  - OMG CORBA 2.6.1 specification, chapter 24
- Real-Time CORBA 2.0: Dynamic Scheduling
  - RFP issued 3/1999
  - Final submissions 6/2001
  - Draft Adopted Specification: 8/2001
  - OMG Document ptc/01-08-34
    - <http://cgi.omg.org/cgi-bin/doc?ptc/01-08-34.pdf>



## Agenda

- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## Applying RT CORBA to Real-time Avionics



### Goals

- Apply COTS & open systems to mission-critical real-time avionics

### Key System Characteristics

- Deterministic & statistical deadlines
  - ~20 Hz
- Low latency & jitter
  - ~250 usecs
- Periodic & aperiodic processing
- Complex dependencies
- Continuous platform upgrades

### Key Results

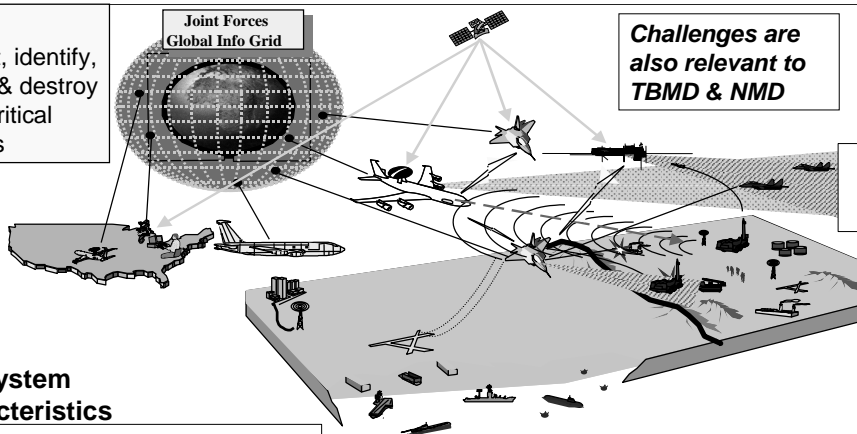
- Test flown at China Lake NAWS by Boeing OSAT II '98, funded by OS-JTF
- Also used on SOFIA project by Raytheon

## Applying RT CORBA to Time-Critical Targets



### Goals

- Detect, identify, track, & destroy time-critical targets



### Key System Characteristics

- Real-time mission-critical sensor-to-shooter needs
- Highly dynamic QoS requirements & environmental conditions
- Multi-service & asset coordination

### Key Solution Characteristics

- Adaptive & reflective
- High confidence
- Safety critical
- Efficient & scalable
- Affordable & flexible
- COTS-based

## Applying RT CORBA to Hot Rolling Mills



### Goals

- Control the processing of molten steel moving through a hot rolling mill in real-time

### System Characteristics

- Hard real-time process automation requirements
  - *i.e.*, 250 ms real-time cycles
- System acquires values representing plant's current state, tracks material flow, calculates new settings for the rolls & devices, & submits new settings back to plant

### Key Software Solution Characteristics

- Affordable, flexible, & COTS
- Product-line architecture
- Design guided by patterns & frameworks
- Windows NT/2000
- Real-time CORBA

## Applying RT CORBA to Image Processing



### Goals

- Examine glass bottles for defects in real-time

### System Characteristics

- Process 20 bottles per sec
  - *i.e.*, ~50 msec per bottle
- Networked configuration
- ~10 cameras

### Key Software Solution Characteristics

- Affordable, flexible, & COTS
- Embedded Linux (Lem)
- Compact PCI bus + Celeron processors
- Remote booted by DHCP/TFTP
- Real-time CORBA

## Real-time CORBA in Action: JTRS—Joint Tactical Radio System



- JTRS Business Case

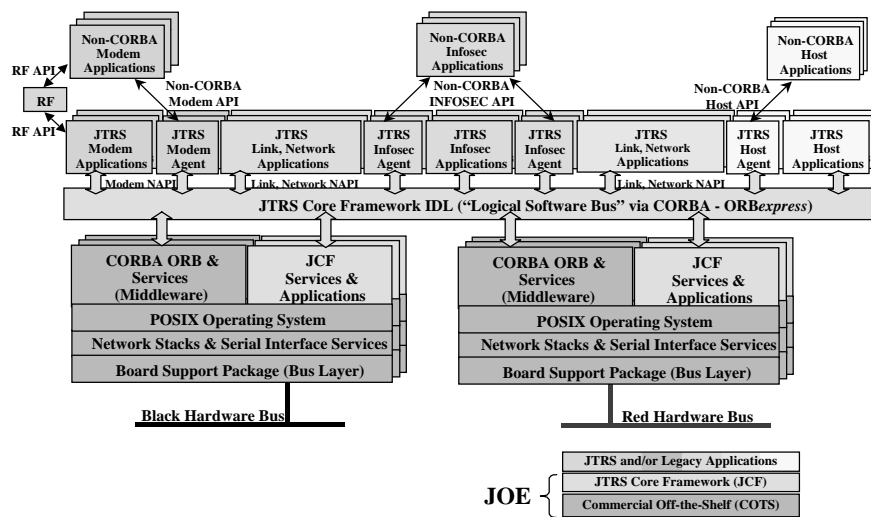
- Today's operating environment requires diverse, interoperable communications
- Single purpose radios sets are a way of the past
- Legacy systems required redundant development of capabilities

**Open-systems architectures allow for:**

- more efficient use of R&D and Procurement dollars
- faster rate of technology insertion
- interoperability through the use of common implementations
- End result:
  - more capable customer with current capabilities



## JTRS Software Structure



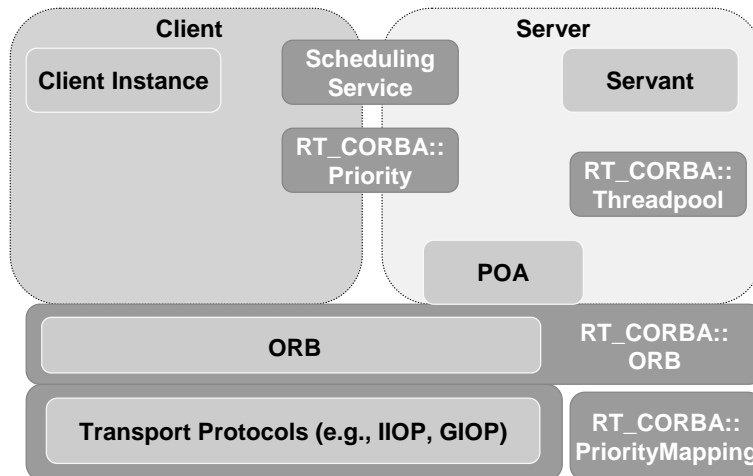
## Agenda

- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## Real-time CORBA 1.0

- Robust, theoretically-sound specifications
  - Avoids distributed priority inversions
  - Propagates scheduling information
- Based on flexible, maintainable CORBA software architecture
- Rich set of competing implementations
- Allows development of software-based time-bounded systems
  - Can utilize predictable transport hardware
  - Without requiring that system design is centered around transport hardware

## Real-Time CORBA 1.0 Components



## Scheduling in Real-Time CORBA 1.0

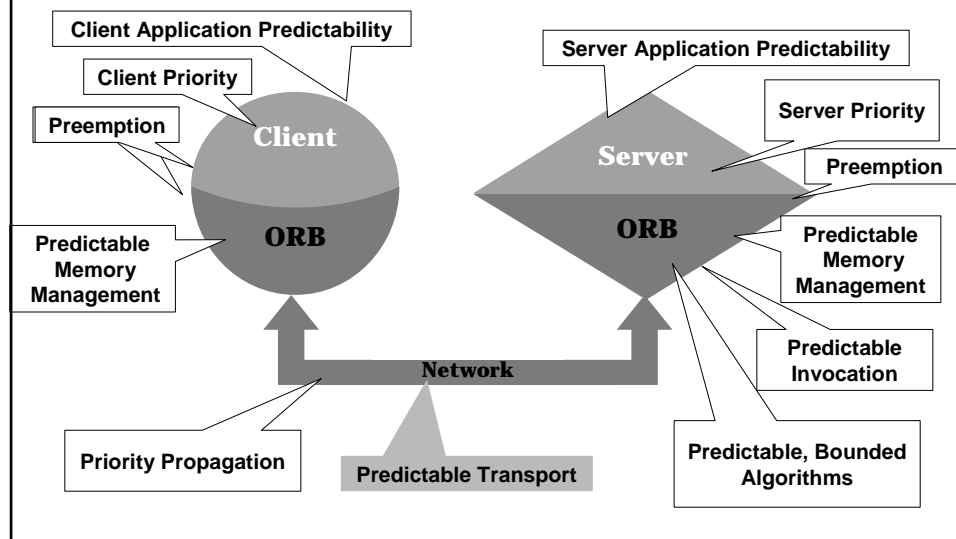


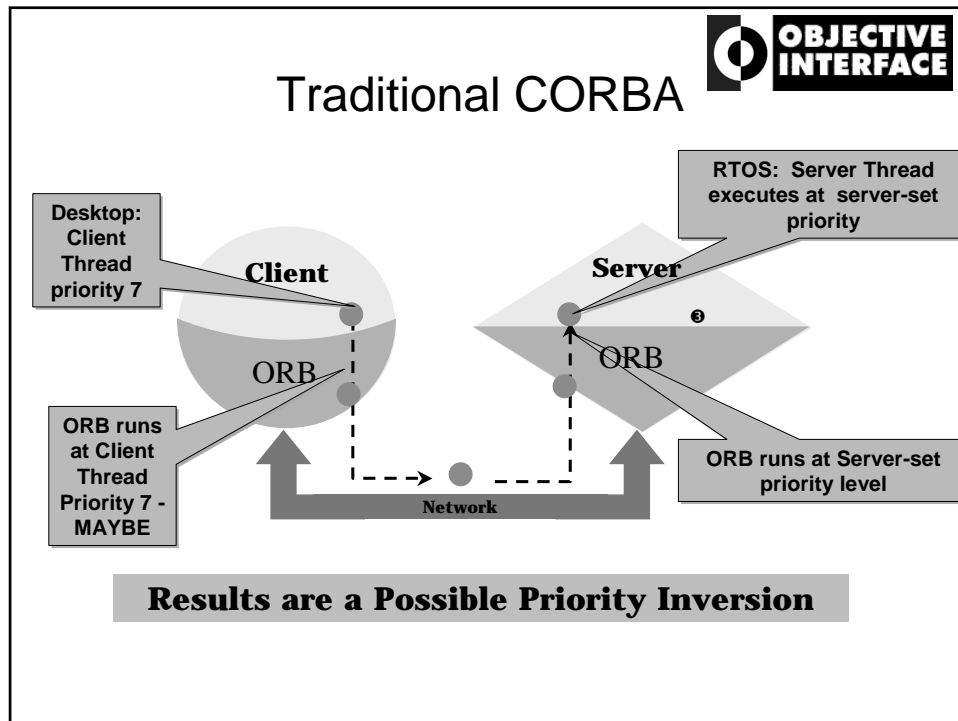
- Fixed Priority (Static) Scheduling
  - Provides a basis for resource allocation
  - Threads have priority based on time constraints
  - Underlying Theoretical Basis
    - Rate Monotonic Scheduling
    - Deadline Monotonic Scheduling
  - Fixed Priority means priorities change only
    - To prevent priority inversion
    - When required by application

## Real-Time CORBA Goals

- Support developers in meeting real-time requirements by:
  - **Facilitating** end-to-end predictability
  - Providing application-level management of ORB resources
- Provide interoperability with traditional ORB's
- Three *magic bullets*
  - “For the purposes of this specification, ‘end-to-end predictability’ of timeliness in a fixed priority CORBA system is defined to mean:
    - respecting thread priorities between client and server for resolving resource contention during the processing of CORBA invocations;
    - bounding the duration of thread priority inversions during end-to-end processing;
    - bounding the latencies of operation invocations.”

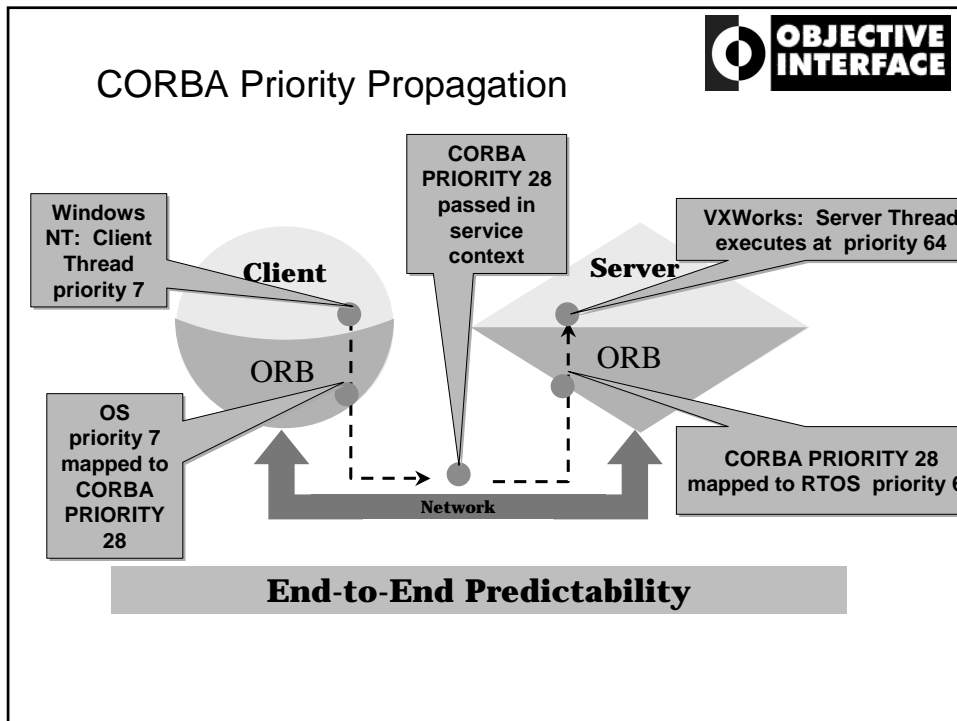
## Real-Time System Predictability



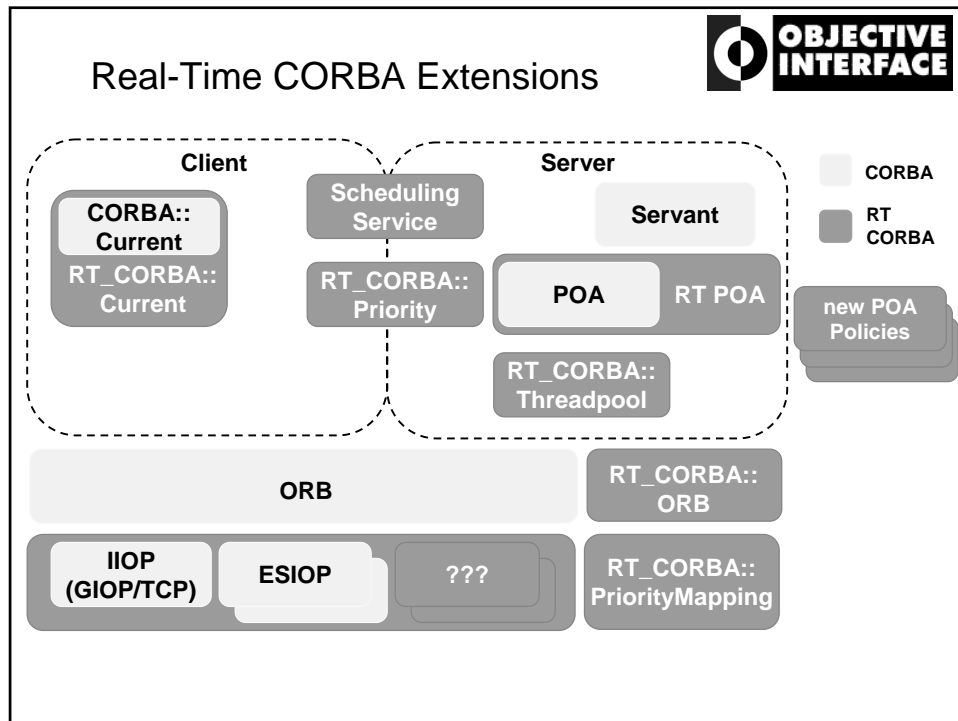


## End-to-End Predictability

- Respect thread priorities between client and server
- Bound duration of thread priority inversions
- Bound latencies of operation invocations
- Provide explicit control over ORB-managed resources



- ## Operating System Assumptions
- Not a portability layer for heterogeneous Operating Systems
  - POSIX is sufficient ... but not necessary
    - POSIX 1003. 1b- 1996 Real- time Extensions
  - Extent to which the ORB is deterministic may be limited by RTOS characteristics
- OBJECTIVE INTERFACE**



- ## RTORB & RT POA
- **RTCORBA::RTORB**
    - Consider as an extension of the CORBA::ORB interface
    - Specified just to provide operations to create and destroy other Real-Time CORBA entities
      - Mutex, Threadpool, Real-Time Policies
    - One RTORB per ORB instance
    - Obtain using:
 

```
ORB::resolve_initial_references("RTORB");
```
  - **RTPortableServer::POA**
    - Adds operations to POA to allow setting of priority on a per-Object basis
    - Implements policies for the RT POA
    - Inherits from PortableServer::POA (so not PIDL)



## New RT POA Policies

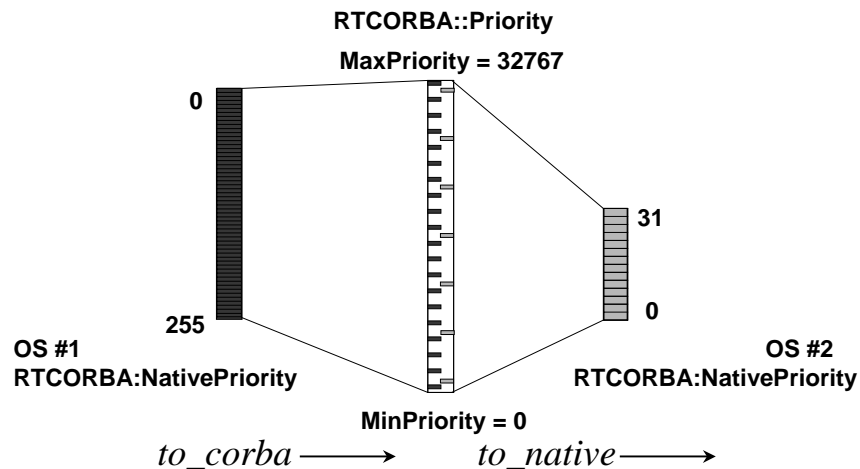
- Priority Model Policy
- Threadpool Policy
- Priority Banded Connection Policy
- Private Connection Policy
- Server Protocol Policy
- Client Protocol Policy



## Universal Real-time Priorities

- RTCORBA::Priority
  - Universal, platform independent priority scheme
  - Allows prioritized CORBA invocations to be made in a consistent fashion between nodes with different native priority schemes
- Priority Mapping
  - Default Priority Mapping
    - Not standardized
    - Real-time ORB must supply a default mapping for each platform (RTOS) that the ORB supports
  - User-defined Priority Mapping
    - Real-time ORB must provide method for user to override default mapping
  - One Priority Mapping installed at any one time per ORB instance
    - installation mechanisms are not standardized

## Real-Time CORBA Priority Mapping



## Priority Model Policy



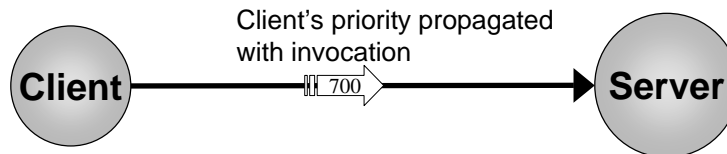
- Two Priority Models Specified
  - Client Propagated
    - Via client-side Policy override
    - Via server-side POA Policy
  - Server Declared
    - Only server-side POA Policy
- A Server-side (POA) Policy
  - configure by adding a PriorityModelPolicy to policy list in POA\_create
  - all objects from a given POA support the same model

## Client Propagated Priority Model



Client thread running  
at RTCORBA::Priority 700

Invocation handled  
at RTCORBA:Priority 700



Scheduling based on end-to-end activities across node boundaries

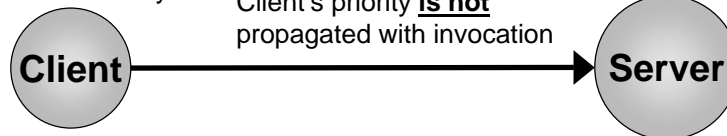
## Server Declared Priority Model



Client running at  
RTCORBA::Priority 700

Client's priority **is not**  
propagated with invocation

Server Priority  
is pre-set



Invocation handled  
at pre-set Server  
priority

Scheduling based on relative priorities of different objects on the same node.

Priority of ORB threads writing to and reading from transport is not defined for Server Declared Priority Model.

## Real-Time CORBA Mutex

- API that gives the application access to the same Mutex implementation that the ORB is using
  - important for consistency in using a Priority Inheritance Protocol
    - e.g. Priority Ceiling Protocol
- The implementation must offer (at least one) Priority Inheritance Protocol
  - No particular protocol is mandated though
  - application domain and RTOS specific

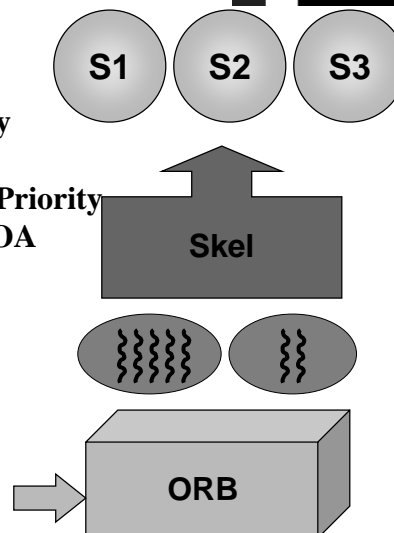
## Threadpools

### Threadpool Benefits

- Control invocation concurrency**
- Thread pre-creation and reuse**
- Partition threads according to Priority**
- Bound thread usage by each POA**

### Multiple Threadpools

- System partitioning**
- Protect resources**
- Integrate different systems more predictably**



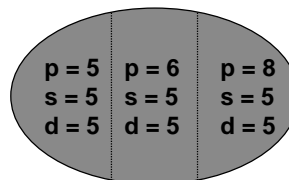
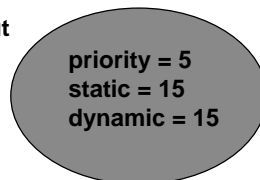
## Threadpool Policy

- A POA may only be associated with one Threadpool
- Threadpools may be shared between POA's
- Threadpool parameters
  - number of static threads
  - dynamic thread limit
    - 0 = no limit. same value as static = no dynamic threads
  - thread stacksize
  - default thread priority

## Laned Threadpools

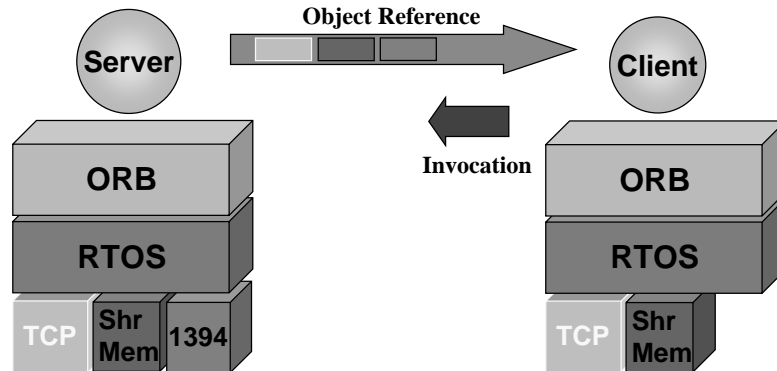
- Alternate way of configuring a Threadpool
  - for applications with detailed knowledge of priority utilization
  - preconfigure 'lanes' of threads with different priorities
  - 'borrowing' from lower priority lanes can be permitted

without  
lanes



with  
lanes

## Protocol Selection and Configuration



On Server, this policy indicates which protocols to publish in Object Reference

On Client, this policy Indicates which protocol to use to connect

## Protocol Properties



- A Protocol Properties interface to be provided for each configurable protocol supported
  - allows support for proprietary and future standardized protocols
- Real-Time CORBA only specifies Protocol Properties for TCP
- TCP Protocol Properties

```
module {  
  interface TCPProtocolProperties : ProtocolProperties  
  {  
    attribute long send_buffer_size;  
    attribute long recv_buffer_size;  
    attribute boolean keep_alive;  
    attribute boolean dont_route;  
    attribute boolean no_delay;  
  };  
};
```

## Client and Server Protocol Policies



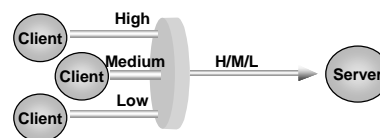
- **Server Protocol Policy**
  - Enables selection and configuration of communication protocols on a per-POA basis
  - Protocols are represented
    - By a unique id
    - And by Protocol Properties defined as ORB/transport level protocol pairs
  - RTCORBA::ProtocolList allows multiple protocols to be supported by one POA
    - Order of protocols in list indicates order of preference
- **Client Protocol Policy**
  - Same syntax as server-side
  - On client, RTCORBA::ProtocolList specifies protocols that may be used to make a connection
    - order indicates order of preference
  - If Protocol Policy not set, order of protocols in target object's IOR used as order of preference

## Connection Management



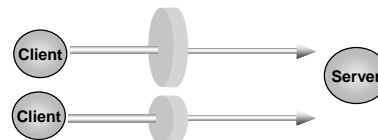
### Connection Multiplexing

- Offered by many ORB's for resource conservation
- Can contribute to priority inversion



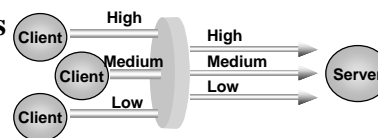
### Private Connection

- Dedicated connection per object



### Priority Banding

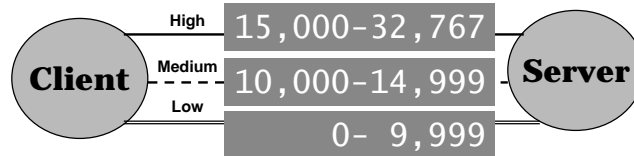
- Several connections between nodes
- Connection to be used based on priority
- Can avoid O/S priority inversions



## Priority Banding Connection Policy



- Multiple connections, to reduce priority inversion
  - each connection may represent a range of priorities
  - may have different ranges in each band, including range of 1

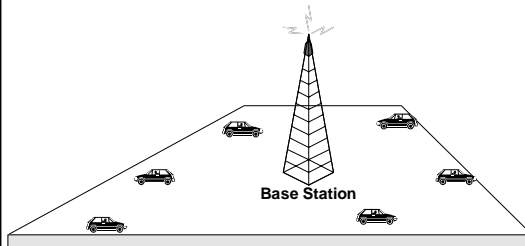


## Agenda



- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

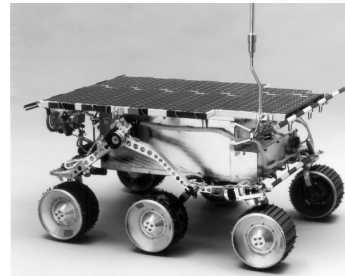
## An Example Distributed Application



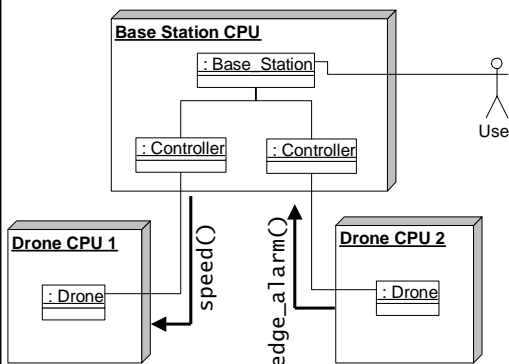
- Consider an application where cooperating *drones* explore a surface & report its properties periodically
  - e.g., color, texture, etc.
- This is a simplification of various autonomous vehicle use-cases



- Drones aren't very "smart,"
  - e.g., they can fall off the "edge" of the surface if not stopped
- Thus, a *controller* is used to coordinate their actions
  - e.g., it can order them to a new position



## Designing the Application



- End-users talk to a *BaseStation* object
  - e.g., they define high-level exploration goals for the drones
- The *BaseStation* object controls the drones remotely using *Drone* objects
- *Drone* objects are proxies for the underlying drone vehicles
  - e.g., they expose operations for controlling & monitoring individual drone behavior

- Each drone sends information obtained from its sensors back to the *BaseStation* via a *Controller* object

## Defining Application Interfaces with CORBA IDL

```

interface Drone
{
    void turn (in float degrees);
    void speed (in short mph);
    void reset_odometer ();
    short odometer ();
    // ...
};

interface Controller
{
    void edge_alarm();
    void battery_low();
};

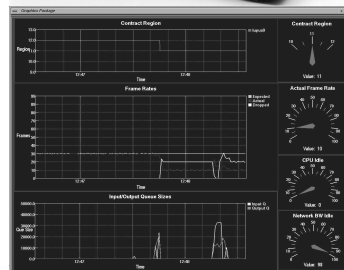
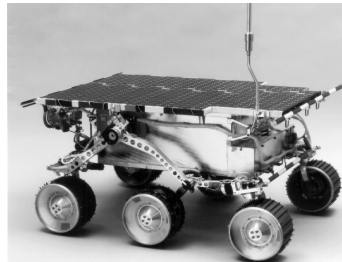
exception Lack_Resources {};

interface BaseStation
{
    Controller new_controller(in string name)
        raises (Lack_Resources);
    void set_new_target (in float x, in float y);
    //.....
};
    
```

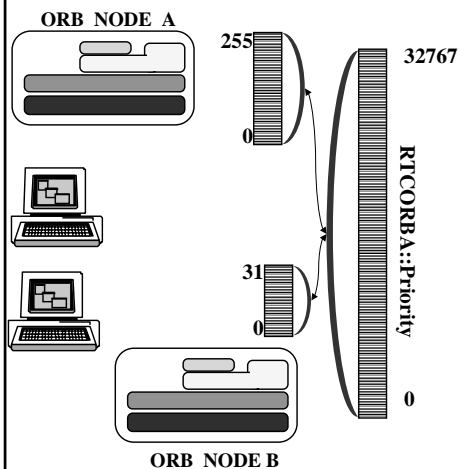
- Each Drone talks to one Controller
  - e.g., Drones send hi-priority alarm messages when they detect an edge
- The Controller should take corrective action if a Drone detects it's about to fall off an edge!
- The BaseStation interface is a Controller factory
  - Drones use this interface to create their Controllers during power up
  - End-users use this interface to set high-level mobility targets

## Real-time Application Design Challenges

- Our example application contains the following real-time design challenges
  1. *Obtaining portable ORB node priorities*
  2. *Preserving priorities end-to-end*
  3. *Enforcing certain priorities at the server*
  4. *Changing CORBA priorities*
  5. *Supporting thread pools effectively*
  6. *Buffering client requests*
  7. *Synchronizing objects correctly*
  8. *Configuring custom protocols*
  9. *Controlling network & node resources to minimize priority inversion*
  10. *Avoiding dynamic connections*
  11. *Simplifying application scheduling*
  12. *Controlling request timeouts*
- The remainder of this tutorial illustrates how these challenges can be addressed by applying RT CORBA capabilities



## Obtaining Portable ORB Node Priorities



- **Problem:** Mapping CORBA priorities to native OS host priorities
- **Solution:** Standard RT CORBA priority mapping interfaces
  - OS-independent design supports heterogeneous real-time platforms
  - CORBA priorities are “globally” unique values that range from 0 to 32767
  - Users can map CORBA priorities onto native OS priorities in custom ways
  - No silver bullet, but rather an “enabling technique”
  - *i.e.*, can’t magically turn a general-purpose OS into a real-time OS!

## Priority Mapping Example

- Define a priority mapping class that always uses native priorities in the range 128-255
  - *e.g.*, this is the top half of LynxOS priorities

```
class MyPriorityMapping : public RTCORBA::PriorityMapping {
    CORBA::Boolean to_native (RTCORBA::Priority corba_prio,
                             RTCORBA::NativePriority &native_prio)
    {
        native_prio = 128 + (corba_prio / 256);
        // In the [128,256) range...
        return true;
    }
    // Similar for CORBA::Boolean to_CORBA ();
};
```

## Preserving Priorities End-to-End

- Problem: Requests could run at the wrong priority on the server
  - e.g., this can cause major problems if `edge_alarm()` operations are processed too late!!
- Solution: Use RT CORBA priority model policies
  - SERVER\_DECLARED
    - Server handles requests at the priority declared when object was created
  - CLIENT\_PROPAGATED
    - Request is executed at the priority requested by client (priority encoded as part of client request)

## Creating PriorityBands Policies

- Drones send critical messages to Controllers in the BaseStation
  - `edge_alarm()` called by and runs at high priority
  - `battery_low()` called by and runs at low priority
- So first we set up a PriorityBands Policy

```
// ----- Obtain a reference to the RTORB
RTCORBA::RTORB_var rt_orb = RTCORBA::RTORB::_narrow(
    orb->resolve_initial_references("RTORB"));

// ----- Obtain the ORB's PolicyManager
CORBA::PolicyManager_var orb_pol_mgr = CORBA::PolicyManager::_narrow(
    orb->resolve_initial_references("ORBPolicyManager"));

RTCORBA::PriorityBands priority_bands(3);
priority_bands.length(3);
priority_bands[0].low = 0;
priority_bands[0].high = 10000;
priority_bands[1].low = 10001;
priority_bands[1].high = 20000;
priority_bands[2].low = 20001;
priority_bands[2].high = 32767;

CORBA::PolicyList policy_list(1);
policy_list.length(1);
policy_list[0] =
    rt_orb->create_priority_banded_connection_policy(priority_bands);

orb_pol_mgr->set_policy_overrides(policy_list, CORBA::ADD_OVERRIDE);
```

## Overriding IOR with PriorityBands

- Then we override and use the new object reference

```
controller_var controller = // get from naming service, etc.
// override the object reference with banding policies
CORBA::Object_var temp_object =
    controller->_set_policy_overrides(
        policy_list, CORBA::ADD_OVERRIDE);

Controller_var rt_controller = Controller::_narrow(temp_object);

// Real-time invocation using priority banding.
// controller servant runs at equivalent of this
// threads priority (high or low) in server.

rt_controller->edge_alarm();

// Normal invocation without priority banding. (BAD!)
// controller servant runs at undefined priority
// (ORB default typically) in server.

controller->edge_alarm();
```

## Changing CORBA Priorities at the Client

- **Problem:** How can RT-CORBA client applications change the priority of operations?
- **Solution:** Use the `RTCurrent` to change the priority of the current thread explicitly
  - An `RTCurrent` can also be used to query the priority
  - Values are expressed in the *CORBA priority* range
  - Behavior of `RTCurrent` is *thread-specific*

```
// Get the ORB's RTCurrent object
obj = orb->resolve_initial_references ("RTCurrent");

RTCORBA::RTCurrent_var rt_current =
    RTCORBA::RTCurrent::_narrow (obj);

// Change the current CORBA priority
rt_current->base_priority (VERY_HIGH_PRIORITY);

// Invoke the request at <VERY_HIGH_PRIORITY> priority
// The priority is propagated (see previous page)
rt_controller->edge_alarm();
```

## Design Interlude: The RTORB Interface

- **Problem:** How can the ORB be extended without changing the CORBA::ORB API?

- **Solution:**

- Use `resolve_initial_references()` interface to obtain the extension
- Thus, non real-time ORBs and applications are not affected by RT CORBA enhancements!

```
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

CORBA::Object_var obj =
    orb->resolve_initial_references ("RTORB");

RTCORBA::RTORB_var rtorb =
    RTCORBA::RTORB::_narrow (obj);
// Assuming this narrow succeeds we can henceforth use RT
// CORBA features
```

## Applying SERVER\_DECLARED

- **Problem:** Some operations must always be invoked at a fixed priority  
–e.g., the `Base_Station` methods are not time-critical, so they should always run at lower priority than the `Controller` methods

- **Solution:** Use the RT CORBA `SERVER_DECLARED` priority model

```
CORBA::PolicyList policies (1); policies.length (1);
policies[0] = rtorb->create_priority_model_policy
    (RTCORBA::SERVER_DECLARED, LOW_PRIORITY);

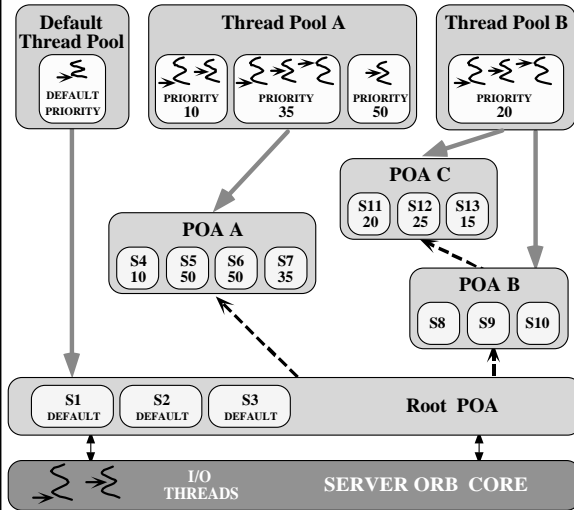
// Create a POA with the correct policies PortableServer::POA_var
base_station_poa =
    root_poa->create_POA ("Base_Station_POA",
        PortableServer::POAManager::_nil (),
        policies);

// Activate the <Base_Station> servant in <base_station_poa>
base_station_poa->activate_object (base_station);
```

- By default, `SERVER_DECLARED` objects inherit the priority of their RTPOA
  - It's possible to override this priority on a per-object basis

**Warning:** `SERVER_DECLARED` does not specify what priority the client and server ORB processing should occur at.

## Supporting Thread Pools Effectively



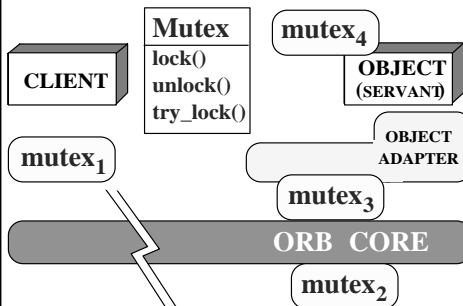
• **Problem:** Pre-allocating threading resources on the server *portably & efficiently*

• e.g., the BaseStation must have sufficient threads for all its priority levels

• **Solution:** Use RT CORBA thread pools to configure server POAs to support

- Different levels of service
- Overlapping of computation & I/O
- Priority partitioning

## Synchronizing Objects Consistently



• **Problem:** An ORB & application may need to use the same *type* of mutex to avoid priority inversions

–e.g., using priority ceiling or priority inheritance protocols

• **Solution:** Use the RTCORBA::Mutex interface to ensure that consistent mutex semantics are enforced across ORB & application domains

```

RTCORBA::Mutex_var mutex = rtorb->create_mutex();
...
mutex->lock();
// Critical section here...
mutex->unlock();
...
rtorb->destroy_mutex(mutex);
    
```

create\_mutex()  
is a factory method

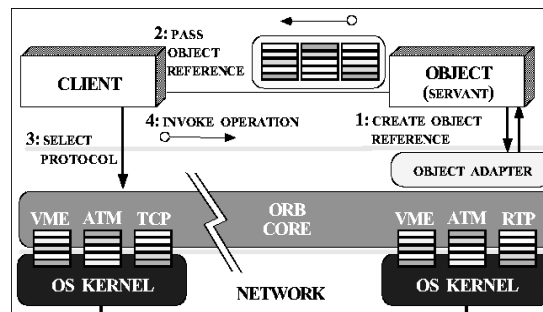
## Configuring Custom Protocols



- **Problems:** Selecting communication protocol(s) is crucial to obtaining QoS
  - TCP/IP is inadequate to provide end-to-end *real-time* response
  - Thus, communication between BaseStation, Controllers, & Drones must use a different protocol
  - Moreover, some messages between Drone & Controller cannot be delayed

- **Solution:** Protocol selection policies

- Both server-side & client-side policies are supported
- Some policies control protocol selection, others configuration
- Order of protocols indicates protocol preference
- Some policies are exported to client in object reference



## Example: Configuring protocols



- First, we create the protocol properties

```
RTCORBA::ProtocolProperties_var tcp_properties
= rtorb->create_tcp_protocol_properties (
    64 * 1024, /* send buffer */
    64 * 1024, /* recv buffer */
    false,    /* keep alive */
    true,     /* dont_route */
    true);   /* no_delay */
```

- Next, we configure the list of protocols to use

```
RTCORBA::ProtocolList plist; plist.length (2);
plist[0].protocol_type = MY_PROTOCOL_TAG;
plist[0].trans_protocol_props =
    /* Use implementation specific interface */
plist[1].protocol_type = IOP::TAG_INTERNET_IOP;
plist[1].trans_protocol_props = tcp_properties;
RTCORBA::ClientProtocolPolicy_ptr policy =
    rtorb->create_client_protocol_policy(plist);
```

## Controlling Network Resources

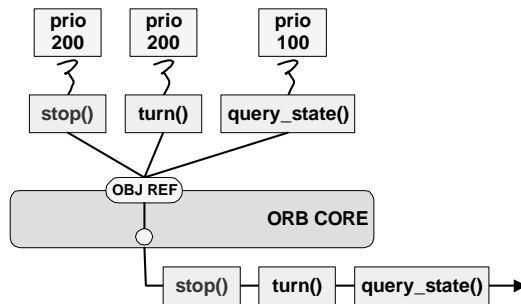


### •Problems:

- Control jitter due to connection setup
- Avoiding request-level (“head-of-line”) priority inversions
- Minimizing thread-level priority inversions

### •Solution: Use explicit binding mechanism

- Priority Banded Connection Policy*
- Invocation priority determines which connection is used



## Priority Banded Connection Policy

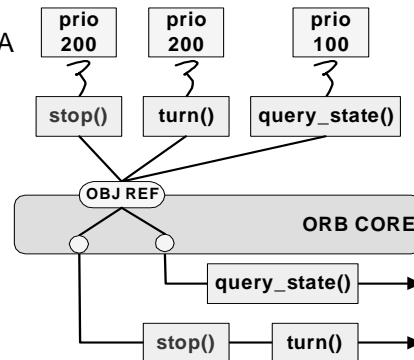


### •Problem: To minimize priority inversions, high-priority operations should not be queued behind low-priority operations

### •Solution: Use different connections for different priority ranges via the RT CORBA PriorityBandedConnectionPolicy

```
// Create the priority bands.
RTCORBA::PriorityBands bands (2);
bands.length (2);
// we can have bands with a range
// of priorities...
bands[0].low = 0;
bands[0].high = 9999;
// ... or just a "range" of 1!
bands[1].low = 10000;
bands[1].high = 19999;
```

```
CORBA::Policy_var policy = rtorb->
create_priority_banded_connection_policy (bands);
```



## Other Relevant CORBA Features



- RT CORBA leverages other advanced CORBA features to provide a more comprehensive real-time ORB middleware solution, e.g.:
  - **Timeouts:** CORBA Messaging provides policies to control roundtrip timeouts
  - **Reliable oneways:** which are also part of CORBA Messaging
  - **Asynchronous invocations:** CORBA Messaging includes support for type-safe asynchronous method invocation (AMI)
  - **Real-time analysis & scheduling:** The RT CORBA 1.0 Scheduling Service is an optional compliance point for this purpose
    - However, most of the problem is left for an external tool
  - **Enhanced views of time:** Defines interfaces to control & query “clocks” (orbos/1999-10-02)
  - **RT Notification Service:** Currently in progress in the OMG (orbos/00-06-10), looks for RT-enhanced Notification Service
  - **Dynamic Scheduling:** Currently in progress in the OMG (orbos/98-02-15) to address additional policies for dynamic & hybrid static/dynamic scheduling

## Controlling Request Timeouts



- **Problem:** Our Controller object should not block indefinitely when trying to stop a drone that's fallen off an edge!
- **Solution:** Override the timeout policy in the Drone object reference

```
// 10 milliseconds (base units are 100 nanosecs)
CORBA::Any val; val <<= TimeBase::TimeT (100000UL);

// Create the timeout policy
CORBA::PolicyList policies (1); policies.length (1);
policies[0] = orb->create_policy
    (Messaging::RELATIVE_RT_TIMEOUT_POLICY_TYPE, val);

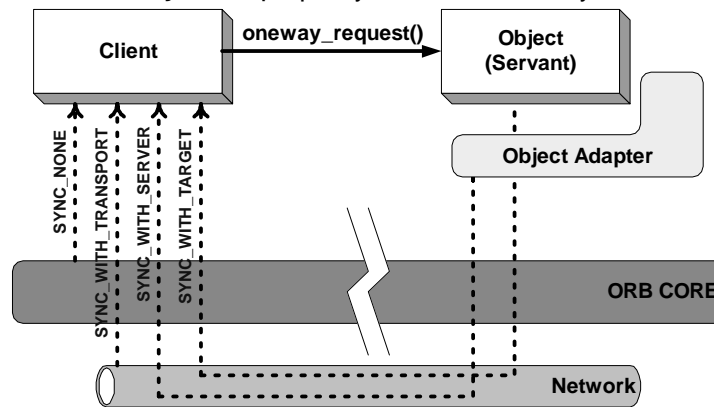
// Override the policy in the drone
CORBA::Object_var obj = drone->_set_policy_overrides
    (policies, CORBA::ADD_OVERRIDE);

Drone_var drone_with_timeout = Drone::_narrow (obj);
try { drone_with_timeout->speed (0); }
catch (const CORBA::TIMEOUT &e) { // Handle exception }
```

## Reliable Oneways



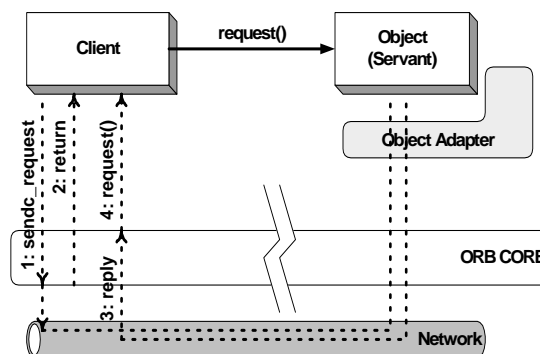
- **Problem:** Traditional CORBA one-way operation semantics are not precise enough for real-time applications
- **Solution:** Use the SyncScope policy to control one-way semantics



## Asynchronous Method Invocation



- **Problem:** clients block waiting for requests to complete, yet some requests take too long, or many requests must be issued simultaneously
- **Solution:** use the AMI interfaces to separate (in time and space) the thread issuing the request from the thread processing the reply





## Agenda

- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary

## Goals of Dynamic Scheduling Real-Time CORBA 2.0



- Generalizes the Real-Time CORBA 1.0
  - Any scheduling discipline may be employed
  - The scheduling parameters may be changed at any time
  - The schedulable entity is a distributable thread
    - Builds upon RT CORBA 1.0 activity concept
    - Spans node boundaries carrying its scheduling context with it



## Scope

- Provides
  - A scheduling framework upon which schedulers can be built
  - Mechanism used for passing information between scheduler instances via GIOP service contexts
    - Does not specify format and content of service contexts
    - On-the-wire interoperability of scheduler implementations is left to future specifications
    - A foundation for future full interoperability
  - An abstraction for distributed real-time programming (distributable thread)



## Scope 2

- Adds interfaces for a small set of well known scheduling disciplines
  - optional compliance points
  - includes interfaces for these disciplines that will allow the development of portable applications
  - interfaces for other scheduling disciplines is left to future specifications
- Does not provide all interfaces necessary for interoperability
  - between scheduling disciplines
  - between different scheduler implementations



## Scope 3

- Does not address
  - fault tolerance
  - content of information propagated along the path of a distributable thread
- Defines ORB/scheduler interfaces for development of portable schedulers
  - Scheduler use of O/S is outside scope of submission
  - Does not provide portability of schedulers except with respect to ORB interactions



## Scheduler

- Realized as an extension to Real-time CORBA
- Utilizes the scheduling needs and resource requirements of one or more applications
- Manages the order of execution on the distributed nodes
- Provides operations for applications to announce their requirements
- Runs in response to specific application requests
  - Defining a new scheduling parameter
  - CORBA invocations (via Portable Interceptor interfaces)



## Scheduler 2

- Utilizes the information provided in these interfaces to execute threads
- Scheduler control of threads
  - Via whatever interfaces the operating system provides
  - Outside of the scope of RTCORBA
- Premise: dist application = set of dist threads
  - May interact in a number of ways
  - Sharing resources via mutexes
  - Sharing transports
  - Parent/offspring relationships
  - etc.



## Scheduler 3

- Interact with Distributable Threads
  - Provides a vehicle for carrying scheduling information across the distributed system
  - Interact with the scheduler at specific scheduling points
    - Application calls
    - Locks and releases of resources
    - Pre-defined locations within CORBA invocations
      - Required because distributable thread may transition to another processor
      - Scheduling information must be reinterpreted on the new processor
    - Creation and termination of a distributable thread



## Scheduler Characteristics

- Does not assume a single scheduling discipline
- Schedulers developed to implement a particular scheduling discipline
- Defines only the interface between the ORB/application and the scheduler
- Intended to foster the development of schedulers that are not dependent on any particular ORB
- While schedulers will likely be dependent on O/S
  - this submission does not address these O/S interfaces



## Scheduler Characteristics 2

- Addresses schedulers that will optimize execution for the application scheduling needs on a processor-by-processor basis
  - ... as the execution of an application distributable thread moves from processor to processor, its scheduling needs are carried along and honored by the scheduler on each processor
  - Doesn't preclude development global optimization schedulers
    - But this proposed specification does not specifically address that type of scheduler.
- Scheduler APIs for a small set of scheduling disciplines is provided

## Scheduler Characteristics 3



- What's missing from the spec that's required for full interoperability?
  - Fully define the semantics of the example scheduling disciplines
  - Define the service contexts to propagate execution context with distributable threads
- Implementation experience is needed before full interoperability is possible

## Scheduling Parameter Elements



- Scheduling parameter = a container of potentially multiple values called scheduling parameter elements
  - the values needed by a scheduling discipline in order to make scheduling decisions for an application
  - A scheduling discipline may have no elements, only one, or several
  - The number and meaning of the scheduling parameter elements is scheduling discipline specific
  - A single scheduling parameter is associated with an executing distributable thread via the `begin_scheduling_segment` operation
  - A thread executing outside the context of a scheduling segment has no scheduling parameter associated with it

## Pluggable Scheduler and Interoperability



- Proposed specification provides a "pluggable" scheduler
- A particular ORB in the system
  - may have any scheduler installed or
  - may have no scheduler
- Applications run on that ORB with a scheduler are "under the purview" of that scheduler

## Pluggable Scheduler and Interoperability 2



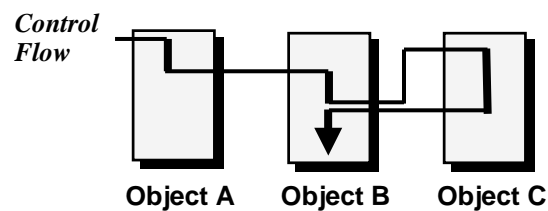
- Application components may interoperate as long as:
  - their ORBs have compatible schedulers installed
    - the schedulers implement the same discipline
    - follow a CORBA standard for that discipline
    - the scheduler implementations use a compatible service context
  - The current proposed specification does not define any standard service contexts
  - Future specifications are anticipated in this area
- A scheduler may choose to support multiple disciplines, but the current proposed specification does not address this

## Distributable Thread

- End-to-end schedulable entity
- Cost-effectiveness demands with application-specific knowledge
- Much of this knowledge can be best captured in the scheduling discipline
- Application-specific scheduling disciplines can be implemented by a pluggable scheduler
- An end-to-end execution model is essential to achieving end-to-end predictability of timeliness
- This is especially important in dynamically scheduled systems

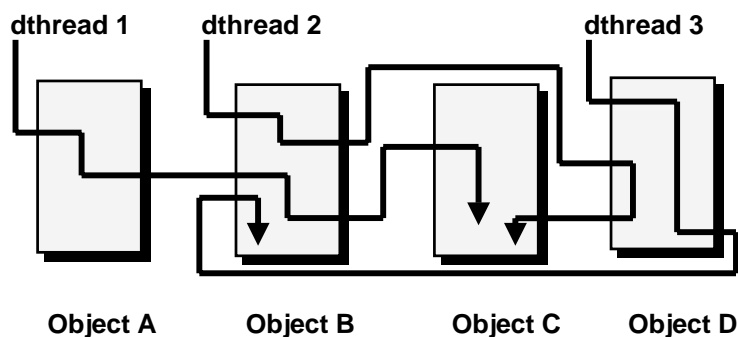
## Distributable Thread 2

- The fundamental abstraction of application execution
- Incorporates the sequence of actions
  - associated with a user-defined portion of the application
  - may span multiple processing nodes
  - but that represents a single logical thread of control



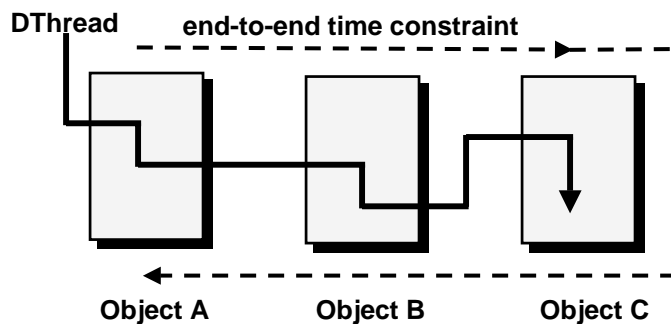
### Distributable Thread 3

- Distributed applications will typically be constructed as several distributable threads that execute logically concurrently
- Locus of execution between significant points in the application



### Distributable Thread 4

- Carries the scheduling context from node to node as control passes through the system
- Might encompass part of the execution of a local (or native) thread or multiple threads executing in sequence on one or more processors





## Distributable Thread 5

- At most one head (execution point) at any moment in time
- CORBA exceptions unwind the distributable thread back to the origin unless caught
- May have a scheduling parameter containing multiple element values associated with it
- These scheduling parameter elements become the scheduling control factor for the distributable thread and are carried with the distributable thread via CORBA requests and replies



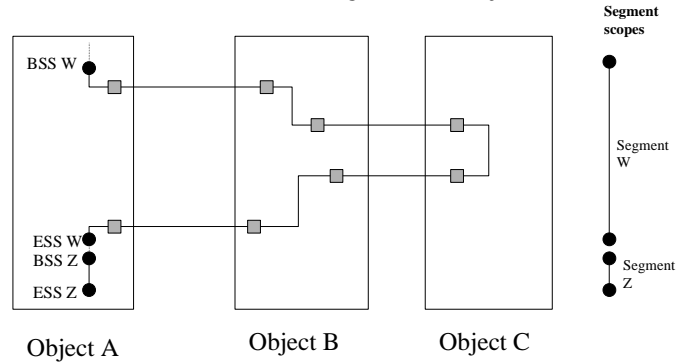
## Distributable Thread 6

- The scheduling parameter elements for a DT are
  - associated with `begin_scheduling_segment()` or `spawn()`
  - updated with `update_scheduling_segment()`
- Branch of control (fork), as occurs with a CORBA oneway invocation
  - The originating distributable thread remains at the client and continues execution according to its eligibility
- Has a globally unique id within the system
  - Fundamental difference from chains of RPCs
  - Via `get_current_id()` operation
  - Reference a distributable thread via `lookup()` operation

## Distributable Thread 7



Distributable Thread Traversing CORBA Objects



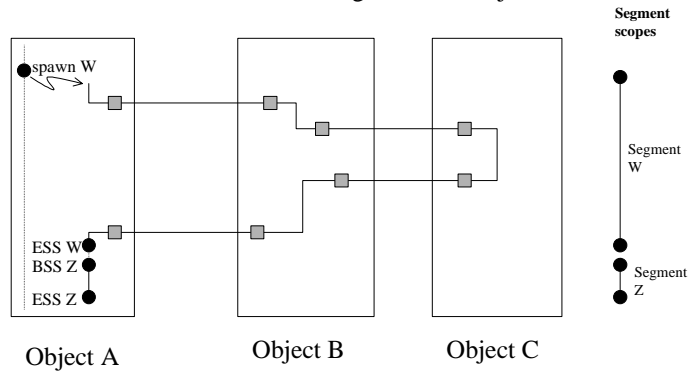
BSS - *begin\_scheduling\_segment*  
 USS - *update\_scheduling\_segment*  
 ESS - *end\_scheduling\_segment*

- Application call
- Portable Interceptor
- Distributable Thread
- ⋯ Normal Thread

## Distributable Thread 8



Distributable Thread Traversing CORBA Objects



- Application call
- Portable Interceptor
- Distributable Thread
- ⋯ Normal Thread

## DistributableThread - IDL

```
module RTScheduling
{
    local interface DistributableThread
    {
        void cancel();
        // raises CORBA::OBJECT_NOT_FOUND if
        // the distributable thread is
        // not known to the scheduler
    };
};
```

## Current – IDL

```
module RTScheduling
{
    local interface Current : RTCORBA::Current
    {
        DistributableThread
        spawn
            (in ThreadAction start,
             in unsigned long stack_size,
             // zero means use the O/S default
             in RTCORBA::Priority base_priority);
    };
};
```

### Current - IDL 2

```
module RTScheduling
{
    local interface Current : RTCORBA::Current
    {
        ...
        typedef sequence<octet> IdType;
        readonly attribute IdType id;
        // a globally unique id

        IdType get_current_id();
        // id of running thread

        DistributableThread lookup(in IdType id);
        // returns a null reference if
        // the distributable thread is
        // not known to the local scheduler
        ...
    };
};
```

### Current - IDL 3

```
module RTScheduling
{
    local interface Current : RTCORBA::Current
    {
        ...
        readonly attribute CORBA::Policy
            scheduling_parameter;

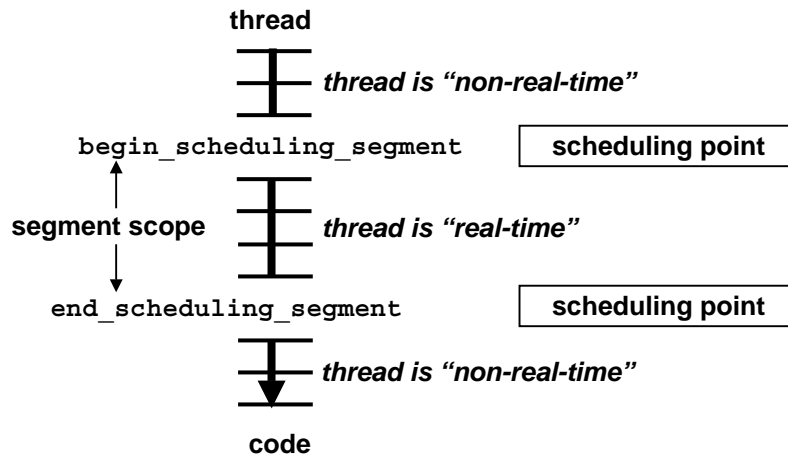
        readonly attribute CORBA::Policy
            implicit_scheduling_parameter;

        typedef sequence<string> NameList;

        readonly attribute NameList
            current_scheduling_segment_names;
    };
};
```

## Scheduling Segments

- Distributable threads consist of one or more (potentially nested) scheduling segments



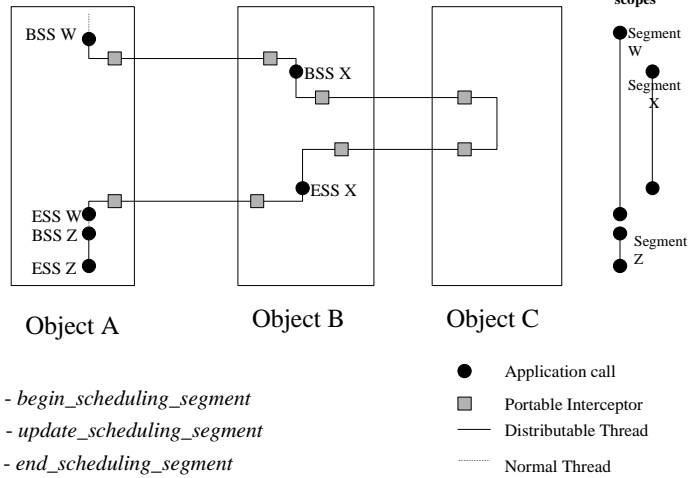
## Scheduling Segments 2

- Represents a sequence of control flow with which a scheduling parameter is associated
- Can be sequential and nested
- Life cycle
  - Starts with `begin_scheduling_segment`
  - Updated with `update_scheduling_segment` (optional)
  - Ends with `end_scheduling_segment`
- May have a segment name
  - Of use by scheduling tools
  - Optional on end statement, allows for error check of nesting

## Scheduling Segments 3



Distributable Thread Traversing CORBA Objects

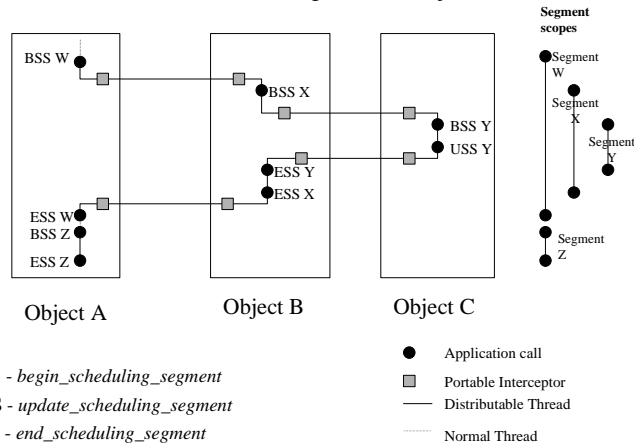


## Scheduling Segments 4



- May span processor boundaries

Distributable Thread Traversing CORBA Objects



## Scheduling Segments - IDL

```
module RTScheduling
{
    local interface Current : RTCORBA::Current
    {
        exception UNSUPPORTED_SCHEDULING_DISCIPLINE {};
        void begin_scheduling_segment
        (in string name,
         in CORBA::Policy sched_param,
         in CORBA::Policy implicit_sched_param)
        raises ( UNSUPPORTED_SCHEDULING_DISCIPLINE );

        void update_scheduling_segment
        (in string name,
         in CORBA::Policy sched_param,
         in CORBA::Policy implicit_sched_param)
        raises ( UNSUPPORTED_SCHEDULING_DISCIPLINE );

        void end_scheduling_segment(in string name);
        ... };
};
```

## Scheduling Points

- Opportunities for scheduler to decide what runs:
  - Creation of a distributable thread (via `begin_scheduling_segment` or `spawn`)
  - Termination or completion of a distributable thread
  - `begin_scheduling_segment`
  - `update_scheduling_segment`
  - `end_scheduling_segment`
  - A CORBA operation invocation, specifically the request and reply interception points provided in the Portable Interceptor specification
  - Creation of a resource manager
  - Blocking on a request for a resource via a call to `RTScheduling::ResourceManager::lock` or `RTScheduling::ResourceManager::try_lock`
  - Unblocking as a result of the release of a resource via a call to `RTScheduling::ResourceManager::unlock`

## Scheduler-Aware Resources

- The application to create a scheduler-aware resource locally via the `create_resource_manager` operation in a `ResourceManager`
- These resources can have scheduling information associated with them via the `register_resource` operation
  - A servant thread could have a priority ceiling if the application were using fixed priority scheduling
  - The scheduler will run when these resources are locked or released, so that the scheduling discipline is maintained
- Any scheduling information associated with these resources is scheduling discipline-specific

## Scheduler & Resources - IDL

```
module RTScheduling
{
    local interface ResourceManager : RTCORBA::Mutex
    {
    };
};
```

## Scheduler & Resources – IDL 2

```
module RTScheduling
{
    local interface Scheduler
    {
        exception INCOMPATIBLE_SCHEDULING_DISCIPLINES {};
        attribute CORBA::PolicyList scheduling_policies;
        readonly attribute CORBA::PolicyList poa_policies;
        readonly attribute string scheduling_discipline_name;

        ResourceManager
            create_resource_manager
                (in string name,
                 in CORBA::Policy scheduling_parameter);

        void set_scheduling_parameter
            (inout PortableServer::Servant resource,
             in string name,
             in CORBA::Policy scheduling_parameter);
    };
};
```

## Exceptions

- CORBA::SCHEDULER\_FAULT
  - Indicates that the scheduler itself has experienced an error
- CORBA::SCHEDULE\_FAILURE
  - Indicates that DT violated the constraints of scheduling parameter
  - Could occur when a deadline has been missed or a segment has used more than its allowed CPU time
- CORBA::THREAD\_CANCELLED
  - Indicates that the DT receiving the exception has been cancelled
  - May occur because a distributable thread cancels another DT thereby causing the CORBA::THREAD\_CANCELLED exception to get raised at the subsequent head of the cancelled DT
- RTScheduling::DistributableThread::UNSUPPORTED\_SCHEDULING\_DISCIPLINE
  - Indicates that the scheduler was passed a scheduling parameter inappropriate for the scheduling discipline(s) supported by the current scheduler

## Well Known Scheduling Disciplines



- Fixed Priority Scheduling
- Earliest Deadline First (EDF)
- Least Laxity First (LLF)
- Maximize Accrued Utility (MAU)

## Fixed Priority Scheduling - IDL



```
module FP_Scheduling
{
    struct SegmentSchedulingParameter
    {
        RTCORBA::Priority base_priority;
    };

    local interface SegmentSchedulingParameterPolicy
    : CORBA::Policy
    {
        attribute SegmentSchedulingParameter value;
    };

    struct ResourceSchedulingParameter
    {
        RTCORBA::Priority resource_priority_ceiling;
    };
};
```

## Fixed Priority Scheduling – IDL 2



```
local interface ResourceSchedulingParameterPolicy
: CORBA::Policy
{
    attribute ResourceSchedulingParameter value;
};

local interface Scheduler : RTScheduling::Scheduler
{
    SegmentSchedulingParameterPolicy
        create_segment_scheduling_parameter
            (in SegmentSchedulingParameter value);

    ResourceSchedulingParameterPolicy
        create_resource_scheduling_parameter
            (in ResourceSchedulingParameter value);
};
};
```

## Earliest Deadline First - IDL



```
module EDF_Scheduling
{
    struct SchedulingParameter
    {
        TimeBase::TimeT deadline;
        long importance;
    };

    local interface SchedulingParameterPolicy : CORBA::Policy
    {
        attribute SchedulingParameter value;
    };

    local interface Scheduler : RTScheduling::Scheduler
    {
        SchedulingParameterPolicy
            create_scheduling_parameter
                (in SchedulingParameter value);
    };
};
```

## Least Laxity First - IDL

```
module LLF_Scheduling
{
    struct SchedulingParameter
    {
        TimeBase::TimeT deadline;
        TimeBase::TimeT estimated_initial_execution_time;
        long importance;
    };
    local interface SchedulingParameterPolicy : CORBA::Policy
    {
        attribute SchedulingParameter value;
    };

    local interface Scheduler : RTScheduling::Scheduler
    {
        SchedulingParameterPolicy
        create_scheduling_parameter
        (in SchedulingParameter value);
    };
};
```

## Maximize Accrued Utility - IDL

```
module Max_Utility_Scheduling
{
    struct SchedulingParameter
    {
        TimeBase::TimeT deadline;
        long importance;
    };

    local interface SchedulingParameterPolicy : CORBA::Policy
    {
        attribute SchedulingParameter value;
    };

    local interface Scheduler : RTScheduling::Scheduler
    {
        SchedulingParameterPolicy
        create_scheduling_parameter
        (in SchedulingParameter value);
    };
};
```



## Conformance

- Makes no changes to the Real-time CORBA 1.0 non-optional IDL (ie. other than changes to Scheduling Service IDL)
- Implementation of this specification is optional for implementations of the Real-time CORBA 1.0 specification
  - Implementation of the basic Dynamic Scheduling infrastructure is the most basic form of compliance
  - Implementation of each scheduler is optional and separate compliance point



## Changes to core

- Adds three system exceptions to core CORBA
  - CORBA::SCHEDULER\_FAULT
  - CORBA::SCHEDULE\_FAILURE
  - CORBA::THREAD\_CANCELLED



## Summary

- Extends Real-time CORBA 1.0 to dynamic systems
- Provides replaceable scheduling
- Allows users to control timeliness of distributed real-time applications



## Agenda

- Motivation
- Introduction to Real-time Principles
- Challenges in Distributing Real-time Systems
- Overview of Real-time CORBA 1.0
- Applying Real-time CORBA 1.0 to problem domains
- Real-time CORBA 1.0 Architecture
- Example Application
- Real-time CORBA 2.0: Dynamic Scheduling
- Summary



## Future of specification

- Real-time CORBA 1.0 Revision Task Force
  - RTF Chair: [bill.beckwith@ois.com](mailto:bill.beckwith@ois.com)
  - OMG mailing list: [rtcorba-rtf@omg.org](mailto:rtcorba-rtf@omg.org)
  - To add an issue: [issues@omg.org](mailto:issues@omg.org)
  - RTF Public Comment Deadline: September 1, 2002
  - RTF Revision Deadline: November 1, 2002
- Dynamic Scheduling Finalization Task Force
  - FTF Chair: [bill.beckwith@ois.com](mailto:bill.beckwith@ois.com)
  - OMG mailing list: [rtcorba-rtf@omg.org](mailto:rtcorba-rtf@omg.org)
  - To add an issue: [issues@omg.org](mailto:issues@omg.org)
  - FTF Public Comment Deadline: March 1, 2002
  - FTF Revision Deadline: April 1, 2002



## Further Information

- Real-time CORBA 1.0
  - OMG CORBA 2.6.1 specification, chapter 24
- Real-time CORBA 2.0: Dynamic Scheduling
  - <http://www.omg.org/cgi-bin/doc?ptc/01-08-34.pdf>
- Tests of various communication technologies
  - Lockheed Martin Advanced Technology Labs' Agent and Distributed Objects Quality of Service (QoS)
  - <http://www.atl.external.lmco.com/projects/QoS/>
- Information about CORBA for Real-Time, Embedded, and High Performance Applications
  - <http://www.ois.com/resources/corb-1.asp>
- Real-time and embedded CORBA discussion forum
  - <http://www.realtime-corba.com>