Distributed Real-Time Combat Systems

Russ Johnston
SPAWAR Systems Center
russ@spawar.navy.mil

1

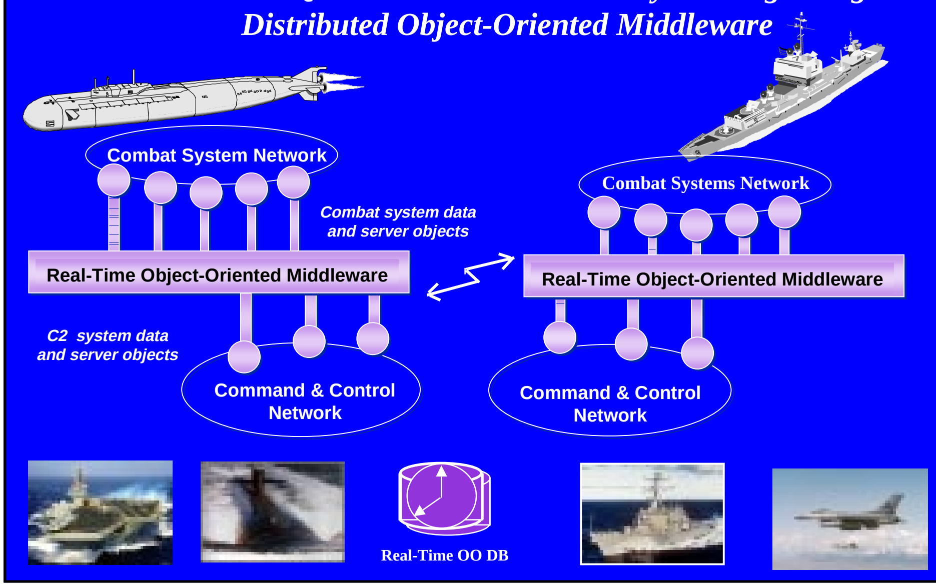
Operational Issues

- **Combat Systems and C2 Systems need to share data and functionality under *real-time constraints*.**
- **A need to maintain a consistent set of data at a specified level of QoS with the ability to enforce QoS tradeoffs in the middleware, databases, operating systems and networks.**
- **There is a need for heterogeneous data access, data sharing and data distribution across multiple platforms which may have different QoS requirements.**
- **Distributed systems need to meet specified timing constraints with the ability to negotiate for the QoS initially statically and then dynamically.**

2

Operational Issues

Real-Time QoS Data and Functionality Sharing Using Distributed Object-Oriented Middleware

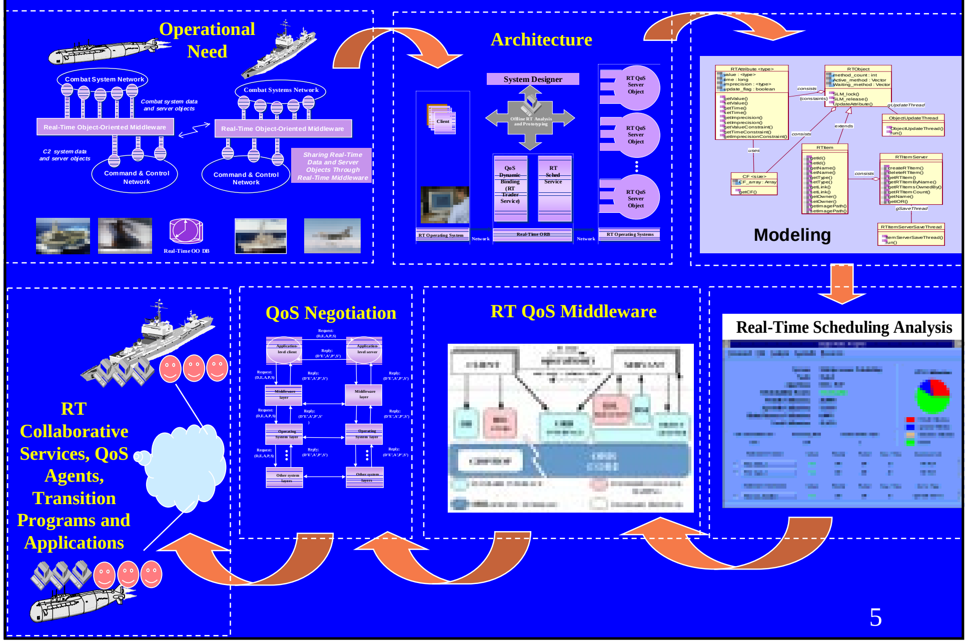


Technical Approach

- Design real-time QoS model which enables the expression of time critical concepts and levels of QoS.
- Develop a multi-layered QoS negotiation schema which will provide a mathematical basis for synthesizing the parameters (real-time, accuracy, fault-tolerance & security).
- Develop a scheduling & analysis capability which provide metrics with respect to the synthesis of the parameters.
- Design a modular framework to support the system design and enable the insertion of custom solutions.
- Design real-time collaborative services which insure a consistent and current view of the data backed by guarantees and enforcement of timing constraints.

4

Technical Approach



Accomplishments

- Implementation of the network-centric real-time QoS middleware algorithms and mechanisms. (static scheduling, dynamic scheduling, load shedding/ reduction, dynamic binding, data replication)
- Implementation of QoS model in International Standard Unified Modeling Language (UML)
- Implementation of real-time QoS analysis tool with input from UML model and output to RT QoS middleware
- Implementation of QoS negotiations among real-time agents. (Accuracy vs Real-Time)
- Transitions:
  - Military programs (Coalition Forces, Virginia Class Sub, COF, Raytheon, Lockheed/Martin, Boeing, Mitre, TRW)
  - International standards (RT CORBA 1.0, RT CORBA 2.0, UML)
  - Commercial products (Analysis Tool, Scheduling Service, UML tools, WindRiver RTOS, Rational Software tools, Lineo Embedded Linux, OIS ORB)
  - Academic publications (IEEE TDPS, 2 Real-Time Systems Journal, conferences) 6

Operational Payoff

• The ability for combat systems and C2 systems to share data and functionality under real-time QoS constraints.

• The ability to design and implement systems using a COTS middleware approach that many programs are adopting.

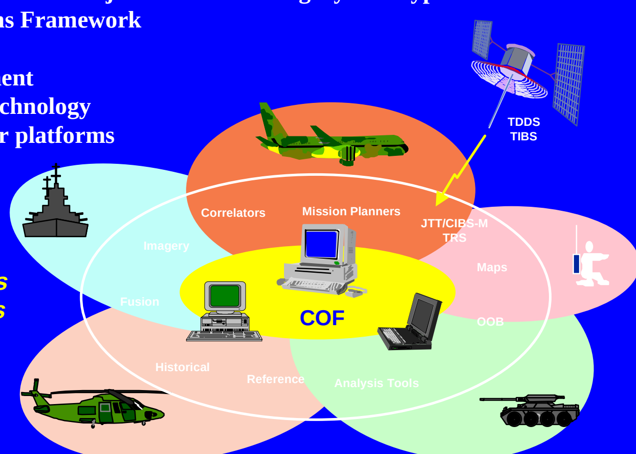
• New algorithms, mechanisms, and analysis techniques for distributed real-time QoS middleware.

Collaboration: Real-Time Support In Common Object Framework (COF)

Trudy Morgan, SPAWAR System Center SD

- Enhanced Data Integration - Object interfaces to legacy data types
- CORBA based Systems Framework
- COTS/GOTS Re-use
- Distributed Environment
- Extendable for new technology
- Transportable to other platforms

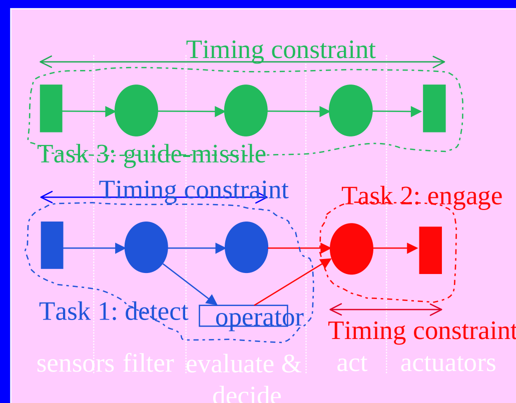
DRCS Real-time QoS middleware enforces real-time constraints in COF CORBA framework



Collaboration: Adaptive Resource Management in Asynchronous Real-Time Distributed Systems

Binoy Ravindran, Virginia Tech University

- Real-time computer systems for mission management
- Significant run-time uncertainties
  - Execution times, communication delays, event arrivals, etc.
- Require decentralization
  - Distributed application resources
  - Survivability
  - Meeting response time requirements

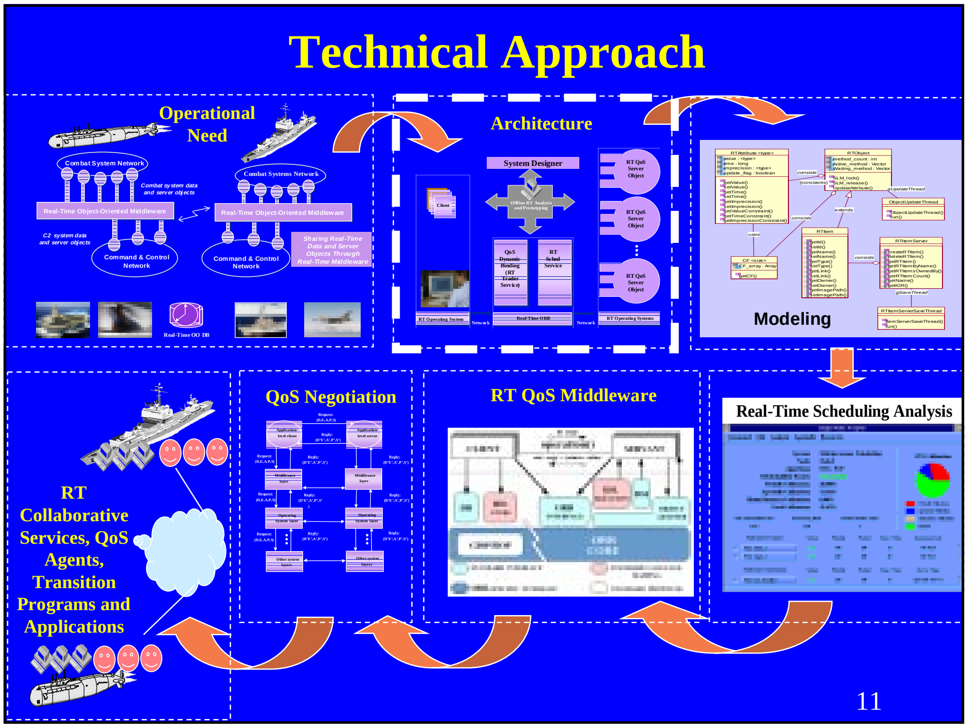


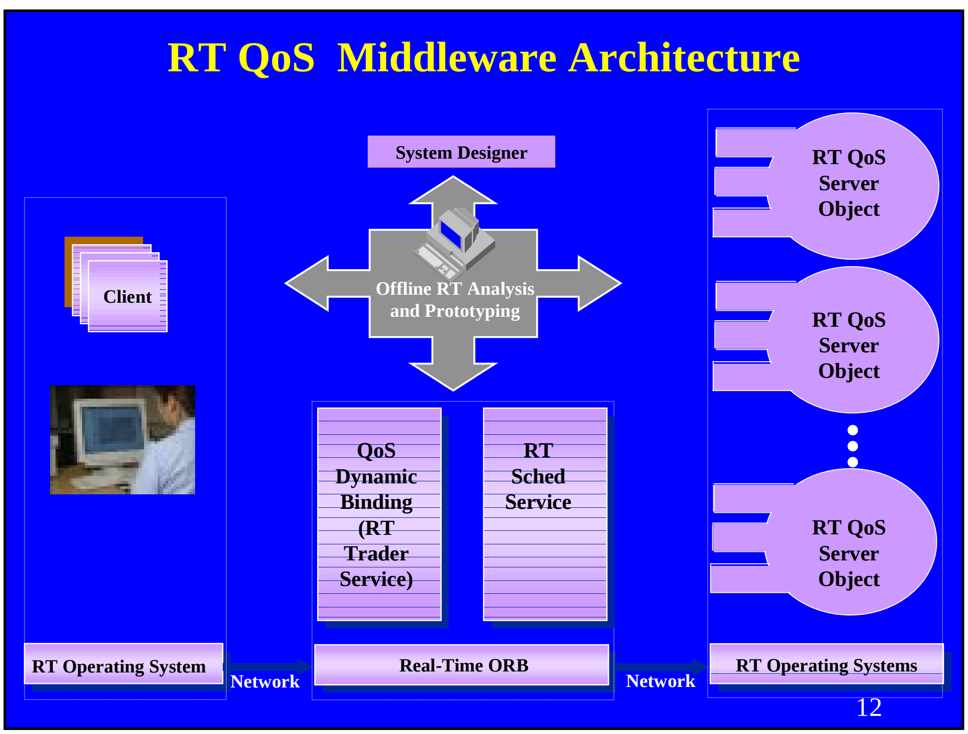
DRCS real-time QoS middleware
algorithm and methodology sharing

Technical Solution

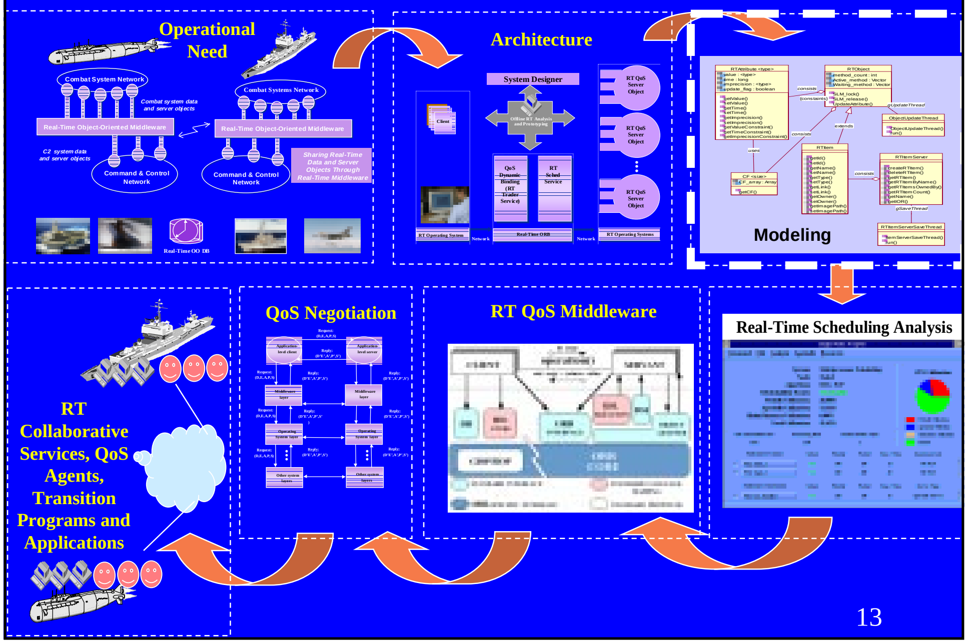
Dr. Victor Fay-Wolfe

University of Rhode Island
wolfe@cs.uri.edu


Technical Approach
Operational Need
Combat System Network
Combat Systems Network
Real-Time Object-Oriented Middleware
Command & Control Network
Architecture
System Designer
Modeling
QoS Negotiation
RT QoS Middleware
Real-Time Scheduling Analysis
RT Collaborative Services, QoS Agents, Transition Programs and Applications
11


RT QoS Middleware Architecture
System Designer
Client
Offline RT Analysis and Prototyping
QoS Dynamic Binding (RT Trader Service)
RT Sched Service
RT QoS Server Object
RT QoS Server Object
RT QoS Server Object
RT Operating System
Network
Real-Time ORB
Network
RT Operating Systems
12

Technical Approach



13

RT QoS Modeling

- UML modeling of timing and QoS constraints
- UML modeling of real-time QoS objects
- Analysis modeling of RT QoS middleware

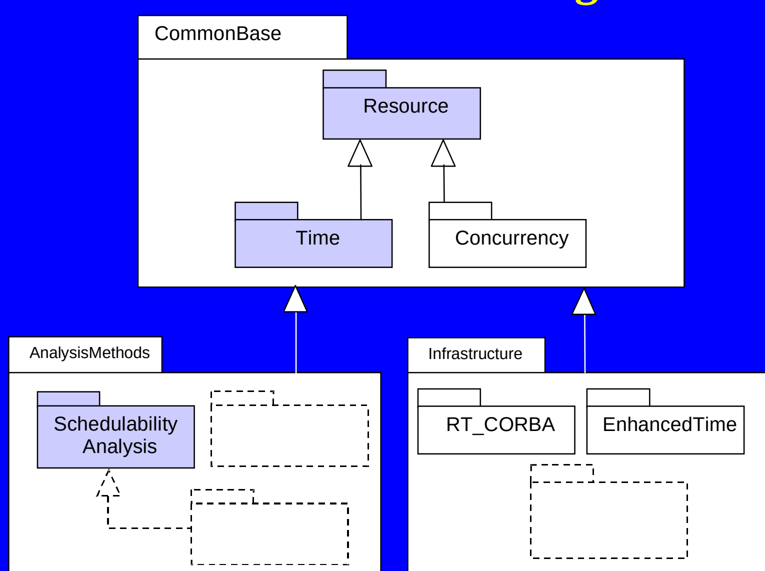
14

QoS in Real-Time UML

- Responded to OMG Request for Proposal For RT Unified Modeling Language (UML)
- Not an extension to the UML metamodel, but a set of domain profiles for UML
- Goals
  - Enable the construction of models that could be used to make quantitative predictions regarding the characteristics of schedulability, performance and time
  - Facilitate communication of design intent between developers in a standard way
  - Enable interoperability between various analysis and design tools

15

Profile Domain Packages



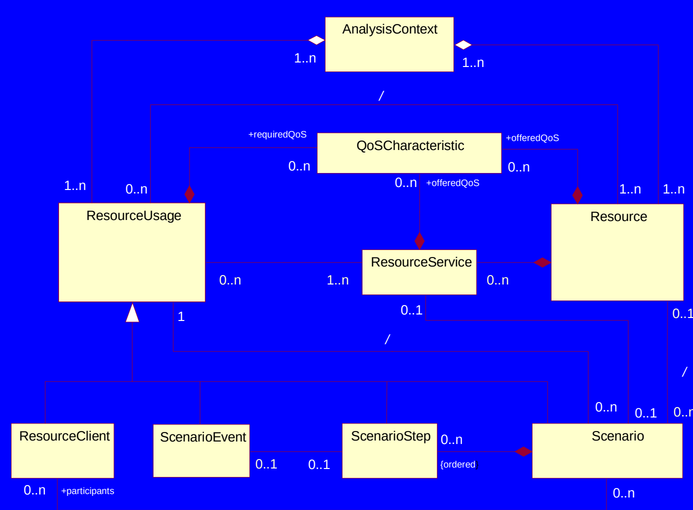
16

Modeling Resources - The QoS Framework

- ***Resource***: a model element that has some finite properties
  - reflects some finite physical quantity
  - may be logical (e.g., buffers) or physical
  - resources offer services (client-server model)
  - need to quantify the demand/supply of services
- ***Quality of Service*** (QoS): a (usually quantitative) specification of:
  - the level of service required by a client from a resource or
  - the level of service offered by a resource to its clients

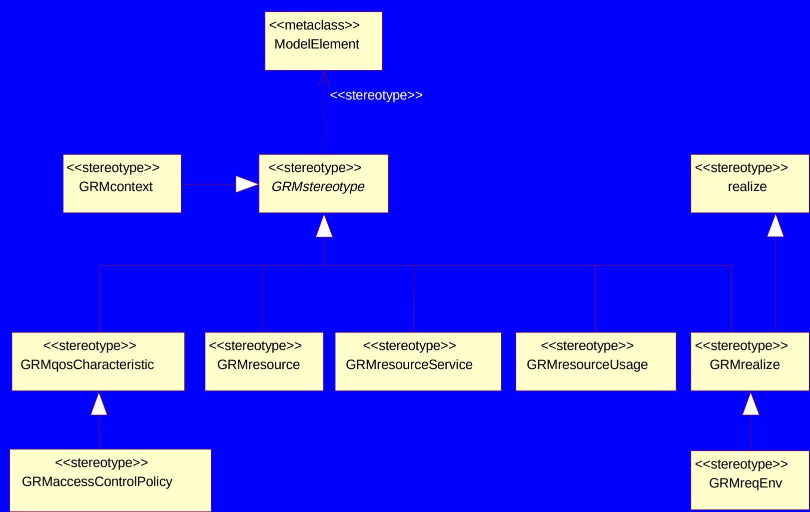
17

The General Resource Model



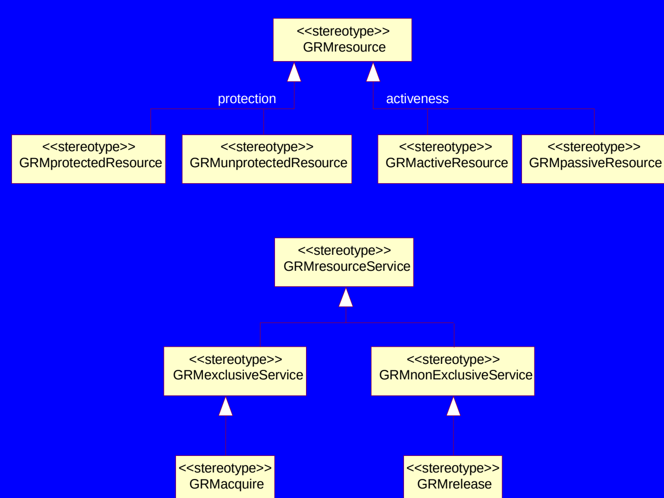
18

Profile Stereotypes for Resource Model



19

Profile stereotypes (cont.)



20

Sample Schedulability Analysis (SA) Profile

SAEvent:

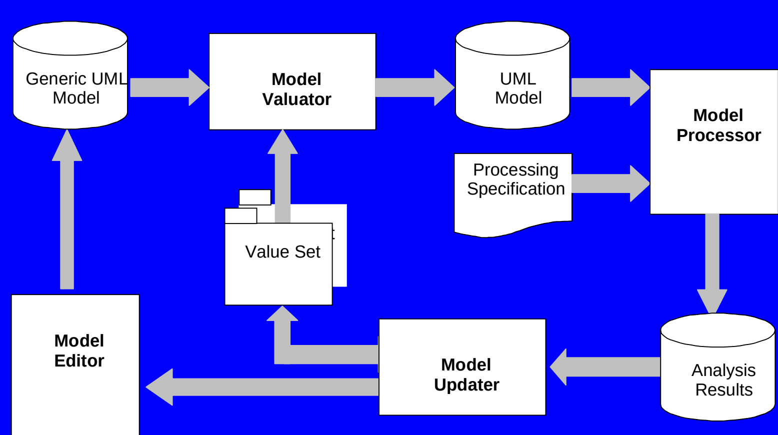
SARelativeDeadline = 400 ms

SAInstances = 1

SAOccurrencePattern = “exponential” 625

21

Model Processing Framework



22

Why Apply RMA to UML?

- UML addresses system structure and function
  - Multiple views of the system
  - Encourages top-down design
  - Ignores timing characteristics
- RMA addresses system timing characteristics
  - Uses the same system system structure
  - Ignores the functional characteristics
- Two abstractions for the same system
  - Same structure
  - Different thinking

23

The Usual Approach

- Focus on function
  - Meet the functional requirements
  - Timing is a requirement, but is too difficult to address
- Timing issues are addressed late in the process
  - Usually during system integration
  - The symptom … sporadic failures
- Architectural changes are very expensive
  - Most timing problems require architectural changes
  - Usual consequence is an “over engineered” system

24

A Better Approach

- Integrate timing considerations throughout the process
  - Start early in the design
  - Refine and update timing in concert with functional refinements
- Make timing specifications visible in the UML
  - Removes the dichotomy of functional and timing abstractions
  - Let tools construct the timing model
- Require timing validation during design, unit test, and integration

25

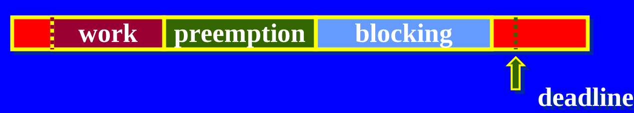
What Does RMA Reveal?

- RMA is static analysis
  - Not dynamic
  - Not simulation
- RMA establishes a bound for schedulability
  - A system is guaranteed to be schedulable within the bound
  - A system may run outside the bound, but is not guaranteed
- RMA shows how system resources are used
  - CPU and other “active” resources
  - Passive resources
  - Physical and logical resources

26

When is a Task Schedulable?

- A task is schedulable if its worst case completion time is less that its deadline.
- Worst case completion time accounts for three classes of time
  - Work - execution of the task itself
  - Preemption - execution of higher priority tasks
  - Blocking - execution by lower or equal priority tasks when the task is ready for execution



27

Analysis Results

- Schedulability
  - System, node, and task level
  - Utilization
  - Worst case completion times
- Analysis quality
  - Spare capacity
  - Blocking - total and by resource
  - Stability - behavior in overload
- “What if” analysis tools
  - The tool suggests changes
  - Rapid assessment for timing and architectural changes

28

More Than Just RMA Analysis

- The analysis alone is very useful
  - Provides interesting information about a system
  - Enables “Timing Design”
- We can apply the analysis to another aspect of the problem
- We can answer the question, “How do I assign threads and priorities?”
  - Assign message priorities
  - Assign execution priorities
  - Assign capsules/activities to physical threads

29

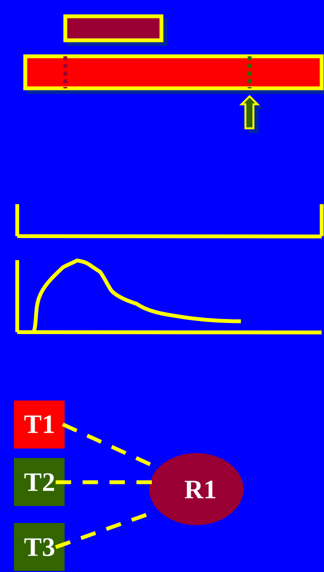
Example System

- Telemetry System
  - Takes real-time data from a set of sensors
  - Filters and processes the data
  - Displays the filtered data to operator
- Display must be updated every 60 ms
- Telemetry data must be gathered every 100ms
- Filtered data update interval is 200ms

30

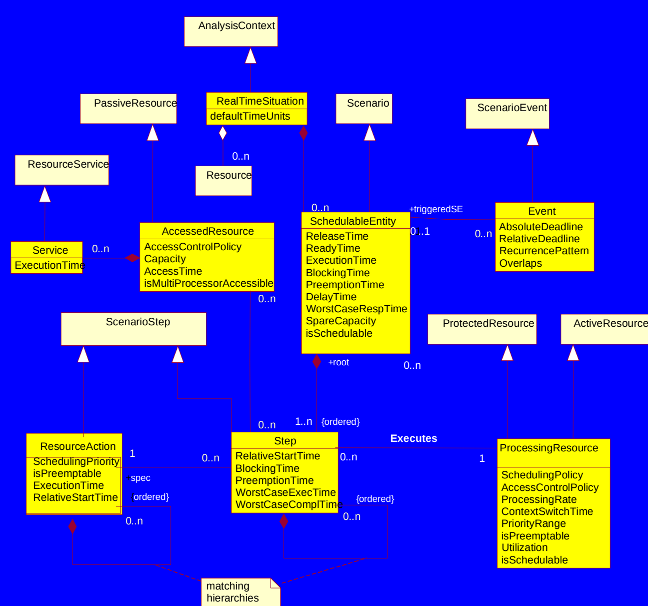
RMA Schedulable Entity Model

- Timing constraints
  - Execution time
  - Deadline
  - Deadline type
- Arrival pattern
  - Periodic
  - Sporadic (hard deadline)
  - Aperiodic (soft deadline)
- Shared Resources
  - Usage pattern
  - Priority inversion



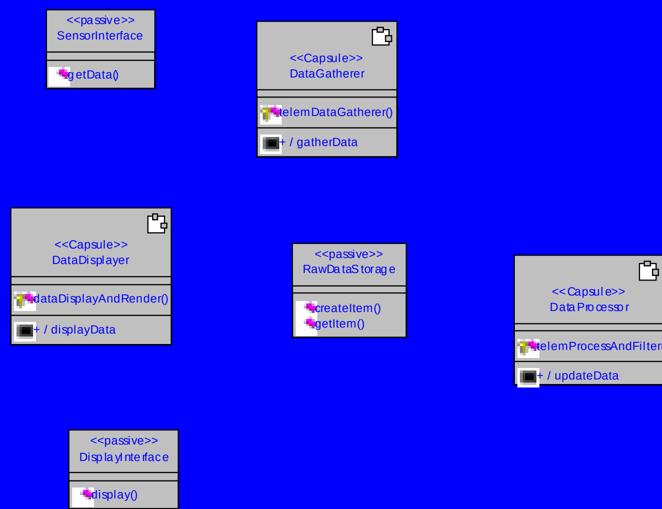
31

RT UML Schedulability Model



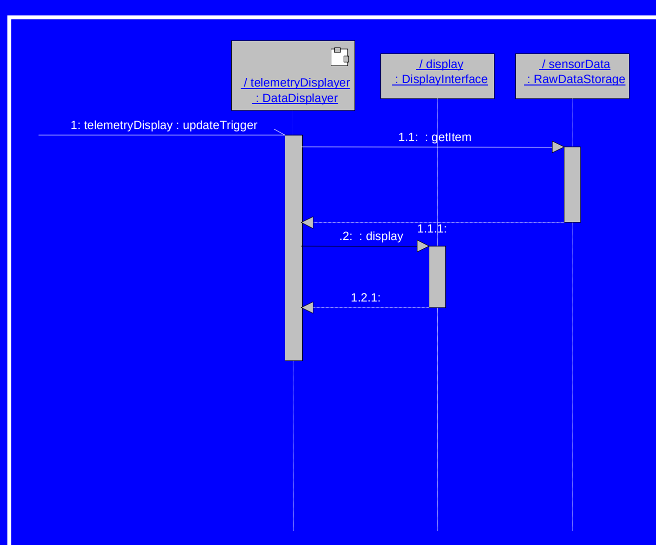
32

RT UML Example: Class Diagram



33

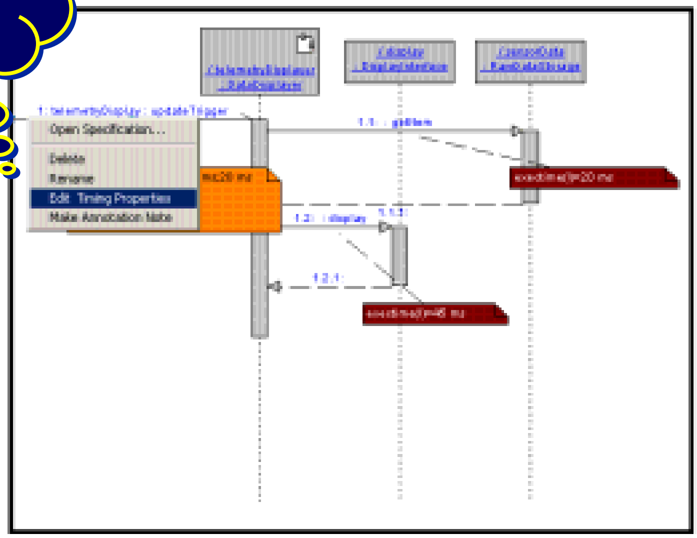
RT UML Example: Sequence Diagram



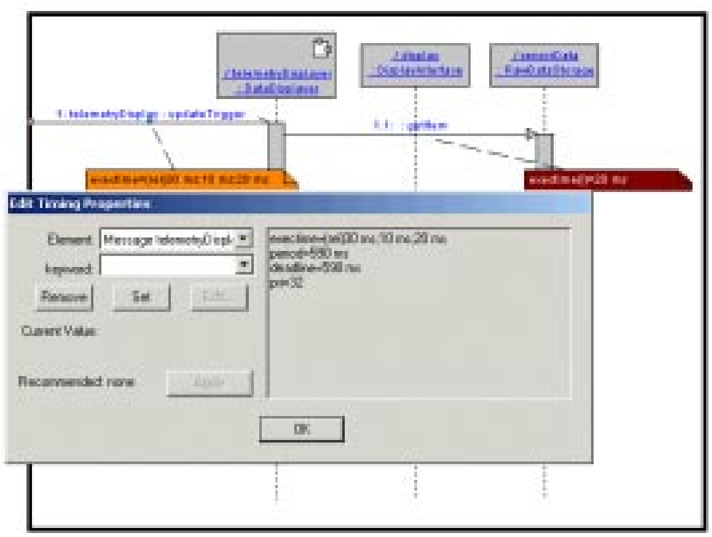
34

Sequence Diagram; Edit Timing Properties

Select any element; popup this menu

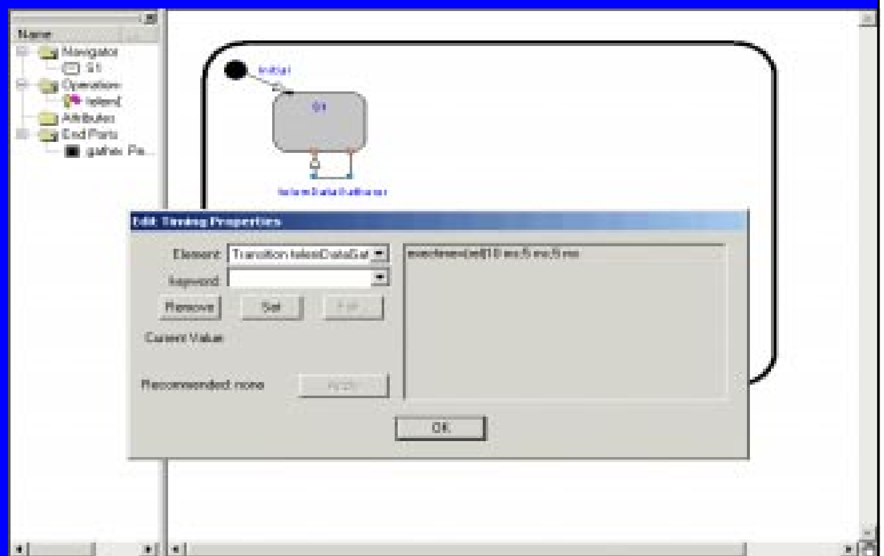


Edit Timing Properties Dialog



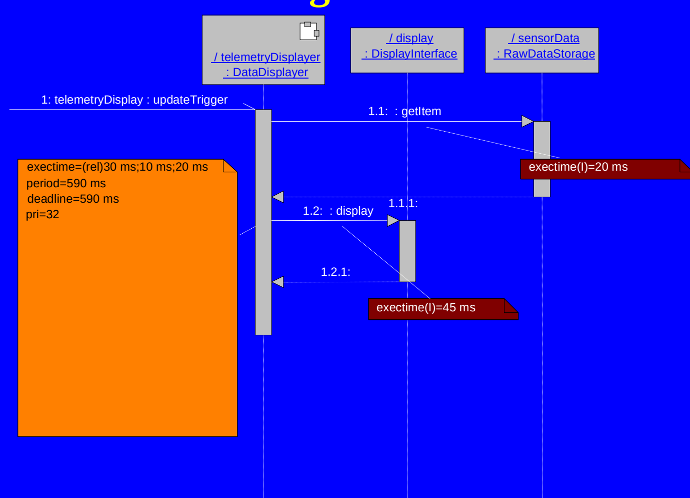
36

Timing Properties on Transitions



37

RT UML Example: Annotated Sequence Diagram



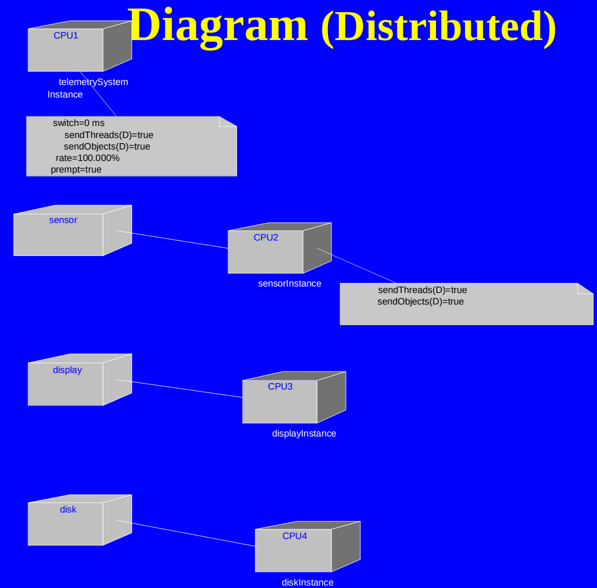
38

RT UML Example: Distributed System

- You can design a distributed system with Rose RealTime
  – Use a deployment diagram to identify the processors
  – Assign package instances to specific processors
- DRCS tools can analyze a distributed system
  – Use end-to-end analysis or ROSA
- Network latency is expressed as another annotation on messages in the Sequence Diagram

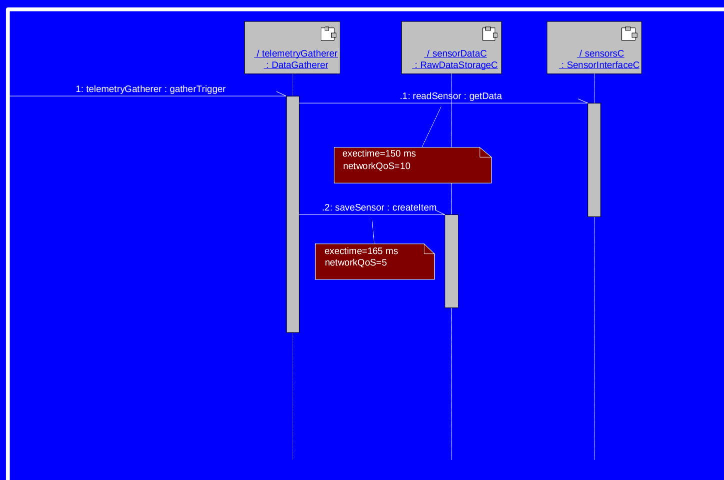
39

RT UML Example: Deployment Diagram (Distributed)



40

RT UML Example: Sequence Diagram (Distributed)

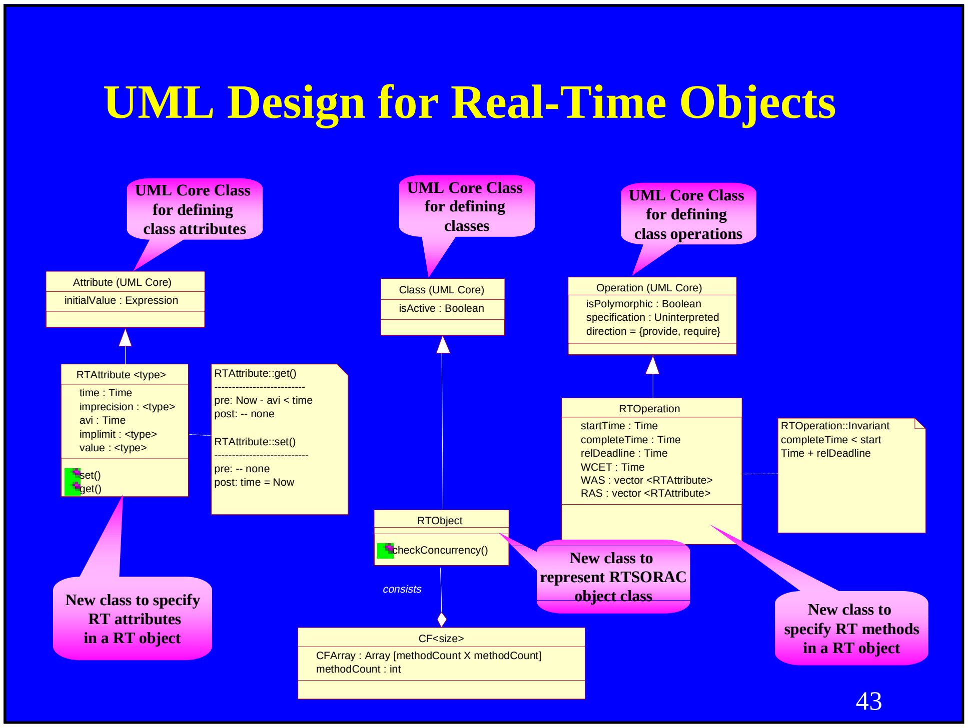


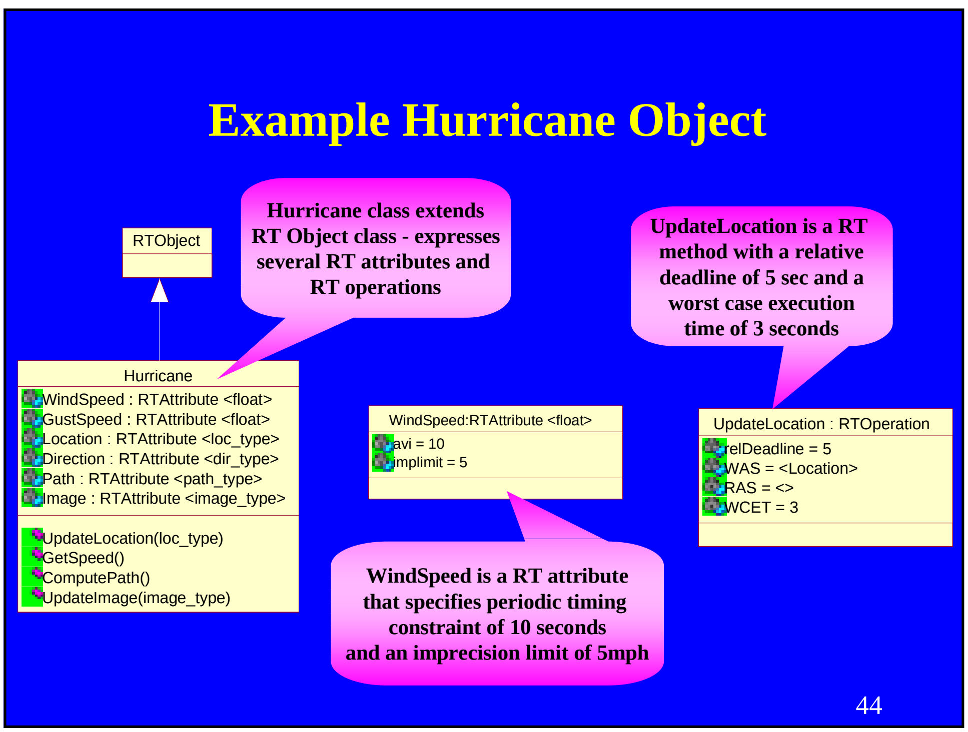
41

Real-Time Objects

- *Definition*: A real-time object is an object in the environment that must be updated periodically to remain valid
- Examples:
  - radar or sonar readings
  - weather data

42


UML Design for Real-Time Objects
UML Core Class for defining class attributes
UML Core Class for defining classes
UML Core Class for defining class operations
Attribute (UML Core)
initialValue : Expression
Class (UML Core)
isActive : Boolean
Operation (UML Core)
isPolymorphic : Boolean
specification : Uninterpreted
direction = {provide, require}
RTAttribute <type>
time : Time
imprecision : <type>
avi : Time
implimit : <type>
value : <type>
set()
get()
RTAttribute::get()
----
pre: Now - avi < time
post: -- none
RTAttribute::set()
----
pre: -- none
post: time = Now
RTOperation
startTime : Time
completeTime : Time
relDeadline : Time
WCET : Time
WAS : vector <RTAttribute>
RAS : vector <RTAttribute>
RTOperation::Invariant
completeTime < start
Time + relDeadline
RTObject
checkConcurrency()
consists
CF<size>
CFArray : Array [methodCount X methodCount]
methodCount : int
New class to specify RT attributes in a RT object
New class to represent RTSORAC object class
New class to specify RT methods in a RT object
43


Example Hurricane Object
RTObject
Hurricane class extends RT Object class - expresses several RT attributes and RT operations
UpdateLocation is a RT method with a relative deadline of 5 sec and a worst case execution time of 3 seconds
Hurricane
WindSpeed : RTAttribute <float>
GustSpeed : RTAttribute <float>
Location : RTAttribute <loc_type>
Direction : RTAttribute <dir_type>
Path : RTAttribute <path_type>
Image : RTAttribute <image_type>
UpdateLocation(loc_type)
GetSpeed()
ComputePath()
UpdateImage(image_type)
WindSpeed:RTAttribute <float>
avi = 10
implimit = 5
UpdateLocation : RTOperation
relDeadline = 5
WAS = <Location>
RAS = <>
WCET = 3
WindSpeed is a RT attribute that specifies periodic timing constraint of 10 seconds and an imprecision limit of 5mph
44

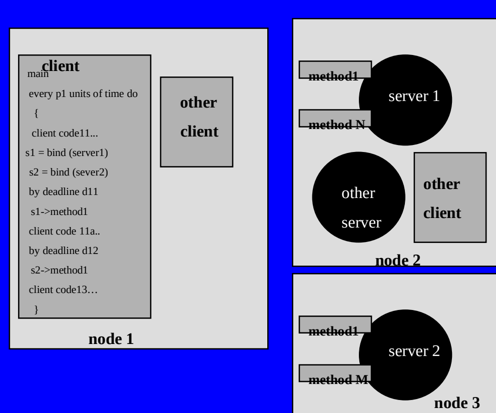
Modeling RT QoS Middleware

- A Real-Time QoS Middleware system is modeled from the client point of view.
- A client is partitioned into a sequence of dependent tasks.
- Middleware Services are modeled as resources.

45

Modeling Real-Time QoS Middleware Application

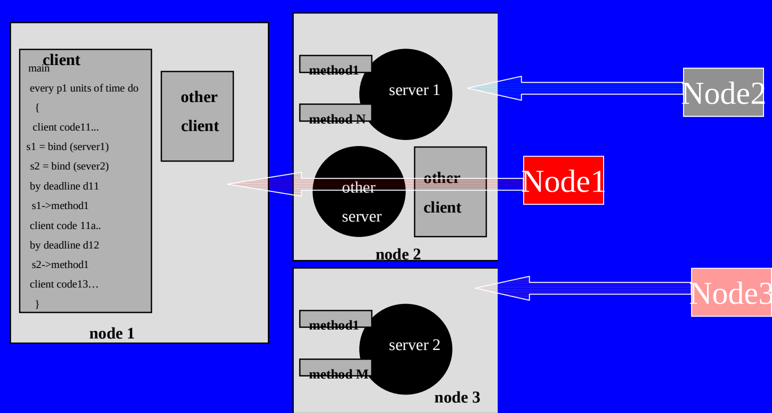
- Clients and Servers



46

Modeling Real-Time QoS Middleware Application

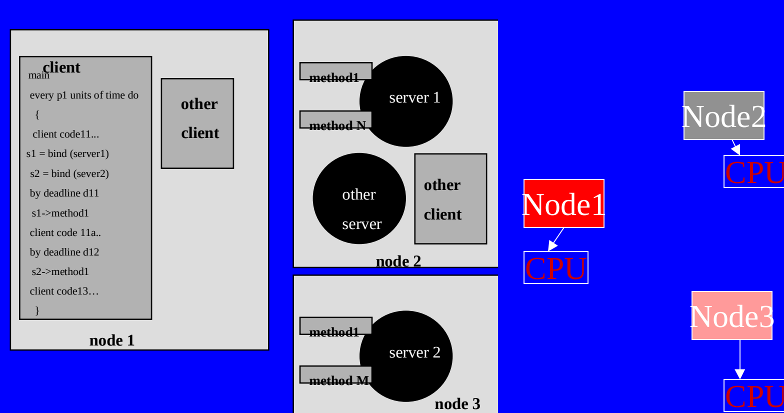
- **All Servers as well as ORB and Services are represented by *resources*.**



47

Modeling Real-Time QoS Middleware Application

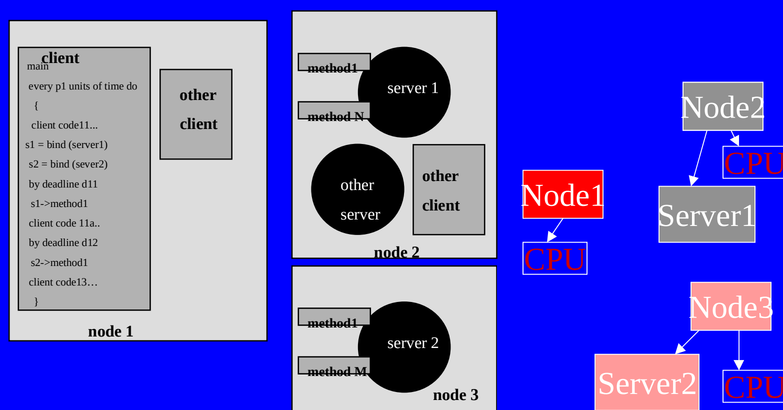
- **Each node has a CPU as the processor.**



48

Modeling Real-Time QoS Middleware Application

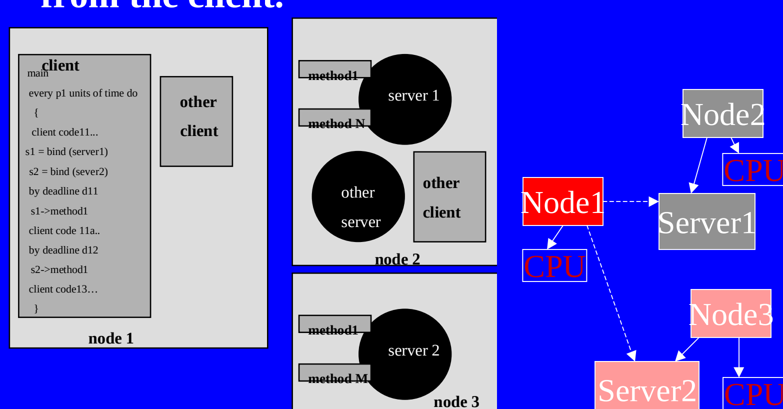
- The servers are represented as resources.



49

Modeling Real-Time QoS Middleware Application

- The dotted arrows indicate access to the servers from the client.



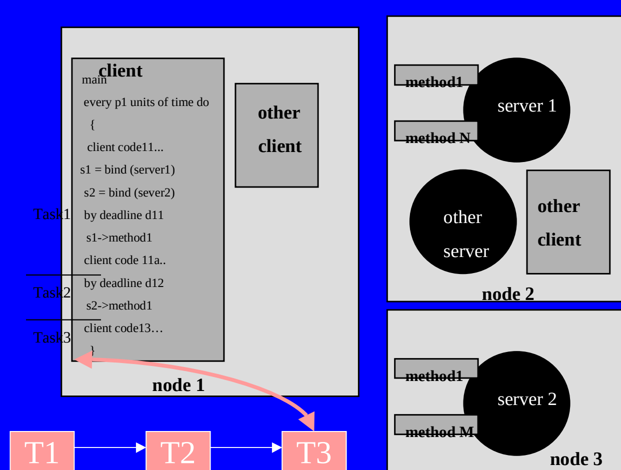
50

Modeling Real-Time QoS Middleware Application

- CORBA Clients can't be mapped directly to *tasks* since Rate Monotonic Analysis does not support the analysis of the
  - *Intermediate Deadlines*,
  - *Network Delay*.
- Each Client with N intermediate deadlines will be modeled as N+1 dependent *tasks*.

51

Modeling Real-Time QoS Middleware Application



52

Modeling Real-Time QoS Middleware Application

- Analysis: *END-TO-END*

  - enables task dependencies analysis
- Priority assignment mechanism: *Deadline Monotonic (DM)*
  - Shorter the relative deadline - higher the priority.
- Resource access protocol: *DASPCP*
  - deadlock-free
  - limited blocking time
  - include network delay

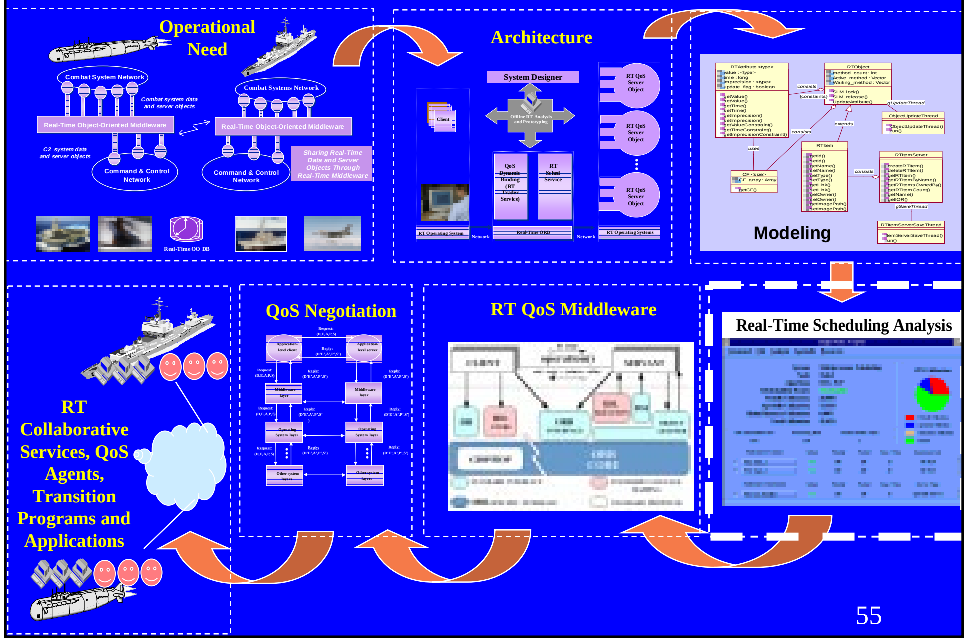
53

Modeling Real-Time QoS Middleware Application

- Analysis assumes that a system has 32K priority levels (consistent with RT CORBA standard).
- Typical operating systems do not permit that many priorities.
- DRCS maps the priority system into the limited priority systems on the network.
- The mapping minimizes the resulting priority inversions.

54

RT QoS Analysis



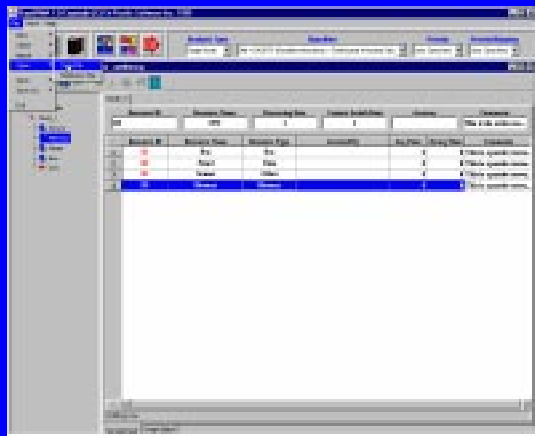
55

Modeling: Task Graph Editor

There are two components to our model. The task graph identifies the tasks of the system to be modeled.

56

Modeling: Resource Graph Editor



The resource graph represents the resources in the system. This is a distributed system with global shared resources.

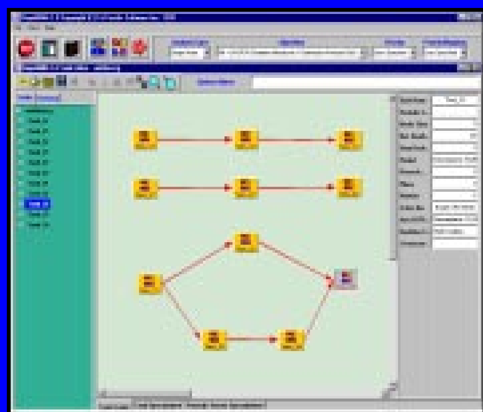
57

Modeling: Schedulability Analyzer

The Scheduler tests the system for schedulability (all tasks meet all their deadlines). There are a variety of scheduling analysis algorithms in the scheduler for various system architectures and scheduling methods.

58

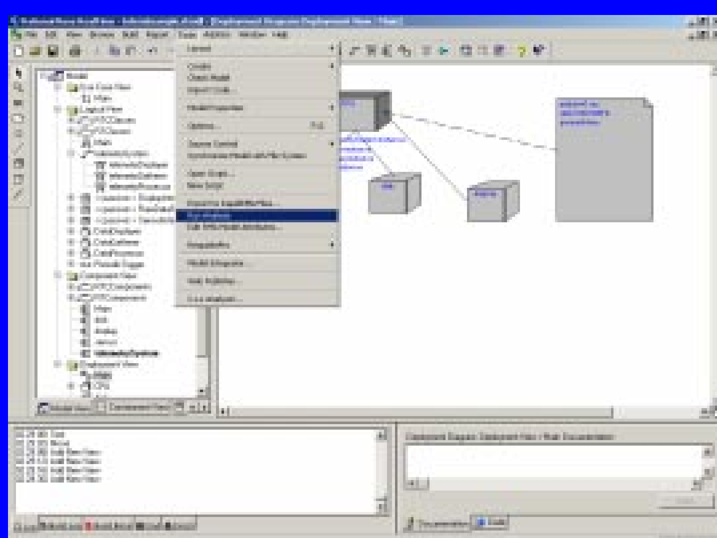
Modeling: End-to-End Analysis



End-to-End Scheduling analyzes the schedulability of a system with one or more paths of execution defined by a series of dependencies between tasks.

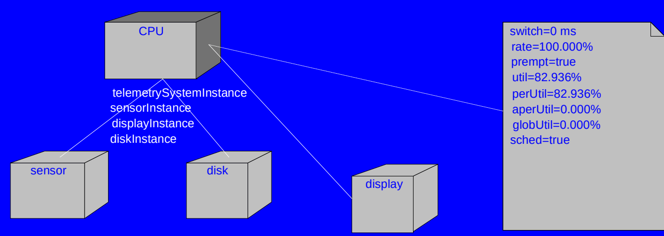
59

RT UML Example: Run RMA Analysis



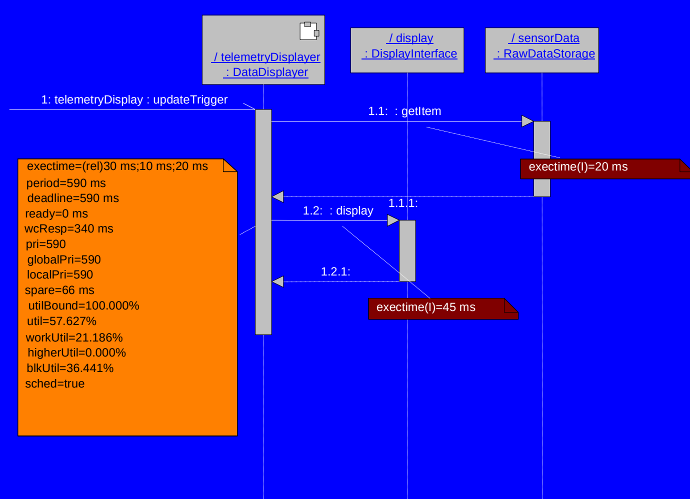
60

RT UML Example: Analysis Results

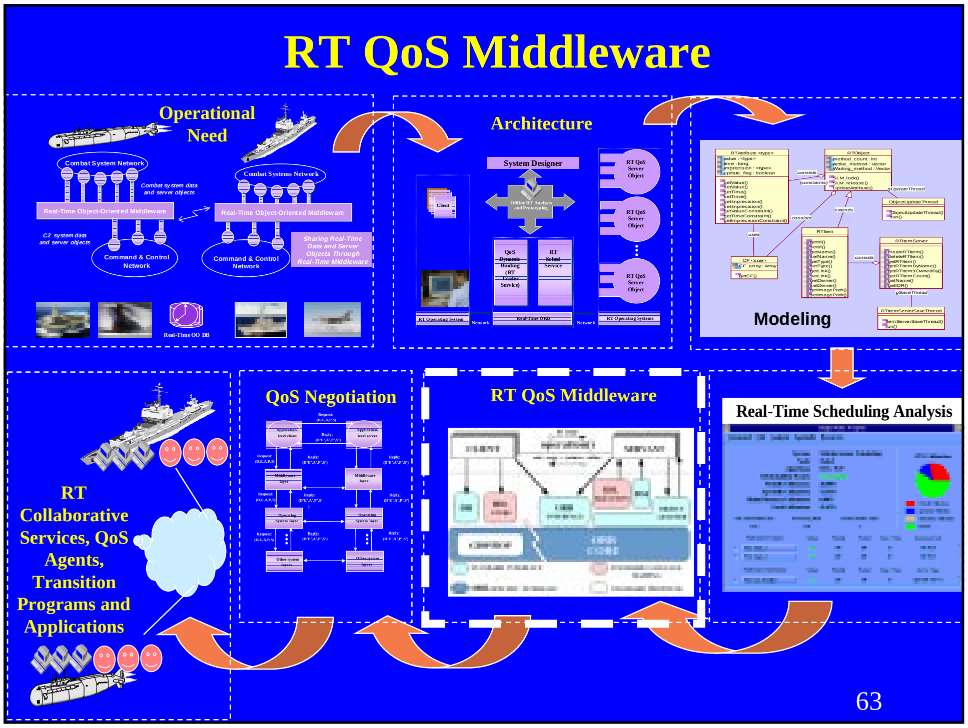


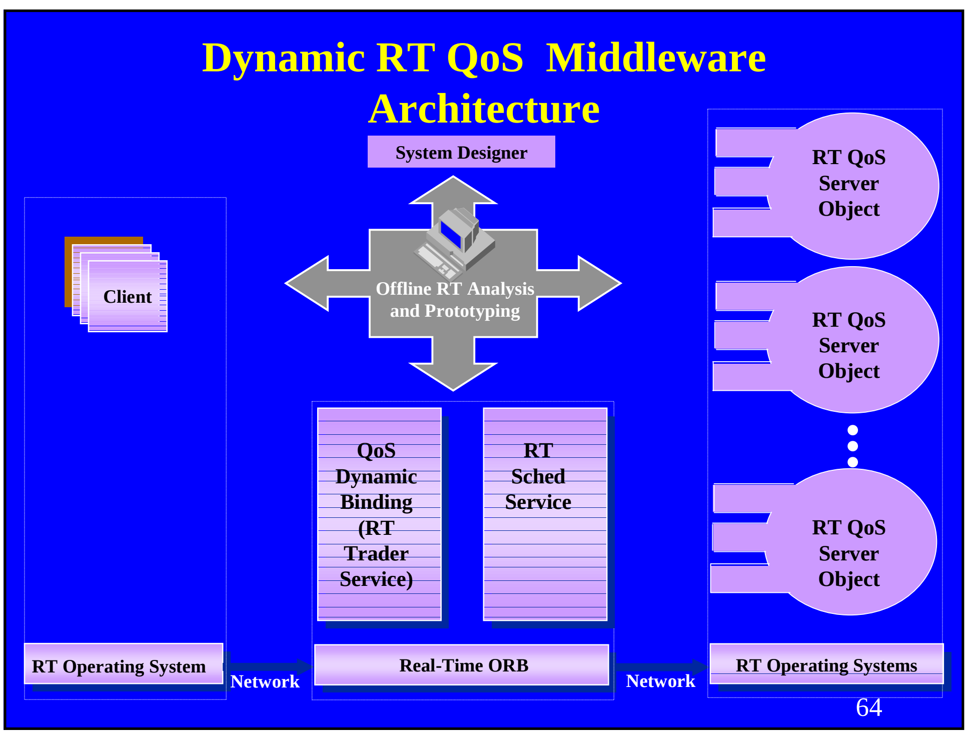
61

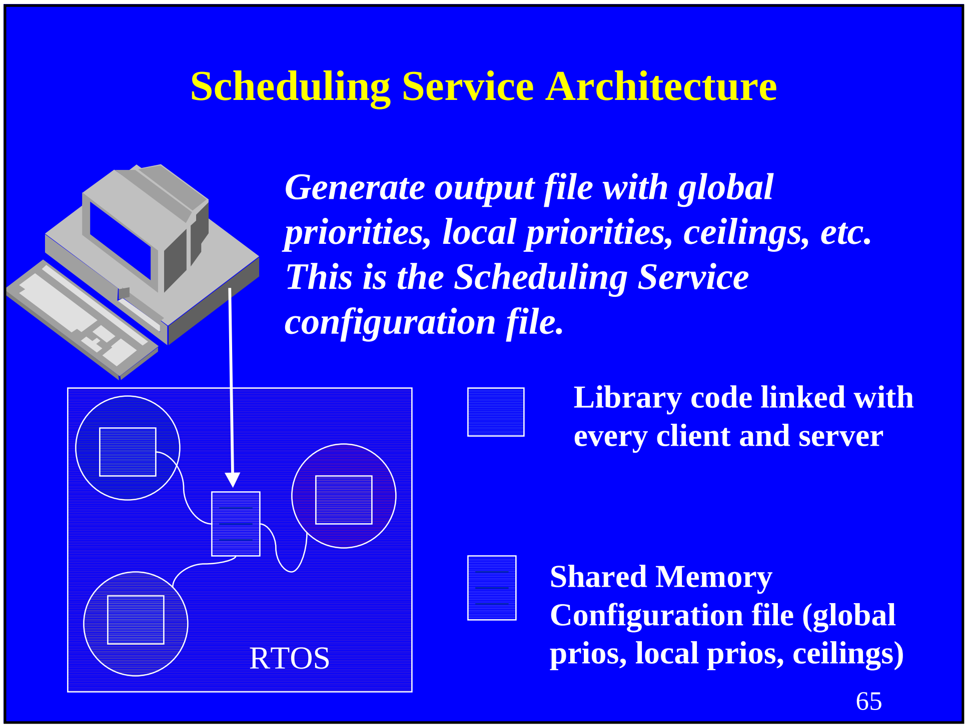
RT UML Example: Analysis Results



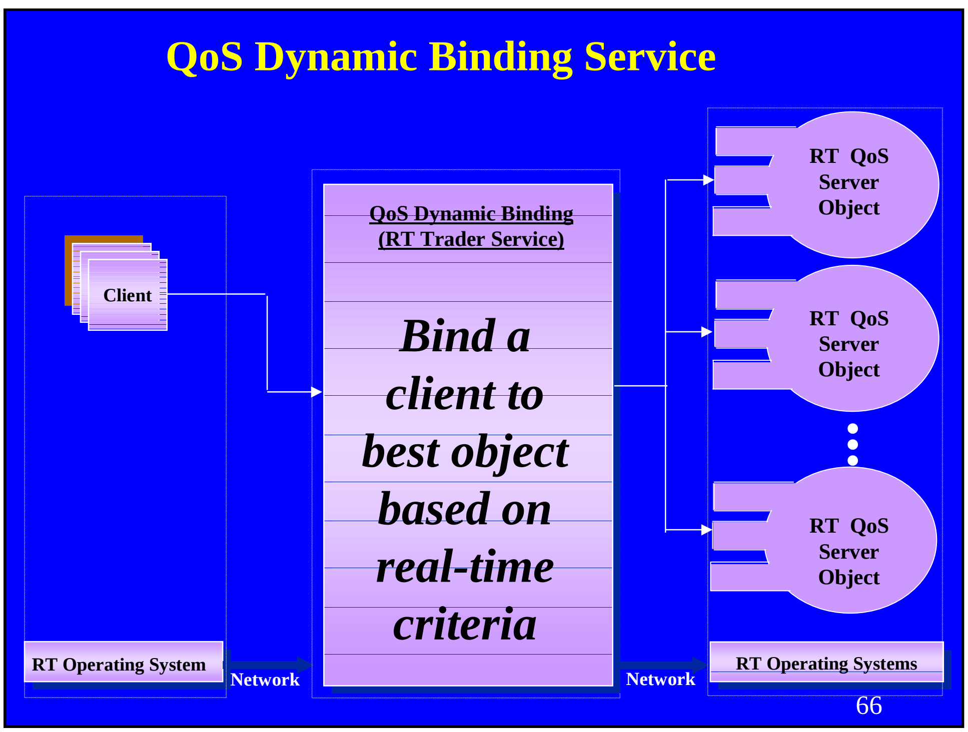
62


RT QoS Middleware
Operational Need
Architecture
Modeling
QoS Negotiation
RT QoS Middleware
Real-Time Scheduling Analysis
RT Collaborative Services, QoS Agents, Transition Programs and Applications
63

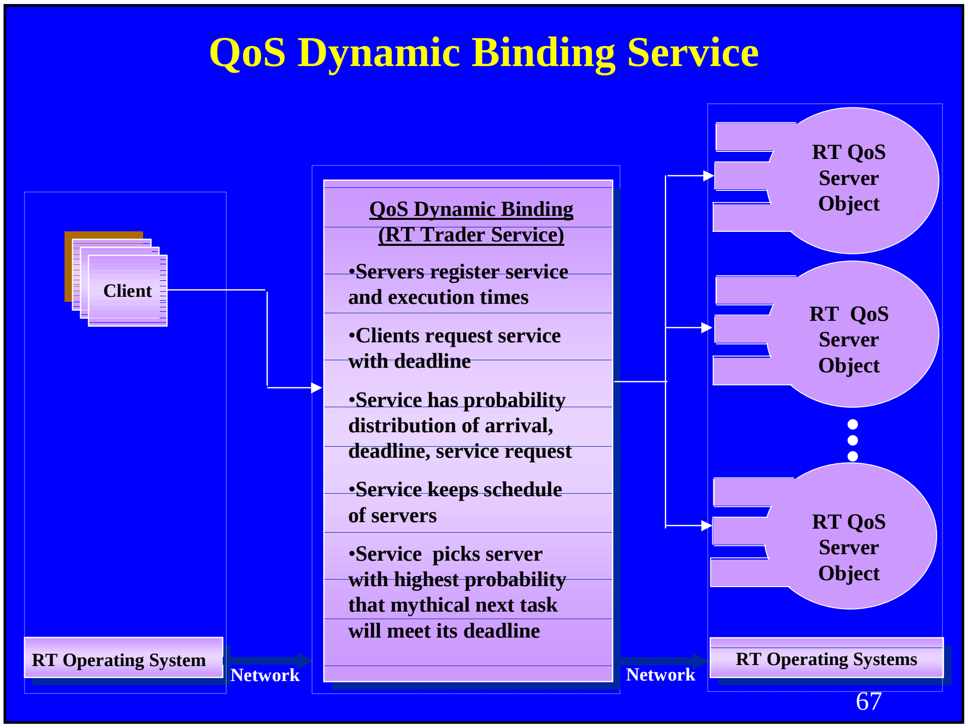

Dynamic RT QoS Middleware Architecture
System Designer
Client
Offline RT Analysis and Prototyping
QoS Dynamic Binding (RT Trader Service)
RT Sched Service
RT QoS Server Object
RT QoS Server Object
RT QoS Server Object
RT Operating System
Network
Real-Time ORB
Network
RT Operating Systems
64

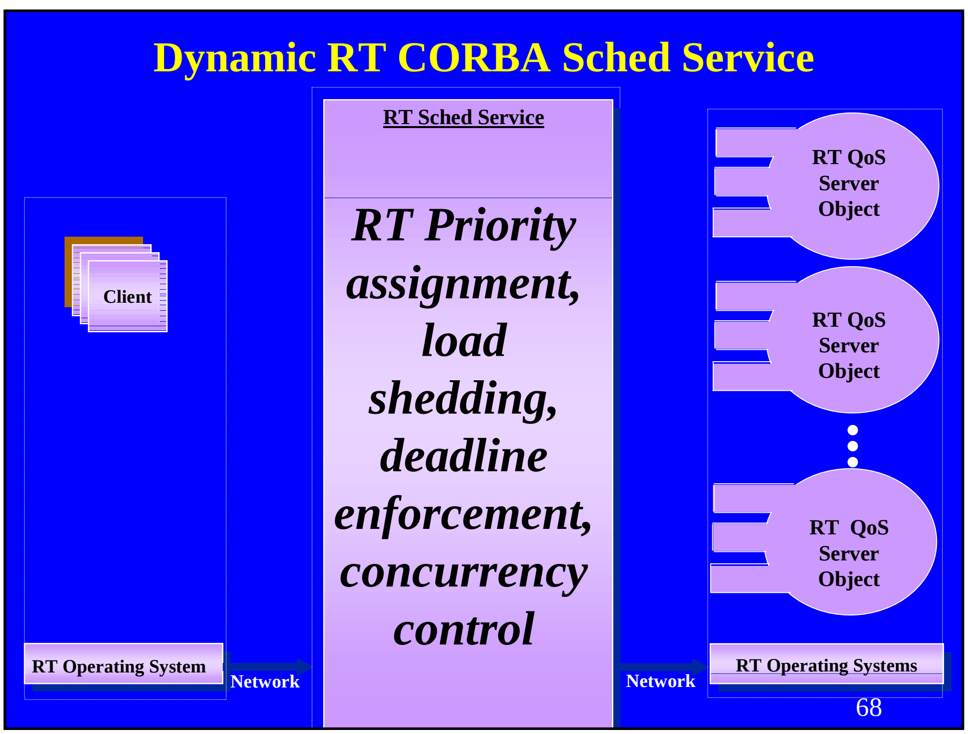


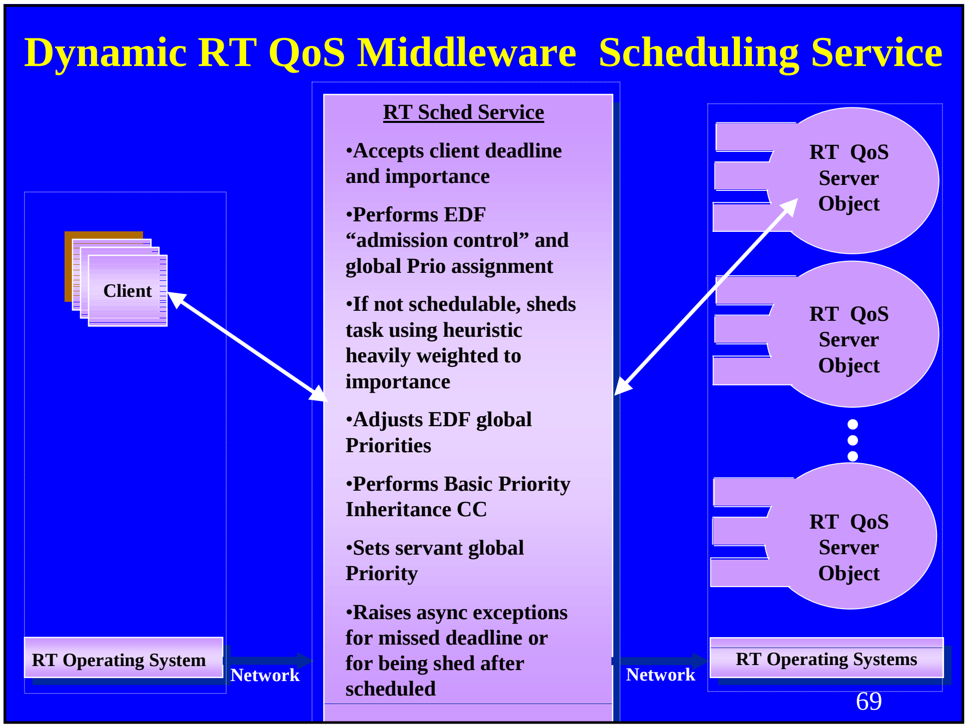
Scheduling Service Architecture
Generate output file with global priorities, local priorities, ceilings, etc. This is the Scheduling Service configuration file.
Library code linked with every client and server
Shared Memory Configuration file (global prios, local prios, ceilings)
RTOS
65

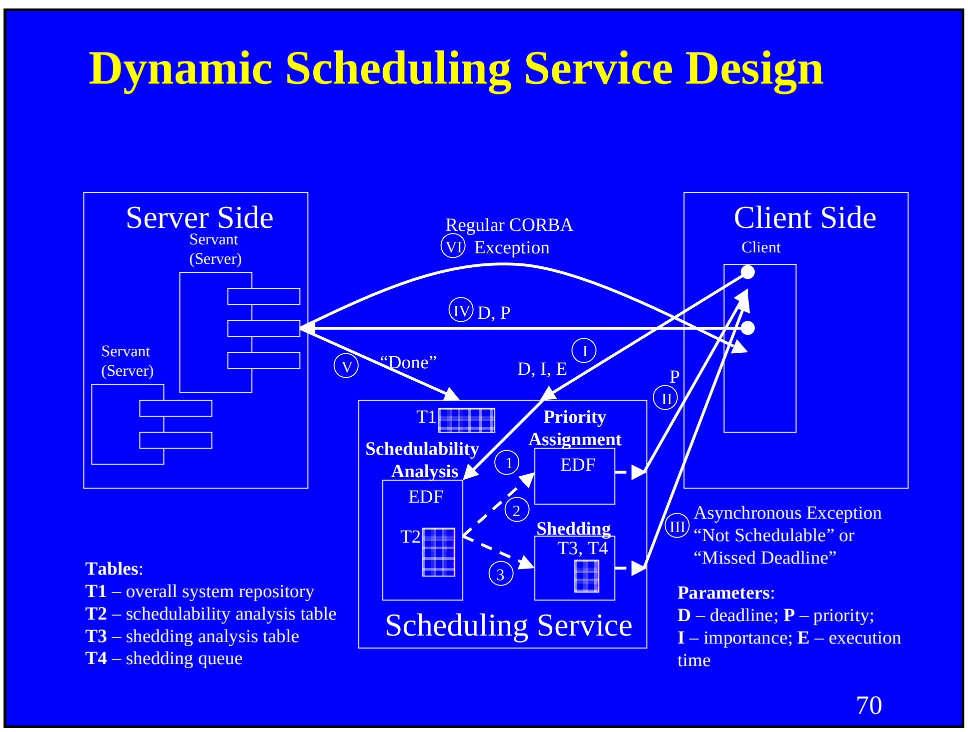


QoS Dynamic Binding Service
Client
QoS Dynamic Binding (RT Trader Service)
Bind a client to best object based on real-time criteria
RT QoS Server Object
RT QoS Server Object
RT QoS Server Object
RT Operating System
Network
Network
RT Operating Systems
66

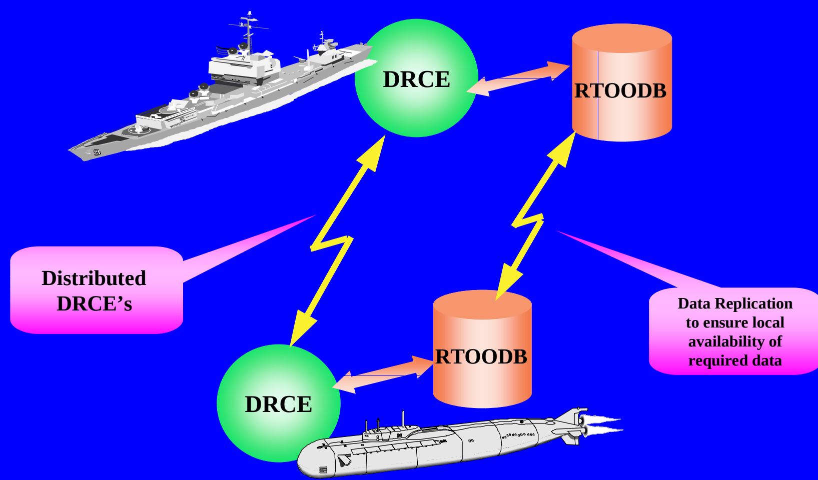

QoS Dynamic Binding Service
Client
QoS Dynamic Binding
(RT Trader Service)
•Servers register service and execution times
•Clients request service with deadline
•Service has probability distribution of arrival, deadline, service request
•Service keeps schedule of servers
•Service picks server with highest probability that mythical next task will meet its deadline
RT QoS Server Object
RT QoS Server Object
RT QoS Server Object
RT Operating System
Network
Network
RT Operating Systems
67


Dynamic RT CORBA Sched Service
RT Sched Service
Client
RT Priority assignment, load shedding, deadline enforcement, concurrency control
RT QoS Server Object
RT QoS Server Object
RT QoS Server Object
RT Operating System
Network
Network
RT Operating Systems
68


Dynamic RT QoS Middleware Scheduling Service
RT Sched Service
•Accepts client deadline and importance
•Performs EDF "admission control" and global Prio assignment
•If not schedulable, sheds task using heuristic heavily weighted to importance
•Adjusts EDF global Priorities
•Performs Basic Priority Inheritance CC
•Sets servant global Priority
•Raises async exceptions for missed deadline or for being shed after scheduled
Client
RT QoS Server Object
RT QoS Server Object
RT QoS Server Object
RT Operating System
Network
Network
RT Operating Systems
69


Dynamic Scheduling Service Design
Server Side
Servant (Server)
Servant (Server)
Client Side
Client
Regular CORBA
VI Exception
IV D, P
V "Done"
D, I, E
I
P
II
T1
Schedulability Analysis
Priority Assignment
EDF
EDF
T2
1
2
3
Shedding
T3, T4
Scheduling Service
III Asynchronous Exception "Not Schedulable" or "Missed Deadline"
Tables:
T1 – overall system repository
T2 – schedulability analysis table
T3 – shedding analysis table
T4 – shedding queue
Parameters:
D – deadline; P – priority;
I – importance; E – execution time
70

RT QoS Middleware Databases



71

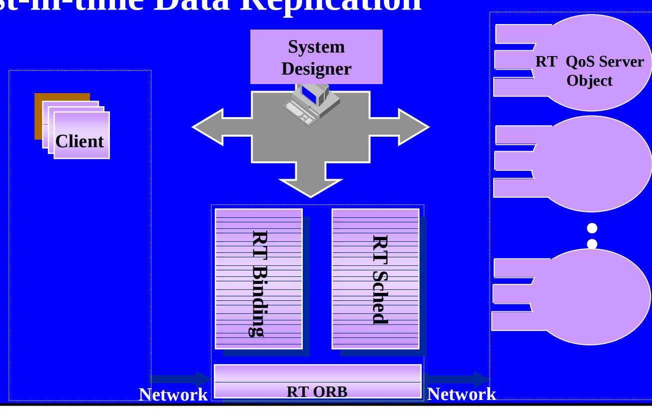
Real-Time QoS Object-Oriented Database Support

- **Share data among collaborating users**
  - **possibly from remote sources**
- **Replicate data on remote sites**
- **Guarantee temporal validity of local copies of objects**
- **Just-In-Time Real-Time Replication Algorithms**

72

New Dynamic RT QoS Middleware Algorithms

- Load shedding (admission control) heuristics
- Dynamic priority mapping
- Affected Set Basic Priority Inheritance
- Just-in-time Data Replication



73

Distributed Affected Set Priority Ceiling Algorithm

- *The conflict priority ceiling of a method* `m` *is the highest priority client that will ever lock a method that is not compatible with* `m`*; where compatibility is defined by affected set semantics.*
- **Typical Priority Ceiling alg. steps used:**
  - grant lock if requesting priority > priority ceilings of all held locks.
  - use priority inheritance to reduce and bound blocking of high priority clients

74

Distributed Affected Set Priority Ceiling Algorithm Properties

- *Consistency* - serializable object operations
- *Tight Priority Inversion Bound*
  - Proof similar to original Priority Ceiling results due to structure of protocol being the same, but granularity and conflict definition changed.
- *Deadlock prevention* - similar to previous Priority Ceiling results
- *Higher concurrency* - less blocking than in original Priority Ceiling algorithms
- Efficient Implementation - compatibility captured in Priority Ceiling check!

75

Priority Mapping Problem Definition

- Real-Time CORBA 1.0 standard allows 32,000+ "CORBA priorities"
- RT OS have limited number of priorities
  - e.g. VXWorks, Lynx have 256 local priorities; Solaris 60
- RT middleware must map this large range of global priorities to RT OS priorities on heterogeneous nodes
- More than one global priority mapped to a local priority causes *priority inversion*
- Priority inversion must be accounted for as additional blocking time for task in analysis

76

Priority Mapping Algorithm Solution

- **Algorithm identifies how many overlapping priorities on each node**
- **Starts with lowest global priority and tries to "squeeze" it with next lowest.**
- **Performs schedulability check that includes new priority inversion blocking.**
- **If schedulable, those two global priorities are mapped to the same local priority. If not, then next highest global priority is tried for "squeeze"**

77

Priority Mapping Heuristics

- We have proven Priority Mapping algorithm to be optimal
- However, solution is NP-hard and takes excessive execution time
- We have developed several heuristics that are fast and near-optimal

78

Dynamic Load Shedding Algorithm

- Let j be the index of "new" (coming) task in the Analysis Table (T2).
- Let n be the number of entries in the Analysis Table (T2).
- $\forall$ k = j..n compute slack time

  $$t_{sl} = D_k - (t_c + \Sigma_{i=1..k} ER_i + B)$$

  Blocking time B is essentially a place holder. It should be considered later.

  The entries in the analysis table above the new one (1.. j-1) are schedulable.
- If t_{sl} is negative then task k is unschedulable.

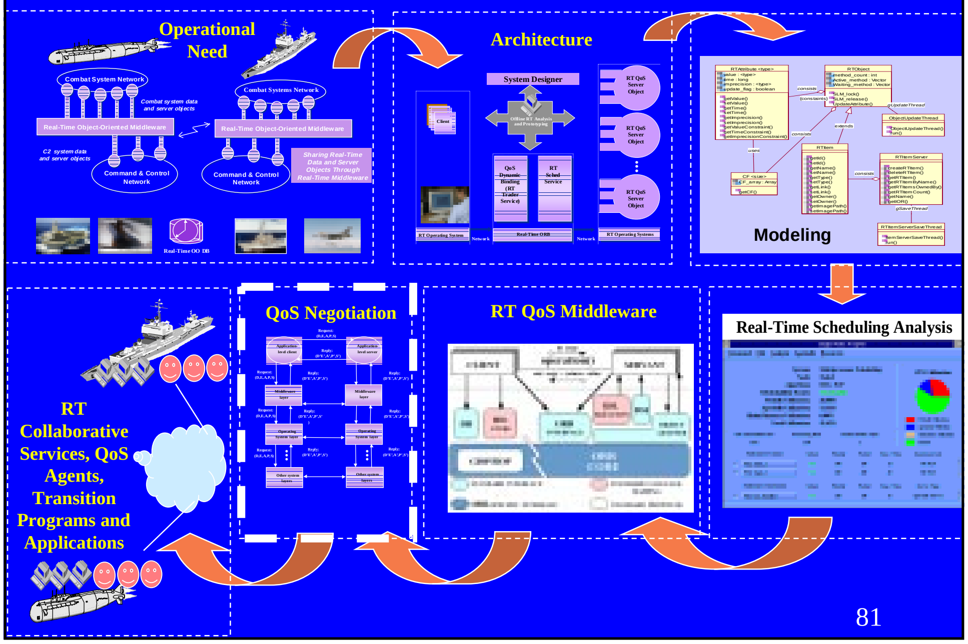
79

JIT-RT Data Replication Algorithms

- Static real-time environment
- Replication transactions
  - copy required data to local site
- Deadline computation
  - necessary and sufficient conditions for guaranteeing that all requests read temporally consistent data
- Replicates at two levels:
  - object level
  - method level - affected set semantics

80

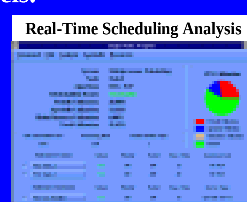
QoS Negotiation



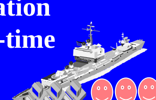
81

QoS Negotiation

- Analysis trades off real-time for quality in middleware and OS levels.

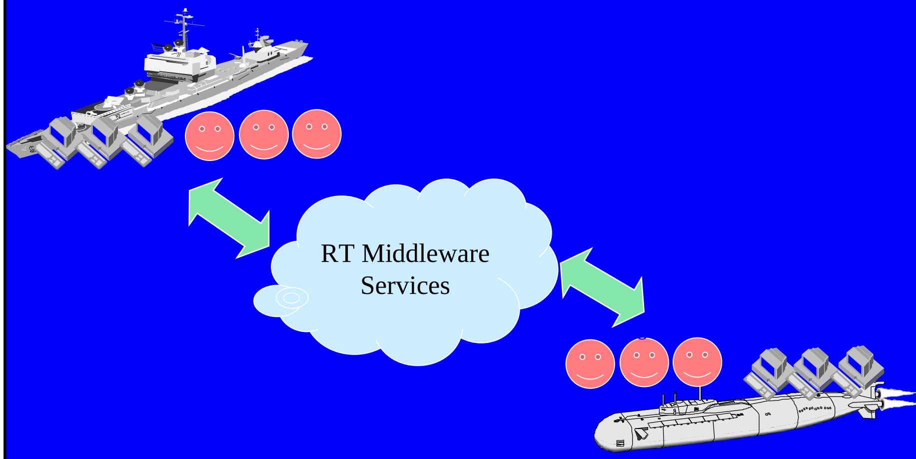


- Establishes middleware and OS priorities using real-time analysis
- Agents negotiate to tradeoff application quality for real-time constraints



82

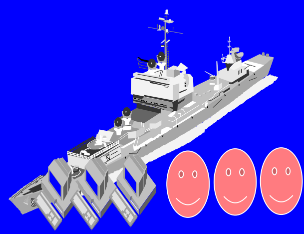
Distributed Real-time Collaborative Environment Architecture - RT Agents for QoS Negotiation



83

Real-Time Agents

- Definition: A *real-time agent* is a flexible, autonomous software entity that must meet its design objectives within specified timing constraints.
  - model
  - architecture
  - communication
  - facilitation
  - scheduling



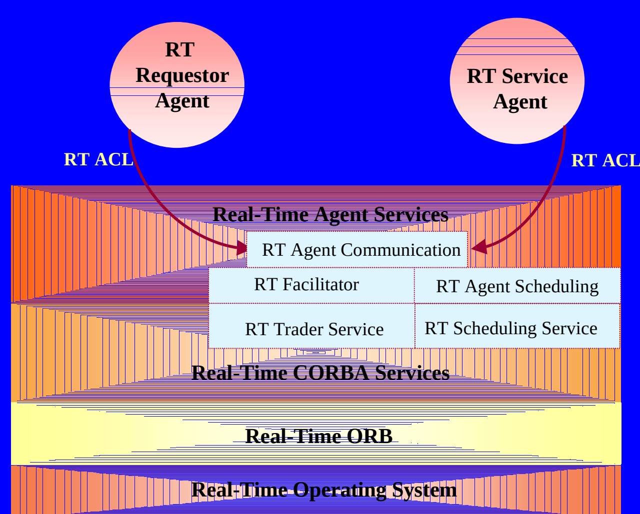
84

Real-Time Agent Model

- RT Agent
  - solvables with multiple execution strategies
  - varying exec time and result quality
- RT Agent Message
  - deadline
  - importance
  - required quality

85

RT Agent Architecture



86

RT Agent Communication

- Extend agent communication language
- Express QoS within:
  - agent capabilities
  - agent requirements

87

RT Agent Communication Language

```
(ask-one
    :sender          UserAgent
    :receiver        TrendWatcher
    :content         Watch(internet)
    :QoS_requirement (dl 15,imp 4,acc 75))
```

Ask a fellow agent to watch a stock trend within 15 seconds

```
(advertise
        :sender     BuyerSeller
        :receiver   Facilitator
        :content    BuyStock(A)
        :QoS_capabilities(
                       (ex 5, acc 85)
                       (ex 2, acc 65)))
```

Advertise QoS capabilities to facilitator including worst case execution times and accuracy levels

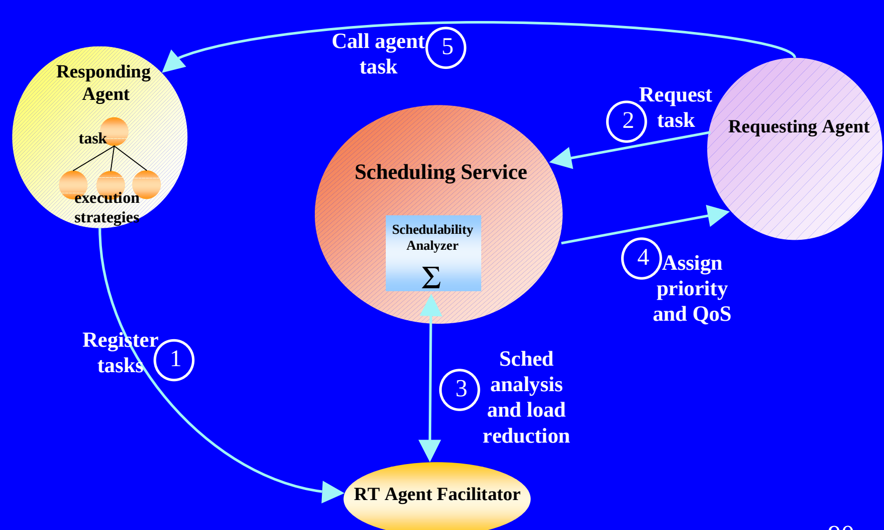
88

RT Agent Scheduling

- Extend RT CORBA Load Shedding scheduling algorithm
- *Load Reduction*
  - if system of tasks cannot be scheduled
  - reduce in quality of one or more tasks to gain more execution time for schedule

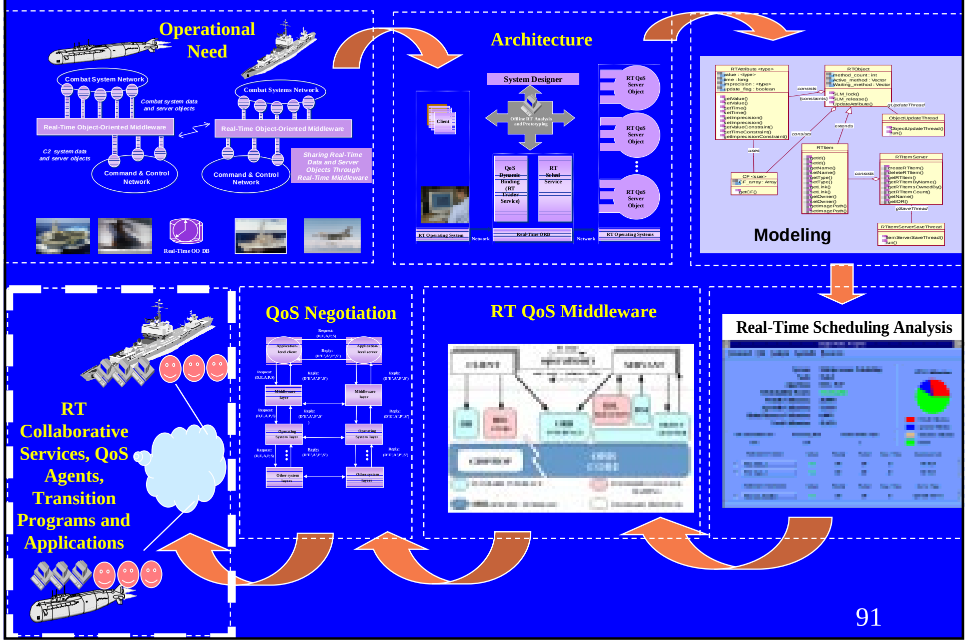
89

RT Agent Scheduling and Facilitation



90

Transitions



Accomplishments

- Implementation of the network-centric real-time QoS middleware algorithms and mechanisms. (static scheduling, dynamic scheduling, load shedding/ reduction, dynamic binding, data replication)
- Implementation of QoS model in International Standard Unified Modeling Language (UML)
- Implementation of real-time QoS analysis tool with input from UML model and output to RT QoS middleware
- Implementation of QoS negotiations among real-time agents. (Accuracy vs Real-Time)
- Transitions:
  - Military programs (Coalition Forces, Virginia Class Sub, COF, Raytheon, Lockheed/Martin, Boeing, Mitre, TRW)
  - International standards (RT CORBA 1.0, RT CORBA 2.0, UML)
  - Commercial products (Analysis Tool, Scheduling Service, UML tools, WindRiver RTOS, Rational Software tools, Lineo Embedded Linux, OIS ORB
  - Academic publications (IEEE TDPS, 2 Real-Time Systems Journal, conferences)

92

Operational Payoff

- The ability for combat systems and C2 systems to share data and functionality under real-time QoS constraints.
- The ability to design and implement systems using a COTS middleware approach that many programs are adopting.
- New algorithms, mechanisms, and analysis techniques for distributed real-time QoS middleware.

93

Web Sites

atticus.spawar.navy.mil/dhda

homepage.cs.uri.edu/research/rtsorac/

www.tripac.com

94