Distributed Real-Time Combat Systems

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

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.
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.
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)
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

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**
- **Architecture**
- **Modeling**
- **Real-Time Scheduling Analysis**

**RT QoS Modeling**

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

Profile Domain Packages
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

The General Resource Model
Profile Stereotypes for Resource Model

- ModelElement
  - GRM stereotype
  - Realize
    - GRM stereotype
  - Context
    - GRM stereotype
  - Resource
    - GRM stereotype
  - Resource Service
    - GRM stereotype
  - Resource Usage
    - GRM stereotype
  - Access Control Policy
    - GRM stereotype
  - Qos Characteristic
    - GRM stereotype

Profile stereotypes (cont.)

- Resource
  - Protected Resource
  - Unprotected Resource
  - Active Resource
  - Passive Resource
- Resource Service
  - Exclusive Service
  - Non-Exclusive Service
  - Acquire
  - Release
Sample Schedulability Analysis (SA) Profile

SA Event:
- SARelativeDeadline = 400 ms
- SAInstances = 1
- SAOccurrencePattern = “exponential”
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 structure
  – Ignores the functional characteristics
◆ Two abstractions for the same system
  – Same structure
  – Different thinking

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
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

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
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

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
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

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
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

RT UML Schedulability Model
RT UML Example: Class Diagram

- Sens orInterface
  - getData()

- RawDataStorage
  - createItem()
  - getItem()

- DisplayInterface
  - display()

- DataGatherer
  - telemDataGatherer()
  + gatherData

- DataProcessor
  - telemProcessAndFilter()
  + updateData

- DataDisplayAndRender
  - dataDisplayAndRender()
  + displayData

RT UML Example: Sequence Diagram

1. telemetryDisplay : updateTrigger
2. display : display
1.1: getItem
1.2: display
1.2.1: gather
1.2.2: getItems
1.2.3: DISPLAY
Sequence Diagram; Edit Timing Properties

Select any element; popup this menu

Edit Timing Properties Dialog
Timing Properties on Transitions

RT UML Example: Annotated Sequence Diagram
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
RT UML Example: Sequence Diagram (Distributed)

1. telemetryGatherer : gatherTrigger
   1.1: readSensor : getData
   1.2: saveSensor : createItem

exectime=150 ms
networkQoS=10

exectime=165 ms
networkQoS=5

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
UML Design for Real-Time Objects

**RTAttribute::get()**
- **pre:** \(\text{Now} - \text{avi} < \text{time} \)
- **post:** \(--\) none

**RTAttribute::set()**
- **pre:** \(--\) none
- **post:** \(\text{time} = \text{Now} \)

**RTObject**
- **checkConcurrency()**
- **consists**

**New class to represent RTSORAC object class**

**New class to specify RT methods in a RT object**

---

Example Hurricane Object

**Hurricane class extends RT Object class - expresses several RT attributes and RT operations**

**WindSpeed**
- RTAttribute <float>
- \(\text{avi} = 10\)
- \(\text{implimit} = 5\)

**WindSpeed is a RT attribute that specifies periodic timing constraint of 10 seconds and an imprecision limit of 5mph**

**UpdateLocation**
- RTOperation
- \(\text{relDeadline} = 5\)
- \(\text{WCET} = 3\)
- \(\text{WAS} = \langle \text{Location} \rangle\)

**UpdateLocation is a RT method with a relative deadline of 5 sec and a worst case execution time of 3 seconds**
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.

Modeling Real-Time QoS Middleware Application

- Clients and Servers
Modeling Real-Time QoS Middleware Application

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

![Diagram of a distributed system with nodes, clients, and servers.]

Each node has a CPU as the processor.

![Diagram of a distributed system with nodes, clients, and servers.]
Modeling Real-Time QoS Middleware Application

- The servers are represented as resources.

Modeling Real-Time QoS Middleware Application

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

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.
Modeling: Task Graph Editor

There are two components to our model. The task graph identifies the tasks of the system to be modeled.
Modeling: Resource Graph Editor

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

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.
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.

RT UML Example: Run RMA Analysis
RT UML Example: Analysis Results

```
CPU telemetrySystemInstance
  sensorInstance
    displayInstance
diskInstance
  sensor
disk
display
switch=0 ms
rate=100.000%
preempt=true
util=82.936%
perUtil=82.936%
aperUtil=0.000%
globUtil=0.000%
sched=true
```

```
exectime=(rel)30 ms;10 ms;20 ms
timeout=500 ms
ready-if true
wcResp=340 ms
pri=590
globalPri=590
localPri=590
spare=66 ms
utilBound=100.000%
util=57.627%
workUtil=21.186%
higherUtil=0.000%
blkUtil=36.441%
sched=true
exectime(I)=20 ms
exectime(I)=45 ms
```
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)

QoS Dynamic Binding Service

*Bind a client to best object based on real-time criteria*
QoS Dynamic Binding 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

Dynamic RT CORBA Sched Service

- RT Priority assignment, load shedding, deadline enforcement, concurrency control
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

Dynamic Scheduling Service Design

Server Side
- Servant (Server)

Client Side
- Client

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

Asynchronous Exception “Not Schedulable” or “Missed Deadline”
RT QoS Middleware Databases

Distributed DRCE’s

Data Replication to ensure local availability of required data

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
New Dynamic RT QoS Middleware Algorithms

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

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
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!

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
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”

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
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
  \[
  t_{sl} = D_k - (t_c + \sum_{i=1..k} ER_i + B)
  \]
  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.

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
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
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
Real-Time Agent Model

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

RT Agent Architecture
RT Agent Communication

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

RT Agent Communication Language

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

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

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

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

RT Agent Scheduling and Facilitation
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)
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.

Web Sites

atticus.spawar.navy.mil/dhda

homepage.cs.uri.edu/research/rtsonac/

www.tripac.com