MicroQoSCORBA

A QoS-Enabled, Reflective, and Configurable Middleware Framework for Embedded Systems

Professors: David E. Bakken, John C. Shovic

Students: A. David McKinnon, Olav Haugan*, Tarana Damania, Kevin Dorrow, Dr. Wesley Lawrence

*Former Student

June 2002
# My Background

- **WSU '85** (BS CptS, BS Math, Minor EE)
- **Boeing (Seattle) 1985-1988**
  - Flight Simulation Lab & Data_Flow tool to parallelize legacy flight simulation code
  - 7J7 Real-Time Executive R&D
  - Taught Ada, C, Unix
- **Grad school at U. Arizona, Tucson 1988-1994**
  - distributed fault tolerance, some “portable parallelism”
- **BBN (Boston) 1994-1999** (BBN built the first Internet in 1969 for DARPA…)
  - Co-architect of **Quality Objects (QuO) framework** for QoS for middleware (CORBA initially), now 7 DARPA projects >$10 million
  - PI on DARPA AQuA project to add replication support to CORBA (with Cornell and U. Illinois at Urbana-Champaign)
  - Replication consultant to BBN/GTE-Internetworking national ISP
- **WSU** June 1999 until retirement…. 🙃
  - Consult to Trusted Information Systems (now “NAI Labs”), TriGeo Net. Security
Outline

• **Embedded systems market background**
  • Middleware and its benefits
  • MicroQoSCORBA Baseline
    – Lifecycle Time Epochs
      • Middleware Taxonomy
    – Base Architecture
    – CASE Tools
    – Profiling (work in progress; done Aug. ’02)
    – Implementation
  • MicroQoSCORBA Quality of Service (QoS)
    – Fault Tolerance (done July ‘02)
    – RealTime (Work in Progress; done Dec. ‘02)
    – Security (Work in Progress; done Dec ‘02)
    – Possible Boeing-MicroQoSCORBA interactions
  • Key Research issues (won’t get to all today….)
Embedded Systems Market

- System size varies
  - Aircraft
  - PDAs
  - Home appliances
- 11 Billion parts per year
- Application volume varies
  - Automobiles, TVs
  - Satellites
The Network is Coming

• Embedding the Internet, CACM, May 2000
  – Smart building materials, wireless computing, smart pills, dog collars, street lights, …

• Wireless computing

• Sensor Networks
  – Environmental monitoring, Home alarm system
  – Wildlife research

• Disposable/One-Time Use
  – Hazardous environments, Battlefield deployment
  – Or simply inexpensive enough to leave behind!
Outline

• Embedded systems market background

• **Middleware and its benefits**

  • MicroQoSCORBA Baseline
    – Lifecycle Time Epochs
      • Middleware Taxonomy
    – Base Architecture
    – CASE Tools
    – Profiling (work in progress; done Aug. ’02)
    – Implementation

  • MicroQoSCORBA Quality of Service (QoS)
    – Fault Tolerance (done July ‘02)
    – RealTime (Work in Progress; done Dec. ‘02)
    – Security (Work in Progress; done Dec ‘02)
    – Possible Boeing-MicroQoSCORBA interactions

• Key Research issues (won’t get to all today....)
Why Middleware?

- Middleware == “A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system”
- Middleware exists to help manage the complexity and heterogeneity inherent in distributed systems
- Middleware provides higher-level building blocks ("abstractions") for programmers than the OS provides
  - Can make code much more portable
  - Can make them much more productive
  - Can make the resulting code have fewer errors
- Middleware sometimes is called “plumbing”
  - Connects parts of a distributed application with “data pipes” and passes data between them
Middleware in Context

Host 1

Distributed Application

Middleware API

Middleware

Middleware

Operating System API

Operating System

Comm. Processing Storage

Host 2

Distributed Application

Middleware API

Middleware

Middleware

Operating System API

Operating System

Comm. Processing Storage

Network
Middleware: Heterogeneity & Transparency

- Middleware’s programming building blocks mask heterogeneity
  - Makes programmer’s life much easier!!
- Kinds of heterogeneity masked by middleware
  - Heterogeneity in network technology always masked
  - Heterogeneity in host CPU always masked
  - Heterogeneity in operating system (or family thereof) usually masked
  - Heterogeneity in programming language usually masked
  - Heterogeneity in vendor implementations sometimes masked
- Middleware can provide transparency with respect to distribution:
  - Location transparency
  - Replication transparency
  - Mobility transparency
  - Concurrency transparency
  - Failure transparency
- Masking heterogeneity and providing transparency makes programming distributed systems much easier to do!
## Middleware, MicroQoS CORBA, and Avionics

- **Good benefits of productivity and**
  - Higher-level building block
  - Less error-prone
  - Analogies: HLL vs. assembler
- **Verifiability/validation**
  - FAA is picky, picky, picky…..
  - Smaller, much more modular/regularized middleware such as MicroQoS CORBA likely much easier to validate with FAA than a huge, general-purpose CORBA ORB
- **Ability to plug in new transports**
  - ARINC 664 (nice complement to our ongoing RT work)
Java

- Java Remote Method Invocation (RMI)
  - Java-centric
  - Lacks cross-language support
- Java VM is ‘large’
  - Java 2 Micro Edition (J2ME)
  - J2ME Connect Device Configuration
    - 2+ MB
  - J2ME Connected Limited Device Configuration
    - 160 MB
Existing CORBA Specifications

- MinimumCORBA
  - Removes support for dynamic interfaces, etc
  - Reduces the memory footprint of an ORB
- Real-time CORBA
  - Provides tools to better predict time delays
  - Enables hard real-time CORBA applications
- Fault-tolerant CORBA
- CORBAsecurity
- Smart Transducers Interface
Existing Small Middleware Frameworks

• Support is lacking for
  – Small memory footprint
  – Generality
  – Power awareness
  – Multi-property QoS (esp. non-RT properties)
  – Fine-grained composability
  – Software Engineering & Analysis Tools

• A few “point solutions” with predefined tradeoffs
Outline

• Embedded systems market background
• Middleware and its benefits

• MicroQoSCORBA Baseline
  – Lifecycle Time Epochs
    • Middleware Taxonomy
  – Base Architecture
  – CASE Tools
  – Profiling (work in progress; done Aug. ’02)
  – Implementation

• MicroQoSCORBA Quality of Service (QoS)
  – Fault Tolerance (done July ‘02)
  – RealTime (Work in Progress; done Dec. ‘02)
  – Security (Work in Progress; done Dec ‘02)
  – Possible Boeing-MicroQoSCORBA interactions

• Key Research issues (won’t get to all today…..)
MicroQoSCORBA Bottom Line

• MicroQoSCORBA the only framework that
  – Is a “bottom-up” rethinking from the device level of what should
    be configurable, and in what ways
  – Is tailorable for a given application to the wide range of
    • Device constraints, and
    • Application-dictated constraints
      with a fine granularity of configuration constraints
  – Will support both “functional” and QoS properties
    • Not just realtime, but security and fault tolerance are also key
      QoS properties
Lifecycle Time Epochs

1. Initial design
2. IDL compilation
3. Code compilation/linkage
4. System initialization/startup
5. Run time execution
# Lifecycle Epoch Table

<table>
<thead>
<tr>
<th>Lifecycle Epoch</th>
<th>Constraint Bound</th>
<th>Representative Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
<td>HW Heterogeneity</td>
<td>Symmetric, asymmetric</td>
</tr>
<tr>
<td></td>
<td>HW Choice</td>
<td>X86, TINI, ColdFire</td>
</tr>
<tr>
<td></td>
<td>Communications HW</td>
<td>Ethernet, Serial, Infrared</td>
</tr>
<tr>
<td></td>
<td>Processing Capability</td>
<td>50 Mhz, 1 Ghz, 8bit, 32bit</td>
</tr>
<tr>
<td></td>
<td>System size</td>
<td>small, medium, large (e.g., transducers to jets)</td>
</tr>
<tr>
<td></td>
<td>Power Usage</td>
<td>line, battery, and/or parasitic power</td>
</tr>
<tr>
<td><strong>IDL Compilation</strong></td>
<td>Communication Style</td>
<td>Passive, Pro-active, Push, Pull</td>
</tr>
<tr>
<td></td>
<td>Stub/Proxy Generation</td>
<td>Inline vs. library usage</td>
</tr>
<tr>
<td></td>
<td>Message Lengths</td>
<td>Fixed, variable length messages</td>
</tr>
<tr>
<td></td>
<td>Parameter Marshalling</td>
<td>Fixed Formats</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Space/Time Optimizations</td>
<td>Loop unrolling, code migration, function and proxy inlining</td>
</tr>
<tr>
<td>Compilation**</td>
<td>Library Usage</td>
<td>Static vs. dynamic library linkage</td>
</tr>
<tr>
<td><strong>System /Application</strong></td>
<td>Device Initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Network Startup</td>
<td>Bootp, dhcp</td>
</tr>
<tr>
<td><strong>Startup</strong></td>
<td>Major QoS adaptation</td>
<td>Select between QoS modules</td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
<td>Minor QoS adaptation</td>
<td>Adjust QoS parameters</td>
</tr>
</tbody>
</table>
## 1. Initial Design

<table>
<thead>
<tr>
<th>Embedded Hardware</th>
<th>Roles</th>
<th>SW I/O</th>
<th>IDL Subsetting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection Setup</strong></td>
<td><strong>Data Flow</strong></td>
<td><strong>Interaction Style</strong></td>
<td><strong>Data Representation</strong></td>
</tr>
<tr>
<td>System Components</td>
<td>Initiates Conn. Setup</td>
<td>Bits In</td>
<td>Sync (Send/Recv)</td>
</tr>
<tr>
<td>• Homogenous</td>
<td>Receive Conn. Requests</td>
<td>Bits Out</td>
<td>Async (One-Way Msgs)</td>
</tr>
<tr>
<td>• Asymmetric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW I/O Support</td>
<td></td>
<td></td>
<td>Protocols</td>
</tr>
<tr>
<td>• Serial, 1-Wire,</td>
<td></td>
<td></td>
<td>• TCP/IP</td>
</tr>
<tr>
<td>Parallel, Digital,</td>
<td></td>
<td></td>
<td>• PPP</td>
</tr>
<tr>
<td>Ethernet, IrDA,</td>
<td></td>
<td></td>
<td>• 1-wire</td>
</tr>
<tr>
<td>Bluetooth, GSM, GPRS</td>
<td></td>
<td></td>
<td><strong>Direct</strong> (i.e., IIOP)</td>
</tr>
<tr>
<td>Resources</td>
<td></td>
<td></td>
<td><strong>Indirect</strong> (i.e., IIOP Gateway)</td>
</tr>
<tr>
<td>• Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Power</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing Capabilities</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• 8-bit, 16-bit, …</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Roles:**
- **Sync**: Initiates Conn. Setup
- **Async**: Receive Conn. Requests
- **Msg Push**: Bits In
- **Msg Pull**: Bits Out
- **Passive**: Bits In/Out
- **Pro-Active**: (Send/Recv)

**Protocols:**
- TCP/IP
- UDP
- PPP
- 1-wire
- Direct (i.e., IIOP)
- Indirect (i.e., IIOP Gateway)
2. Interface Compilation

- IDL Subsets
  - Verify conformance with design choices
- Application specific code generation
  - Leverage IDL subset meta-data
  - Optimize client stub/server skeleton code
- Representation Optimizations
  - Optimize static string references
  - Object references
3. Application Compilation/Linkage

• Space/Time optimizations
  – Use existing compiler techniques

• Library usage
  – Static
  – Dynamic

• Future choices (work in progress)
  – MicroQoSCORBA ORB library vs. custom-generated library
  – MicroQoSCORBA Orb library or custom-generated library
4. System Initialization/Startup

- Device initialization
  - Use reflection to load appropriate modules
  - Initialize QoS subsystems

- Network initialization
  - Locate other key nodes on the network
5. Run Time Execution

• Fixed functionality
  – Most common case

• Dynamic functionality
  – Increased Resource Usage
# Lifecycle Epoch Table

<table>
<thead>
<tr>
<th>Lifecycle Epoch</th>
<th>Constraint Bound</th>
<th>Representative Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
<td>HW Heterogeneity</td>
<td>Symmetric, asymmetric</td>
</tr>
<tr>
<td></td>
<td>HW Choice</td>
<td>X86, TINI, ColdFire</td>
</tr>
<tr>
<td></td>
<td>Communications HW</td>
<td>Ethernet, Serial, Infrared</td>
</tr>
<tr>
<td></td>
<td>Processing Capability</td>
<td>50 Mhz, 1 Ghz, 8bit, 32bit</td>
</tr>
<tr>
<td></td>
<td>System size</td>
<td>small, medium, large (e.g., transducers to jets)</td>
</tr>
<tr>
<td></td>
<td>Power Usage</td>
<td>line, battery, and/or parasitic power</td>
</tr>
<tr>
<td><strong>IDL Compilation</strong></td>
<td>Communication Style</td>
<td>Passive, Pro-active, Push, Pull</td>
</tr>
<tr>
<td></td>
<td>Stub/Proxy Generation</td>
<td>Inline vs. library usage</td>
</tr>
<tr>
<td></td>
<td>Message Lengths</td>
<td>Fixed, variable length messages</td>
</tr>
<tr>
<td></td>
<td>Parameter Marshalling</td>
<td>Fixed Formats</td>
</tr>
<tr>
<td><strong>Application Compilation</strong></td>
<td>Space/Time Optimizations</td>
<td>Loop unrolling, code migration, function and proxy inlining</td>
</tr>
<tr>
<td></td>
<td>Library Usage</td>
<td>Static vs. dynamic library linkage</td>
</tr>
<tr>
<td><strong>System/Application Startup</strong></td>
<td>Device Initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Network Startup</td>
<td>Bootp, dhcp</td>
</tr>
<tr>
<td></td>
<td>Major QoS adaptation</td>
<td>Select between QoS modules</td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
<td>Minor QoS adaptation</td>
<td>Adjust QoS parameters</td>
</tr>
</tbody>
</table>
MicroQoSCORBA over UDP

Cons of TCP:

- Own model for retransmission of data
- Designed to transmit bulk data; 1 bit to indicate "interactive data"; no common equipment checks that bit
- Strictly sequential: no way to access data "as soon as received", or to ignore/discard "obsolete data“ (Ex. You don’t want to retransmit old temperatures from sensors since the temperature may already have changed)
- Kernel resources occupied by TCP stack
  - Typical stack 50-60KB in small devices
**MicroQoS CORBA over UDP (cont.)**

- UDP used to overcome TCP limitations
- Reliability added on top of UDP
- UDP-based protocol needs to be highly aware of data semantics (not just syntax) of what it carries => usage specific
  (Ex. Temperature sensor pinging the temperature every 5 mins. may not need reliability unless certain value turns on fire alarm)

MS thesis on this work completing in July, 2002.
Configurability

- Each embedded system is slightly different
- Not everyone will need the kitchen sink
- Let the developer pick and choose
- Configure QoS properties, too
  - Multiple implementations/strengths for one property to choose from (more later in talk)
  - Choose at startup with reflection/introspection
CASE Tools

- Not all developers are created equal
- Make it easy for the casual programmer
  - Domain expert, but QoS novice
  - Lifecycle support personnel
  - Temporary/contract employees
- Let the tools choose compatible components based upon
  - QoS requirements
  - Resource configuration
CASE Tools
### (Very) Initial Profiling Results (Dec. 01)

```java
module sensor {
    interface volts {
        long read_smart_transducer(in long arg1);
    }
};
```

<table>
<thead>
<tr>
<th>Module</th>
<th>Class File Size on Linux (bytes)</th>
<th>Executable File Size on TINI (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
</tr>
<tr>
<td>MicroQoS-CORBA</td>
<td>35,182</td>
<td>43,412</td>
</tr>
<tr>
<td>ZEN</td>
<td>356,080</td>
<td>351,704</td>
</tr>
<tr>
<td>JacORB</td>
<td>6,596,196</td>
<td>6,592,640</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Average call time (ms)</th>
<th>Memory usage on TINI (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linux</td>
<td>TINI</td>
</tr>
<tr>
<td>MicroQoS-CORBA</td>
<td>0.56</td>
<td>210.00</td>
</tr>
<tr>
<td>ZEN</td>
<td>2.03</td>
<td>N/A</td>
</tr>
<tr>
<td>JacORB</td>
<td>1.46</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Future Profiling Directions

• Memory footprint
  – Total vs. RAM vs. ROM
  – Growth rate w.r.t. parameters, methods, …

• Timing & CPU utilization
  – End-to-End latency & jitter
  – Hot spot analysis
    • Network I/O, Parameter marshalling, Buffer management

• Power consumption

• Non-Java language (C++) support in some cases/configurations
Outline

• Embedded systems market background
• Middleware and its benefits
• MicroQoSCORBA Baseline
  – Lifecycle Time Epochs
    • Middleware Taxonomy
  – Base Architecture
  – CASE Tools
  – Profiling (work in progress; done Aug. ’02)
  – Implementation

• MicroQoSCORBA Quality of Service (QoS)
  – Fault Tolerance (done July ‘02)
  – RealTime (Work in Progress; done Dec. ‘02)
  – Security (Work in Progress; done Dec ‘02)
  – Possible Boeing-MicroQoSCORBA interactions

• Key Research issues (won’t get to all today…..)
Quality of Service (work in progress)

- Quality of Service (QoS): the “non-functional” parts of the program
  - “functional” part is the “what”
  - “non-functional” part is the “how”

- Security
  - Multiple strengths/Algorithms

- Fault Tolerance
  - Quantity and types of faults tolerated

- Real-time Behavior
  - Scheduling Algorithms, Network performance

- Resource Issues
  - Memory footprint
  - Power awareness

- Which combos of QoS modules are legal???
## Fault Tolerance Taxonomy

<table>
<thead>
<tr>
<th>Redundancy</th>
<th>Reliability</th>
<th>Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Temporal</strong>&lt;br&gt;– Multiple transmits</td>
<td><strong>Group Communication</strong>&lt;br&gt;– Best Effort&lt;br&gt;– Reliable&lt;br&gt;– Uniform&lt;br&gt;– Atomic</td>
<td><strong>Sender FIFO</strong>&lt;br&gt;<strong>Causal</strong>&lt;br&gt;– Logical Timestamping</td>
</tr>
<tr>
<td><strong>Value</strong>&lt;br&gt;– Checksums, CRC</td>
<td><strong>Failure Detection</strong></td>
<td></td>
</tr>
</tbody>
</table>
MicroQoSCORBA Initial QoS Architecture

• Note: Security and RealTime work in progress....
MicroQoSCORBA, RealTime, and Boeing

• Real-Time research
  – Profiling of components and application for timing, not memory (as current profiling is)
  – 1-2 CPU scheduling mechanisms TBD
  – MS thesis work, finishing in Dec. 2002 by Dr. Wesley Lawrence

• Possible Boeing and Real-Time collaboratory project
  – Add ARINC 644 (or subset(s) thereof?) to MicroQoSCORBA
    • Nice Real-Time network mechanisms to augment above MS work!
  – Target a few extra specific processors
Conclusions

• MicroQoSCORBA the only framework that
  – Is a “bottom-up” rethinking from the device level of what should be
    configurable, and in what ways
  – Is tailorable for a given application to the wide range of
    • Device constraints, and
    • Application-dictated constraints
      with a fine granularity of configuration constraints
  – Will support both “functional” and QoS properties
    • Not just realtime, but security & fault tol. also key QoS properties

• MicroQoSCORBA now interacts with TAO ORB, others easy (via IIOP)
  – Access to naming service, trader service, fault tolerance
    service (non-transparent), etc…..
  – Access to bigger servers and the world at large!

• Questions? Comments? Potential Collaborations?
BACKGROUND SLIDES

[Outline]

• Embedded systems market background
• Related middleware research
• MicroQoSCORBA
  – Lifecycle Time Epochs
    • Middleware Taxonomy
  – Quality of Service
  – Architecture
  – CASE Tools
  – Implementation
• Key Research issues
Outline

• Embedded systems market background
• Middleware and its benefits
• MicroQoSCORBA Baseline
  – Lifecycle Time Epochs
    • Middleware Taxonomy
  – Base Architecture
  – CASE Tools
  – Profiling (work in progress; done Aug. ’02)
  – Implementation
• MicroQoSCORBA Quality of Service (QoS)
  – Fault Tolerance (done July ‘02)
  – RealTime (Work in Progress; done Dec. ‘02)
  – Security (Work in Progress; done Dec ‘02)
  – Possible Boeing-MicroQoSCORBA interactions
• Key Research issues (won’t get to all today….)

MicroQoSCORBA © 2002 David E. Bakken
Key Questions

• **Q1: Baseline Constraints:**
  - What constraints must be imposed on middleware to achieve a small footprint (ignoring QoS for now)?

• **Q2: QoS Properties:**
  - What strength or “flavor” of QoS properties (security, fault tolerance, real-time) can reasonably be supported in resource-starved embedded systems?

• **Q3: Network-Wide Composition:**
  - What issues, optimizations, and tradeoffs must be made when a network of QoS-enabled devices integrated? How can we support global properties, metrics, and goals involving the entire distributed embedded system?

• **Q4: Software Engineering Issues:**
  - How can we use QoS aspects to make the middleware more manageable and quantifiable? How can we use patterns and tools to help the developer to be more productive?
Key Questions in Context

Q4: Software Engineering

Aspects
Patterns
Tools

Embedded Systems
Application Developer

Q3: Local-Global Interactions

Node 1

Q2: QoS
Security, Realtime, Fault Tolerance

Power
Memory

Q1: Baseline Functionality

Application Logic

•

•

•

•
Question 1: Baseline Functionality

• What constraints must be imposed on middleware to achieve a small footprint (ignoring QoS for now)?
  – 1a. Architecture
  – 1b. Costs
  – 1c. Models
  – 1d. Composition
  – 1e. Bottom line
1a: Architecture

• What middleware components or subsystems should be constrained?

• What should their architecture be?
  – Covered previously in this presentation
1b: Costs

- What memory footprint and power costs are associated with each subsystem or facet?
  - What HW hooks are available for gathering profiling information?
  - Can SW profiling hooks be added?
  - Can rough, empirical rules-of-thumb be derived for cost estimation?
- How much can be gained by constraining a subsystem’s flexibility or removing it?
1c: Models

• What models and heuristics for predicting memory and power usage can be developed, to map from larger environments with adequate profiling and other “hooks” onto very small embedded environments where this is not possible?

• How can spreadsheet-like online tools use these models to allow developers to project the costs of various possible tradeoffs and constraints without having to code them (like with 1b)?
1d: Composition

- How do combinations of these constraints compose or interact?
- What combinations are legal or useful?
- How do we best allow a developer to constrain them?
  - Where in the SW architecture will these constraints be implemented?
  - Do the HW choices impose mutually exclusive constraints?
  - What constraints will provide the greatest benefit?
  - Must some constraints be implemented in order to support QoS?
  - Should the CASE tool
    - allow the developer to have access to all constraint choices
    - provide a standard set of design constraint patterns
1e: Bottom Line

• How small and power stingy can we get, and with what constraints?
  – How can we instrument the HW/SW?
    • Are hooks provided for memory usage?
    • Can we gather power consumption information?
    • Can a SW code analysis provide the required information?
  – How does the choice of HW cpu/platform affect small size/power?
    • Battery power, parasitically powered devices, …
  – What size/power trade-offs exist?
    • Batching messages saves power but increases code size
    • How much ‘hand tweaking’ of the code is justifiable at the cost of maintainability?
Question 2: QoS Properties

• What strength or “flavor” of QoS properties (security, fault tolerance, realtime) can reasonably be supported in resource-starved embedded systems?
  – 2a. Mechanisms
  – 2b. QoS Subsets
  – 2c. Costs
  – 2d. Models
  – 2e. Qos Composition
  – 2f. Composition with Baseline
• What mechanisms are used to support each QoS property?
  – What HW design choices ensure QoS or detract from it?
    • eg, dedicated transmission lines ensure message confidentiality whereas wireless links are open to eavesdroppers
    • What are the costs of SW vs. HW QoS implementations?
  – Security
    • Physical protection, Data encryption, Access control, Digital signatures, A/Symmetric encryption, Physical tokens, Shared secrets, Challenge/Response protocols, Identity attributes, Privileges
  – Fault Tolerance
    • Simple retransmits, Passive vs. Active replication, Redundant HW
  – Real-Time
    • Static vs. Dynamic scheduling, Determinism, Jitter control
2b: QoS Subsets

• What subsets of these mechanisms exist for each property that is reasonable in resource-starved environments?
  – Is HW support required for some mechanisms?
  – How much hardware support is required for each mechanism?
  – How many interchangeable implementations should be developed for a given QoS sub-property?
  – Are some subsets effectively mutually exclusive?
  – How many subsets should be exposed to the developer?
  – Can we derive reasonable Subset Design Patterns?
2c: QoS Costs

- What are the costs of these mechanisms?
  - Can we estimate (pre-implementation) costs of a QoS mechanism?
  - What HW costs exist?
  - What SW costs exist?
  - Does HW support exist for profiling the QoS Costs?
  - Do we need to implement SW hooks for gathering profiling/cost information?
  - When do the increased costs become prohibitive?
  - Is there a point where the application should be transitioned to a full-featured ORB?
2d. Models

• What models and heuristics for predicting QoS costs can be developed, to map from larger environments with adequate profiling and other “hooks” onto very small embedded environments where this is not possible?
• How can spreadsheet-like online tools use these models to allow developers to project the costs of various possible tradeoffs and constraints without having to code them?
2e: QoS Composition

• What combinations of these mechanisms are useful for resource-starved environments?
  – What are some common ‘Use Cases’ that require multi-property QoS?
  – Can we develop some useful design patterns for these ‘Use Cases’?
  – At what level (or multiple levels?) should these multi-property QoS combinations be developed/described?
  – Are the resource constraints for combinations additive?
What interactions do the QoS subsystems have with the functional subsystems and constraints?

- Which functional and QoS subsystems are co-located?
- Which functional and QoS subsystems (or the implementation of their mechanisms) are orthogonal?
- Do some mechanisms compose better with the functional aspects than others?
- Are some subsystem interactions mutually exclusive?
Q. 3: Network-Wide Composition

• What issues, optimizations, and tradeoffs must be made when a network of QoS-enabled devices integrated?

• How can we support global properties, metrics, and goals involving the entire distributed embedded system?

  – 3a. Local-Global Interactions
3a: Local-Global Interactions

• What basic issues, optimizations, and tradeoffs are associated with networks of QoS enabled embedded systems?
  – What new metrics need to be developed?
  – Can existing metrics be QoS-parameterized?

• How does power awareness impact the local-global interactions?
Q. 4: Software Engineering Issues

• How can we use QoS aspects to make the middleware more manageable and quantifiable?
• How can we use patterns and tools to help the developer to be more productive?
  – 4a. Aspects
  – 4b. Patterns
  – 4c. Tools
4a: Aspects

• How can the QoS properties in Question 2 be organized into aspects that are orthogonal?
• How can this be woven cleanly in the architecture?
  – Aspect-oriented IDL compiler generated code
  – How will aspect weaving affect the cost models produced in Questions 1 & 2
4b: Patterns

• What canonical patterns involving combinations of functional constraints and QoS requirements are common in various embedded system domains?
  – What are the common use cases?

• How can these be organized to help middleware novices avoid choosing from $2N$ or similar number of combinations?
4c: Tools

• An initial set of tools has been developed.
• How can the tools be extended in order to encompass the cost projections and models developed in Questions 1 & 2?
• How can the tools present design patterns to the software developer?
Optional/Related Questions

• What middleware services (naming, etc.) and variants thereof are useful for embedded systems, and what are their costs?
Conclusions

• MicroQoSCORBA the only framework that
  – Is a “bottom-up” rethinking from the device level of what should be configurable, and in what ways
  – Is tailorable for a given application to the wide range of
    • Device constraints, and
    • Application-dictated constraints
    • with a fine granularity of configuration constraints
  – Will support both “functional” and QoS properties
    • Not just realtime, but security and fault tolerance are also key QoS properties
  – Will support the Developer with SW Eng. Tools