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

A. David McKinnon, Tarana R. Damania, David E. Bakken, Kevin E. Dorow, Wesley E. Lawrence

July 2002
MicroQoSCORBA Objectives

• Go beyond Minimum & Real-Time CORBA
• Be Small
  – Distributed sensor networks, home appliances
• Have QoS
  – Run time performance, security, fault tolerance
• Maintain interoperability
• Create S/W Eng. tools that aid the developer
MicroQoS CORBA Overview

• Lifecycle choices
  – Exploit limiting choices from the initial idea until the end of execution

• Quality of Service choices
  – Decide what to support (e.g., Security, Fault Tolerance, Real-Time performance)

• Development tools
  – Design Patterns
  – Profiling metrics & heuristics
## Lifecycle Time Epochs

<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><strong>Run Time</strong></td>
<td>Major QoS adaptation</td>
<td>Select between QoS modules</td>
</tr>
<tr>
<td></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>Interaction Style</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>SW I/O</strong></td>
<td><strong>IDL Subsetting</strong></td>
<td></td>
</tr>
<tr>
<td>System Comp. • Homogenous • Asymmetric</td>
<td>Initiates Conn. Setup</td>
<td>Bits In</td>
<td>Sync</td>
<td>Data Representation</td>
</tr>
<tr>
<td>HW I/O Support • Serial, 1-Wire, Parallel, Digital, Ethernet, IrDA, Bluetooth, GSM, GPRS</td>
<td>Receive Conn. Requests</td>
<td>Bits Out</td>
<td>Async</td>
<td>• CORBA CDR</td>
</tr>
<tr>
<td>Resources • Memory • Power</td>
<td></td>
<td>Bits In/Out</td>
<td>(One-Way Msgs)</td>
<td>• MQC CDR</td>
</tr>
<tr>
<td>Processing Capabilities • 8-bit, 16-bit, …</td>
<td></td>
<td></td>
<td></td>
<td>• …</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Data Types
- CORBA CDR
- MQC CDR
- …
- TCP/IP
- UDP
- PPP
- 1-wire
- IIOP
- 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
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
Quality of Service

- 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
# Fault Tolerance Taxonomy

<table>
<thead>
<tr>
<th>Redundancy</th>
<th>Reliability</th>
<th>Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Temporal</td>
<td>• Group Communication</td>
<td>• Sender FIFO</td>
</tr>
<tr>
<td>– Multiple</td>
<td>– Best Effort</td>
<td>• Causal</td>
</tr>
<tr>
<td>transmits</td>
<td>– Reliable</td>
<td>– Logical Timestamping</td>
</tr>
<tr>
<td>• Spatial</td>
<td>– Atomic</td>
<td>• Total</td>
</tr>
<tr>
<td>– Multiple</td>
<td>– Uniform</td>
<td>– Sequencer / Token based</td>
</tr>
<tr>
<td>Channels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Replicated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Servers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Value</td>
<td>• Failure Detection</td>
<td></td>
</tr>
<tr>
<td>– Checksums, CRC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ordering:
- Sender FIFO
- Causal
- Logical Timestamping
- Total
- Sequencer / Token based
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
  – 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
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

Q1: Baseline Functionality
- Application Logic
- Power
- Memory
Node 1

Q2: QoS
- Security, Realtime, Fault Tolerance

Q3: Local-Global Interactions

Q4: Software Engineering
- Aspects
- Patterns
- Tools

Embedded Systems
Application Developer
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 tailororable 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
Follow-up

• Please visit our poster for more details and a chance to ask questions!
  – Poster Session: Wednesday, 1500 – 1600
  – Memory footprint results
  – Performance profiling results
• MicroQoSCORBA website
  – http://microqoscorba.eecs.wsu.edu