Flexible Fault Tolerance In Configurable Middleware For Embedded Systems

Kevin Dorow

Masters Thesis Defense Presentation
19 November 2002
Acknowledgment

• Dr. Bakken—advisor
• Committee members
• MicroQoSCORBA team
• My Family—Angie, KJ, and Ethan
Presentation Outline

• Introduction
• Architectural Taxonomy
• MicroQoSCORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
Introduction—1

• Explosion of embedded systems
  – Advancements in computing technology
  – 11 billion parts per year
  – Ubiquitous networking (soon all will be interconnected)
  – Provide a better and more integrated level of service

• Opportunities and difficulties associated with embedded system development
  – Restricted resources of devices
  – Reliance on proprietary, low-level mechanisms
  – Rapid evolution of computing technologies
Introduction—2

• Middleware technologies
  – Overcome difficulties in distributed systems (not specifically embedded systems)
  – “Plumbing” of an information system
  – Manage complexity and heterogeneity
  – Common Object Request Broker Architecture (CORBA)

• Middleware applied to embedded systems
  – General purpose frameworks fail to scale down
  – Do not have the flexibility to cover the wide space of the embedded systems market
  – Overlook the fundamental problems of partial failures and consistent ordering of distributed events (fault tolerance)
Introduction—3

• MicroQoSCORBA (MQC)
  – Configurable middleware framework for embedded systems
  – Tailorable to hardware and application requirements
  – Improved taxonomy of terms / orthogonal components
  – CASE toolkit to easily configure system / application constraints
Introduction—4

• Focus of this thesis
  – Integrate fault tolerance into MQC

• Contributions
  – Analysis, design, implementation of several fault-tolerant mechanisms into the MQC framework
  – Analysis, design, implementation of a highly configurable fault-tolerant group communication system (stand alone / integrated into MQC)
Presentation Outline

• Introduction
• **Architectural Taxonomy**
• MicroQoSCORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
Architectural Taxonomy—1

• Embedded system applications and hardware characteristics (wide space)
• Need to categorize this wide space (completely and orthogonally) for MQC
• We divided the properties of embedded systems and middleware architecture into a more fine-grained definition
## Architectural Taxonomy—2

<table>
<thead>
<tr>
<th>Embedded H/W</th>
<th>Roles (Client/Server/Peer)</th>
<th>SW I/O</th>
<th>IDL Subsetting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Control Flow</td>
<td>Data Flow</td>
<td>Interaction Style</td>
</tr>
<tr>
<td>System Comp.</td>
<td>Connection Setup</td>
<td>Data Direction</td>
<td>Sync (Send/Receive)</td>
</tr>
<tr>
<td>• Homogenous</td>
<td>• Initiates Setup</td>
<td>• Bits In</td>
<td>Async (One-Way Msgs)</td>
</tr>
<tr>
<td>• Asymmetric</td>
<td>• Receive Setup</td>
<td>• Bits Out</td>
<td>Msg Push</td>
</tr>
<tr>
<td>HW I/O Support</td>
<td>• Requests</td>
<td>• Bits In/Out</td>
<td>Msg Pull</td>
</tr>
<tr>
<td>• Serial</td>
<td>• Service Location</td>
<td>• Parallelism</td>
<td>Passive</td>
</tr>
<tr>
<td>• 1-Wire</td>
<td>• Hardwired Logic</td>
<td>• 1 msg in transit</td>
<td>Pro-Active</td>
</tr>
<tr>
<td>• Parallel (Digital)</td>
<td>• Config. File</td>
<td>• N msgs in transit</td>
<td>Event &amp; Notif. Services</td>
</tr>
<tr>
<td>• Ethernet</td>
<td>• Name Service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• IrDA</td>
<td>• Other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Bluetooth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• GSM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• GPRS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td></td>
<td></td>
<td></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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Data Representation
- CORBA CDR
- MQC CDR
- …
- Protocols
- TCP/IP
- UDP
- PPP
- 1-wire
- Direct (i.e., IIOP)
- Indirect (i.e., IIOP Gateway)

- Message Types
- Parameter Types
- Data Types
- Exceptions
- Message Payload
Presentation Outline

• Introduction
• Architectural Taxonomy
• MicroQoSCORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
MicroQoSCORBA Base Architecture—1

• Lifecycle Epochs
  – Important characteristics / constraints are provided in each epoch
  – MQC architecture and toolkit span epochs to capture as much of these characteristics as possible
  – Lifetime of an MQC project divided into 5 epochs: Design, IDL Compilation, Application Compilation, System / Application Startup, and Run Time
  – Each epoch provides opportunities to refine key facets of the application
## MicroQoS CORBA Base Architecture—2

<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>Bootp, dhcp</td>
</tr>
<tr>
<td></td>
<td>Network Startup</td>
<td></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 Base Architecture—3
MicroQoSCORBA Base Architecture—4

- **IDL Compiler**
  - “one size fits all” does not apply
  - Customized ORB implementation support
- **Customized ORBs and POAs**
  - Automatically generated via CASE toolkit or “hand coded”
- **Communications Layer**
  - Support for lighter-weight communications is provided
    - UDP, TCP, IIOP, IIOPlite
- **Quality of Service**
  - Realtime performance, security, and fault tolerance (focus of this thesis)
Presentation Outline

• Introduction
• Architectural Taxonomy
• MicroQoSCORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
Configurable Fault-Tolerant Mechanisms for MQC—1

• Most embedded applications require some level of dependability
• Dependability—The measure in which reliance can be justifiably be placed on the service delivered by a system
• Need for dependability (fly-by-wire systems, anti-lock brake systems, remote banking systems)
• Because of this need it is important that MQC have fault tolerance mechanisms
# Configurable Fault-Tolerant Mechanisms for MQC—2

## Redundancy
- **Temporal**
  - Multiple Transmits
- **Spatial**
  - Multiple Channels
- **Value**
  - Checksums
  - CRC

## Reliability
- **Group Communication**
  - Best Effort
  - Reliable
  - Uniform
  - Atomic
- **Failure Detection**

## Ordering
- **Sender FIFO**
  - Logical Timestamping
- **Causal**
  - Moving-Sequencer based
- **Total**
  - Moving-Sequencer based
Configurable Fault-Tolerant Mechanisms for MQC—3
Configurable Fault-Tolerant Mechanisms for MQC—4

• Temporal Redundancy
  – Doing the same thing more than once

• Spatial Redundancy
  – Having several copies of the same component

• Value Redundancy
  – Adding extra information about the value of data being stored or sent
Configurable Fault-Tolerant Mechanisms for MQC—5

• Reliability
  – Group Communication
    • Multi-point to multi-point communication through groups
    • Reliability aspects and ordering aspects
    • Fundamental building block of fault-tolerant distributed systems
  – Failure Detection
    • Detect and diagnose failed components in a system
Presentation Outline

• Introduction
• Architectural Taxonomy
• MicroQoSCORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
Implementation and Integration—1

- Temporal Redundancy
  - Communication Channel
  - Tolerate $k$ omission failures by transmitting messages $k+1$ times
  - Number of retransmissions configurable from CASE toolkit
  - Client and Server code modified to handle retransmissions and duplicate filtering
Implementation and Integration—2

• Spatial Redundancy
  – Communications path
  – Provides message omission and channel failure tolerance
  – Communication paths to be supported is configurable from the CASE toolkit
  – Client code modified to filter duplicate messages received
Implementation and Integration—3

• Value Redundancy
  – Checksum included in messages
  – CRC32 checksum selected
  – Selectable from the CASE toolkit

• Group Communication
  – Developed as a stand alone system
  – Integrated into MQC (Server side—replication)
Implementation and Integration—4

- Group Communication (cont.)
  - Nonuniform Failure-Atomic Multicast (2 phase)
  - Dynamically Uniform Failure-Atomic Multicast (3 phase)
  - Ordering
    - FIFO Ordered Multicast (ID based)
    - Causal Ordered Multicast (vector timestamp)
    - Totally Ordered Multicast (moving sequencer)
  - All modes are configurable from the CASE toolkit
Implementation and Integration—5

• Failure Detection
  – Implemented as part of the Group Communications package
  – User can specify a timeout in the CASE toolkit
  – Group Communications package monitors the transmission of messages
  – If a message is late it is assumed that the group member has crashed (removed from group)
Presentation Outline

• Introduction
• Architectural Taxonomy
• MicroQoSCORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
Results and Analysis—1

• Testing Platforms
  – Two desktop systems (1.5GHz P4, Linux, 100 BaseT network)
  – Two TINI boards (40Mhz Dallas Semiconductor DS80C390 CPU)
  – Two Systronixs SaJe boards (100Mhz aJile CPU)
Results and Analysis—2

- Impact on memory footprint
  - “Simple” application selected for analysis
  - Client and Server class file sizes examined
  - Some numbers for TINI and AJILE not available due to UDP

<table>
<thead>
<tr>
<th>Application: Simple -- (use TCP and GIOP unless stated otherwise)</th>
<th>Server Class File Size (bytes)</th>
<th>Linux</th>
<th>TINI</th>
<th>AJILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Fault Tolerance</td>
<td></td>
<td>68,654</td>
<td>32,882</td>
<td>253,985</td>
</tr>
<tr>
<td>Temporal Redundancy</td>
<td></td>
<td>69,067</td>
<td>33,022</td>
<td>254,161</td>
</tr>
<tr>
<td>Spatial Redundancy (TCP and Unreliable UDP)</td>
<td></td>
<td>88,287</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform)</td>
<td></td>
<td>137,800</td>
<td>63,014</td>
<td>NA</td>
</tr>
<tr>
<td>Group Comm (Uniform)</td>
<td></td>
<td>142,596</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Value Redundancy</td>
<td></td>
<td>70,109</td>
<td>33,506</td>
<td>254,793</td>
</tr>
</tbody>
</table>
Results and Analysis—3

- Impact on Memory Footprint (cont.)
  - Impact to class file size is very small (except for Group Communication options)
  - Group Communication is not optimized for MQC (designed as a stand alone—uses its own transports, etc.)

<table>
<thead>
<tr>
<th>Application: Simple -- (use TCP and GIOP unless stated otherwise)</th>
<th>Linux</th>
<th>TINI</th>
<th>AJILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Fault Tolerance</td>
<td>53,189</td>
<td>26,886</td>
<td>243,496</td>
</tr>
<tr>
<td>Temporal Redundancy</td>
<td>53,812</td>
<td>27,805</td>
<td>243,764</td>
</tr>
<tr>
<td>Spatial Redundancy (TCP and Unreliable UDP)</td>
<td>62,488</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform)</td>
<td>53,307</td>
<td>26,911</td>
<td>NA</td>
</tr>
<tr>
<td>Group Comm (Uniform)</td>
<td>53,307</td>
<td>26,911</td>
<td>NA</td>
</tr>
<tr>
<td>Value Redundancy</td>
<td>54,913</td>
<td>27,594</td>
<td>244,395</td>
</tr>
</tbody>
</table>
Results and Analysis—4

• Impact on Execution Times
  – “Timing” application selected (simple program with one method that passes a long from the client to the server and back)
  – Average call time calculated from many iterations
Results and Analysis—5

<table>
<thead>
<tr>
<th>Application: Timing (Use TCP and GIOP unless stated otherwise)</th>
<th>Execution Time (microseconds)</th>
<th>Linux</th>
<th>TINI</th>
<th>AJILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Fault Tolerance</td>
<td></td>
<td>307</td>
<td>212,667</td>
<td>5,355</td>
</tr>
<tr>
<td>Temporal Redundancy (2 retrans)</td>
<td></td>
<td>387</td>
<td>278,750</td>
<td>9,534</td>
</tr>
<tr>
<td>Temporal Redundancy (4 retrans)</td>
<td></td>
<td>457</td>
<td>407,750</td>
<td>17,349</td>
</tr>
<tr>
<td>Temporal Redundancy (6 retrans)</td>
<td></td>
<td>519</td>
<td>589,000</td>
<td>26,660</td>
</tr>
</tbody>
</table>

- Temporal Redundancy
  - Increase in time is due the retransmitting of the message along with the duplicate filter processing
  - Impact is linear and small on PC, but as CPU power goes down the impact is much more significant
Results and Analysis—6

<table>
<thead>
<tr>
<th>Application: Timing (Use TCP and GIOP unless stated otherwise)</th>
<th>Linux</th>
<th>TINI</th>
<th>AJILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (microseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Fault Tolerance</td>
<td>307</td>
<td>212,667</td>
<td>5,355</td>
</tr>
<tr>
<td>Spatial Redundancy (TCP and Unreliable UDP)</td>
<td>564</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

• Spatial Redundancy
  – Impact is about double (exactly what would be expected—twice as long to send the message down two paths)
  – Numbers not available on TINI and AJILE due to problems with UDP implementation
Results and Analysis—7

- **Value Redundancy**
  - Impact is very dependent on CPU processing power (due to CRC32 calculation)
  - Insertion and extraction of checksum value also has an impact

<table>
<thead>
<tr>
<th>Application: Timing (Use TCP and GIOP unless stated otherwise)</th>
<th>Execution Time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linux</td>
</tr>
<tr>
<td>No Fault Tolerance</td>
<td>307</td>
</tr>
<tr>
<td>Value Redundancy</td>
<td>388</td>
</tr>
</tbody>
</table>
Results and Analysis—8

• Group Communication
  – Replicated server configuration
  – Impact is significant (expected)
    • Not tightly integrated (uses own transport)
    • Group Communication requires the transmission of a large amount of messages (1 message becomes 10 in the worst case)

<table>
<thead>
<tr>
<th>Application: Timing (Use TCP and GIOP unless stated otherwise)</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Fault Tolerance</td>
<td>307</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform--2 members no order)</td>
<td>655</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform--2 members total order)</td>
<td>795</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform--3 members no order)</td>
<td>860</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform--3 members total order)</td>
<td>1019</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform--4 members no order)</td>
<td>1053</td>
</tr>
<tr>
<td>Group Comm (Non-Uniform--4 members total order)</td>
<td>1120</td>
</tr>
<tr>
<td>Group Comm (Uniform--2 members no order)</td>
<td>764</td>
</tr>
<tr>
<td>Group Comm (Uniform--2 members Total order)</td>
<td>787</td>
</tr>
<tr>
<td>Group Comm (Uniform--3 members no order)</td>
<td>921</td>
</tr>
<tr>
<td>Group Comm (Uniform--3 members Total order)</td>
<td>1060</td>
</tr>
<tr>
<td>Group Comm (Uniform--4 members no order)</td>
<td>1151</td>
</tr>
<tr>
<td>Group Comm (Uniform--4 members Total order)</td>
<td>1170</td>
</tr>
</tbody>
</table>
Presentation Outline

- Introduction
- Architectural Taxonomy
- MicroQoSCORBA Base Architecture
- Configurable Fault-Tolerant Mechanisms for MicroQoSCORBA
- Implementation and Integration
- Results and Analysis
- Related Work
- Conclusion
Related Work—1

• MQC allows constraints to be chosen at much finer granularity
  – Designed from the device level up
• We have found no middleware framework that allows the hardware and application constraints to be used for tailoring the middleware
• We have found no middleware framework for small embedded devices that supports multiple QoS properties (fault tolerance tends to be completely overlooked)
Related Work—2

- Projects / Products that are starting to address embedded system issues:
  - MinimumCORBA (cut down version of full CORBA)
    - Still too large to be useable on most small embedded devices
  - e*ORB framework by Vertel
    - Smaller, but no flexibility provided
  - LegORB
    - Very few details available—some customizations allowed (much less granularity) and no QoS
  - OMG Smart Transducers Interface RFP
    - Focused on lightweight communications only
  - MMLite
    - Targeted for embedded systems and real time—narrow QoS breadth
  - Aegis Weapon System (AWS)
    - CORBA with custom UDP implementation
Presentation Outline

• Introduction
• Architectural Taxonomy
• MicroQoS CORBA Base Architecture
• Configurable Fault-Tolerant Mechanisms for MicroQoS CORBA
• Implementation and Integration
• Results and Analysis
• Related Work
• Conclusion
Conclusion—1

• Project was successful!—we were able to identify, design, implement, and integrate several important fault-tolerant mechanisms into MQC

• Areas for Future Work
  – Configurable means of how to deal with duplicate messages that are received due to redundancy (error detection, voting, etc.)
  – Incorporate more sophisticated error correction codes for value redundancy
  – Support for more modes of communication in the group communication package (virtual synchrony, TO with causality)
  – Tighter integration of the group communication package into MQC (use MQC’s existing transport mechanism, message header)
  – Group communication support on the client
  – Failure detection as an independent mechanism
Conclusion—2

• Publications:
Conclusion—3

• Questions / Comments??
• Thanks!!!!