Real-time Middleware for Distributed and Embedded Systems

Douglas C. Schmidt

Associate Professor & Computer Engineering Dept.
www.cs.wustl.edu/~schmidt/
University of California, Irvine

Sponsors
NSF, DARPA, Bellcore/Telcordia, BBN, Boeing, CDI/GDIS, Experian, Global MT, Hughes, Kodak, Krones, Lockheed, Lucent, Microsoft, Motorola, Nokia, Nortel, OCI, OTI, Raytheon, SAIC, Siemens SCR, Siemens MED, Siemens ZT, Sprint, USENIX

November 1999
Motivation: the Telecom Software Crisis

- **Symptoms**
  - Network element *hardware* gets smaller, faster, cheaper
  - Communication *software* gets larger, slower, more expensive

- **Culprits**
  - *Inherent* and *accidental* complexity

- **Solution Approach**
  - Manage & control network elements via *QoS-enabled middleware*
Goal: Apply Embedded Middleware to Network Element Mgmt & Control

Domain Challenges

- High-speed (20 Gbps) ATM switches w/embedded controllers
- Low-latency and statistical real-time deadlines
- COTS infrastructure, standards-based open systems, and small footprint

Washington University, St. Louis
Problem: Low-level Switch Control & Management (e.g., GSMP & VSI)

Features
- Setup & release connections
- Add & delete point-to-multipoint leaves
- Manage ATM switch ports
- Request configuration information & statistics

Drawbacks
- Non-portable, tedious, and error-prone programming APIs

Washington University, St. Louis
Real-time and Embedded ORBs

Proxy Server Configuration

Features

- Supports standard CORBA programming API
- Can use standard ORB
- Transparent to existing GSMP & VSI servers
- Scales to distributed configuration
  - *i.e.*, one CP can control multiple switches

Features

- Supports standard CORBA programming API
- Can use standard ORB
- Transparent to existing GSMP & VSI servers
- Scales to distributed configuration
  - *i.e.*, one CP can control multiple switches
Embedded ORB Configuration

Features

- Leverages middleware flexibility and standardization
- Multiple protocols can be supported
  - GSMP & VSI in-line bridging, GIOP/IIOP, etc.
- Minimal footprint
Context: Levels of Abstraction in Internetworking and Middleware

INTERNETWORKING ARCH
- DNS
- TELNET
- FTP
- HTTP
- UDP
- IP
- ETHERNET
- ATM
- FDDI
- FIBRE CHANNEL

MIDDLEWARE ARCH
- CORBA SERVICES
- CORBA APPLICATIONS
- WIN NT
- LINUX
- LYNXOS
- SOLARIS
- VXWORKS
Problem: Lack of QoS-enabled Middleware

- Many telecom applications require QoS guarantees
  - e.g., call-processing, network/switch management, wireless systems
- Building these applications manually is hard
- Existing middleware doesn’t support QoS effectively
  - e.g., CORBA, DCOM, DCE, Java
- Solutions must be integrated horizontally & vertically
Candidate Solution: CORBA

Goals of CORBA

- Simplify distribution by automating
  - Object location & activation
  - Parameter marshaling
  - Demultiplexing
  - Error handling
- Provide foundation for higher-level services

www.cs.wustl.edu/~schmidt/corba.html
Real-time and Embedded ORBs

Caveat: Requirements/Limitations of CORBA for Telecom

www.cs.wustl.edu/~schmidt/RT-ORB.ps.gz

Requirements
- Location transparency
- Performance transparency
- Predictability transparency
- Reliability transparency

Limitations
- Lack of QoS specifications
- Lack of QoS enforcement
- Lack of real-time programming features
- Lack of performance optimizations
Overview of the Real-time CORBA Specification

Features

1. End-to-end priority propagation
2. Protocol properties
3. Thread pools
4. Explicit binding
5. Standard synchronizers

www.cs.wustl.edu/~schmidt/oorc.ps.gz
Real-time and Embedded ORBs

Our Approach: The ACE ORB (TAO)

TAO Overview

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX/UNIX, Win32, & RTOS platforms
  - e.g., VxWorks, Chorus, LynxOS
- Leverages ACE

www.cs.wustl.edu/~schmidt/TAO.html
ACE Overview →

- A concurrent OO networking framework
- Available in C++ and Java
- Ported to POSIX, Win32, and RTOSs

Related work →

- x-Kernel
- SysV STREAMS

www.cs.wustl.edu/~schmidt/ACE.html
ACE and TAO Statistics

- Over 35 person-years of effort
  - ACE > 200,000 LOC
  - TAO > 125,000 LOC
  - TAO IDL compiler > 100,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - www.cs.wustl.edu/~schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, Boeing, Ericsson, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE ➔ www.riverace.com
  - TAO ➔ www.ociweb.com

Washington University, St. Louis
Applying TAO to Avionics Mission Computing

Domain Challenges

- Deterministic & statistical real-time deadlines
- Periodic & aperiodic processing
- COTS and open systems
- Reusable components
- Support platform upgrades

www.cs.wustl.edu/~schmidt/TAO-boeing.html

www.cs.wustl.edu/~schmidt/JSAC-98.ps.gz
Applying TAO to Other Performance-Sensitive Applications

Medical Imaging

Satellite Surveillance
Problem: Optimizing Complex Software

Common Problems →

- Optimizing complex software is hard
- Small “mistakes” can be costly

Solution Approach (Iterative) →

- Pinpoint overhead via white-box metrics
  - e.g., Quantify and VMEtro
- Apply patterns and framework components
- Revalidate via white-box and black-box metrics

www.cs.wustl.edu/~schmidt/JSAC-99.ps.gz
Solution 1: Patterns and Framework Components

Definitions

- **Pattern**
  - A solution to a problem in a context

- **Framework**
  - A “semi-complete” application built with components

- **Components**
  - Self-contained, “pluggable” ADTs

www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz

Definitions

- **Pattern**
  - A solution to a problem in a context

- **Framework**
  - A “semi-complete” application built with components

- **Components**
  - Self-contained, “pluggable” ADTs

www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz
Real-time and Embedded ORBs

Solution 2: ORB Optimization
Principle Patterns

Definition

- **Optimization principle patterns** document rules for avoiding common design and implementation problems that can degrade the performance, scalability, and predictability of complex systems

Key Principle Patterns Used in TAO

<table>
<thead>
<tr>
<th>#</th>
<th>Principle Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optimize for the common case</td>
</tr>
<tr>
<td>2</td>
<td>Remove gratuitous waste</td>
</tr>
<tr>
<td>3</td>
<td>Replace inefficient general-purpose functions with efficient special-purpose ones</td>
</tr>
<tr>
<td>4</td>
<td>Shift computation in time, <em>e.g.</em>, precompute</td>
</tr>
<tr>
<td>5</td>
<td>Store redundant state to speed-up expensive operations</td>
</tr>
<tr>
<td>6</td>
<td>Pass hints between layers and components</td>
</tr>
<tr>
<td>7</td>
<td>Don’t be tied to reference implementations/models</td>
</tr>
<tr>
<td>8</td>
<td>Use efficient/predictable data structures</td>
</tr>
</tbody>
</table>

Washington University, St. Louis
ORB Latency and Priority Inversion Experiments

Method

- Vary ORBs, hold OS constant
- Solaris real-time threads
- High priority client $C_0$ connects to servant $S_0$ with matching priorities
- Clients $C_1 \ldots C_n$ have same lower priority
- Clients $C_1 \ldots C_n$ connect to servant $S_1$
- Clients invoke two-way CORBA calls that cube a number on the servant and returns result

www.cs.wustl.edu/~schmidt/RT-perf.ps.gz
ORB Latency and Priority Inversion Results

Synopsis of Results

- TAO’s latency is lowest for large # of clients
- TAO avoids priority inversion
  - i.e., high priority client always has lowest latency
- Primary overhead stems from concurrency and connection architecture
  - e.g., synchronization and context switching
ORB Jitter Results

Definition

- Jitter $\rightarrow$ standard deviation from average latency

Synopsis of Results

- TAO’s jitter is lowest and most consistent
- CORBAplus’ jitter is highest and most variable
Problem: Improper ORB Concurrency Models

Common Problems

- High context switching and synchronization overhead
- Thread-level and packet-level priority inversions
- Lack of application control over concurrency model

www.cs.wustl.edu/~schmidt/CACM-arch.ps.gz
Problem: ORB Shared Connection Models

Common Problems

- Request-level priority inversions
  - Sharing multiple priorities on a single connection
- Complex connection multiplexing
- Synchronization overhead

www.cs.wustl.edu/~schmidt/RTAS-98.ps.gz
Real-time and Embedded ORBs

**Problem: High Locking Overhead**

Common Problems

- Locking overhead affects latency and jitter significantly
- Memory management commonly involves locking

www.cs.wustl.edu/~schmidt/RTAS-98.ps.gz
Solution: TAO’s ORB Endsystem Architecture

Solution Approach
- Integrate scheduler into ORB endsystem
- Co-schedule threads
- Leader/followers thread pool

Principle Patterns
- Pass hints, precompute, optimize common case, remove gratuitous waste, store state, don’t be tied to reference implementations & models
Thread Pool Comparison Results

Worker Thread Pool

Leader/Follower Thread Pool

Performance Improvement

Threads

Load

Washington University, St. Louis
Problem: Reducing Demultiplexing Latency

Design Challenges

- Minimize demuxing layers
- Provide $O(1)$ operation demuxing through all layers
- Avoid priority inversions
- Remain CORBA-compliant

www.cs.wustl.edu/~schmidt/POA.ps.gz
Real-time and Embedded ORBs

Solution: TAO’s Request Demultiplexing Optimizations

(A) LAYERED DEMUXING, LINEAR SEARCH

(B) LAYERED DEMUXING, DYNAMIC HASHING

(C) LAYERED DEMUXING, PERFECT HASHING

(D) DE-LAYERED ACTIVE DEMUXING

Demuxing

- www.cs.wustl.edu/~schmidt/{ieee_tc-97,COOTS-99}.ps.gz

Perfect hashing

- www.cs.wustl.edu/~schmidt/gperf.ps.gz

Washington University, St. Louis
POA Demultiplexing Results

Synopsis of Results
- Active demux is efficient & predictable for both transient and persistent object references.

Principle Patterns
- Precompute, pass hints, use special-purpose & predictable data structures

Washington University, St. Louis
Real-time and Embedded ORBs

Servant Demultiplexing Results

Synopsis of Results
- Linear demux is costly
- Active demux is most efficient & predictable

Principle Patterns
- Precompute, pass hints, use special-purpose & predictable data structures

Washington University, St. Louis
Operation Demultiplexing Results

Synopsis of Results

- Perfect Hashing
  - Highly predictable
  - Low-latency
- Others strategies slower

Principle Patterns

- Precompute, use predictable data structures, remove gratuitous waste
TAO Request Demultiplexing Summary

<table>
<thead>
<tr>
<th>Demultiplexing Stage</th>
<th>Absolute Time ((\mu s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Request parsing</td>
<td>2</td>
</tr>
<tr>
<td>2. POA demux</td>
<td>2</td>
</tr>
<tr>
<td>3. Servant demux</td>
<td>3</td>
</tr>
<tr>
<td>4. Operation demux</td>
<td>2</td>
</tr>
<tr>
<td>5. Parameter demarshaling</td>
<td>operation dependent</td>
</tr>
<tr>
<td>6. User upcall</td>
<td>servant dependent</td>
</tr>
<tr>
<td>7. Results marshaling</td>
<td>operation dependent</td>
</tr>
</tbody>
</table>
Real-time and Embedded ORBs

Real-time ORB/OS Performance Experiments

Method

- Vary OS, hold ORBs constant
- Single-processor Intel Pentium II 450 Mhz, 256 Mbytes of RAM
- Client and servant run on the same machine
- Client $C_i$ connects to servant $S_i$ with priority $P_i$
  - $i$ ranges from $1 \ldots 50$
- Clients invoke two-way CORBA calls that cube a number on the servant and returns result

requests

[p]Priority

Client

servants

[O]bject Adapter

ORB Core

I/O SUBSYSTEM

Server

Pentium II

www.cs.wustl.edu/~schmidt/RT-OS.ps.gz
Real-time ORB/OS Performance Results

High-priority Client Latency

Low-priority Clients Latency
Real-time ORB/OS Jitter Results

High-priority Client Jitter

Low-priority Clients Jitter

LynxOS
NT
VxWorks
Linux
Solaris

Low Priority Clients

Two-way Jitter, usec

0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

Low Priority Clients

Two-way Jitter, usec

0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

Two-way Jitter, usec
Problem: Hard-coded ORB Messaging and Transport Protocols

- GIOP/IIOP are not sufficient, *e.g.*:
  - GIOP message footprint may be too large
  - TCP lacks necessary QoS
  - Legacy commitments to existing protocols

- Many ORBs do not support “pluggable protocols”
  - This makes ORBs inflexible and inefficient
One Solution: Hacking GIOP

- GIOP requests include fields that aren’t needed in homogeneous embedded applications
  - e.g., GIOP magic #, GIOP version, byte order, request principal, etc.
- These fields can be omitted without any changes to the standard CORBA programming model
- TAO’s `--ORBgioplite` option save 15 bytes per-request, yielding these calls-per-second:

<table>
<thead>
<tr>
<th></th>
<th>Marshaling-enabled</th>
<th>Marshaling-disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>GIOP</td>
<td>2,878</td>
<td>2,937</td>
</tr>
<tr>
<td>GIOPlite</td>
<td>2,883</td>
<td>2,978</td>
</tr>
</tbody>
</table>

- The result is a measurable improvement in throughput/latency
  - However, it’s so small (2%) that hacking GIOP is of minimal gain except for low-bandwidth links

Washington University, St. Louis
Better Solution: TAO’s Pluggable Protocols Framework

Features

- Pluggable ORB messaging and transport protocols
- Highly efficient and predictable behavior

Principle Patterns

- Replace general-purpose functions (protocols) with special-purpose ones

Washington University, St. Louis
### CORBA Protocol Interoperability Architecture

#### Standard CORBA Programming API

<table>
<thead>
<tr>
<th>ORB Messaging Component</th>
<th>GIOP</th>
<th>GIOPlite</th>
<th>ESIOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORB Transport Adapter Component</td>
<td>IIOP</td>
<td>VME-IOP</td>
<td>ATM-IOP RELIABLE SEQUENCED</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>TCP</td>
<td>VME DRIVER</td>
<td>AAL5</td>
</tr>
<tr>
<td>Network Layer</td>
<td>IP</td>
<td>ATM</td>
<td></td>
</tr>
</tbody>
</table>

#### Protocol Configurations

- **Presentation layer**
  - e.g., CDR
- **Message formats**
  - e.g., GIOP
- **Transport assumptions**
  - e.g., TCP
- **Object addressing**
  - e.g., IIOP IOR

www.cs.wustl.edu/~schmidt/pluggable_protocols.ps.gz
Embedded System Benchmark Configuration

VxWorks running on 200 Mhz PowerPC over a 320 Mbps VME & 10 Mbps Ethernet

Washington University, St. Louis
Synopsis of Results

- VME protocol is much faster than Ethernet
- No application changes are required to support VME

Washington University, St. Louis
Pinpointing ORB Overhead with VMetro Timeprobes

- Timeprobes use VMetro monitor, which measures end-to-end time
- Timeprobe overhead is minimal, *i.e.*, 1 $\mu$sec

Washington University, St. Louis
Real-time and Embedded ORBs

ORB & VME One-way Overhead Results

Synopsis of Results

- ORB overhead is relatively constant and low
  - e.g., \( \sim 110 \) \( \mu \)secs per end-to-end operation
- Bottleneck is VME driver and OS, not ORB

Washington University, St. Louis
ORB & Transport Overhead Results

**Synopsis of Results**

- ORB overhead is relatively constant and low
  - *e.g.*, $\sim 49 \mu$secs per two-way operation
- Bottleneck is OS and I/O operation

Washington University, St. Louis
Lessons Learned Developing Real-time ORBs

- Avoid dynamic connection management
- Minimize dynamic memory management and data copying
- Avoid multiplexing connections for different priority threads
- Avoid complex concurrency models
- Integrate ORB with OS and I/O subsystem and avoid reimplementing OS mechanisms
- Guide ORB design by empirical benchmarks and patterns
Summary of TAO Research Project

Completed work
- First POA and first deployed real-time CORBA scheduling service
- Pluggable protocols framework
- Minimized ORB Core priority inversion and non-determinism
- Reduced latency via demuxing optimizations
- Co-submitters on OMG’s real-time CORBA spec

Ongoing work
- Dynamic/hybrid scheduling
- Distributed QoS, ATM I/O Subsystem, & open signaling
- Implement CORBA Real-time, Messaging, and Fault Tolerance specs
- Tech. transfer via DARPA Quorum program and www.theaceorb.com
  - Integration with Flick IDL compiler, QuO, TMO, etc.
Concluding Remarks

- Researchers and developers of distributed, real-time telecom applications confront many common challenges
  - e.g., service initialization and distribution, error handling, flow control, scheduling, event demultiplexing, concurrency control, persistence, fault tolerance
- Successful researchers and developers apply patterns, frameworks, and components to resolve these challenges
- Careful application of patterns can yield efficient, predictable, scalable, and flexible middleware
  - i.e., middleware performance is largely an “implementation detail”
- Next-generation ORBs for telecom will be highly QoS-enabled, though many research challenges remain
Web URLs for Additional Information

- **Real-time CORBA 1.0 spec:**
  
  
  www.cs.wustl.edu/~schmidt/oorc.ps.gz

- **More information on TAO:**
  
  www.cs.wustl.edu/~schmidt/TAO.html

- **TAO real-time event channel:**
  
  www.cs.wustl.edu/~schmidt/JSAC-98.ps.gz

- **TAO static scheduling:**
  
  www.cs.wustl.edu/~schmidt/RT-ORB.ps.gz

- **TAO dynamic scheduling:**
  
  www.cs.wustl.edu/~schmidt/dynamic.ps.gz