Patterns and Performance of Real-time Middleware for Embedded Systems

Douglas C. Schmidt

Associate Professor & Director of the Center for Distributed Object Computing

Lockheed Martin

November 1st, 1999

Motivation: the QoS-enabled Software Crisis

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

- Culprits
  - Inherent and accidental complexity

- Solution Approach
  - Standard communication middleware

Candidate Solution: CORBA

Goals of CORBA

- Simplify distribution by automating
  - Object location & activation
  - Parameter marshaling
  - Demultiplexing
  - Error handling

- Provide foundation for higher-level services
Overview of the Real-time CORBA Specification

Features
1. End-to-end priority propagation
2. Protocol properties
3. Thread pools
4. Explicit binding
5. Mutex IDL

Limitations
- Lack of QoS specifications
- Lack of QoS enforcement
- Lack of real-time programming features
- Lack of performance optimizations

Relevance to Real-time and Embedded ORBs
- End-to-end priority
- Protocol properties
- Thread pools
- Explicit binding
- Mutex IDL

Related work
- x-Kernel
- SysV STREAMS
Real-time and Embedded ORBs

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

Over 35 person-years of effort

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

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
**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**

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

**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

**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

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

Problem: Reducing Demultiplexing Latency

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

Solution: TAO's Request Demultiplexing Optimizations

Demuxing

Perfect hashing
- www.cs.wustl.edu/~schmidt/gperf.ps.gz
**TAO Request Demultiplexing Summary**

Demultiplexing Stage | Absolute Time (μs)
--- | ---
1. Request parsing | 2
2. POA demux | 2
3. Servant demux | 3
4. Operation demux | 2
5. Parameter demarshaling | operation dependent
6. User upcall | servant dependent
7. Results marshaling | operation dependent

**Servant Demultiplexing Results**

- **Active Demux** is most efficient & predictable
- **Linear Demux** is costly

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

**Synopsis of Results**
- Perfect Hashing
  - Highly predictable
  - Slower
- Dynamic Hashing
  - Others strategies
- Linear Search
  - Remove gratuitous waste
- Binary Search
  - Use predictable data structures, remove gratuitous waste
- Perfect Hashing
  - Highly predictable
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 saves 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>min</td>
<td>2,878</td>
<td>2,937</td>
</tr>
<tr>
<td>max</td>
<td>2,963</td>
<td>2,996</td>
</tr>
<tr>
<td>avg</td>
<td>2,906</td>
<td>2,949</td>
</tr>
</tbody>
</table>

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

Problem: Hard-coded ORB Messaging

Many ORBs do not support “pluggable protocols,” e.g., GIOP message footprint may be too large, TCP lacks necessary QoS, legacy commitments to existing protocols.

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

Better Solution: TAO's Pluggable Protocols Framework

**CLIENT**

**STUBS**

**SKELETONS**

TCP

MULTICAST

IOP

VME

UDP

ORB MESSAGING COMPONENT

ORB TRANSPORT ADAPTER COMPONENT

ESIOP

REAL-TIME

IOP

EMBEDDED IOP

RELIABLE, BYTE-STREAM

ATM

TCP

MEMORY MANAGEMENT

CONCURRENCY

MODEL

OTHER ORB CORE SERVICES

COMMUNICATION INFRASTRUCTURE

HIGH SPEED NETWORK INTERFACE

REAL-TIME I/O SUBSYSTEM

ORB MESSAGE FACTORY

ORB TRANSPORT ADAPTER FACTORY

OBJECT ADAPTER

GIOP

GIOPLITE

ADAPTIVE Communication Environment (ACE)

OBJECT (SERVANT) operation (args)

IN  ARGS

OUT  ARGS  &  RETURN  VALUE

POLICY

CONTROL

Features

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

Principle Patterns

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

Features

- 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

Washington University, St. Louis

Embedded System Benchmark Configuration

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

Ethernet & VME Two-way Latency

Synopsis of Results

- VME protocol is much faster than Ethernet

- No application changes are required to support VME
Pinpointing ORB Overhead with VMEtro Timeprobes

- Timeprobes use VMEtro monitor, which measures end-to-end time
- Timeprobe overhead is minimal, i.e., 1 μsec

Synopsis of Results
- ORB overhead is relatively constant and low
  - e.g., ~110 μsecs per two-way operation
- Bottleneck is VME driver and OS, not ORB

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
Concluding Remarks

- Researchers and developers of distributed, real-time 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 will be highly QoS-enabled, though many research challenges remain

Web URLs for Additional Information

- **Real-time CORBA 1.0 spec:**
- **More information on TAO:**
  [www.cs.wustl.edu/~schmidt/TAO.html](http://www.cs.wustl.edu/~schmidt/TAO.html)
- **TAO static scheduling:**
  [www.cs.wustl.edu/~schmidt/RT-ORB.ps.gz](http://www.cs.wustl.edu/~schmidt/RT-ORB.ps.gz)
- **TAO dynamic scheduling:**
  [www.cs.wustl.edu/~schmidt/dynamic.ps.gz](http://www.cs.wustl.edu/~schmidt/dynamic.ps.gz)