Patterns and Performance of Real-time Object Request Brokers

Douglas C. Schmidt
Associate Professor  Elec. & Comp. Eng. Dept.
schmidt@uci.edu  University of California, Irvine
www.ece.uci.edu/~schmidt/  (949) 824-1901

Sponsors
NSF, DARPA, ATD, BBN, Boeing, Cisco, Comverse, GDIS, Experian, Global MT, Hughes, Kodak, Krone, Lockheed, Lucent, Microsoft, Mitre, Motorola, NASA, Nokia, Nortel, OIC, Oresis, OTI, Raytheon, SAIC, Siemens SCR, Siemens MED, Siemens ZT, Sprint, Telcordia, USENIX

High-performance, Real-time ORBs

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
  - Standards-based COTS Hardware & Software

www.arl.wustl.edu/arl/

UC Irvine

Context: Levels of Abstraction in Internetworking and Middleware

- Context
  - Adopting COTS hardware & software is increasingly essential for real-time mission-critical systems

- Problems
  - Inherent and accidental complexity
  - Integration woes

- Solution Approach
  - Standards-based adaptive COTS middleware

UC Irvine
Problem: Lack of QoS-enabled Middleware

- Many applications require QoS guarantees
  - e.g., avionics, telecom, WWW, medical, high-energy physics
- Building these applications manually is hard and inefficient
- 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

Overview of the Real-time CORBA Specification

Features

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

Caveat: Requirements/Limitations of CORBA for QoS-enabled Systems

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
**Features**

- Designed to support heterogeneous real-time platforms
- CORBA priorities range from 0 - 32767
- Users can map CORBA priorities to native OS priorities
- No silver bullet, but rather an "enabling technique"

---

**End-to-End Priority Propagation**

(A) **SERVER DECLARED MODEL**

(1) Server priority is pre-set

(2) Priority is exported in IOR

(3) Client's priority is not propagated by invocation

(B) **CLIENT PROPAGATED MODEL**

GLOBAL CORBA PRIORITY = 100

Features

- Client priorities can propagate end-to-end
- Servers can also declare priority
Explicit Binding

_validate_connection (out CORBA::PolicyList inconsistent_policies);

Features

- Enables pre-establishment of connections
  - Priority-banded connections
  - Private connections
  - Protocol policies

Standard Synchronizers

Mutex

lock() unlock() try_lock()

Features

- A portable Mutex API
  - e.g., lock, unlock, try_lock
- Necessary to ensure consistency between ORB and application synchronizers
  - e.g., priority inheritance and priority ceiling protocols
- Locality constrained

Buffering Requests

Requests are buffered when all threads are busy
Buffering can be specified in terms of:

- Number of bytes
- Number of requests

When buffers are full:

- A transient exception is thrown to client
- Request is dropped by server
- Request can be reissued later by client
High-performance, Real-time ORBs

Additional Information on Real-time CORBA

- Real-time CORBA 1.0 specification
- Many papers at my Web site
  - www.cs.wustl.edu/~schmidt/corba-research-realtime.html
- Upcoming OMG Real-time and Embedded CORBA Workshop
  - www.omg.org/meetings/realtime/
- Real-time ORBs
  - HighComm → www.highcomm.com
  - ORB Express → www.ois.com
  - TAO → www.theaceorb.com

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

ACE Overview →

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

ACE and TAO Statistics

- Over 50 person-years of effort
  - ACE > 200,000 LOC
  - TAO > 200,000 LOC
  - TAO IDL compiler > 130,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - ACE → www.riverace.com
  - TAO → www.theaceorb.com

The ADAPTIVE Communication Environment (ACE)

ACE Overview →

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

Related work →

- x-Kernel
- SysV STREAMS

ACE and TAO Statistics

- Currently used by dozens of companies
  - Bellcore, BBN, Boeing, Ericsson, Hughes, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE → www.riverace.com
  - TAO → www.theaceorb.com

ACE Overview

- Over 50 person-years of effort
  - ACE > 200,000 LOC
  - TAO > 200,000 LOC
  - TAO IDL compiler > 130,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - ACE → www.riverace.com
  - TAO → www.theaceorb.com
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

High-speed (20 Gbps) ATM switches

High latency and statistical real-time deadlines

COTS infrastructure, standards-based open systems, and small footprint

Domain Challenges

- High scalability and group communication
- High throughput and low latency
- "Interactive" real-time
- Multi-platform

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

Key Challenges

- Alleviate priority inversion and non-determinism
- Reduce demultiplexing latency/jitter
- Ensure protocol flexibility
- Specify QoS requirements
- Schedule operations
- Eliminate (de)marshaling overhead
- Minimize footprint
**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

---

**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

---

**Definitions**
- **Pattern** – A solution to a problem in a context
- **Framework** – A “semi-complete” application built with components
- **Components** – Self-contained, “pluggable” ADTs

---

**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 return result
**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 → standard deviation from average latency

**Synopsis of Results**
- TAO's jitter is lowest and most consistent
- CORBAplus' jitter is highest and most variable
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

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
High-performance, Real-time ORBs

TAO Request Demultiplexing Summary

<table>
<thead>
<tr>
<th>Demultiplexing Stage</th>
<th>Absolute Time (us)</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>2</td>
</tr>
<tr>
<td>4. Operation demarshaling</td>
<td>operation dependent</td>
</tr>
<tr>
<td>5. Parameter demarshaling</td>
<td>servant dependent</td>
</tr>
<tr>
<td>6. User upcall</td>
<td></td>
</tr>
<tr>
<td>7. Results marshaling</td>
<td>operation dependent</td>
</tr>
</tbody>
</table>

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 . . . 50
- Clients invoke two-way CORBA calls that cube a number on the servant and returns result

Real-time ORB/OS Performance Results

Real-time ORB/OS Jitter Results
Problem: Hard-coded ORB Messaging and Transport Protocols

Many ORBs do not support “pluggable 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>Protocol</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
High-performance, Real-time ORBs  Douglas C. Schmidt

Embedded System Benchmark Configuration

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

UC Irvine

High-performance, Real-time ORBs  Douglas C. Schmidt

Pinpointing ORB Overhead with VMEtro Timeprobes

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

UC Irvine

High-performance, Real-time ORBs  Douglas C. Schmidt

ORB & VME One-way Overhead Results

- ORB overhead is relatively constant and low – e.g., ~110 μsec per end-to-end operation
- Bottleneck is VME driver and OS, not ORB

UC Irvine
High-performance, Real-time ORBs

Workstation Benchmark Configuration

Debian Linux running on 400 Mhz workstation over Local IPC

Client Whitebox Latency Results

<table>
<thead>
<tr>
<th>Direction</th>
<th>Client Activities</th>
<th>Absolute Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing</td>
<td>1. Initialization</td>
<td>6.30</td>
</tr>
<tr>
<td></td>
<td>2. Get object reference</td>
<td>15.6</td>
</tr>
<tr>
<td></td>
<td>3. Parameter marshal</td>
<td>0.74 (param. dependent)</td>
</tr>
<tr>
<td></td>
<td>4. ORB messaging send</td>
<td>7.78</td>
</tr>
<tr>
<td></td>
<td>5. ORB transport send</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>6. I/O</td>
<td>8.70 (op. dependent)</td>
</tr>
<tr>
<td></td>
<td>7. ORB transport recv</td>
<td>50.7</td>
</tr>
<tr>
<td></td>
<td>8. ORB messaging recv</td>
<td>9.25</td>
</tr>
<tr>
<td></td>
<td>9. Parameter demarshal</td>
<td>op. dependent</td>
</tr>
</tbody>
</table>

Xeon platform is quad-CPU 400 Mhz with 1 Gigabytes RAM

Synergy of Results

- Local IPC more efficient than TCP/IP over loopback

Server Whitebox Latency Results on Xeon/NT

<table>
<thead>
<tr>
<th>Direction</th>
<th>Server Activities</th>
<th>Absolute Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incoming</td>
<td>1. I/O</td>
<td>7.0 (op. dependent)</td>
</tr>
<tr>
<td></td>
<td>2. ORB transport recv</td>
<td>24.8</td>
</tr>
<tr>
<td></td>
<td>3. ORB messaging recv</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>4. Parsing object key</td>
<td>4.6</td>
</tr>
<tr>
<td></td>
<td>5. POA demux</td>
<td>1.39</td>
</tr>
<tr>
<td></td>
<td>6. Servant demux</td>
<td>4.6</td>
</tr>
<tr>
<td></td>
<td>7. Operation demux</td>
<td>4.52</td>
</tr>
<tr>
<td></td>
<td>8. User upcall</td>
<td>3.84 (op. dependent)</td>
</tr>
<tr>
<td>Outgoing</td>
<td>9. ORB messaging send</td>
<td>4.56</td>
</tr>
<tr>
<td></td>
<td>10. ORB transport send</td>
<td>93.6</td>
</tr>
</tbody>
</table>
**One-Way Delayed Buffering Strategy**

- Copy params to new buffer
- Requests buffered in the Transport Adaptor
- Flush at byte count or timeout
- Send as one ORB message but multiple requests
- Server demultiplexes individual requests

**Shared Buffer Strategy**

- Request free buffer
- Add to Send queue
- Return to Free pool
- Request free buffer
- Add to Revc queue
- Return to Free pool

**ORB & Transport Overhead Results**

![Graph showing overhead results for different transport protocols.](image)

**Synopsis of Results**

- ORB overhead is relatively constant and low
  - e.g., ~49 μs per two-way operation
- Bottleneck is OS and I/O operation

**Data Copies in the Pluggable Protocols**

- Marshal parameters, data copy to CDR stream
- VME send, data copy from CDR stream to VME buffers
- DMA, data copy over VME Bus
- VME read, data copy to CDR stream
- Demarshal parameters, data copy to method parameters
**Problem: Overly Large Memory Footprint**

- **Problem**
  - ORB footprint is too big for some embedded apps

- **Unnecessary Features**
  - DSI, DII, & Dynamic Any
  - Interface Repository
  - Advanced POA features
  - CORBA/COM interworking

---

**Solution: Minimum CORBA**

<table>
<thead>
<tr>
<th>Component</th>
<th>CORBA</th>
<th>Minimum CORBA</th>
<th>Percentage Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>POA</td>
<td>282k</td>
<td>207k</td>
<td>26.5%</td>
</tr>
<tr>
<td>ORB Core</td>
<td>347k</td>
<td>330k</td>
<td>4.8%</td>
</tr>
<tr>
<td>Dynamic Any</td>
<td>131k</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>CDR Interpreter</td>
<td>69k</td>
<td>69k</td>
<td>0%</td>
</tr>
<tr>
<td>IDL Compiler</td>
<td>10k</td>
<td>11k</td>
<td>0%</td>
</tr>
<tr>
<td>Pluggable Protocols</td>
<td>15k</td>
<td>15k</td>
<td>0%</td>
</tr>
<tr>
<td>Default Resources</td>
<td>8k</td>
<td>8k</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>862k</td>
<td>640k</td>
<td>25.8%</td>
</tr>
</tbody>
</table>

Applying Minimum CORBA subsetting to TAO reduces memory footprint by ~25% (on SPARC with EGCS) and increases ORB determinism.

---

**Problem: Providing QoS to CORBA Operations**

- **Design Challenges**
  - Specifying/enforcing QoS requirements
  - Focus on Operations upon Objects
    - Rather than on communication channels or threads/synchronization
  - Support static and dynamic scheduling

- **Solution Approach**
  - Servants publish resource (e.g., CPU) requirements and (periodic) deadlines
  - Most clients are also servants

---

**Solution: TAO’s Real-time Static Scheduling Service**

struct **RT_Info** {
  Time worstcase_exec_time_;  
  Period period_;  
  Criticality criticality_;  
  Importance importance_;  
}

1: CONSTRUCT CALL CHAINS OF RT_OPERATIONS
2: IDENTIFY THREADS
3: POPULATE RT_INFO REPOSITORY
4: ASSESS SCHEDULABILITY
5: ASSIGN OS THREAD PRIORITIES AND DISPATCH QUEUE ORDERING SUBPRIORITIES

www.cs.wustl.edu/~schmidt/TAO.ps.gz
TAO's RT Event Service Architecture

Features →
- Integrated with RT Scheduler
- Stream-based architecture
  - Enhance pluggability
- Source and type-based filtering
- Event correlations
  - Conjunctions (A+B+C)
  - Disjunctions (A\lor B\lor C)

www.cs.wustl.edu/~schmidt/JSAC-98.ps.gz

Features/Domain Characteristics

- I/O driven
  - Periodic processing requirements
- Complex dependencies
  - e.g., I/O Facades depend on multiple sensor proxies
- Real-time constraints
  - Deterministic and statistical deadlines
  - Static scheduling (e.g., rate monotonic)
- Single-Processor (VxWorks)
  - Single address space
  - No distribution requirements (yet)
Candidate Solution: COS Event Service

- Features
  - Decoupled consumers and suppliers
  - Transparent group communication
  - Asynchronous communication
  - Abstraction for distribution
  - Abstraction for concurrency

Applying the COS Event Service to Real-time Avionics

- TAO is currently used at Boeing for avionics mission computing
  - Initial flight dates are mid-summer 1998
- Extensive benchmarks demonstrate it is possible to meet stringent performance goals with real-time CORBA
  - e.g., for Boeing, target latency for CORBA oneway operations is $150 \mu\text{secs}$ for 100 MHz PowerPC running over MVME 177 boards
- Technology transfer to commercial vendors via OMG RT SIG and DARPA Quorom program & OCI

Overview of Avionics Mission Computing

- Typical Interactions
  - I/O arrives
  - Proxies demarshal data
  - Proxies push to channel
  - EC pushes to facades
  - Facades process data
- Advantages:
  - Anonymous consumers/suppliers
  - Group communication
  - Asynchronous pushes

Issues Not Addressed by COS Event Service

- No support for complex event dependencies
  - Consumer-specified event filtering
  - Event correlations (e.g., waiting for events A and B before pushing)
- No support for real-time scheduling policies
  - Priority-based dispatching (e.g., which consumer is dispatched first)
  - Priority-based preemption policies and mechanisms
  - Interval timeouts for periodic processing
  - Deadline timeouts for "failed" event dependencies
**TAO’s Event Service Architecture**

- **Features**
  - Stream-based architecture
    - Enhance pluggability
  - Subscription/filtering
    - Source and type-based filtering
  - Event correlations
    - Conjunctions (A+B+C)
    - Disjunctions (A|B|C)

**Collaborations in the RT Event Channel**

- Well-defined event structure
  - CORBA Anys are inefficient
- Augmented COS interfaces:
  - Extra QoS structure to connect suppliers and consumers

**Real-Time Event Dispatching with TAO’s Event Service**

- **Features**
  - Run-time scheduler
    - Determines event priority
  - 2-level priority queues
    - Preemption groups
    - Priority queues
  - Dispatcher
    - Encapsulates concurrency policy

**Real-time Event Channel Dispatching Experiments**

(A) FIFO Dispatching  
(B) RTU Dispatching  
(C) Threaded Dispatching
**Multi-Threaded Dispatching**

**Single-Threaded Dispatching**

---

**Dimensions of ORB Extensibility**

1. Extensible to retargeting on new platforms
2. Extensible via custom implementation strategies
3. Extensible via dynamic configuration of custom strategies

---

**Applying Patterns to Develop Extensible ORBs**

- Factories produce Strategies
- Strategies implement interchangeable policies
- Concurrency strategies use Reactor and Active Object
- Acceptor-Connector decouple transport from GIOP operations
- Service Configurator permits dynamic configuration

www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz
### Addressing ORB Portability and Typesafety Challenges

#### Problem
- Building an ORB using low-level system APIs is hard

#### Forces
- Low-level APIs are tedious to program
- Low-level APIs are error-prone
- Low-level APIs are non-portable

#### Solution
- Apply the Wrapper Facade pattern to encapsulate low-level OS programming details

### Using the Wrapper Facade Pattern in TAO

- TAO's wrapper facades are based on the ACE framework
- The Wrapper Facade pattern substantially increased portability and reduced the amount of ad hoc code

### Addressing ORB Demuxing/Dispatching Challenges

#### Problem
- ORBs must process many different types of events simultaneously

#### Forces
- Multi-threading may not be available
- Multi-threading may be inefficient
- Multi-threading may be inconvenient
- Tightly coupling general event processing with ORB-specific logic is inflexible

#### Solution
- Use the Reactor pattern to decouple generic event processing from ORB-specific processing
Enhancing Demuxing with the Reactor Pattern

- **Intent**
  - Decouples synchronous event demuxing/dispatching from event handling

- **Forces Resolved**
  - Demuxing events efficiently within one thread
  - Extending applications without changing demux infrastructure

Intent – Decoupling synchronous event demuxing/dispatching from event handling

Forces Resolved – Demuxing events efficiently within one thread – Extending applications without changing demux infrastructure

www.cs.wustl.edu/~schmidt/Reactor.ps.gz

Using the Reactor Pattern in TAO

- The Reactor pattern and ACE Reactor are widely used

SunSoft IIOP TAO

High-performance, Real-time ORBs

Enhancing Endpoint Initialization with the Acceptor-Connector Pattern

- **Intent**
  - Decouple connection establishment and service handler initialization from subsequent service processing

Intent – Decouple connection establishment and service handler initialization from subsequent service processing

Addressing ORB Endpoint Initialization Challenges

- **Problem**
  - The communication protocol used between ORBs is often orthogonal to its connection establishment and service handler initialization protocols

Problem – The communication protocol used between ORBs is often orthogonal to its connection establishment and service handler initialization protocols

- **Forces**
  - Low-level connection APIs are error-prone and non-portable
  - Separating initialization from processing increases software reuse

Forces – Low-level connection APIs are error-prone and non-portable – Separating initialization from processing increases software reuse

- **Solution**
  - Use the Acceptor-Connector pattern to decouple passive/active connection establishment and GIOP connection handler initialization from the subsequent ORB interoperability protocol (e.g., IIOP)

Solution – Use the Acceptor-Connector pattern to decouple passive/active connection establishment and GIOP connection handler initialization from the subsequent ORB interoperability protocol (e.g., IIOP)
Using the Acceptor-Connector Pattern in TAO

**Problem**
- Multi-threaded ORBs are needed since Reactive ORBs are often inefficient, non-scalable, and non-robust

**Forces**
- Multi-threading can be very hard to program
- No single multi-threading model is always optimal

**Solution**
- Use the *Active Object* pattern to allow multiple concurrent server operations using an OO programming style

Enhancing ORB Concurrency with the Active Object Pattern

**Intent**
- Decouple thread of request execution from thread of request reception

**Forces Resolved**
- Allow blocking operations
- Permit flexible concurrency strategies

TAO supports several variants of Active Objects (e.g., Thread-per-Connection, Thread-per-Request, Thread Pool, etc.)
Reducing Lock Contention and Priority Inversions with the Thread-Specific Storage Pattern

- Problem
  - It is important to minimize the amount of locking required to serialize access to resources shared by an ORB

- Forces
  - Locks increase performance overhead
  - Locks increase potential for priority inversion
  - Different concurrency schemes yield different locking costs

- Solution
  - Use the Thread-Specific Storage pattern to maximize threading-model flexibility and minimize lock contention and priority inversion

Forces Resolved
- Minimizes overhead and priority inversion

Minimizing ORB Locking with the Thread-Specific Storage Pattern

Intent
- Allows multiple threads to use one logically global access point to retrieve ORB thread-specific data without incurring locking overhead for each access

Forces Resolved
- Minimizes overhead and priority inversion

Addressing ORB Flexibility Challenges

- Problem
  - Real-world ORBs must be flexible to satisfy the requirements of many different types of end-users and applications

- Forces
  - Ad hoc schemes for ORB flexibility are too static and non-extensible
  - Flexibility often has many (related) dimensions

- Solution
  - Use the Strategy pattern to support multiple transparently “pluggable” ORB strategies

Using Thread-Specific Storage in TAO

www.cs.wustl.edu/~schmidt/TSS-pattern.ps.gz
Enhancing ORB Flexibility with the Strategy Pattern

- **Intent**
  - Factor out similarity among algorithmic alternatives
- **Forces Resolved**
  - Orthogonally replace behavioral subsets transparently
  - Associating state with an algorithm

Using the Strategy Pattern in TAO

- **Intent**
  - Integrate all strategies used to configure an ORB
- **Forces Resolved**
  - Consolidates customization of many strategies
  - Ensures semantically-compatible strategies

Addressing ORB Configurability Challenges

- **Problem**
  - Aggressive use of Strategy pattern creates a configuration nightmare
- **Forces**
  - Managing many individually configured strategies is hard
  - It’s hard to ensure that groups of semantically compatible strategies are configured
- **Solution**
  - Use the Abstract Factory pattern to consolidate multiple ORB strategies into semantically compatible configurations
Using the Abstract Factory Pattern in TAO

Problem
- Prematurely committing ourselves to a particular ORB configuration is inflexible and inefficient

Forces
- Certain ORB configuration decisions can’t be made efficiently until run-time
- Forcing users to pay for components they don’t use is undesirable

Solution
- Use the Service Configurator pattern to assemble the desired ORB components dynamically

Enhancing Dynamic ORB Extensibility with the Service Configurator Pattern

Intent
- Decouples ORB strategies from time when they are configured

Forces Resolved
- Reduce resource utilization
- Support dynamic (re)configuration

www.cs.wustl.edu/~schmidt/Svc-Conf.ps.gz

Using the Service Configurator Pattern in TAO

TAO PROCESS
- Priority-based Dispatching
- Perfect Hashing
- Medical Imaging Concrete Factory

DLLs
- Active Demuxing
- FIFO Dispatching

svc.conf
- dynamic ORB Service_Object *
  avionics_orb:make_orb() "ORBport 2001"
Quantifying the Benefits of Patterns

- **Statistics**
  - Patterns greatly reduce code complexity
    - e.g., Most TAO components have \( \nu(G) < 10 \)
  - TAO components are substantially smaller than SunSoft IIOP
    - e.g., connection management reduced by a factor of 5

Macabe Complexity Metric Scores for TAO and SunSoft IIOP

Lessons Learned Developing QoS-enabled 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
High-performance, Real-time ORBs

Synopsis of TAO’s Pattern-Oriented ORB Design

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.

Summary: Real-time Optimizations in TAO

Next Steps: New TAO Features and Optimizations

Forthcoming Features

- CORBA Component Model (CCM)
- Real-time and Minimum CORBA
- CORBA Messaging
- Fault-Tolerant CORBA
- Notification Service
  www.cs.wustl.edu/~schmidt/TAO-status.html
Next Steps: Integrating QoS-Enabled CORBA Component Model with TAO

**Features**
- Select optimal communication reflectively
- Re-factor component QoS aspects into their containers
- Dynamically load/unload component implementations

～schmidt/RIO.ps.gz

---

Next Steps: Integrating TAO with ATM I/O Subsystem

**Features**
- Vertical integration of QoS through ORB, OS, and ATM network
- Real-time I/O enhancements to Solaris kernel
- Provides rate-based QoS end-to-end
- Leverages APIC features for cell pacing and zero-copy buffering

～schmidt/RIO.ps.gz

---

Next Steps: Strategized Scheduling Framework

1. Specify RT_operation
2. Populate RT_info repository
3. Configure queues based on dispatching queue configuration
4. Dispatching queues assign dynamic portions of dispatching hierarchy
5. Supply dispatching queue configuration
6. Supply static portions of dispatching hierarchy
7. Supply static portions of dispatching hierarchy

www.cs.wustl.edu/~schmidt/dynamic.ps.gz

---

Next Steps: Open ATM Signaling & Control

---
Key Themes

- Decouple functional path from QoS path
- Emphasize integration and configuration

Next Steps: Adaptive Middleware (e.g., QuO/TAO)

Efficiency
- Sockets for data transfer to get high performance

Flexibility
- Uses CORBA for control messages and properties

Web URLs for Additional Information

- These slides: ~schmidt/TAO4.ps.gz
- More information on CORBA: ~schmidt/corba.html
- More info on ACE: ~schmidt/ACE.html
- More info on TAO: ~schmidt/TAO.html
- TAO Event Channel: ~schmidt/JSAC-98.ps.gz
- TAO static scheduling: ~schmidt/TAO.ps.gz
- TAO dynamic scheduling: ~schmidt/dynamic.ps.gz
- ORB Endsystem Architecture: ~schmidt/RIO.ps.gz
- Pluggable protocols: ~schmidt/pluggable_protocols.ps.gz
Web URLs for Additional Information (cont’d)

- Network monitoring, visualization, & control: ~schmidt/NMVC.html

- Performance Measurements:
  - Demuxing latency: ~schmidt/COOTS-99.ps.gz
  - SII throughput: ~schmidt/SIGCOMM-96.ps.gz
  - DII throughput: ~schmidt/GLOBECOM-96.ps.gz
  - ORB latency & scalability: ~schmidt/ieee_tc-97.ps.gz
  - IOOP optimizations: ~schmidt/JSAC-99.ps.gz
  - Concurrency and connection models: ~schmidt/RT-perf.ps.gz
  - RTOS/ORB benchmarks:
    ~schmidt/RT-OS.ps.gz
    ~schmidt/words-99.ps.gz