Real-time Object Request Brokers

Principles and Patterns of High-performance
Motivation

- Many applications require QoS guarantees
  - e.g., telecom, avionics, WWW
- Existing middleware doesn’t support QoS effectively
  - e.g., CORBA, DCOM, DCE
- Solutions must be *integrated*
  - *Vertically* and *horizontally*
Candidate Solution: CORBA

- Goals of CORBA
  - Simplify distribution
  - Provide foundation for higher-level services

- Limitations of CORBA
  - Poor performance
  - Lack of QoS features

http://www.cs.wustl.edu/~schmidt/corba.html
The ACE ORB (TAO)

- **TAO Overview**
  - A high-performance, real-time ORB
    * Networking and avionics focus
  - Leverages the ACE framework
    * Ported to VxWorks, POSIX, and Win32

- **Related work**
  - QuO at BBN
The ADAPTIVE Communication Environment (ACE)

- **ACE Overview**
  - A concurrent OO networking framework
  - Very widely used in industry
  - Available in C++ and Java
  - Ported to VxWorks, POSIX, and Win32

- **Related work**
  - x-Kernel

http://www.cs.wustl.edu/~schmidt/ACE.html
Applying ORBs to Real-time Avionics

- **Domain Challenges**
  - Periodic hard real-time deadlines
  - COTS infrastructure
  - Open systems

- **Related work**
  - Deng, Liu, and J. Sun ’96
  - Gopalakrishnan and Parulkar ’96
  - Wolfe et al. ’96
Applying ORBs to Real-time Network Management

- **Domain Challenges**
  - Periodic statistical real-time deadlines
  - COTS infrastructure
  - Open systems

- **Related work**
  - Deng, Liu, and J. Sun ’96
  - Gopalakrishnan and Parulkar ’96
  - Wolfe et al. ’96
Research Objectives

- Identify features and architectural patterns needed for real-time ORBs
- Both hard real-time and statistical real-time ORBs
- Determine changes needed to CORBA specification
- E.g., Gigabit bandwidth and ∼10 microsecond latency
- e.g., APIs for defining end-to-end QoS requirements

High-performance, Real-time ORBs

Douglas C. Schmidt
Real-time Features and Optimizations in TAO

Diagram showing the architecture of TAO with various components and their interactions, including:

- CLIENT
- ORB QoS INTERFACE
- RIDL STUBS
- OBJECT IMPLEMENTATION
- REAL-TIME OBJECT ADAPTER
- RIDL SKELETON

Key interactions include:

- In args
- Out args + return value

Optimizations include:

- Presentation Layer Optimizations
- Data Copying Optimizations
- Request Demultiplexing and Dispatching Optimizations
- Protocol Engine Optimizations
- I/O Subsystem Optimizations
- Network Adapter Optimizations

Additional components:

- ORB CORE
- RIOP
- OS KERNEL
- OS I/O Subsystem
- Network Adapters
- Network
Experimental Setup for CORBA/ATM Testbed

Client

Ultra 2

ATM Switch

Ultra 2

Server

Object Adapter

ORB Core

Services

Requests

Douglas C. Schmidt

High-performance, Real-time ORBs

Washington University, St. Louis
Problem: Meeting End-to-End QoS Requirements

- Design Challenges
  - Specifying QoS requirements
  - Reducing demultiplexing latency
  - Meeting scheduling deadlines
  - Reducing presentation layer overhead

1) CLIENT MARSHALING
2) CLIENT PROTO QUEUEING
3) NETWORK DELAY
4) SERVER PROTO QUEUEING
5) THREAD DISPATCHING
6) REQUEST DISPATCHING
7) SERVER DEMARSHALING
8) METHOD EXECUTION

Washington University, St. Louis
Problem: Reducing Demultiplexing Latency

- **Design Challenges**
  - Minimize demuxing layers
  - Provide $O(1)$ operation demuxing
  - Avoid priority inversions
  - Remain CORBA-compliant
Solution: De-layered Active Demultiplexing

**Solution Approach**
- Pre-negotiate demuxing keys
- Tunnel demuxing key with Object key
- Use ACT pattern for validation

**Related Work**
- Yau and Lam ’97
- Dittia and Parulkar ’97
- Engler and Kaashoek ’96
Demultiplexing Performance Experiments

- Linear search based on Orbix demuxing strategy
- Perfect hashing based on GNU gperf
  - http://www.cs.wustl.edu/~schmidt/gperf.ps.gz
- Results at http://www.cs.wustl.edu/~schmidt/GLOBECOM-97.ps.gz
Demultiplexing Performance Results

Synopsis

Demultiplexing scheme

Linear (100 methods)
Linear (10 methods)
GPERF (100 methods)
GPERF (10 methods)
GPERF (1 method)
Active Demux (100 methods)
Active Demux (10 methods)
Active Demux (1 method)

Number of Objects

Latency in microseconds

- Active demuxing isn't compatible, but it is dynamic.
- GPERF solution is 100% compatible, but it isn't.

High-performance, real-time ORBS
Problem: Meeting CORBA Request Deadlines

- Design Challenges
  - Specifying/enforcing QoS requirements
  - Focus on *Objects* and *Operations*

- Assumptions
  - Static scheduling
  - Non-distributed (initially)

---

Washington University, St. Louis
Solution 1: Real-time Object Adapter

- **Solution Approach**
  - Integrate RT dispatcher into ORB
  - Support multiple request scheduling strategies
    * e.g., RMS, RMS with Deferred Preemption, and EDF

- **Related work**
  - Zinky, Bakken, and Schantz, ’95
  - Lee, Rajkumar, and Mercer ’96
Solution 2: Real-time Scheduling Service

1: Object Adapter receives RIOP request from ORB Core.

2: Run-time scheduler determines priority of target RT_task.

3: Request queued according to RT_task’s priority.

4: Request dequeued by thread with suitable OS priority.

5: Request dispatched.
Scheduling Service Roles

- **Components**
  - **Offline**
    - Assess schedule feasibility
    - Assign thread and queue priorities
  - **Online**
    - Supply priorities to Object Adapter’s dispatcher

http://www.cs.wustl.edu/~schmidt/TA0.ps.gz
Scheduling Service Interfaces

- **Components**
  - Application interface
    * Use RTInfos
  - Privileged interface
    * Used by system tasks and services

```c
struct RT_Info {
    wc_exec_time_;  
    cached_exec_time_;  
    period_;  
    importance_;  
    dependencies_;  
};

interface Scheduler {
    register_task();
    schedule();
    priority();
};
```
Scheduling Steps During Configuration Run

1: APPLICATIONS CONSTRUCT RT_INFOS

2: COMPILE AND LINK PROGRAM
   (USE -DSCHEDCONFIG=1)

3: RUN PROGRAM
   A: APPLICATIONS REGISTER WITH EVENT CHANNEL (WHICH REGISTERS OPERATIONS WITH SCHEDULER) OR
   B: APPLICATIONS REGISTER DIRECTLY WITH SCHEDULER
   C: PROGRAM INFORMS EVENT CHANNEL THAT REGISTRATIONS ARE COMPLETE

4: COMPLETE PROGRAM
   A: EVENT CHANNEL CALLS SCHEDULER'S SCHEDULE() METHOD
   B: PROGRAM EXITS

---

struct RT_Info
{
  wc_exec_time_;  
  cached_exec_time_;  
  period_;  
  importance_;  
  dependencies_;  
};

RTRT__INFOINFO
REPOSITORYREPOSITORY
OFF-OFF-LINELINE
SCHEDULLERSCHEDULER
REQUESTREQUEST
QUEUESQUEUES
RT_TaskRT_Task
RT_TaskRT_Task
RT_TaskRT_Task
Scheduling Service Internal Repository

- **Components**
  - RT_Info references
  - Vector of RT_Tasks called by each RT_Task
    * Vector records dependencies
  - Called-task chains are traversed to compute total CPU time and minimum period
Real-time Dispatching Experiments

(A) FIFO Dispatching

1: Incoming Request
2: Enqueue Request
3: Dequeue Request
4: Dispatch Request

Dispatcher

(B) RTU Dispatching

1: Incoming Request
2: Enqueue Request
3: Dequeue Request
4: Dispatch Request

Dispatcher
Run-Time Scheduler

(C) Threaded Dispatching

1: Incoming Request
2: Enqueue Request
3: Dequeue Request
4: Dispatch Request

Dispatcher
Run-Time Scheduler

OBJECT ADAPTER

- Available at http://www.cs.wustl.edu/~schmidt/oopsla.html
Key Patterns in TAO

- **Definition**
  - “A recurring solution to a design problem in a particular context”

- **Benefits of Patterns**
  - Facilitate design reuse
  - Preserve crucial design information
  - Guide design choices
  - Document common traps and pitfalls

---

**STRATEGIC PATTERNS**

- Active Object
- Acceptor
- Connector
- Reactor
- Half-Sync/Half-Async
- Asynchronous Completion Token (ACT)

---

**TACTICAL PATTERNS**

- Proxy
- Strategy
- Abstract Factory
- Adapter

---

Washington University, St. Louis
Real-time Event Channel Overview

- Real-time Event Channel Features
  - Scheduling
  - Correlation dependencies
  - Filtering
Collaboration in the RT Event Channel

- CONNECT_PUSH
- CONSUMER
- Object Ref
- Scheduling QoS
- Correlation Specs
- Subscription Info
- Timeout Registration

- EVENT CHANNEL
  - Consumer Proxies
  - Dispatching Module
  - Event Correlation
  - Subscription & Filtering
  - Priority Timers
  - Supplier Proxies

- CONNECT_PUSH
- SUPPLIER
- Publish Types
- Object Ref

Washington University, St. Louis
RT Event Channel Use-cases

**CONSUMERS**

- Air Frame
- HUD
- Nav

**3: PUSH (DEMARSHALED DATA)**

**EVENT CHANNEL**

**SUPPLIERS**

- Sensor proxy
- Sensor proxy
- Sensor proxy

**CONSUMERS**

- Con Proxy
- Con Proxy
- Con Proxy

**3: PUSH (DEMARSHALED DATA)**

**EVENT CHANNEL**

**SUPPLIERS**

- Attr proxy
- Attr proxy
- Attr proxy

**Avionics**

**Network management**
High-performance, Real-time ORBS

Timeline for FIFO Object Adapter

Task Time Line Viewer

Douglas C. Schmidt
**Domain Challenges**

- Large volume of “Blob” data
  - e.g., 10 to 40 Mbps
- Lossy compression isn’t viable
- Prioritization of requests
Problem: Reducing Protocol Engine Overhead

- Design Challenges
  - Small memory footprint
  - Predictable performance
  - Minimize the typeid interpreter overhead
Solution: TypeCode Interpreter Optimizations

- Solution Approach
  - Optimized Typecode Interpreter
  - Based on SunSoft IIOP engine

- Related work
  - Hoschka ’97
  - O’Malley, Proebsting, and Montz ’94
### TypeCode Layout for Sequence of BinStructs

<table>
<thead>
<tr>
<th>TypeCode Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TKSEQUENCE</td>
<td>Sequence of BinStructs</td>
</tr>
<tr>
<td>ULONG LENGTH</td>
<td><strong>128</strong></td>
</tr>
<tr>
<td>OCTET* BUFFER</td>
<td></td>
</tr>
<tr>
<td>BYTE ORDER</td>
<td><strong>0</strong></td>
</tr>
<tr>
<td>ELEMENT TypeCode Kind</td>
<td>TK_STRUCT</td>
</tr>
<tr>
<td>ENCAPSULATION Length</td>
<td><strong>112</strong></td>
</tr>
<tr>
<td>Bounds of the Sequence</td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

- **Byte Order of Encapsulation:** 0
- **Length of String ID:** 1
- **Actual String ID:** 0
- **Length of String Struct Name:** 1
- **Actual Name of Struct:** 0
- **Number of Members in Struct:** 0
- **Length of String Name for Struct Member of Type Short:** 6
- **Actual Name of Member of Type Short:** 1
- **TypeCode Kind for Member of Type Short:** TK_SHORT

#### TypeCode Description in CDR format

```c
// 32 bytes
struct BinStruct{
    short s; char c; long l;
    octet o; double d;
    octet pad[8];
};
typedef sequence<BinStruct> StructSeq;
```
Throughput of the SunSoft IIOP Implementation

- Experimental design
  - Transfer 64 Mbytes of “oneway” data
  - Various types of data
Douglas C. Schmidt

High-performance, Real-time ORBS

Challenges of Optimizing Complex Software

Optimizing complex software is hard

Small “mistakes” are costly over high-speed networks

- Optimize using principles
- Validate using white-box and black-box metrics
- Pinpoint sources of overhead using white-box metrics

Solution Approach (Iterative)

- Apply optimization principles
- e.g., quantity, TRF, etc.

Problem

- Validate via white-box and black-box metrics
<table>
<thead>
<tr>
<th>Number</th>
<th>Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optimize for the common case</td>
</tr>
<tr>
<td>2</td>
<td>Eliminate gratuitous waste</td>
</tr>
<tr>
<td>3</td>
<td>Replace inefficient general-purpose methods with efficient special-purpose ones</td>
</tr>
<tr>
<td>4</td>
<td>Precompute values, when possible</td>
</tr>
<tr>
<td>5</td>
<td>Store redundant state to speed up expensive operations</td>
</tr>
<tr>
<td>6</td>
<td>Pass information between layers</td>
</tr>
</tbody>
</table>
Sender-side Analysis of SunSoft IIOP Implementation

Percent Execution Time for doubles and structs

Washington University, St. Louis
Receiver-side Analysis of SunSoft IIOP Implementation

Percent Execution Time for doubles and structs
Problems

- Invocation overhead for small, frequently called methods

Solution

- Inline method calls

Principle

- Optimize for the common case

Problems and Solutions
Throughput After 1st Optimization

Throughput for doubles and structs

Douglas C. Schmidt

High-performance, Real-time ORBs

Washington University, St. Louis
Receiver-side Analysis of IIOP Implementation (1st Opt)

Throughput for doubles and structs
Douglas C. Schmidt

High-performance, Real-time ORBS

Problems

Problems

- Lack of C++ compiler support for aggressive inlining

Solution

- Replace inline methods with preprocessor macros

Principle

- Optimize for the common case
Throughput After 2nd Optimization

Throughput for doubles and structs

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

Throughput After 2nd Optimization

Throughput for doubles and structs

Washington University, St. Louis
Receiver-side Analysis of IIOP Implementation (2nd Opt)

Percent Execution Time for doubles and structs
Problems

Toomanymethodcallsofthe"samequantity"
Computingthesamequantityrepeatedly
Too many method calls

Principles

– Convert generic methods to special-purpose ones
– Pass information through layers
– Add extra state
– Precalculate

Problems and Solutions

High-performance Real-time ORBS

Douglas C. Schmidt
Throughput After 3rd Optimization

Throughput for doubles and structs

Douglas C. Schmidt

High-performance, Real-time ORBs

double

struct

Washington University, St. Louis
Receiver-side Analysis of IIOP Implementation (3rd Opt)

Percent Execution Time for doubles and structs
Douglas C. Schmidt
High-performance Real-time ORBS

Problems

- Problems
- Expensive no-ops for memory deallocation
- Eliminate gratuitous waste
- Specialize generic methods
- Principles

Washington University, St. Louis
Throughput After Optimizations

Throughput for doubles and structs

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

Washington University, St. Louis
Receiver-side Analysis of IIOP Implementation after Optimizations

Percent Execution Time for doubles and structs

- Double:
  - Read: 54.93%
  - Typecode::traverse: 44.21%

- Struct:
  - TypeCode::traverse: 91.94%
  - TypeCode::typecode_param: 3.78%
Throughput Comparisons

Throughput for SunSoft and TAO Versions of IIOP

Original SunSoft

Optimized TAO
Results available at http://www.cs.wustl.edu/~schmidt/IIOP.ps.gz

- 4.2 times for structs
- 5 times for chars/shorts
- 3.75 times for shorts
- 3.3 times for longs
- 1.8 times for doubles

Our measurement-driven, principle-based optimization process improved TAO's IIOP protocol engine performance as follows.

Results for Typecode Interpreter Optimizations

High-performance, real-time ORBS

Douglas C. Schmidt
– Real-time Event Channels and Multimedia Streaming Service

– Real-time Object Services

– Demultiplex, schedule, and dispatch client requests in real-time

– Real-time Object Adapter

– Multi-threaded ORB run-time system based on ACE

– ACE ORB Core

– Enhancements

– Optimized version of Sun’s GIOP/IIOP protocol engine with real-time

– GIOP Protocol Engine

– Based on Sun’s "IDL" front-end and our back-end

– IDL Compiler

Current Status of TAO

High-performance, Real-time ORBs

Douglas C. Schmidt
Developing an ORB Core with ACE

- Components
  - Acceptor/Connector
    * Parameterized via *strategies*
  - Reactor
    * Demuxes client requests
  - Active Objects
    * Processes client requests
Douglas C. Schmidt

High-performance, Real-time ORBs

Concluding Remarks

Current Focus: High-performance, Real-time ORBs

Future Work

- Applying optimization principles to presentation layer
  * i.e., support static request scheduling
  - Enforcing periodic deadlines via Real-time Object Adapter
  - Applying optimization principles to TypeCode Interpreter
  - Reducing latency via de-layered active demuxing

- TypeCode compiler optimizations
- Distributed QoS and integration with RT I/O Subsystem
- Dynamic scheduling of requests
- Pointing non-determinism and priority inversions in ORBs

Pinpoint non-determinism and priority inversions in ORBs

Washington University, St. Louis