Design Patterns for Developing and Using Real-time CORBA Object Request Brokers

Douglas C. Schmidt
schmidt@cs.wustl.edu

Washington University, St. Louis
www.cs.wustl.edu/~schmidt/TAO4.ps.gz

Sponsors
Bellcore, Boeing, CDI, DARPA, Kodak, Lucent, Motorola, NSF, OTI, SAIC, Siemens SCR, Siemens MED, Siemens ZT, and Sprint

Motivation: the Distributed Software Crisis

- Symptoms
  - Hardware gets smaller, faster, cheaper
  - Software gets larger, slower, more expensive

- Culprits
  - Accidental and inherent complexity

- Solutions
  - Middleware, frameworks, components, and patterns
Sources of Complexity for Distributed Applications

- **Inherent complexity**
  - Latency
  - Reliability
  - Partitioning
  - Ordering

- **Accidental Complexity**
  - Low-level APIs
  - Poor debugging tools
  - Algorithmic decomposition
  - Continuous re-invention

Techniques for Improving Software Quality and Productivity

- **Proven solutions**
  - Components
    * Self-contained, “pluggable” ADTs
  - Frameworks
    * Reusable, “semi-complete” applications
  - Patterns
    * Problem/solution pairs in a context
  - Architecture
    * Families of related patterns and components
Motivation for Real-time Middleware

- 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 by automating
    * Object location and activation
    * Parameter marshaling
    * Demultiplexing
    * Error handling
  - Provide foundation for higher-level services

www.cs.wustl.edu/~schmidt/corba.html
Limitations of CORBA for Real-time Systems

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

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

The ACE ORB (TAO)

- TAO Overview
  - A high-performance, real-time ORB
    - Telecom and avionics focus
  - Leverages the ACE framework
    - Runs on RTOSs, POSIX, and Win32

- Related work
  - QuO at BBN

www.cs.wustl.edu/~schmidt/TAO.html
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User's guide</td>
<td><a href="http://www.cs.wustl.edu/~schmidt/ACE-">www.cs.wustl.edu/~schmidt/ACE-</a></td>
</tr>
<tr>
<td>Supported commercially</td>
<td>•</td>
</tr>
<tr>
<td>Related work</td>
<td>•</td>
</tr>
<tr>
<td>ACE overview</td>
<td>•</td>
</tr>
<tr>
<td>ACE statistics</td>
<td>•</td>
</tr>
<tr>
<td>SYV Streams</td>
<td>•</td>
</tr>
<tr>
<td>x-kernel</td>
<td>•</td>
</tr>
<tr>
<td>Win32</td>
<td>•</td>
</tr>
<tr>
<td>POSIX</td>
<td>•</td>
</tr>
<tr>
<td>Runs on RTOS</td>
<td>•</td>
</tr>
<tr>
<td>Java</td>
<td>•</td>
</tr>
<tr>
<td>Ported to C++</td>
<td>•</td>
</tr>
<tr>
<td>Framework networking</td>
<td>•</td>
</tr>
<tr>
<td>Concurrent OO</td>
<td>•</td>
</tr>
</tbody>
</table>

The ADAPTIVE Communication Environment (ACE)

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

• Domain Challenges
  – Periodic
deterministic
real-time deadlines
  – COTS infrastructure
  – Open systems

Visualizing Periods for Avionics Operations
Real-time Features and Optimizations in TAO

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

- **Initial focus**
  - Deterministic deadlines
  - Static scheduling

- **Solution approach**
  - Servants publish resource (e.g., CPU) requirements and (periodic) deadlines
  - Most clients are also servants
TAO’s Real-time Scheduling Service

**Components**

- **Offline**
  * Assess schedule feasibility
  * Assign thread and queue priorities

- **Online**
  * Supply priorities to ORB endsystem dispatcher via $O(1)$ table lookup

**Solution Approach**

- Integrate RT dispatcher into ORB endsystem
- Support multiple request scheduling strategies
  * e.g., RMS, RMS with Deferred Preemption, and EDF
- Requests ordered across thread priorities by OS dispatcher
- Requests ordered within priorities based on *data dependencies* and *importance*

www.cs.wustl.edu/~schmidt/TAO.ps.gz
Priority Inversion Experiments

- One high priority client
- 1..n low priority clients
- Server factory implements thread-per-rate
  - Highest real-time priority for high priority client
  - Lowest real-time priority for low priority clients

Priority Inversion Experiment Results

- Synopsis of results
  - Chorus’ latency is lower for small # of clients
    * ~1.2msec vs. ~1.4sec vs. ~2.0sec
  - TAO’s latency is much lower for large # of clients
    * ~2.3msec vs. ~7.6msec vs. ~14.1msec
  - TAO avoids priority inversion
    * i.e., high priority client always has lowest latency
Douglas C. Schmidt

Jitter Experiment Results

- **Definition**
  - Jitter is the variance from the average latency

- **Synopsis of results**
  - TAO's jitter is lowest and most consistent
  - MT-Orbix's jitter is highest and more variable

Problem: Improper ORB Concurrency Model

- **Common Problems**
  - High overhead
    - Context switching
    - Synchronization
  - Priority inversions
    - FIFO request queueing
    - Improper thread priorities
  - Lack of application control over concurrency model
Problem: ORB Shared Connection Model

- Common Problems
  - Priority inversions
    * Sharing multiple priorities on a single connection
  - Complex connection multiplexing
  - Synchronization overhead

TAO’s Inter-ORB Connection Topology

[Diagram of TAO’s Inter-ORB Connection Topology]
**Problem: Reducing Demultiplexing Latency**

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

**Demultiplexing Performance Experiments**

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

- **Synopsis**
  - Linear search is far too costly
  - Dynamic hashing is too erratic
  - gperf solution is 100% compatible, but static
  - Active demuxing may not be 100% compatible, but is dynamic

---

Integrating TAO with RT I/O Subsystem and ATM

- **Key Features**
  - Vertical integration of QoS through ORB, OS, and ATM network
  - Provides rate-based QoS end-to-end
  - Leverages APIC features for cell pacing
Dimensions of ORB Extensibility

- Extensible to retarget on new platforms
- Extensible via custom implementation strategies
- Extensible via dynamic configuration of custom strategies

Applying Patterns to Develop Extensible ORBs

- 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
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

---

Enhancing Portability and Typesafety with the Wrapper Facade Pattern

- **Intent**
  - Encapsulates low-level, stand-alone system mechanisms within type-safe, modular, and portable class interfaces

- **Forces Resolved**
  - Avoid tedious, error-prone, and non-portable system APIs
  - Create cohesive abstractions
Using the Wrapper Facade Pattern in TAO

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

Addressing ORB Demuxing and Dispatching Challenges

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

- **Forces**
  - Multi-threading is not always available
  - Multi-threading is not always efficient
  - 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

---

Using the Reactor Pattern in TAO

- The ACE Reactor pattern is widely used by industry
Addressing ORB Endpoint Initialization Challenges

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

- **Forces**
  - Low-level connection APIs are error-prone and non-portable.
  - Separating initialization from subsequent processing increases software reuse for many types of communication software.

- **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).

---

Enhancing Endpoint Initialization with the Acceptor-Connector Pattern

- **Intent**
  - Decouple connection establishment and service handler initialization from subsequent service processing.
Addressing ORB Concurrency Challenges

- **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**
  - Decouples the thread of request execution from the thread of request reception

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

Using the Active Object Pattern in TAO

- TAO supports several variants of Active Objects (e.g., Thread-per-Connection, Thread-per-Request, Thread Pool, Thread-per-Rate, 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

---

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
Douglas C. Schmidt  

High-performance, Real-time ORBs

Using Thread-Specific Storage in TAO

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


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
Douglas C. Schmidt

High-performance Real-time ORBs

Centralizing ORB Configurability with the Abstract Factory Pattern

Problem

Aggressive use of the Strategy pattern can create a configuration nightmare. It’s hard to manage large numbers of individually configured strategies. It’s hard to ensure that groups of semantically compatible strategies are configured.

Forces

- Ensures semantic compatibility of strategies.
- Consolidates customizations of many strategies.
- Forces Resolved

Centralizing ORB Configurability

Solution

- Use the Abstract Factory pattern to consolidate multiple ORB strategies into semantically compatible configurations.

Intent

Integrate all strategies used to configure an ORB

Implementation

- Concrete Factory_1
  - make_product_A()
  - make_product_B()
- Concrete Factory_2
  - make_product_A()
  - make_product_B()

Abstract Factory

- make_product_A()
- make_product_B()

Abstract Product

- Product_A
  - Product_A1
  - Product_A2
- Product_B
  - Product_B1
  - Product_B2

Douglas C. Schmidt

High-performance Real-time ORBs
Using the Abstract Factory Pattern in TAO

Addressing ORB Dynamic Configurability Challenges

- **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

CORBA::ORB_init (int &argc, char *argv[]) {
    // Configure the ORB.
    Service_Config tao (argc, argv);
    // Perform initialization...
}

Using the Service Configurator Pattern in TAO

Run-time Configuration
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

Current Status of TAO
The next-generation of ORBs will provide much better QoS support.

ORBs are an effective way to achieve reuse of distributed software components. ORBs are an effective way to achieve communication frameworks and components design patterns to create communication frameworks and components that developers can use to solve problems.

Successful developers resolve these challenges by applying appropriate performance principles to Type Code interpreter. Eliminating the round trip delays for CORBA requests can improve performance.

Future Work

- Pinpointing non-deterministic and priority inversions in ORBs.
- Future Work
  - Applying optimization principles to presentation layer
  - IDL compiler and optimized stub generation
  - Dynamic scheduling of requests
  - Pinpointing non-deterministic and priority inversions in ORBs
  - Distributed QoS and integration with RT/IOL Subsystem
  - Type code compiler optimizations
  - Support static scheduling for CORBA requests
  - i.e., support static scheduling for CORBA requests

Concluding Remarks

Developers of distributed applications confront recurring challenges that require frequent attention. Successful developers resolve these challenges by applying appropriate patterns to design communication frameworks and components. ORBs are an effective way to achieve reuse of distributed software components. The next-generation of ORBs will provide much better QoS support.