Design and Implementation of the Portable Object Adapter for a Real-time ORB

Irfan Pyarali
irfan@cs.wustl.edu

Washington University, St. Louis
www.cs.wustl.edu/~irfan

Advisor: Dr. Douglas C. Schmidt

January 18, 2000

Outline of Presentation

• Research Contributions
• CORBA Architecture
• Object Adapter
• Portable Object Adapter
  – Design Goals
  – Architecture
  – Real-time features
• Concluding Remarks

Washington University, St. Louis

CORBA Architecture

• Goals of CORBA
  – Simplify distribution by automating
    * Object location and activation
    * Parameter marshaling
    * Demultiplexing
    * Error handling
  – Provide foundation for higher-level services

Research Contributions

• Design and implement the POA specification
• Adapt the POA for TAO, our real-time ORB
  – Minimize locking
  – Optimize demultiplexing and dispatching
  – Provide predictable behavior

Washington University, St. Louis
Object Adapter

- Functionality
  - Request demultiplexing
  - Operation dispatching
  - Object reference generation
  - Servant activation and deactivation

Interoperable Object Reference (IOR)

- Protocol Id
- Time Stamp
- Object Id
- Communication Endpoint
- Object Adapter Id

URL format

Overview of the Portable Object Adapter (POA)

- Basic Object Adapter (BOA) begone!
  - Abandon by OMG due to lack of specificity
  - No longer publishes BOA specs
- POA is a portable Object Adapter
  - i.e., servants are portable between different ORB implementations
- Standardized skeleton classes
- Standardized interactions between servants and object adapter
  - POA interface is defined in IDL

POA Design Goals

- Portability
- Persistent Identity
  - e.g., database objects
- Automation
  - Transparent activation of servants and objects
- Conserving Resources
  - Single servant supports multiple objects
- Nested POAs
- Flexibility
  - Lifetime of objects and servants
  - User defined Object Ids
- POA behavior governed by policies
- SSI and DSI support
POA Architecture

POA Manager Processing States

- States
  - Active
  - Holding
  - Discarding
  - Inactive
  - Shuting down

POA Policies

1. Threading
2. Servant Retention
3. Request Processing
4. Implicit Activation
5. Object Id Uniqueness
6. Lifespan
7. Object Id Assignment
**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

---

**POA Features for TAO**

1. De-layered demultiplexing
2. POA Active Object tables
3. Different ORB Core and POA configurations
4. POA synchronization
5. Upcall optimization
6. Collocation classes
7. Predictability

---

**De-layered Demultiplexing**

- **Consequences**
  - O(1) lookups
  - User specified Ids not allowed

---

**POA Active Object Table Strategies**

- **Features**
  - Strategy pattern
  - Extensible design
**POA-per-ORB configuration**

- **Benefits**
  - No synchronization in POA
    - Only accessed by one thread

- **Drawbacks**
  - Complicated servant registration
    - Servants must register with multiple POAs

**Global POA Configuration**

- **Drawbacks**
  - Synchronization required in POA
    - Accessed by multiple threads

- **Benefits**
  - Simple servant registration
    - Servants register with one POA

**POA Synchronization**

- Certain ORB configurations don't require synchronization in POA
  - If only one thread uses the POA
  - If the POA state does not change during run-time
  - Servants and servant managers are registered at startup
- Control synchronization by using a TAO specific POA creation policy:

```idl
enum SynchronizationPolicyValue
{
    NULL_LOCK, THREAD_LOCK, DEFAULT_LOCK
};

interface SynchronizationPolicy : CORBA::Policy
{
    readonly attribute SynchronizationPolicyValue value;
};

SynchronizationPolicy
create_synchronization_policy (in SynchronizationPolicyValue value);
```

**Class Hierarchy of POA Locks**

- **Features**
  - External Polymorphism pattern
  - Extensible design