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

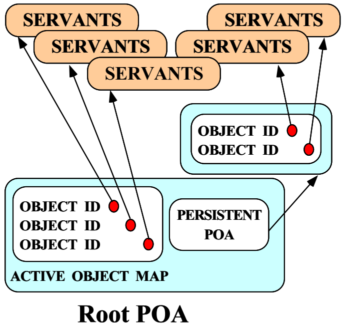
Irfan Pyarali
irfan@cs.wustl.edu

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

Advisor: Dr. Douglas C. Schmidt

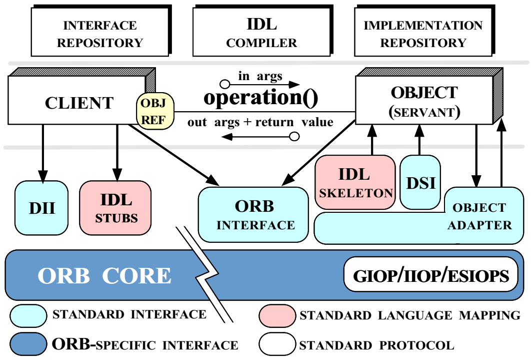
January 18, 2000

Outline of Presentation


SERVANTS
SERVANTS
SERVANTS
SERVANTS
SERVANTS
OBJECT ID
OBJECT ID
OBJECT ID
OBJECT ID
OBJECT ID
PERSISTENT POA
ACTIVE OBJECT MAP
Root POA

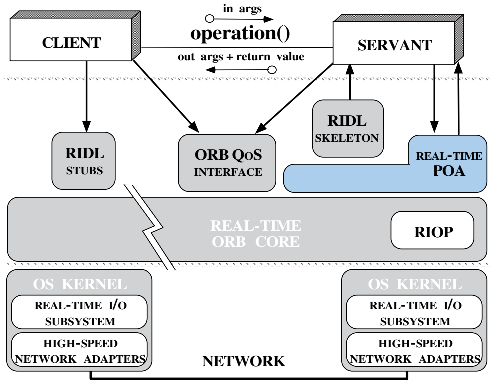
- Research Contributions
- CORBA Architecture
- Object Adapter
- Portable Object Adapter
  - Design Goals
  - Architecture
  - Real-time features
- Concluding Remarks

CORBA Architecture


INTERFACE REPOSITORY
IDL COMPILER
IMPLEMENTATION REPOSITORY
CLIENT
OBJ REF
in args
operation()
out args + return value
OBJECT (SERVANT)
DII
IDL STUBS
ORB INTERFACE
IDL SKELETON
DSI
OBJECT ADAPTER
ORB CORE
GIOP/IIOP/ESIOPS
STANDARD INTERFACE
STANDARD LANGUAGE MAPPING
ORB-SPECIFIC INTERFACE
STANDARD PROTOCOL

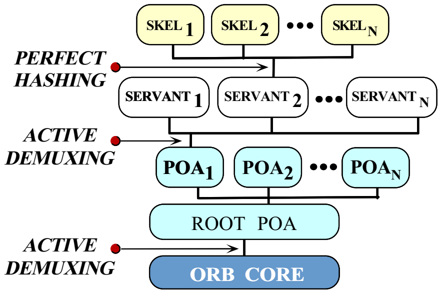
- **Goals of CORBA**
  - Simplify distribution by automating
    - ∗ Object location and activation
    - ∗ Parameter marshaling
    - ∗ Demultiplexing
    - ∗ Error handling
  - Provide foundation for higher-level services

Research Contributions


CLIENT
in args
operation()
out args + return value
SERVANT
RIDL STUBS
ORB QoS INTERFACE
RIDL SKELETON
REAL-TIME POA
REAL-TIME ORB CORE
RIOP
OS KERNEL
REAL-TIME I/O SUBSYSTEM
HIGH-SPEED NETWORK ADAPTERS
NETWORK
OS KERNEL
REAL-TIME I/O SUBSYSTEM
HIGH-SPEED NETWORK ADAPTERS

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

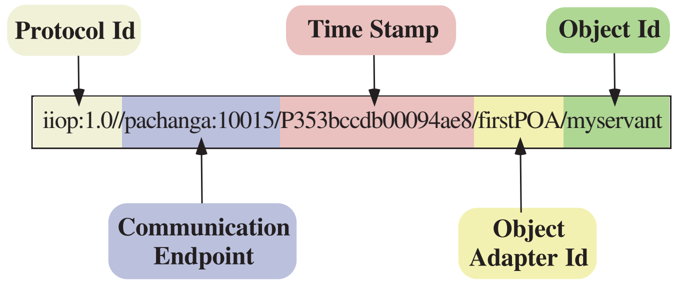
Object Adapter



SKEL 1
SKEL 2
SKEL N
PERFECT HASHING
SERVANT 1
SERVANT 2
SERVANT N
ACTIVE DEMUXING
POA$_1$
POA$_2$
POA$_N$
ROOT POA
ACTIVE DEMUXING
ORB CORE

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

Interoperable Object Reference (IOR)



Protocol Id
Time Stamp
Object Id
iiop:1.0//pachanga:10015/P353bccdb00094ae8/firstPOA/myservant
Communication Endpoint
Object Adapter Id

URL format

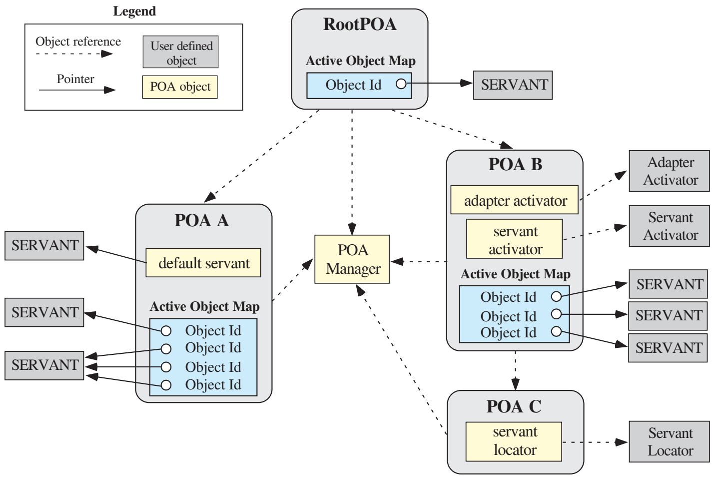
Overview of the Portable Object Adapter (POA)

- 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

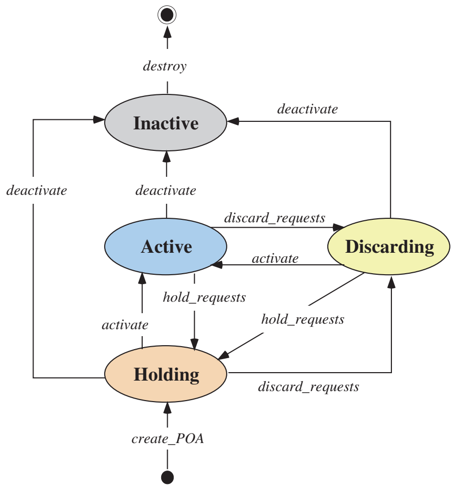
POA Design Goals

- 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

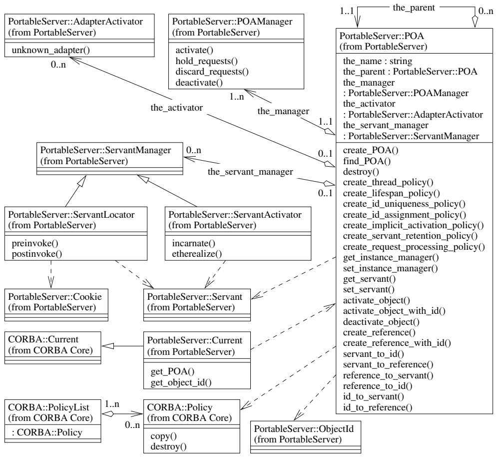


POA Manager Processing States



- **States**
  - Active
    - ∗ Process requests
  - Holding
    - ∗ Queue requests
  - Discarding
    - ∗ Discard requests
  - Inactive
    - ∗ Shuting down

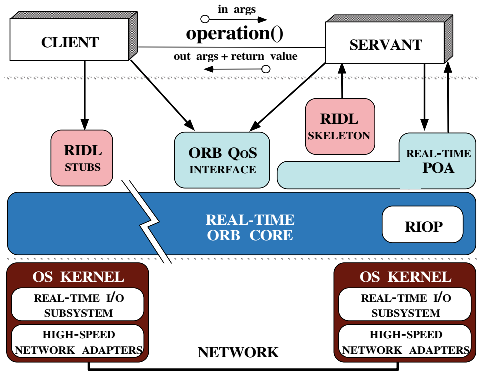
POA Architecture in UML



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)

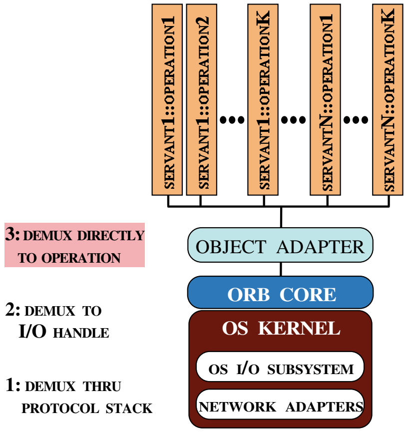


- **TAO Overview**
  - A high-performance, real-time ORB
    - ∗ Telecom and avionics focus
  - Leverages the ACE framework
    - ∗ Runs on RTOSs, POSIX, and Win32

POA Features for TAO

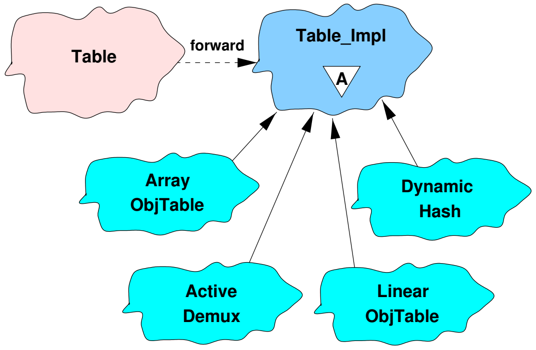
1. De-layered demultiplexing
2. POA Active Object tables
3. Different ORB Core and POA configurations
4. POA synchronization
5. Upcall optimization
6. Collocation classes
7. Predictability

De-layered Demultiplexing



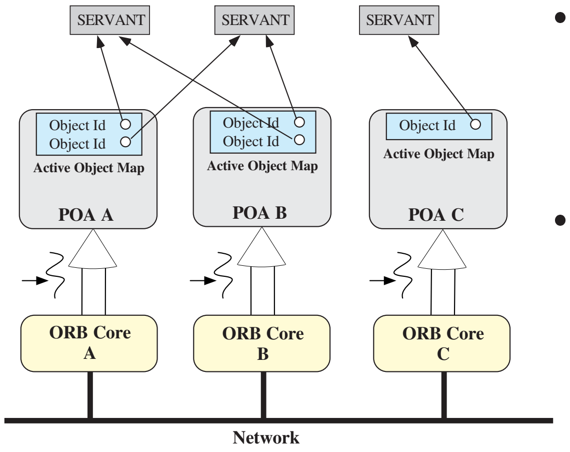
- **Consequences**
  - O(1) lookups
  - User specified Ids not allowed

POA Active Object Table Strategies



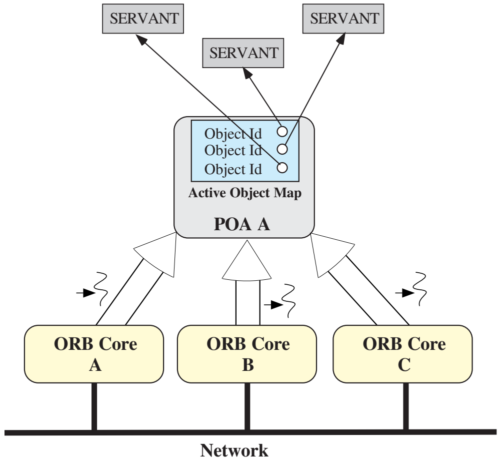
- **Features**
  - Strategy pattern
  - Extensible design

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



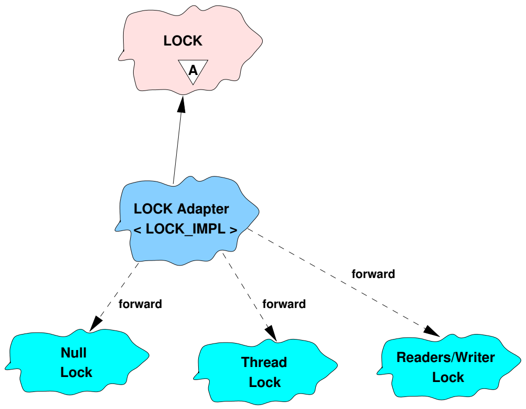
- **Drawbacks**
  - Synchronization required in POA
    - Accessed by multiple threads
- **Benefits**
  - Simple servant registration
    - Servants register with one POA

POA Synchronization

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

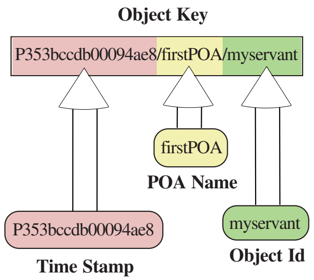
```
// IDL
enum SynchronizationPolicyValue
{
  NULL_LOCK, THREAD_LOCK, DEFAULT_LOCK
};

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

SynchronizationPolicy
  create_synchronization_policy (in SynchronizationPolicyValue value);
```

Class Hierarchy of POA Locks

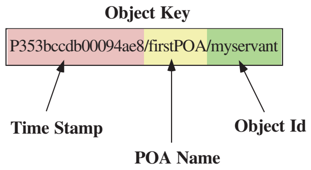


- **Features**
  - External Polymorphism pattern
  - Extensible design

Upcall Optimization

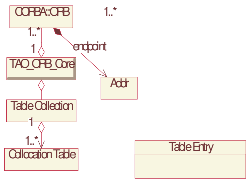


Naive Parsing



Optimized Parsing

Collocation Class



- **Consequences**
  - Transparent to clients
  - Minimal overhead
    - Inlined forwarding method
  - Features bypassed
    1. Interceptors
    2. POA Managers
    3. Servant Managers
    4. Active Object Map
    5. POA::Current
    6. Forwarding

Configuring the POA for Predictability

- De-layered demultiplexing
  - Bounded dispatch latency
- POA per ORB Core
  - No contention for POA state
- No user-defined callback objects
  - No Servant Managers
  - No Adapter Activators
- No queuing in POA
  - No POA Manager

Concluding Remarks

- An Object Adapter is an integral part of CORBA
- POA has many features
  - Simple things are easy
- Design choices made to adapt the POA for TAO
  - High performance
  - Predictability
- First publicly available implementation
  - www.cs.wustl.edu/∼schmidt/TAO.html