

Introduction to Distributed Computing

Session 2: Advance CORBA Components

OOMWorks

Irfan Pyarali and Pradeep Gore

irfan@oomworks.com and pradeep@oomworks.com

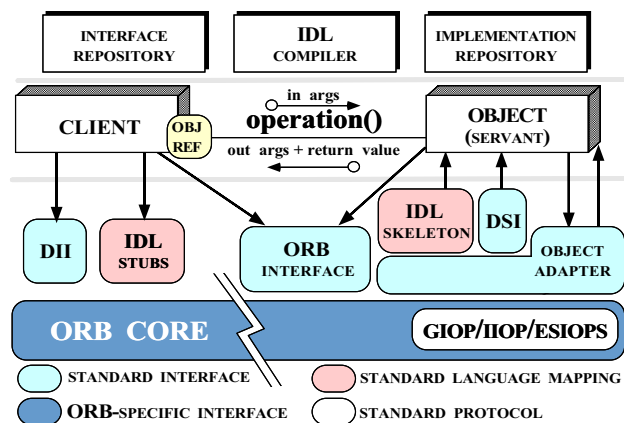
May 3, 2001

– Typeset by FoilTEX –

Session 2: Advanced CORBA Components

- *Recap of last class - including quiz*
 - *Make sure we are intimately familiar with CORBA vocabulary*
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

CORBA ORB Architecture



Goals of CORBA

- Simplify distribution by automating
 - Object location & activation
 - Parameter marshaling
 - Demultiplexing
 - Error handling
- Provide foundation for higher-level services

www.cs.wustl.edu/~schmidt/corba.html

CORBA Vocabulary

- **ORB Object Request Broker**
 - Component that allows clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and hardware.
- **Client**
 - Obtains references to objects and invokes operations on them regardless of the object location relative to the client.
- **Object**
 - In CORBA, an Object is an instance of the OMG Interface Definition Language (IDL) interface. Each Object is identified by an object reference which associates one or more paths through which a client can access an object on a server.

Some more terms ...

- **Servant**
 - This component implements operations defined by an OMG IDL interface. Servants are implemented as class instances. A client does not interact with servants directly but always through objects identified by object references. (Bridge pattern)
- **IDL Stubs and Skeletons**
 - Serve as the “glue” between clients and servants, respectively, and the ORB
- **POA - Portable Object Adapter**
 - Demultiplexes and dispatches client requests to servants. (Only needed by servers)
- **CORBA Server**
 - Hosts Servants and makes them accessible via CORBA Object interfaces.

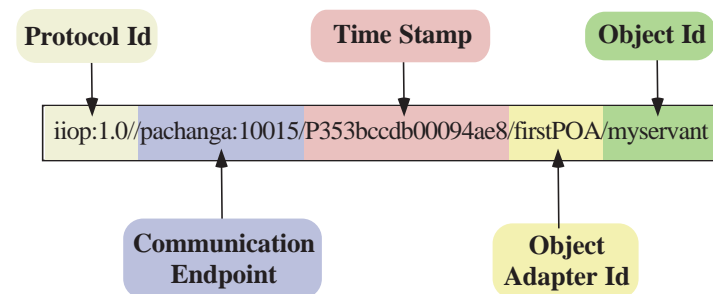
Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- **Object References**
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

Object References

- An “object reference” is an opaque handle to an object
 - It identifies the object's location
- Object references may be passed among processes on separate hosts
 - The underlying CORBA ORB will correctly convert object references into a form that can be transmitted over the network
 - The ORB provides the receiver with a pointer to a proxy in its own address space
 - * This proxy refers to remote object implementation
- Object references are a powerful feature of CORBA
 - e.g., supports *peer-to-peer* interactions and *distributed callbacks*

Object Reference in URL Format



- Transient object reference

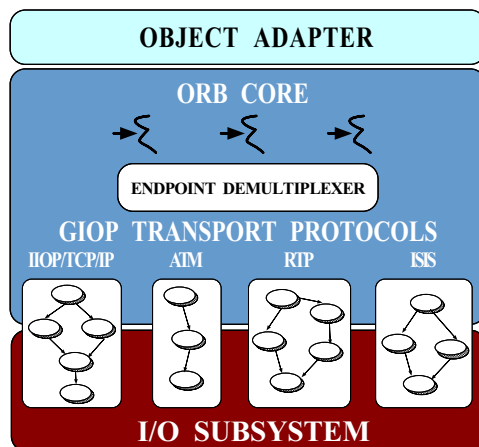
Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- *Complete Quoter C++ example from last class*
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- *Advanced CORBA ORB Components*
 - *ORB Core; POA; Pluggable Protocols*
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

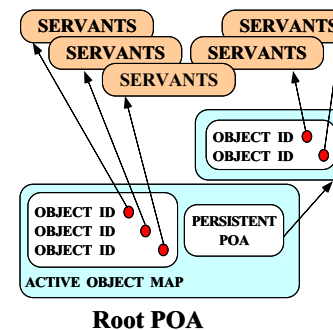
Overview of the ORB Core



Features

- Connection/memory management
- Request transfer
- Endpoint demuxing
- Concurrency control

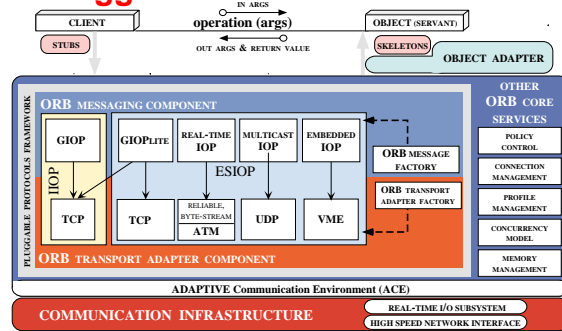
Portable Object Adapter (POA)



• Functionality

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

Pluggable Protocols Framework



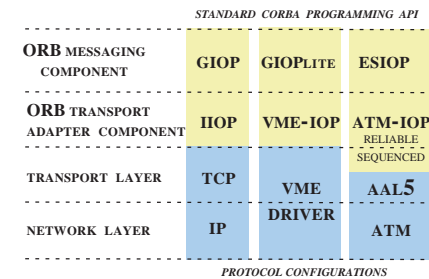
Features

- Pluggable *ORB* messaging and transport protocols
- Highly efficient and predictable behavior

Principle Patterns

- Replace general-purpose functions (protocols) with special-purpose ones

CORBA Interoperability Protocols

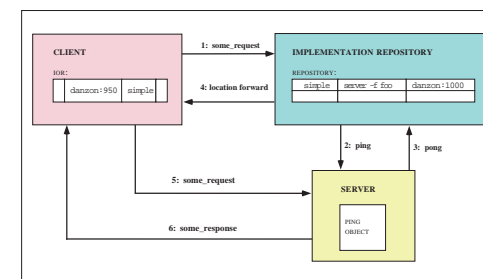


- **GIOP**
 - Enables ORB-to-ORB interoperability
- **IIOIP**
 - IIOIP adds to GIOP semantics for TCP/IP connection management.
- **ESIOPs**
 - *e.g.*, DCE, DCOM, wireless, etc.

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- **CORBA Services**
 - *Naming; Trading; Event and Notification; Time*
 - *Asynchronous Method Invocation (AMI)*
 - *CORBA Component Model (CCM)*
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

Implementation Repository



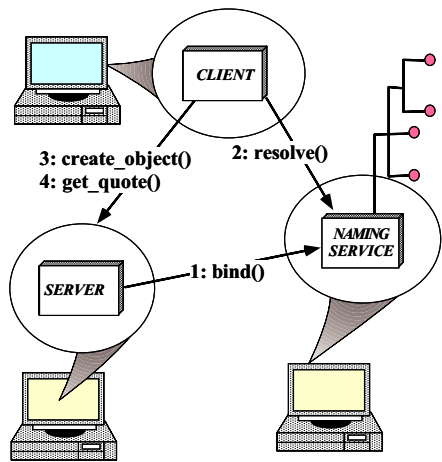
Repository for persistent objects

- Persistent objects can outlive originating server processes

Transparent/on-demand activation and migration

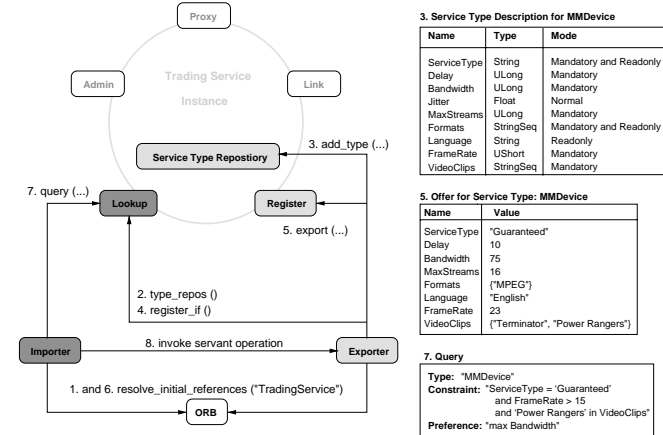
- Restarts and forwards to servers
- Maintains restart info on servers
- Keeps track of currently running servers

CORBA Naming Service



- **Purpose**
 - Maps strings to object references
- **Implementation**
 - Naming Context can be a hierarchically nested graph
 - Typically written using Hash Maps

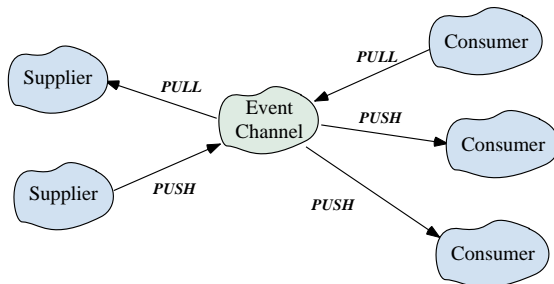
CORBA Trading Service



- Services advertised based on properties
- Clients perform constraint lookups (TCL)

- Dynamic properties
- Trader federation

CORBA Event and Notification Service



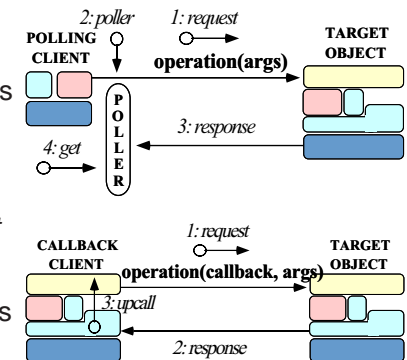
- Anonymous consumers/suppliers
 - Publish and subscribe model
- Group communication
 - Supplier(s) to consumer(s)
- Decoupled communication
 - Asynchronous delivery
- Abstraction for concurrency
 - Can facilitate concurrent event handling

Asynchronous Method Invocation (AMI)

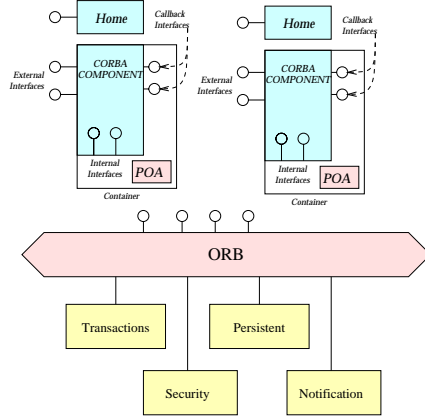
- Specifies two *asynchronous method invocation* (AMI) models

1. Poller model
2. Callback model

- Standardizes *time-independent invocation* (TII) model
 - Used for store/forward routers



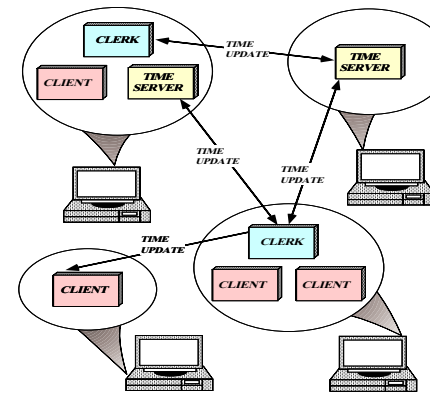
CORBA Component Model



• Features

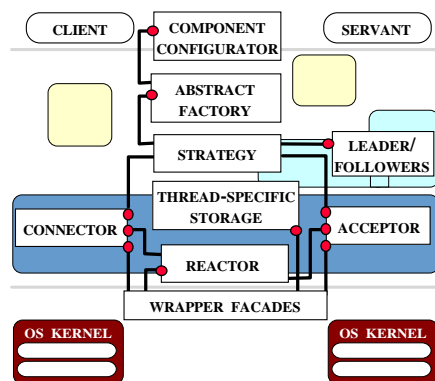
- Navigation among interfaces supported by components
- Standardized system-component interaction
- Standardized component life-cycle management
- Component interconnections
- Standardized component configuration
- Standardized ORB services interfaces

CORBA Time Service



- Service gets current time +/- inaccuracy estimate
- Globally synchronized time
- Time Service Server
 - answers queries for time made by clerks
 - can be activated on demand
- Time Service Clerk
 - updates time from the various servers
 - answers queries for time from the clients

Design Interlude: Patterns and Frameworks



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

Definitions

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

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- *Quoter example using the Naming Service*
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- [Quoter example in Java](#)
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- [Coping with Changing requirements](#)
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- Quiz and Assignments

Coping with Changing Requirements

- [New Quoter features](#)
 - Format changes to extend functionality
 - New interfaces and operations
- [Improving existing Quoter features](#)
 - Batch requests
- [Leveraging new ORB features](#)
 - Asynchronous Method Invocations (AMI)

New Formats

For example, percentage that stock increased or decreased since start of trading day, volume of trades, etc.

```
module Stock
{
    // ...

    interface Quoter
    {
        long get_quote (in string stock_name,
                       out double percent_change,
                       out long trading_volume)
        raises (Invalid_Stock);
    };
};
```

Note that even making this simple change would involve a great deal of work for a sockets-based solution...

Adding Features Unobtrusively

Interface inheritance allows new features to be added without breaking existing interfaces

```
module Stock
{
  // ...
  interface Quoter { /* ... */ };

  interface Stat_Quoter : Quoter // a Stat_Quoter IS-A Quoter
  {
    long get_stats (in string stock_name,
                   out double percent_change,
                   out long trading_volume)
      raises (Invalid_Stock);
  };
};
```

Note that there are no changes to the existing Quoter interface

New Interfaces and Operations

For example, adding a trading interface

```
module Stock {
  // Interface Quoter_Factory and Quoter same as before.
  interface Trader {
    void buy (in string name,
             inout long num_shares,
             in long max_value) raises (Invalid_Stock);

    void sell (in string name,
              inout long num_shares,
              in long min_value) raises (Invalid_Stock);
  };
  interface Trader_Factory { /* ... */ };
};
```

Multiple inheritance is also useful to define a full service broker:

```
interface Full_Service_Broker : Stat_Quoter, Trader {};
```

Batch Requests

Improve performance for multiple queries or trades

```
interface Batch_Quoter : Stat_Quoter
{ // Batch_Quoter IS-A Stat_Quoter
  typedef sequence<string> Names;
  struct Stock_Info {
    string name;
    long value;
    double change;
    long volume;
  };
  typedef sequence<Stock_Info> Info;
  exception No_Such_Stock { Names stock; };

  void batch_quote (in Names stock_names,
                   out Info stock_info) raises (No_Such_Stock);
};
```

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- *Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)*
- Quiz and Assignments

Static Invocation Interface (SII)

```
// Get object reference.
Quoter_var quoter = // ...

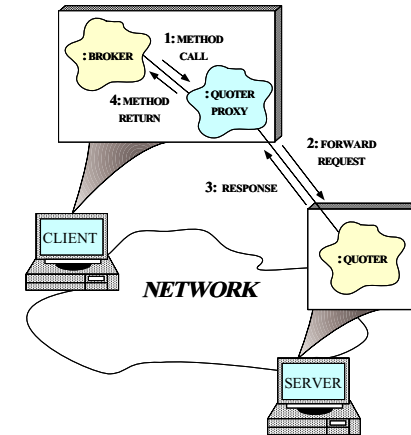
const char *name =
    "ACME ORB Inc.";

CORBA::Long value =
    quoter->get_quote (name);
cout << name << " = "
    << value << endl;
```

- The common way to use OMG IDL is the “Static Invocation Interface” (SII)
- All operations are specified in advance and are known to client via *stubs*
 - Stubs marshal operation calls into request messages

Primary advantages of SII are *simplicity*, *typesafety*, and *efficiency*

Stubs use the Proxy Pattern



Intent: provide a surrogate for another object that controls access to it

Dynamic Invocation Interface (DII)

- A less common programming API is the “Dynamic Invocation Interface” (DII)
 - Enables clients to invoke operations on objects that aren’t known until run-time
 - * *e.g.*, MIB browsers
 - Allows clients to “push” arguments onto a request stack and identify operations via an ASCII name
 - * Type-checking via meta-info in “Interface Repository”
- The DII is more flexible than the SII
 - *e.g.*, it supports *deferred synchronous* invocation
- However, the DII is also more complicated, less typesafe, and inefficient

An Example DII Client

```
// Get Quoter reference.
Stock::Quoter_var quoter_ref = // ...
CORBA::Long value;

// Create request object.
CORBA::Request_var request =
    quoter_ref->_request ("get_quote");

// Add parameter.
request->add_in_arg () <=< "IONAY";
request->set_return_type (CORBA::_tc_long);

// Call method.
request->invoke ();

// Retrieve/print value.
if (request->return_value () >=> value)
    cout << "Current value of IONA stock: "
        << value << endl;
```

This is *much* more complicated and inefficient than using SII...

Static and Dynamic Skeleton Interface

- The Static Skeleton Interface (SSI) is generated automatically by the IDL compiler
 - The SII performs the operation demuxing/dispatching and parameter demarshaling
- The Dynamic Skeleton Interface (DSI) provides analogous functionality for the server-side that the DII provides on the client-side
 - It is defined primarily to build ORB “Bridges”
 - The DSI lets server code handle arbitrary invocations on CORBA objects

Session 2: Advanced CORBA Components

- Recap of last class - including quiz
 - Make sure we are intimately familiar with CORBA vocabulary
- Object References
- Complete Quoter C++ example from last class
- Advanced CORBA ORB Components
 - ORB Core; POA; Pluggable Protocols
- CORBA Services
 - Naming; Trading; Event and Notification; Time
 - Asynchronous Method Invocation (AMI)
 - CORBA Component Model (CCM)
- Quoter example using the Naming Service
- Quoter example in Java
- Coping with Changing requirements
- Static Invocation Interface (SII) and Dynamic Invocation Interface (DII)
- [Quiz and Assignments](#)