CORBA & Design Pattern

Csc8350 Advanced Software Engineering
Georgia State University
Dr. Xiaolin Hu
Before we start

Hands-on project topic
Agenda

• An OO example
• The broker pattern
• CORBA
• IDL (Interface Definition Language)
• A CORBA example
• Middleware
• Note: this class is more about “how to design CORBA” instead of “how to use CORBA”
A OO Program

ClassA

aMethod()

ClassB

sayHello()

ClassB b = new ClassB();
b.sayHello();

return "Hello world";
Question

Assume ClassA and ClassB run on two different computers.

How can we make ClassA and ClassB work as before (i.e., in OO fashion)? In other words, how can we structure a distributed system so that application developers don't have to concern themselves with the details of remote communication?

How to make this work? What are needed?
How to Design this System?

• Let’s start with a class diagram with two classes: ClassA and ClassB
• You have 5 minutes ……

• Response:
  – I don’t understand the problem
  – I am totally lost
  – I am on something…

As will be seen, at least three design patterns will be used to make this work.
Let’s Design Together

- Both ClassA and ClassB belong to their own processes on their own computers.
- To make ClassA call ClassB’s method just like in OO, ClassA needs to get an object reference of ClassB.
- Let’s first assume there is a **magic** operation, which can return the object reference of ClassB when called.
- With this reference, e.g., `b`, what happens when ClassA executes `b.sayHello()`?
- Keep in mind that the only way of interaction between the two computers is through message passing
- A message will be generated on computerA and sent to computerB. – How?
Let’s Design Together

• A message will be generated on computerA and sent to computerB.
• ClassB will be triggered to execute sayHello().
• A returning message will be generated on ComputerB and sent to ComputerA.
• ClassA gets the result.
• What will the message look like (what is the necessary information in that message)?
• Which object handles the message passing?
• What design pattern can we use to support this design?
Review the Proxy Design Pattern

- Interface inheritance is used to specify the interface shared by **Proxy** and **RealSubject**.
- Delegation is used to catch and forward any accesses to the **RealSubject** (if desired)
Proxy Design Pattern

• Use the proxy design pattern to design the following interaction.
• Note that client and server are on two different computers.
Let’s Finish the Design

• Again, start with a class diagram with two classes: ClassA and ClassB
• You have 5 minutes……

• Response:
  – I am totally lost.
  – I know I need to apply the proxy pattern, but still don’t know how to do that.
  – I am on something…
Figure 2: Structure with distribution (from Microsoft web site)

What is the purpose of each method?

Why do we need the ServerProxy?

How to implement performFunctionA() in ClientProxy, ServerProxy, and Server?
Behavior with distribution (from Microsoft web site)
Proxy

• The proxies would do all the marshaling and unmarshaling of data, security control, transfer channel configuration, and any other additional work.

• The client would simply invoke the `performFunctionA` method on the client proxy as if it were a local call because the client proxy actually implements the `ServerInterface`. 
Revisit a claim that we made before

- Let’s first assume there is a **magic** operation, which can return the object reference when called.
- With this reference, e.g., $b$, what happens when ClassA executes $b.sayHello()$?

- Which object does $b$ refer to?
  - The proxy (which proxy?).

- What parameters are needed for the magic operation to create this reference? In other words, how to set up the entire system?
A Problem

• The proxies do all the marshaling and unmarshaling of data, security control, transfer channel configuration, and any other additional work.

• The basic parameters needed for the magic operation to create this reference.
  • Object name
  • Host’s address

• Problem of this design.
  • Fixed and static binding between the client and server.

• A better design?
Broker

• However, because the client proxy communicates with the server proxy directly, the client must be able to find the location of the server at compile time. This means that you cannot change or move the server to a different location at run time.

• To overcome this limitation, you need to avoid exposing the exact location of the server. Instead, you deploy a new component, the broker component, at a well-known location and then expose that location to the client.

• The broker component is then responsible for locating the server for the client. This type of function is often called a naming service.

• The server is responsible to register itself to the broker.
Let’s Finish the Design

• Start from the previous class diagram with proxies.
• You have 5 minutes to add new things to make the broker pattern work …..

• Response:
  – I don’t understand the problem.
  – I am totally lost.
  – I am on something…
What is each method used for?

Broker (from Microsoft web site)
In some situations, however, direct communication between client and server is not desirable. For example, for security reasons, you may want to host all the servers in your company's private network, which is behind a firewall, and only allow access to them from the broker.

(from Microsoft web site)
After processA (the process of ClassA) sends the message, will it wait (doing nothing) before receiving the return message? What happens if it takes extremely long time for ClassB to process the request?

Synchronous messaging, asynchronous messaging

If the request is synchronous, the calling system must wait for the provider to finish processing the request before it can continue. If the request is asynchronous, the calling system can continue processing after it makes the request while the providing system independently continues to process the request. In terms of time, the asynchronous call is more loosely coupled than the synchronous call.

How to support the asynchronous messaging? Use a design pattern to propose a solution.
Asynchronous Method Invocation

• How to get the result?
  – Polling
  – Callback

• Callback and the Observer Design Pattern
Distributed Callback – the observer pattern
(synchronous vs. asynchronous messaging)

Douglas C. Schmidt, Steve Vinoski, Object Interconnections Distributed Callbacks and Decoupled Communication in CORBA (Column 8), SIGS C++ Report magazine, October 1996

Dr. Xiaolin Hu
Finally, One More Issue

• ClassA and ClassB run on two different computers, and are likely designed by different designers. How does ClassA know that ClassB has the sayHello() method?

• To make ClassA use ClassB’s services, ClassA needs to know ClassB’s interfaces.

• Client – server

• How to establish a contract between client and server?

This leads to the paradigm of interface-based programming.
ServiceInterface – contract

Figure 2: Structure with distribution (from Microsoft web site)
CORBA

CORBA - Common Object Request Broker Architecture

OMG -- Object Management Group
Develops CORBA standards, but does not implement

See: http://www.omg.org/ for more information on OMG and CORBA
Solution prior to CORBA

• Remote procedure calls (RPCs)
  – Enabling functions to be invoked by clients on one machine and executed by servers on a different machine
  – Low-level; provides very little in the way of encapsulation
  – Other problems include queuing of requests, event notification, transaction management…
Remote Procedure Call (RPC)

- Remote procedure call (RPC) is an Inter-process communication technology that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. That is, the programmer would write essentially the same code whether the subroutine is local to the executing program, or remote.

- Note that there are many different technologies commonly used to accomplish this which are often incompatible.
  - Socket
  - Middleware

Distributed Objects

• The term distributed objects usually refers to software modules that are designed to work together, but reside either in multiple computers connected via a network or in different processes inside the same computer.
• One object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.
• Examples
  – Distributed objects are used in Java RMI.
  – CORBA lets one build distributed mixed object systems.
  – DCOM is a framework for distributed objects on the Microsoft platform.

http://en.wikipedia.org/wiki/Distributed_object
CORBA – an Integration of RPCs and Object Oriented Concept

- Gives rise to notions of distributed components and interface-based programming
- Clients program to an interface where different implementations of the interface (components) are possible
- Notions like inheritance, delegation, dynamic binding are kept in CORBA
Essential CORBA concepts

- A **CORBA component** is an object
- A **CORBA interface** is specified in CORBA IDL
- A **CORBA class** implements one or more CORBA interfaces
- CORBA is programming-language neutral
- The execution of a CORBA component is location transparent
IDs and Names of Objects

• All CORBA objects have globally unique identifiers called **Interoperable Object References (IORs)**, which are machine generated and not intended for human use

• A **named object** has a well-known, unique, human-friendly name that clients use to access it

• A server process advertises one or more named objects; client processes have no named objects
Access to Remote Objects

- Clients access remote objects via references --- light-weight objects that have the same interface as remote objects, except their task is to forward all methods invocations to the server for remote object execution
Object Request Broker (ORB)

- Runs in every CORBA process (both client and server), responsible for communicating with other ORBs
- Acts as a message bus between objects
- Interacts with each other via General Inter-ORB Protocol (GIOP)
What an ORB Does

- Locate the object to which the message is to be sent
- If the object is at another process, pass the message to the ORB of that process
- If the object is within the same process as the ORB, ensures that the object is ready to receive the request and pass the parameters to the object
- Dynamic invocation of an object.
CORBA Interface Definition Language (IDL)

- IDL is the basis of CORBA interface-centric programming, which enables the development of language-neutral interfaces that CORBA objects implements.
- Language-specific compilers will translate IDL specifications into language-specific source codes that is used to implement those interfaces.
Suppose the server is written in language L1, and the client is written in L2

1. Define the interface (I) of server object by CORBA IDL (language neutral)

2. Compile I to I’ written in L1; the server object implements I’

3. Compile I to I’’ written in L2
Interface Centric Programming (cont.)

4. The client object creates a reference of the server object that has I” as its interface.

5. Whenever the client object needs to call the server object to perform some operation, it calls the corresponding method of the reference instead.

6. CORBA will automatically forward the method call to the remote server object.
IDL Syntax

- Based on C++ concepts and syntax
- Supports typedefs, unions, enumerated types, sequences
- Needs language-specific compilers to translate it into language-specific interface
//M.idl
Module M {
    exception Oops {
        string whatiswrong;
    };
    interface MyInfo {
        string name;
        void setName( in string aName);
    }
    interface InfoManager {
        void register ( in MyInfo info ) raises (Oops);
        boolean find ( in string name, out MyInfo info);
    }
};
Development Process

IDL Developer

IDL Source

IDL-to-Java Compiler

IDL-to-C++ Compiler

Client Developer

App.java
Stubs.java

Client Executable

Server Developer

Stubs.cc
impl.cc

Server Executable

ORB

Dr. Xiaolin Hu
A CORBA example

**Step 1** Create an interface for the server using the CORBA interface definition language (IDL)

Hello.idl

```java
interface Hello {
    string sayHello();
};
```

**Step 2** Compile the IDL interface

`idl Hello.idl`

This creates a directory called `java_output` which contains:
Hello.java, HelloSkeleton.java HelloHelper.java _HelloImplBase.java _HelloStub.java HelloHolder.java _HelloOperations.java _tie_Hello.java
Step 3 Implementing the Server using **Inheritance**

```java
public class HelloImplementation extends _HelloImplBase {
    public String sayHello() {
        return "Hello World";
    }
}
```

**HelloServer**

```java
public class HelloServer {
    public static void main (String args[]) {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
        try {
            Hello server = new HelloImplementation();
            _CORBA.Orbix.impl_is_ready( "HelloServer" );
        }
        catch ( org.omg.CORBA.SystemException corbaError) {
            System.out.println("Exception ", corbaError);
        }
    }
}
```
Step 3 Implementing the Server using delegation (the Tie model, see an example at http://java.sun.com/j2se/1.4.2/docs/guide/idl/jidlTieServer.html)

```java
public class HelloImplementation implements Hello {
    public String sayHello() {
        return "Hello World";
    }
}

HelloServer
public class HelloServer {
    public static void main (String args[]) {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
        try {
            Hello server = new HelloImplementation();
            // create a tie, with servant being the delegate.
            _tie_Hello tie = new _tie_Hello (server );
            tie._this(orb);
            _CORBA.Orbix.impl_is_ready( "HelloServer" );
        }
        catch ( org.omg.CORBA.SystemException corbaError) {
            System.out.println("Exception ", corbaError);
        }
    }
}
```

Dr. Xiaolin Hu
Note: these two class diagrams illustrate the difference between the two approaches. They may not represent the actual relationships among the classes.
A CORBA example

Step 4 Compiling the Server

Step 5 Registering and running the Server

Make sure that the OrbixWeb daemon (orbixdj) is running on the server machine

You start the daemon by the command:
orbixdj -textConsole
Now register the server via:
putit HelloServer ...
Now run the server via
java HelloServer

Dr. Xiaolin Hu
A CORBA example

Step 6 Writing the client

HelloClient.java

```java
public class HelloClient {
    public static void main(String args[]) {
        ORB.init();
        String hostname = "eli.sdsu.edu";
        String serverLabel = ":HelloServer";
        Hello server = HelloHelper.bind( serverLabel, hostname);
        System.out.println( server.sayHello() );
    }
}
```

The magic operation
A CORBA example

Step 7 Compiling and Running the client

Compile the classes:
_HelloStub.java
HelloClient.java

Now run the client with the command:

java HelloClient
Middleware

General-purpose software, defined by an API, that facilitates application elements to interoperate at a logical level, be distributed across multiple systems, or ported to another platform, despite differences in underlying communication protocols, operating systems, or other basic services.
Types of Middleware

• Database-oriented middleware
  – high-level DBMS use
• Function-oriented middleware
  – ‘traditional’ distributed computing such as RPC
• Message-oriented middleware
  – communication-oriented distr. computing
• Object-oriented middleware
  – object-oriented distributed computing
Literature Review Presentation

• Each group’s presentation is 45 minutes, including 5 minutes Q&A.

• LR is 32% of the total grade.
  – 60% presentation
  – 40% final report

• Grading of LR presentation