4. CORBA & IDL

www.EJBTutorial.com

Dr. Imed Bouchrika
Dept of Mathematics & Computer Science
University of Souk-Ahras
imed@imed.ws
What is CORBA

- **Common Object Request Broker Architecture**

- A *specification* for creating and using distributed objects

- It’s a specification rather than an implementation

- CORBA provides an **RPC mechanism** to allow distributed objects:
  - to *interoperate* in a *heterogeneous* environment
  - Objects can be implemented in different programming languages
  - and deployed on different platforms
**What is CORBA**

- CORBA can be thought as a language-independent RMI:
  - Similar lifecycle including definition of interfaces, generation of stubs and skeletons
  - Used to connect together application written in different languages on different platforms

- **interfaces** must be specified in a *language-independent* format
Object Management Group (OMG)

- A *standard* developed by OMG to help in distributed programming
- OMG *was* initially created in 1989 by 11 companies.
- OMG has over 800 members today
- One of the largest industry consortiums
- Does not develop implementations
- Formal process for defining specifications
  - Members submit proposals, develop implementations
- Emphasizing cooperation and compromise
  - Most specs are fusion of ideas
- Non-profit
- To learn more: [http://www.omg.org/gettingstarted/](http://www.omg.org/gettingstarted/)
“...promote the theory and practice of object-oriented technology in software development...”

“...promote the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments...”
CORBA Features

- Heterogeneity

- Object Orientation
  - Inheritance
  - Encapsulation
  - Instantiation
  - Polymorphism

- Transparency
  - Migration, Access, Recovery ...

- Dynamic Binding
## CORBA Products

<table>
<thead>
<tr>
<th>ORB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java 2 ORB</td>
<td>comes with Sun's Java 2 SDK</td>
</tr>
<tr>
<td>VisisBroker for Java</td>
<td>A popular Java ORB from <strong>Inprise Corporation</strong>. VisiBroker is also embedded in other products</td>
</tr>
<tr>
<td>OrbixWeb</td>
<td>A popular Java ORB from <strong>Iona Technologies</strong></td>
</tr>
<tr>
<td>WebSphere</td>
<td>A popular application server with an ORB from <strong>IBM</strong></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Basic Architecture

naming service

object client

stub

ORB

network

operating system

logical data flow

physical data flow

object implementation

skeleton

ORB

network

operating system

Cross-language CORBA application

object client written in Java

stub in Java generated by compiling the CORBA object interface

ORB written in Java

object implementation written in C++

skeleton in C++ generated by compiling the CORBA object interface

ORB written in C++
The Basic Architecture

Client

Dynamic Invocation
Static Stub

ORB Core

Object Implementation

operation()

args + return value

ORB Core

Dynamic Skeleton Interface
Static Skeleton

Network

Object Adapter

ORB-dependent implementation
Application specific Stub and Skeleton
Same interface, ORB-independent
ORB core:

- is the critical, well defined middleware that allows different CORBA implementations to communicate seamlessly.
- Is responsible for communication of requests.
- Transfers request to object implementation.
Object adapter

- Provides an interface between the ORB and the object implementation and enables their communication

- Maintains a mapping of object references to their implementations

- Creates remote object references for CORBA objects

- Dispatches client requests to server objects

- Activates and deactivates objects
The Portable Object Adapter

There are different types of CORBA object adapters.

The Portable Object Adapter, or POA, is a particular type of object adapter that is defined by the CORBA specification.

An object adapter that is a POA allows an object implementation to function with different ORBs, hence the word portable.
CORBA components: Object Interface

- A distributed object is defined using an **IDL** file similar to the remote interface in RMI.

**IDL → Interface Definition Language**

- Since CORBA is language independent, the **IDL** interface is defined using a universal language with a distinct syntax.

- The syntax of CORBA IDL is similar to Java and C++. However, object defined in a CORBA IDL file can be implemented in a large number of diverse programming languages, including C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLScript.
Inter-ORB Protocols : GIOP/IIOP

- Multiple vendors implementing different ORBs

- A need for ORBs to interface with one another in a standard way: General Inter-ORB Protocol

- GIOP: generic protocol, no implementation

- IIOP (Internet Inter-ORB Protocol) is the TCP/IP implementation of GIOP
The IIOP specification includes the following elements:

- **Transport management requirements**: specifies the connection and disconnection requirements.

- **Definition of common data representation**: a coding scheme for marshalling and unmarshalling data of each IDL data type.

- **Message formats**: different types of message format are defined. The messages allow clients to send requests to object servers and receive replies.
As in Java RMI, a CORBA distributed object is located using an **object reference**.

Since CORBA is language-independent, a CORBA object reference is an abstract entity mapped to a language-specific object reference by an ORB.

For interoperability, OMG specifies a protocol for the abstract CORBA object reference object, known as the **Interoperable Object Reference (IOR)** protocol.
An IOR is a string that contains encoding for the following information:

- The type of the object.
- The host where the object can be found.
- The port number of the server for that object.
- An object key, a string of bytes identifying the object.
  - The object key is used by an object server to locate the object.
The following is an example of the string representation of an IOR:

IOR:0000000000000000d49444c3a677269643a312e30000000000000000483a5c756c7472612e6475626c696e2e696f6e612e69653a677269643a303a3a49523a67726964003a
The **Naming Service** serves as a directory for CORBA objects, and, as such, is platform independent and programming language independent.

The Naming Service permits ORB-based clients to obtain references to objects they wish to use.

To export a distributed object, a CORBA object server contacts a Naming Service to **bind** a symbolic name to the object.

To obtain a reference to the object, an object client requests the Naming Service to **look up** the object associated with the name.
CORBA Object Services

CORBA specify services commonly needed in distributed applications such as:

- **Naming Service**:
- **Concurrency Service**: for event synchronization;
- **Event Service**: for event logging;
- **Event Service**: for event scheduling;
- **Logging Service**: for event logging;
- **Scheduling Service**: for event scheduling;
- **Security Service**: for security management;
- **Trading Service**: for locating a service by the type (instead of by name);
- **Time Service**: a service for time-related events;
- **Notification Service**: for events notification;
- **Object Transaction Service**: for transactional processing.
Steps for CORBA

1. Define an interface using IDL
2. Use the IDL compiler.
3. Implement the Hello object
4. Implement the server
5. Run the ORB processes.
6. Compile and run the server.
7. Implement the client
8. Run
Steps for CORBA

1. Define Interfaces in IDL
2. Compile Interfaces with IDL Compiler
3. Implement the Client side
4. Implement the functionality of the Interfaces
5. Implement the server side
6. Compile to byte-code
7. Start CORBA Service
8. Start Server and Client Application

IDL Stubs
IDL Skeletons

Define Interfaces in Java
Implement the Client side
Implement the functionality of the Interfaces
Compile to byte-code
Use RMI compile (rmic)
Start Server and Client Application
RMI Stubs
RMI Skeletons
CORBA vs. Java RMI

- CORBA differs from the architecture of Java RMI in one significant aspect:
  - RMI is a proprietary facility developed by Sun Microsystems, Inc., and supports objects written in the Java programming language only.
  - CORBA is an architecture that was developed by the Object Management Group (OMG), an industrial consortium.
RMI vs CORBA: Similarities

- Both provide a framework for developing distributed applications
  - Provides a lot of services to support and ease the development
  - Compared with socket programming, developers concentrate more on business logic rather than low-level protocols

- Both provide tools to generate stubs and skeletons for application
  - RMI use RMI compiler (rmic) and CORBA use various compilers (e.g. idlj for java)

- Both provide name service to register and discover service by name

- Both support static and dynamic method invocation
  - RMI use Java reflection and CORBA use DII and DSI via interface repository
RMI vs CORBA: Differences

Language and platform support
- RMI is designed only for Java and only works under JVM.
- CORBA is designed to work with multiple languages and platforms

Communication Protocol
- RMI uses Java Remote Method Protocol (JRMP) which utilizes Java object Serialization
- CORBA uses language independent General Inter-ORB Protocol (GIOP) / Internet Inter ORB Protocol (IIOP)

Programming Model
- RMI is a pure object-oriented programming model
- CORBA supports both object-oriented programming and non object-oriented programming

Garbage Collection
- RMI support automatic distributed garbage collection
- CORBA does not support distributed garbage collection
RMI vs CORBA: Differences

Name Schema in Name Service
- RMI makes use of URL based name schema to look up object
- CORBA constructs a hierarchical structure of object’s name

Security
- RMI utilizes the build-in Java security framework to grant various permissions
- CORBA has its own security service to handle security issue

Simplicity
- RMI is much simpler to learn and use
- CORBA is a big specification and hard to learn

Interoperability
- RMI supports if IIOP is used as transport
- CORBA supports interaction between implementations in various languages and platforms
Interface Definition Language (IDL)

IDL is object-oriented
- specify interfaces containing operation and attributes
- supports interface inheritance (single and multiple)

IDL designed to map to multiple programming languages

IDL similar to Java interfaces and C++ abstract classes
OMG IDL Features

- modules
- interfaces
- operations
- attributes
- inheritance
- basic types
- arrays
- sequence
- struct
- enum
- typedef
- const
- exceptions
Modules

- IDL module defines naming scope for a set of IDL definitions
- Group interface and other IDL type definitions into logical name space
- Avoid name clashes

```idl
// IDL
module finance
  interface account {...};
};
```

- Fully scoped name of interface `account` is `finance::account`
The basic unit is the interface defining the interface to a type of object

interface account {
    // operation and attribute definitions
};
interface account {
   // operations
   void makeDeposit (in float amount);
   boolean makeWithdrawal (in float amount,
                           out float balance);
}

- Look like C++/Java member functions
- parameters must be named
- parameter passing mode must be specified
Parameter Passing Modes

- **in** passed from caller to called object
- **out** passed from called object to caller
- **inout** passed in both directions

- CORBA must know the direction
  - to know how to pass the values
Raising Exceptions in IDL Operations

- Operations can raise exceptions to indicate occurrence of an error

- Two types of exceptions:
  - System exceptions - standard set defined by CORBA
  - User-defined exceptions

```
// IDL
module finance {
    interface account {
        exception WithdrawalFailure {
            string reason;
        }
        void MakeWithdrawal(in float amount, out float newBalance)
            raises(WithdrawalFailure);
    }
}
```
Typically these map to two functions in the programming language:

- set the value
- get the value

A `readonly` attribute maps only to a function to get the value.
Inheritance

- An IDL interface can inherit all elements of one or more other interfaces

```java
interface checkingAccount : account {
    attribute float overdraftLimit;
    boolean orderCheckBook();
};
```

- All attributes and operations of account are valid on objects of interface checkingAccount
Basic Types

- short (16-bit)
- long (32-bit)
- unsigned short (16-bit)
- unsigned long (32-bit)
- long long (64-bit)
- unsigned long long (64-bit)
- float
- double
- char (8-bit)
- wchar (16-bit)
- boolean (TRUE or FALSE)
- octet (8-bit - no conversion)
- any (arbitrary IDL type)
- string (can be bounded)
**Enum**

```cpp
enum currency {pound, dollar, yen, franc};
```

**Struct**

```cpp
struct customerDetails {
    string name;
    string age;
};
```

**Union**

```cpp
union Date switch (short) {
    case 1: string stringFormat;
    case 2: long digitalFormat;
    default: DateStructure structFormat;
};
```
Other IDL Types

- **String** - max can be specified
  
  ```
  attribute string sortCode<10>;
  ```

- **Sequence** - 1D array- can be bounded or unbounded
  
  ```
  sequence<account, 50> accounts;
  ```

- **Array** - can be multidimensional - always fixed-size
  
  ```
  account accounts[3];
  ```

- **Constant**
  
  ```
  const long MaxAccounts = 10000;
  ```
Mapping to a Programming Language

- e.g. Java client’s view

```java
public interface account extends org.omg.CORBA.Object {
    public float balance();
    public void MakeDeposit(float f);
    public void Withdraw(float f);
}
```

- call these methods to make a call to a CORBA object
rmic vs IDL

- Not sure about IDL syntax, You can auto generate the IDL file using the rmic from your RMI Java Interface:

- Use the following command : rmic -idl AdditionInterface
The IDL file named : Addition.idl

```idl
/**
 * @author imed
 */
module AdditionApp
{
  interface Addition
  {
    long add(in long a, in long b);
    oneway void shutdown();
  };
};
```
Example : AdditionObj.java

import AdditionApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

class AdditionObj extends AdditionPOA {
    private ORB orb;
    public void setORB(ORB orb_val) {
        orb = orb_val;
    }

    public int add(int a, int b) {
        int r = a+b;
        return r;
    }

    public void shutdown() {
        orb.shutdown(false);
    }
}

import AdditionApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

public class MainServer {
    public static void main(String args[]) {
        try {
            ORB orb = ORB.init(args, null);
            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();

            // create servant and register it with the ORB
            AdditionObj addobj = new AdditionObj();
            addobj.setORB(orb);
        }
    }
}
Example : MainServer.java

```java
// get object reference from the servant
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(addobj);
Addition href = AdditionHelper.narrow(ref);

org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

NameComponent path[] = ncRef.to_name("ABCD");
ncRef.rebind(path, href);
System.out.println("Addition Server ready and waiting ...");
// wait for invocations from clients
for (;;){
    orb.run();
}
}

catch (Exception e) {
    System.err.println("ERROR: " + e);
}

System.out.println("HelloServer Exiting ...");
```
Running the Server Example

- You need to compile the IDL file.
  ```
  idlj Addition.idl
  ```

- You need to run the ORB process daemon as:
  ```
  start orbd -ORBInitialPort 1050
  ```

- Once you have compiled the project, you can run it as:
  ```
  java MainServer -ORBInitialPort 1050 -ORBInitialHost localhost
  ```
import AdditionApp.*;

import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import java.io.*;
import java.util.*;

public class StartClient {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try {
            ORB orb = ORB.init(args, null);
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            Addition addobj = (Addition) AdditionHelper.narrow(ncRef.resolve_str("ABC"));
        }
    }
}
Scanner c = new Scanner(System.in);
System.out.println("Welcome to the addition system:");
for (;;) {
    System.out.println("Enter a:");
    String aa = c.nextLine();
    System.out.println("Enter b:");
    String bb = c.nextLine();
    int a = Integer.parseInt(aa);
    int b = Integer.parseInt(bb);
    int r = addobj.add(a, b);
    System.out.println("The result for addition is : "+r);
    System.out.println("-----------------------------------");
}
catch (Exception e) {
    System.out.println("Hello Client exception: " + e);
}
}
Client Side

- You need to compile the IDL file on the Client **also** 
idlj Addition.idl

- Once you have compiled the Client Project, you can run it as:
  java MainClient -ORBInitialPort 1050 -ORBInitialHost localhost

Imed Bouchrika. Distributed Objects, Uni of Souk-Ahras 2013-2014

http://www.imed.ws