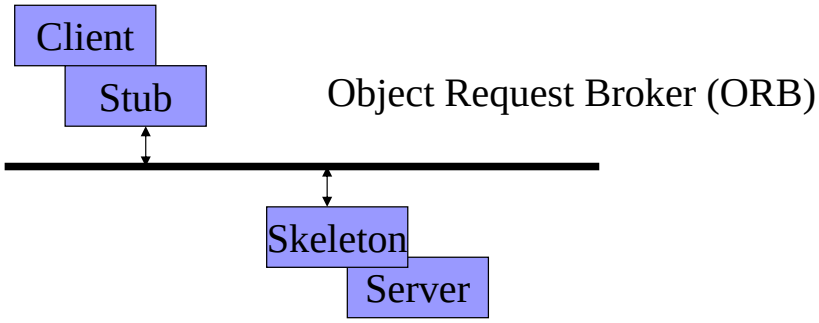
Common Object Request Broker Architecture (CORBA)

CORBA

- CORBA is similar in high level concepts to RMI →RMI is basically a simplified form of CORBA
- Adds cross-platform, multiple language interfaces, more bells and whistles
- Widely used standard

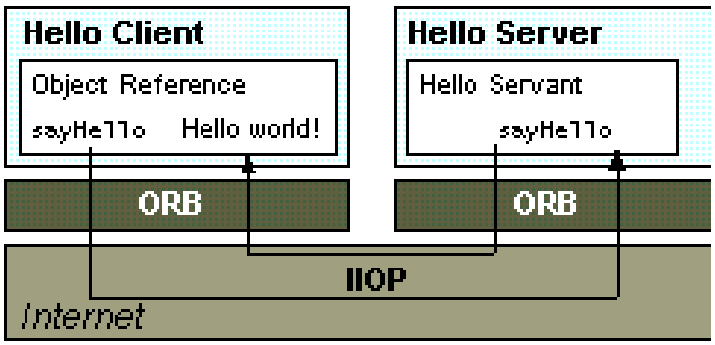
CORBA

- At the top level the architecture is similar to RMI



ORB and IIOP

The Internet Inter-ORB Protocol (IIOP) is a protocol by which ORBs communicate (Similar to JRMP in RMI)



CORBA

- The Object Request Broker (ORB) is the “bus” that connects objects across the network. It’s a virtual bus; some network software that runs on each machine
- It locates the object on the network, communicates the message, and waits for results
- Not all ORBs are alike—there are some standardization moves around, but it’s not generally true that a client using one ORB can talk to a remote object using a different ORB
- Major vendors: Java 2 ORB, VisiBroker, WebSphere ORB, Iona (Orbix).

CORBA Services

- CORBA has a standard, the IIOP, that (supposedly) allows you to communicate with CORBA objects running on another vendor’s ORB over TCP/IP
- The CORBA infrastructure also provides services for naming, transactions, object creation, etc. These are CORBA services implemented in CORBA.

Stub & Skeleton Creation

- In RMI we had an RMI compiler run on the .class file we wanted to be remote. It's different under CORBA
- The interface is written in Interface Definition Language, IDL. IDL is language-neutral (by being a different language than anything else) though highly reminiscent of C++ (multiple inheritance, similar syntax, etc.)
- IDL has a set of primitive data types similar to most languages—integer, float, string, byte, etc.

CORBA Development Process

1. Write an IDL file which describes the interface to the distributed object.
2. Run idlj on the IDL file. This generates Java code that implements the stub and the skeleton
3. Run nameserver
4. Implement the servant (implement the IDL interface)
5. Implement the server (register the servant with the naming service and wait for requests)
6. Implement the client
7. Run the server
8. Run the client

Interface Definition Language (IDL)

- Compile the IDL code with idlj. This creates java source code files in the Tracker package. Note the use of the IDL primitive type “string”
- idlj **-fclient -fserver –oldImplBase** Tracker.idl

```
module Tracker {
 interface Time {
  string getTime();
 };
};
```

Interface Definition Language (IDL) *(Contd.)*

- A pure description language.
  - enables developers to describe an interface they wish to use remotely in a language-independent fashion.
- Language specific compiler required for each participating language (e.g., idlj for Java).
  - creates files required by each language for CORBA related tasks.

Idlj: IDL to Java Compile output

- Idlj generates some horror-story java code in the Tracker directory.
  - _TimeStub (the stub for the client)
  - Time (the interface for the distributed object)
  - TimeHelper (Kitchen sink functionality)
  - TimeHolder (Serialization & communications; out & inout parameters)
  - TimeOperations (initial server implementation)
  - _TimeImplBase (base class for servant)

Start the nameserver

- CORBA uses a technique similar to that of the RMI registry. Servers register with the name server, and clients ask the name server where to find the server.
- orbd –ORBInitialPort <myPort>

  Example: orbd –ORBInitialPort 900
- This is a replacement for the older tnsnameserver
- Note that *myPort* must match up with the properties used to find the name server in the client and server

Client-side Code

- Very similar to RMI
  - Get reference to remote object
  - Cast it to what you want
  - Call methods on the object via the local proxy object

  On the other hand, there are more annoyances due to CORBAs multi-language background

Client.java:

```
package ex1;
import Tracker.*; // The package containing our stubs.
import org.omg.CosNaming.*; // Client will use the naming service.
import org.omg.CORBA.*; // All CORBA applications need these classes.

public class Client {
 public static void main (String args[]) {
  try {
   // Create and initialize the ORB
   ORB orb = ORB.init (args, null);
   // Get the root naming context
   org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references ("NameService");
   NamingContext ncRef = NamingContextHelper.narrow (objRef);
   // Resolve the object reference in naming
   NameComponent nc = new NameComponent ("TimeServer", "");
   NameComponent path[] = {nc};
   Time timeRef = TimeHelper.narrow (ncRef.resolve (path));
   // Call the time server object and print results
   String time = "Time on the Server is " + timeRef.getTime ();
   System.out.println (time);
  } catch (Exception e) {
   e.printStackTrace ();
  }
 }
}
```

Server-side Code

- On the server-side, there is the “Server” code, and the “Servant” code. The servant code is the object implementation. The server code holds one or more servants.

The servant Code

```
package ex1;
 // The package containing our stubs.
import Tracker.*;
// Server will use the naming service.
import org.omg.CosNaming.*;
// The package containing special exceptions thrown by the name service.
import org.omg.CosNaming.NamingContextPackage.*;
// All CORBA applications need these classes.
import org.omg.CORBA.*;
import java.util.*;
import java.text.*;

class TimeServer extends _TimeImplBase
{
    public String getTime ()
    {
    SimpleDateFormat formatter = new SimpleDateFormat ("MMMMM dd,
    yyyyy GGG, hh:mm:ss:SSS aaa"); Date date = new Date (); return
    formatter.format ( date );
    }
}
```

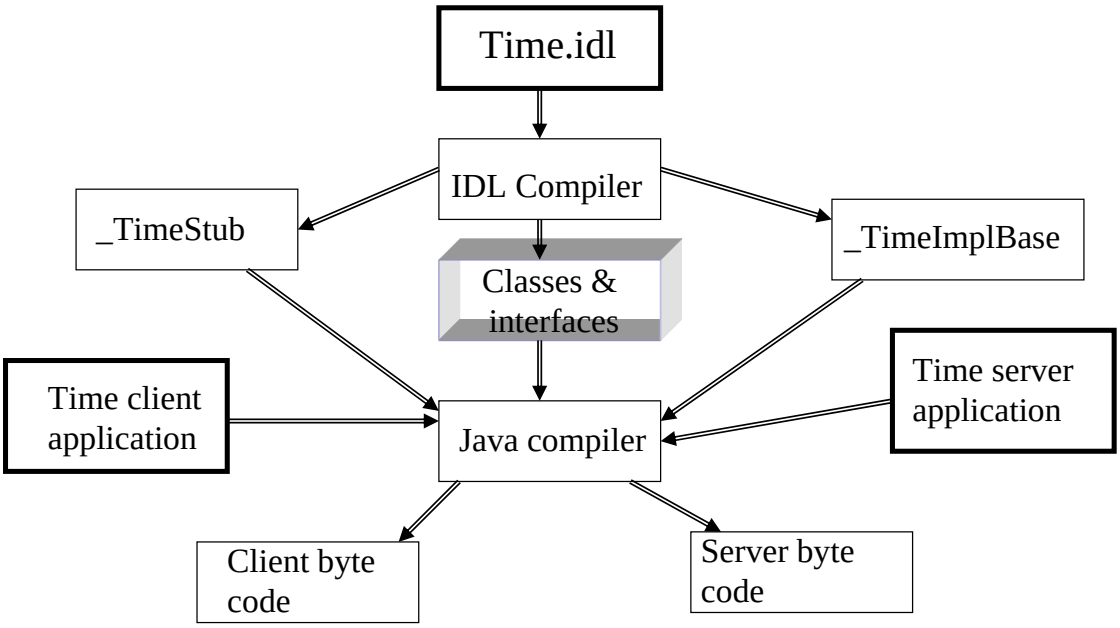
The Server Code

- The server code does the following:
  - Creates a new Servant object
  - Registers the Servant with the naming service
  - Listens for incoming requests (Makes sure we don't exit)

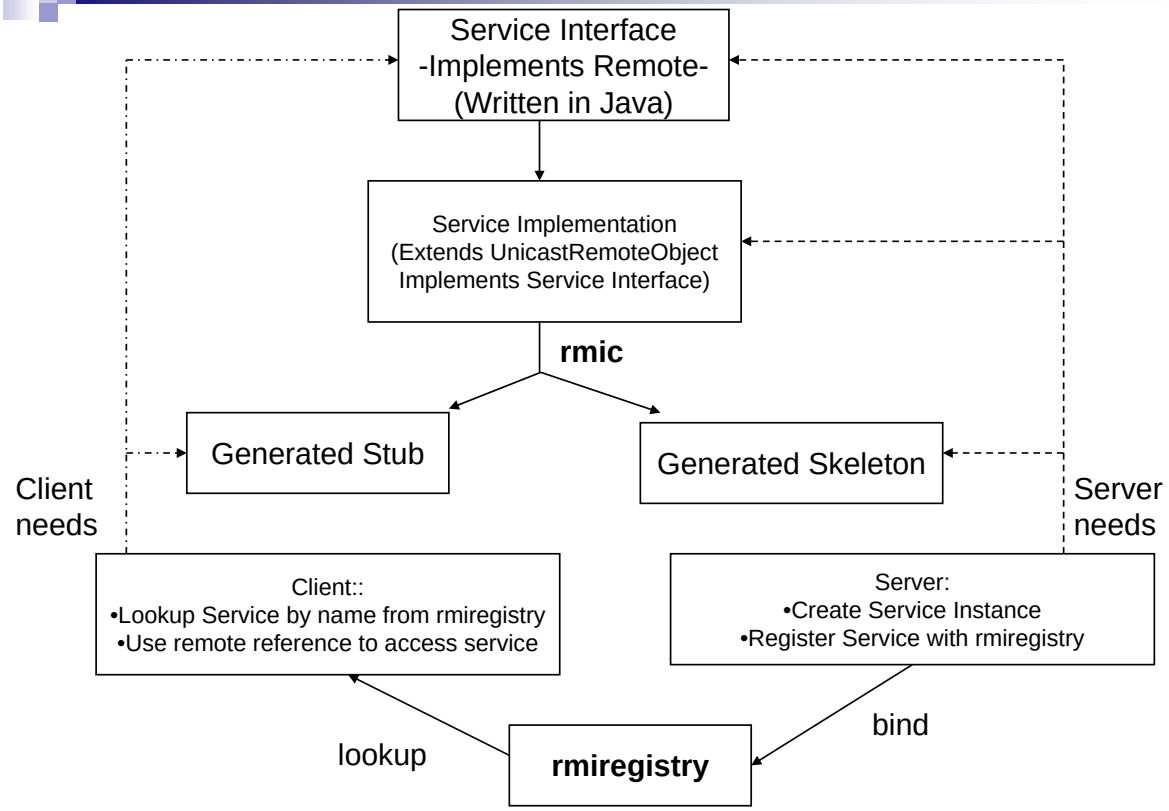
The Server Code

```
package ex1;

// The package containing our stubs.
import Tracker.*;
// Server will use the naming service.
import org.omg.CosNaming.*;
// The package containing special exceptions thrown by the name service.
import org.omg.CosNaming.NamingContextPackage.*;
// All CORBA applications need these classes.
import org.omg.CORBA.*;

public class Server {
 public static void main (String args[]) {
  try {
   // Create and initialize the ORB
   ORB orb = ORB.init (args, null);
   // Create the servant and register it with the ORB
   TimeServer timeRef = new TimeServer ();
   orb.connect (timeRef);
   // Get the root naming context
   org.omg.CORBA.Object objRef = orb.resolve_initial_references ("NameService");
   NamingContext ncRef = NamingContextHelper.narrow (objRef);
   // Bind the object reference in naming
   NameComponent nc = new NameComponent ("TimeServer", "");
   NameComponent path[] = {nc};
   ncRef.rebind (path, timeRef);
   // Wait forever for current thread to die
   Thread.currentThread ().join ();
  } catch (Exception e) {
   e.printStackTrace ();
  }
 }
}
```

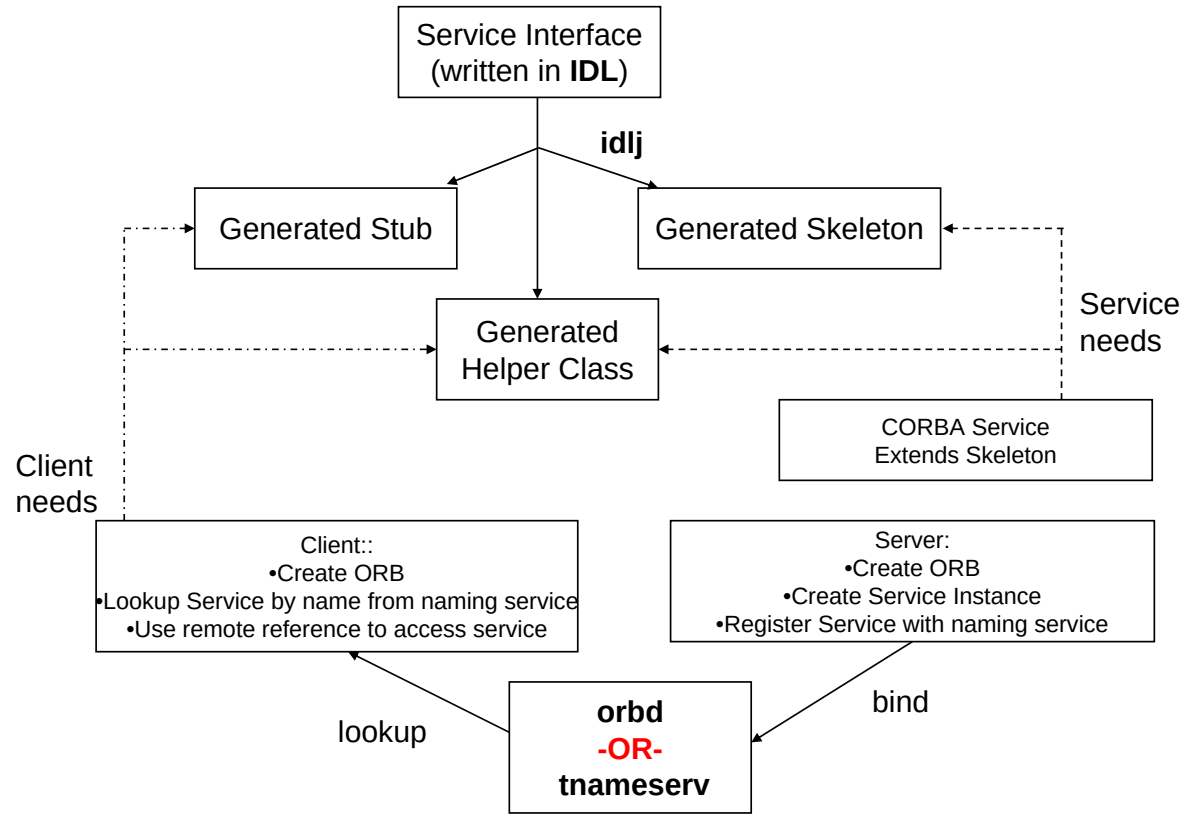
Overview of Compilation Process



RMI In One Slide



CORBA In One Slide



Reference

- CORBA, The MOVE Institute: Naval Postgraduate School.