Information Systems 2

3. Distributed Information Systems I: CORBA

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Business Economics and Information Systems
& Institute for Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

Example Scenario

Assume, you have to set up an information system that informs business customers about products you offer and the prices you charge.

A later stage of the system should allow

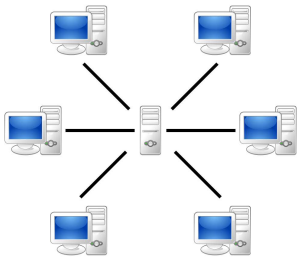
- product managers to add, edit and remove products,
- marketing staff to set prices,
- customers to place orders and
- sales staff to mark orders as shipped
- etc.

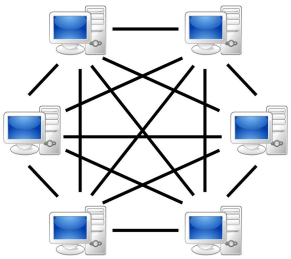
To accomplish this, many different persons have to collaborate on different aspects of the data and the process.

Thus, the whole system has to be distributed.

Paradigms of Distributed Systems

In general, one distinguishes two types of distributed systems:





Client/Server Applications:
A central server hosts the shared part of the data and offers services to different clients, e.g., access to the data as well as communication between the clients.

Peer-to-Peer Applications:
There is no central server, but the data is distributed over a network of clients (called peers). Peers may communicate directly with each other as well as indirectly by routing throught the peer network.

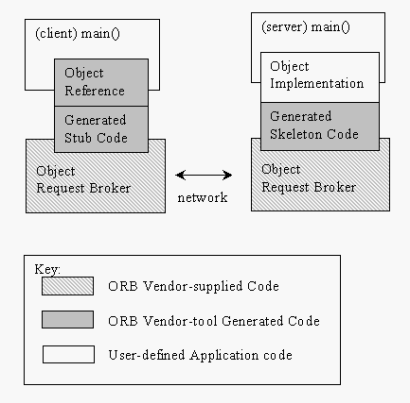
CORBA

The **Common Object Request Broker Architecture** (**CORBA**) allows programs

- on different computers,
- written in different languages

to communicate.

Communication is mediated by so called **Object Request Brokers** (**ORB**s).



CORBA Standard

CORBA is an open standard developed by the Object Management Group (OMG):

- CORBA 1.0 / Oct. 1991 (Object model, IDL, core DII; C language binding)
- CORBA 2.0 / Aug. 1996 (GIOP, IIOP; C++ and Smalltalk language bindings)
- CORBA 2.2 / Feb. 1998 (POA; Java language binding)
- CORBA 3.0 / Jul. 2002
- CORBA 3.1 / Jan. 2008

CORBA is widespread. E.g., an implementation ships with every Sun JDK release.

Benefits of CORBA [McH07]

- **Maturity:**
  CORBA is developed since 1991.
- **Open Standard:**
  CORBA is standardized by the Object Management Group (OMG).
- **Wide platform support:**
  CORBA is available for mainframes (e.g., IBM OS/390s), Unix & Linux, Windows, AS/400, Open VMS, OS X and several embedded operating systems.
- **Wide language support:**
  CORBA has language bindings for C, C++, Java, Smalltalk, Ada, COBOL, PL/I, LISP, Python and IDLScript.
- **Efficiency:**
  CORBA marshals data, i.e., converts data from programming-language types into binary representations that can be transmitted efficiently.
- **Scalability:**
  CORBA servers can handle huge server-side data as well as high communication loads from thousands of client applications.

General Procedure

The implementation of a distributed system with CORBA requires the following four steps:

1. **Interface:** create the interface description.
2. **Implementation:** implement the interface.
3. **Server:** implement a server application offering remote access to objects.
4. **Client:** implement a client application using the remote objects.

Step 1: Interface / IDL Description

To specify the interface there is a programming language unaware **Interface Definition Language** (**IDL**):

- Interfaces are grouped in **modules** ($\equiv$ Java packages).
- Each interface consists of a set of methods with arguments, return type and exceptions.
- Arguments and return values can have the usual elementary datatypes or interfaces themselves.
- The grammar is very close to Java.
- Interfaces are mapped to specific programming language interfaces by the use of a tool (e.g., `idlj`).

Step 1: Interface / IDL Description

```
1 module ismll_commerce {
2   interface Offer {
3     string name();
4     double price();
5   };
6 };
```

Figure 4: Offer.idl: Interface description for offers.

Step 1: Interface / IDL to Java/C++ Binding

IDL	Java	C++
module	package	namespace
interface	interface	abstract class
operation	method	member function
attribute	pair of methods	pair of functions
exception	exception	exception

IDL type	Java type
boolean	boolean
char / wchar	char
octet	byte
short / unsigned short	short
long / unsigned long	int
long long / unsigned long long	long
float	float
double	double
string / wstring	String

To create the Java base class `Offer.java` and other derived classes (see below):

```
idlj Offer.idl
```

Creates class Offer.java in package ismll_commerce.

Step 1: Interface / Derived Java Interface

```
module ismll_commerce {
    interface Offer {
        string name();
        double price();
    };
};
```

Figure 5: Offer.idl: Interface description for offers.

```
package ismll_commerce;


/**
* ismll_commerce/OfferOperations.java .
* Generated by the IDL-to-Java compiler (portable), version '
* from Offer.idl
* Monday, May 26, 2008 11:48:56 AM CEST
*/

public interface OfferOperations
{
  String name ();
  double price ();
} // interface OfferOperations
```

Figure 6: OfferOperations.java: derived Java interface.

Step 2: Implementation

The implementation has to be derived from the abstract **server skeleton** or **servant** class,
in the Sun JDK: `OfferPOA`.

The implementation may not contain any CORBA specific code.

The servant is generated automatically from the IDL spec.

The servant implements the programming language specific interface.
In Sun JDK: `Offer`.

Step 2: Implementation

```
1 package ismll_commerce;
2
3 public class OfferImpl extends OfferPOA {
4     public OfferImpl(String name, double price) {
5         this.name = name; this.price = price;
6     }
7     public String name() { return name; }
8     public double price() { return price; }
9
10    protected String name;
11    protected double price;
12 }
```

Figure 7: OfferImpl.java: implementation of the interface methods.

Step 3: Server

The server application has to

1. connect to the ORB infrastructure,
   (a) create an `ORB` with a specific hostname and port,
   (b) retrieve a reference to the root **Portable Object Adapter** (**POA**) and
   (c) activate it.

2. create application objects,
   (a) using the implementation / servant class from step 2.

3. output references to them,
   (a) by looking up string representations of references of the servants,
   in the simplest case **Interoperable Object References** (**IORs**).

4. wait for connections to the application objects.

Step 3: Server

```
1 package ismll_commerce;
2 import org.omg.CORBA.ORB;
3 import org.omg.PortableServer.*; // POA, POAHelper
4
5 public class OfferServer {
6   public static void main(String args[]) {
7     try{
8       // a. connect to ORB infrastructure:
9       String[] argsOrb = new String[] { "-ORBInitialPort", "9090", "-ORBInitialHost", "localhost"};
10      ORB orb = ORB.init(argsOrb, null);
11      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
12      rootpoa.the_POAManager().activate();
13
14      // b. create application objects:
15      OfferImpl offer_PC = new OfferImpl("PC Core 2 Quad 6600", 899.90);
16
17      // c. create references to them:
18      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(offer_PC);
19      System.out.println(orb.object_to_string(ref));
20
21      // d. wait for connections to the application objects:
22      orb.run();
23    } catch (Exception e) { System.err.println("ERROR: " + e); e.printStackTrace(System.out); }
24  }
25 }
```

Figure 8: OfferServer.java: Simple server.

Step 4: Client

The client application has to

1. connect to the ORB infrastructure,
   (a) create an `ORB` with a specific hostname and port,

2. retrieve references to the application objects,
   (a) by looking up CORBA objects (represented by **client stubs**) by their IOR and
   (b) casting them to the interfaces from step 1 using helper classes.

3. do something with the application object references,
   (a) using the interface from step 1.

Step 4: Client

```
1 package ismll_commerce;
2 import org.omg.CORBA.ORB;
3
4 public class OfferClient {
5   public static void main(String args[]) {
6     try{
7       // a. connect to the ORB infrastructure:
8       String[] argsOrb = new String[] { "-ORBInitialPort", "9090", "-ORBInitialHost", "localhost"};
9       ORB orb = ORB.init(argsOrb, null);
10
11      // b. retrieve application object by reference (here: args[0] command line):
12      String refString = args[0];
13      org.omg.CORBA.Object ref = orb.string_to_object(refString);
14      Offer offer_pc = OfferHelper.narrow(ref);
15
16      // c. do something with them:
17      System.out.println("Obtained a handle on server object: " + offer_pc);
18      System.out.println("name:  " + offer_pc.name());
19      System.out.println("price: " + offer_pc.price());
20    } catch (Exception e) { System.out.println("ERROR : " + e); e.printStackTrace(System.out); }
21  }
22 }
```

Figure 9: OfferClient.java: Simple client.

General Procedure

Example: offers.

1. **Interface:** create the interface description `Offer.idl` and derive the
   (a) Java interface `Offer.java`,
   (b) Java implementation base class `OfferPOA.java` and
   (c) Java helper class `OfferHelper.java`

   by running `idlj` on it.

2. **Implementation:** derive the implementation class `OfferImpl.java` from `OfferPOA.java` implementing the specified interface methods.

3. **Server:** implement a server application `OfferServer.java` that
   (a) connects to the ORB infrastructure,
   (b) creates application objects,
   (c) outputs references to them and
   (d) wait for connections to the application objects.

4. **Client:** implement a client application `OfferClient.java` that
   (a) connects to the ORB infrastructure,
   (b) retrieves application objects by references, and
   (c) does something with them.

Running the example

To run the server:

```
orbd -ORBInitialPort 9090 -ORBInitialHost localhost
java ismll_commerce.OfferServer
```

The offer server writes the reference to the PC offer object to the console, something like

```
IOR:000000000000001d49444c3a69736d6c6c5f636f6d6d6572
```

To run the client:

```
java ismll_commerce.OfferClient IOR:000000000000001d
```


Required Files

file	function	derived	server	client
Offer.idl	interface	–	–	–
OfferOperations.java	interface	+	+	+
Offer.java	interface	+	+	+
OfferHelper.java	helper	+	+	+
_OfferStub.java	client stub	+	+	+
OfferPOA.java	server skeleton	+	+	–
OfferImpl.java	implementation	–	+	–
OfferServer.java	server	–	+	–
OfferClient.java	client	–	–	+

Class Hierarchy

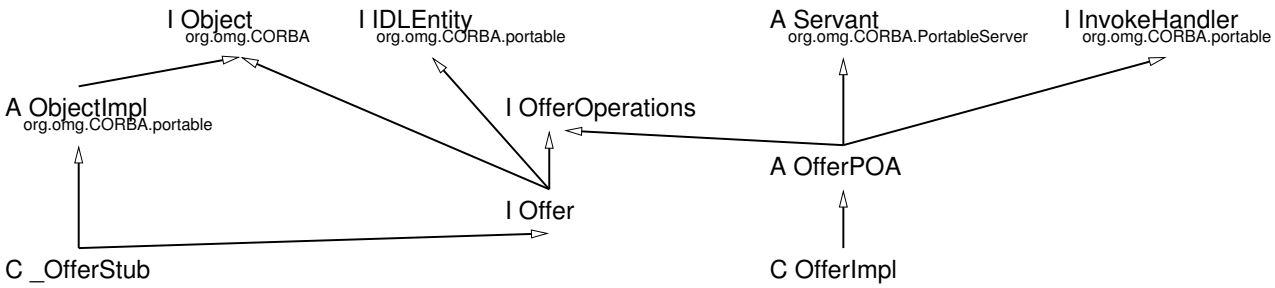


Figure 10: Class hierarchy for the offer example.

Get/Setter Methods

A pair of Get/Setter methods can be specified more easily by an attribute.

```
module test {
   interface AttributeTest {
      readonly attribute string name;
      attribute double price;
   };
};
```

Figure 11: attributes.idl: Alternative interface description for offers.

```
package test;


/**
* test/AttributeTestOperations.java .
* Generated by the IDL-to-Java compiler (portable), version "
* from attributes.idl
* Tuesday, June 3, 2008 8:02:59 AM CEST
*/

public interface AttributeTestOperations
{
  String name ();
  double price ();
  void price (double newPrice);
} // interface AttributeTestOperations
```

Figure 12: AttributeTestOperations.java: derived Java interface.

1. Introduction

2. Offering and Using Remote Objects

3. Publishing and Requesting Objects by Names

Name Services

In practice, using IORs may be too inflexible.

Name Services can be used instead:

- Each ORB allows access to a name service, an object of class `NamingContextExt` by the initial reference `NameService`.

- The name service object allows to
  1. bind names to object references (`bind`, `rebind`) and
  2. resolve names to object references (`resolve`)
  3. names are represented as sequences of `NameComponent`s – for us: strings; “paths”.

```
1 package ismll_commerce;
2 import org.omg.CORBA.ORB;
3 import org.omg.PortableServer.*; // POA, POAHelper
4 import org.omg.CosNaming.*;
5
6 public class OfferServer_NS {
7     public static void main(String args[]) {
8         try{
9             // a. connect to ORB infrastructure:
10            String[] argsOrb = new String[] { "-ORBInitialPort", "9090", "-ORBInitialHost", "localhost"};
11            ORB orb = ORB.init(argsOrb, null);
12            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
13            rootpoa.the_POAManager().activate();
14            org.omg.CORBA.Object nsObj = orb.resolve_initial_references("NameService");
15            NamingContextExt ns = NamingContextExtHelper.narrow(nsObj);
16
17            // b. create application objects:
18            OfferImpl offer_PC = new OfferImpl("PC Core 2 Quad 6600", 899.90);
19
20            // c. bind application objects to names:
21            org.omg.CORBA.Object refObj = rootpoa.servant_to_reference(offer_PC);
22            Offer ref = OfferHelper.narrow(refObj);
23            NameComponent path[] = ns.to_name(offer_PC.name());
24            ns.rebind(path, ref);
25
26            // d. wait for connections to the application objects:
27            orb.run();
28        } catch (Exception e) { System.err.println("ERROR: " + e); e.printStackTrace(System.out); }
```



```
29    }
30 }
```

Figure 13: Simple server with name service.

Client with Name Service

```
1 package ismll_commerce;
2 import org.omg.CORBA.ORB;
3 import org.omg.CosNaming.*;
4
5 public class OfferClient_NS {
6    public static void main(String args[]) {
7       try{
8          // a. connect to the ORB infrastructure:
9          String[] argsOrb = new String[] { "-ORBInitialPort", "9090", "-ORBInitialHost", "localhost"};
10         ORB orb = ORB.init(argsOrb, null);
11         org.omg.CORBA.Object nsObj = orb.resolve_initial_references("NameService");
12         NamingContextExt ns = NamingContextExtHelper.narrow(nsObj);
13
14         // b. retrieve application object by name:
15         String name = args[0];
16         org.omg.CORBA.Object ref = ns.resolve_str(name);
17         Offer offer_pc = OfferHelper.narrow(ref);
18
19         // c. do something with them:
20         System.out.println("Obtained a handle on server object: " + offer_pc);
21         System.out.println("name:  " + offer_pc.name());
22         System.out.println("price: " + offer_pc.price());
23      } catch (Exception e) { System.out.println("ERROR : " + e); e.printStackTrace(System.out); }
24   }
25 }
```

Figure 14: Simple client with name service.

Running the example

To run the server:

```
orbd -ORBInitialPort 9090 -ORBInitialHost localhost
java ismll_commerce.OfferServer_NS
```

To run the client:

```
java ismll_commerce.OfferClient_NS "PC Core 2 Quad 6600"
```

Browsing the Name Service

```
1 import org.omg.CORBA.*;
2 import org.omg.CosNaming.*;
3
4 public class NamespaceBrowser {
5    public static void main(String args[]) {
6       try {
7          // a. connect to the ORB infrastructure:
8          String[] argsOrb = new String[] { "-ORBInitialPort", "9090", "-ORBInitialHost", "localhost"};
9          ORB orb = ORB.init(argsOrb, null);
10         NamingContextExt ns = NamingContextExtHelper.narrow(
11                         orb.resolve_initial_references("NameService"));
12         // b. get bindings and print them:
13         BindingListHolder bl = new BindingListHolder();
14         BindingIteratorHolder blIt = new BindingIteratorHolder();
15         ns.list(1000, bl, blIt);
16         Binding[] bindings = bl.value;
17         for (int i = 0; i < bindings.length; i++) {
18            System.out.print(bindings[i].binding_type == BindingType.ncontext ? "Context: " : "Object: ");
19            System.out.print(bindings[i].binding_name[0].id);
20            for (int j = 1; j < bindings[i].binding_name.length; j++)
21               System.out.print(" / " + bindings[i].binding_name[j].id);
22            System.out.println();
23         }
24      } catch (Exception e) { System.out.println("ERROR : " + e); e.printStackTrace(System.out); }
25   }
26 }
```

Figure 15: Simple nameservice browser.

Summary

- CORBA allows programs on different computers, written in different languages to communicate.
- Services are described by interface description in a specific language, the interface description language IDL.
- Programming language-specific interfaces are derived from the IDL descriptions automatically.
- Implementations based on generated servant base classes may contain no CORBA specific code.
- To allow clients to locate objects, name services are available.
- The name service itself is a CORBA object; for bootstrapping, initial references by standard names (“NameService”) are available.

References

[AKS05] Markus Aleksy, Axel Korthaus, and Martin Schader. *Implementing Distributed Systems with Java and CORBA*. Springer, 2005.

[McH07] Ciaran McHale. *CORBA Explained Simply*. http://www.ciaranmchale.com/corba-explained-simply/, 2007.