Topic 3
Distributed Objects

Part A

Overview

- Early Distribution – RPC
- CORBA – Common Object Request Broker Architecture
- RMI – Remote Method Invocation

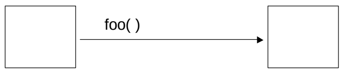
Distributed Computing

- The search for ways to unify multiple networked machines so that they can
  - share information
  - share resources
- Driving force:
  - workstations and local area networks
- But, progress has been slow

Difficulties in distributed computing

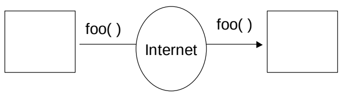
- Heterogeneous environments
  - different operating systems, languages
- Network reliability
  - life is easier on a single machine

The Goal of Distributed Object Computing



Objects in a single address space

The Goal of Distributed Object Computing



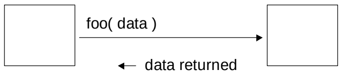
talk to objects
in a different address space
as if the object is local

Remote Procedure Call (RPC)

- Available in pre-Java era
- Allows a procedure call to be made from one machine to another
- To the programmer it looks like a local call
- RPC requires programmers to register their programs with Port Mapper

Remote Procedure Call

- Allows a thread of control in one process to call a function in another process – perhaps on another machine



RPC

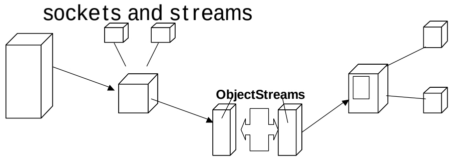
```
if (x > 3)
   foo ();      Lives on local machine
else
   bar();       Lives on remote machine –
                193.164.1.20 on port 2345
```

Vocabulary

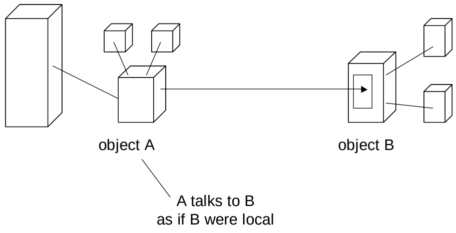
- remote object
  - an object that can be called from another machine
  - implements a remote interface
  - also called a server
- client
  - an object that talks to a remote object; the call can come from an applet or application

Programming with Sockets and Streams

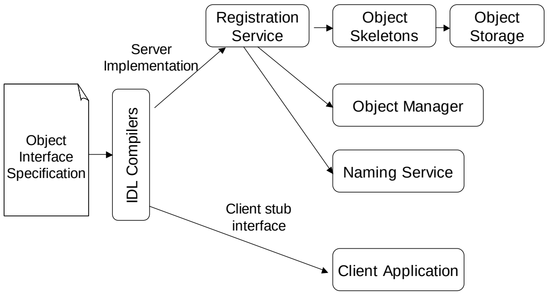
- PRO
  - Efficient, programmer is in control
- CON
  - Programmer must be in control
  - Some object must “know” about the sockets and streams



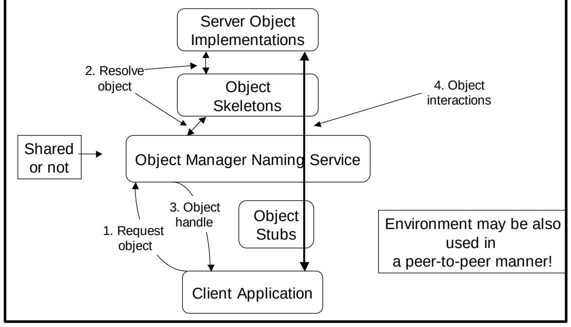
Ideal World (no Sockets or Streams)



General architecture for distributed object systems



Remote object transactions at runtime



Environment may be also used in a peer-to-peer manner!

Object Interface Specification

- Consider a truly open system for distributing objects:
  - clients should be able to access regardless of their impl. details
    - hardware platform, software language
  - server should be able to implement an object in whatever way it needs to
    - option of wrapping existing services with object interfaces

- Platform-independent means for specifying object interfaces:
  - Interface Definition Language (IDL) in CORBA
  - Interface Specification Language (ISL) in Xerox's Inter-Language Unification (ILU) system
  - Component Model Language (COM) in Microsoft's DCOM system

Object manager

- manages the object skeletons and object references on an object server
- Its role (object creation, call/result routing, destruction) is similar to
  - CORBA's Object Request Broker (ORB)
  - RMI's registry system

- Further roles:
  - dynamic object activation/deactivation
    - via corresponding registered methods
  - persistent objects
    - via a method for storing/retrieving state after de/activation
- Where to put object manager?

Registration/Naming Service

- Implementation of an interface needs to be registered so that it can be addressed by clients
  - routes clients' requests/method invocations to proper object server
  - helps OM in supporting object de/activation, and persistent objects

Object Communication Protocol

- A general protocol for handling remote object requests
  - a means for transmitting and receiving object references, method references, and data in the form of objects of basic data types.

Development Tools

- Object i/f editors
- Project managers
- Language cross-compilers
- Symbolic debuggers
- Tools for monitoring and diagnosing object systems
- Load simulation and testing tools

Security

- Agents making requests of the object broker
  - authentication, authorization, access control
- Transactions between agents and the remote objects
  - encryption

Distributed object schemes for Java

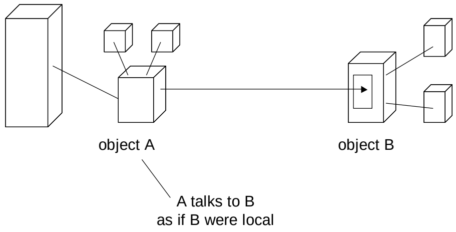
- To be explained using an Example involving a generic problem solver, which we will distributed using both CORBA and RMI
  - **`Solver`**: acts as a generic computing engine that solves numerical problems
  - **`ProblemSet`**: holds all information describing a problem and fields for solution

```
package dcj.examples;
import java.io.OutputStream;
public interface Solver {
  // Solve the current problem set
  public boolean solve();

  // Solve the given problem set
  public boolean solve(ProblemSet s, int numIters);

  // Get/set the current problem set
  public ProblemSet getProblem();
  public void setProblem(ProblemSet s);

  // Get/set the current iteration setting
  public int getIterations();
  public void setIterations(int numIter);

  // Print solution results to the output stream
  public void printResults(OutputStream os);
}
```

A Problem Set Class

```
package dcj.examples;

public class ProblemSet {
  protected double value = 47.0;
  protected double solution = -1.0;

  public double getValue() { return value; }
  public double getSolution() { return solution; }
  public void setValue(double v) { value = v; }
  public void setSolution(double s) { solution = s; }
}
```

CORBA (Common Object Request Broker Adapter)



CORBA

- Based on a consortium of over 700 companies called the Object Management Group (OMG)
  - except Microsoft which has its own Distributed Component Object Model (DCOM)
- Designed to allow components to find and talk to each other on an Object BUS

CORBA

- 1991 – Specification for object interaction
  - based on IDL – Interface Definition Language
- 1994 – CORBA 2.0
  - defined interoperability between objects in heterogeneous systems
- IIOP – Internet Inter-ORB Protocol
  - for interoperability over the Internet

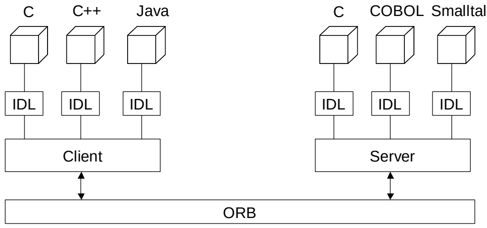
CORBA

- meant to be platform- and language-independent
  - client stub interfaces to the objects
  - the server implementations of these object interfaces

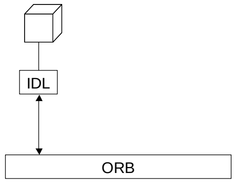
    can be specified in any programming language

Elements of CORBA framework

- An Object Request Broker (ORB)
  - means to make/receive requests
- Methods for specifying interfaces that objects in the system support
  - IDL (static) and DII (dynamic)
- Inter-ORB Protocol (IIOP)
  - a binary protocol for communication between ORBs

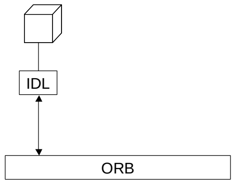


IDL



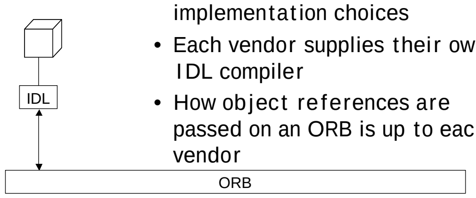
- Interface Definition Language
- The CORBA "glue"
- Language independent interfaces to the ORB (the BUS)

ORB



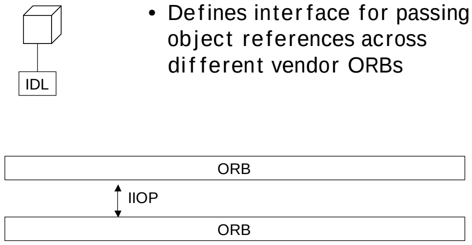
- The object "BUS" = middleware
- Allows objects make requests to – and receive responses from other objects on the bus

CORBA's ORB is an interface specification



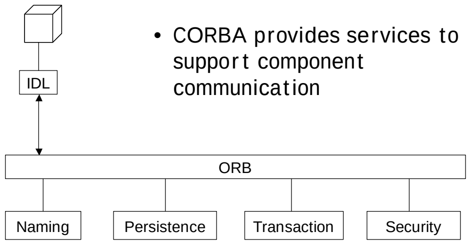
- Different vendor ORBs may make very different implementation choices
- Each vendor supplies their own IDL compiler
- How object references are passed on an ORB is up to each vendor

IIOP
Internet Inter-ORB Protocol



- Defines interface for passing object references across different vendor ORBs

CORBA Services



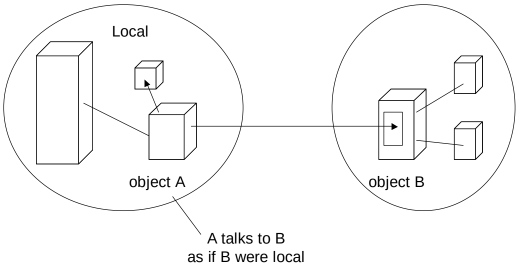
- CORBA provides services to support component communication

CORBA ORB Vendors

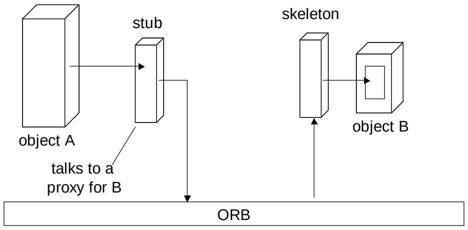
- Visigenic
- Iona
- Inprise

CORBA Development

Ideal World (No Sockets of Streams)

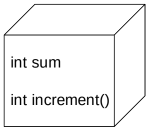


CORBA World



Example: Remote Object Count

```
int sum;

int increment()
// increments and returns sum
```



need to define an IDL interface

IDL Types vs Java Types

IDL Type	Java Mapping
• long	• int
• short	• short
• float	• float
• double	• double
• char	• char
• boolean	• boolean
• octet	• byte

Count IDL

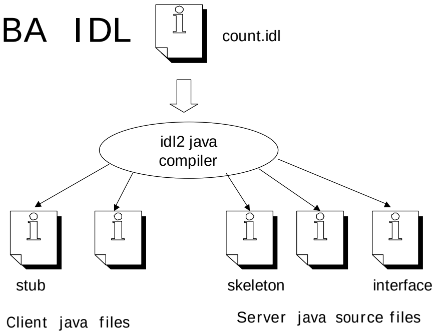
```
module Counter {
  interface Count {
    attribute long sum;
    long increment()
  };
};
```

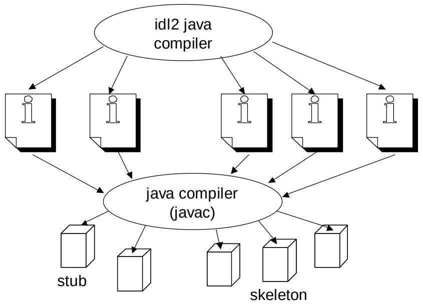
Uses IDL datatypes

```
module DCJ {
  module examples {
      interface ProblemSet {
           double getValue();
           double getSolution();
           void setValue(in double v);
           void setSolution(in double s);
      };
      interface Solver {
           boolean solveCurrent();
           boolean solve(inout ProblemSet s,
                         in long numIters);
           ProblemSet getProblem();
           void setProblem(inout ProblemSet s);
           unsigned long getIterations();
           void setIterations(in unsigned long numIter);
      };
  };
};
```

Uses IDL datatypes
No constructors
No method overloading

CORBA IDL





Java Interface (generated by idl2.java)

```
public interface Count extends CORBA.Object
{
  public int sum() throws CORBA.SystemException;
  public void sum(int val) throws CORBA.SystemException;
  public int increment() throws CORBA.SystemException;
}
```

idl2java JavaIDL

modules converted to packages
inout method args -> holder types

```
package DCJ.examples;
public interface Solver extends org.omg.CORBA.Object
{
    boolean solveCurrent();
    boolean solve(DCJ.examples.ProblemSetHolder s,
                  int numIters);
    DCJ.examples.ProblemSet getProblem();
    void setProblem(DCJ.examples.ProblemSetHolder p);
    int getNumIterations();
    void setNumIterations(int i);
}
```

The holder classes act as streamable versions of the main class;
ORB uses these to xmit instances of the i/f as remote method args

Client stubs

- The compiler also generates client stubs for interfaces in IDL that implements the Java base class for the object:

```
public class _SolverStub
 extends org.omg.CORBA.portable.ObjectImpl
 implements dcj.examples.Solver {
 . . .
```

ObjectImpl class provides the i/f used by client ORB to un/marshal remote method args.

Server skeleton

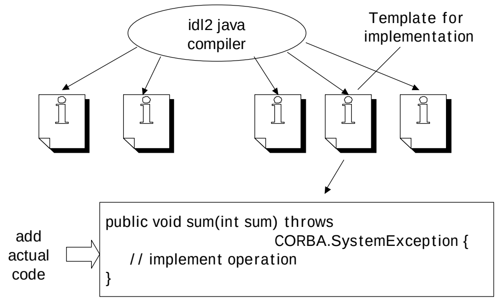
- The compiler also generates a skeleton for object implementation:

_ProblemSetImplBase class is also generated

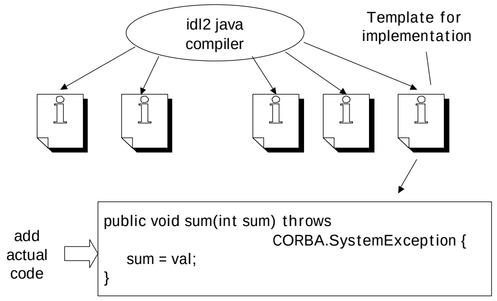
```
public abstract class _SolverImplBase
 extends org.omg.CORBA.portable.ObjectImpl
 implements dcj.examples.Solver
 implements org.omg.CORBA.portable.Skeleton{
 . . .
```

The server ORB will be looking for Skeleton interface when invoking methods on the object implementation

- The last step in setting up our remote object for business is:
  - to extend the _SolverImplBase class and the _ProblemSetImplBase class
  - and to implement the methods defined in their base interfaces.



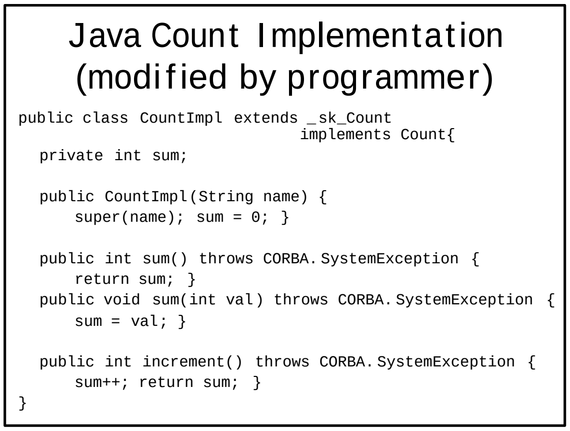
```
public void sum(int sum) throws
                              CORBA.SystemException {
    // implement operation
}
```



```
public void sum(int sum) throws
                              CORBA.SystemException {
    sum = val;
}
```

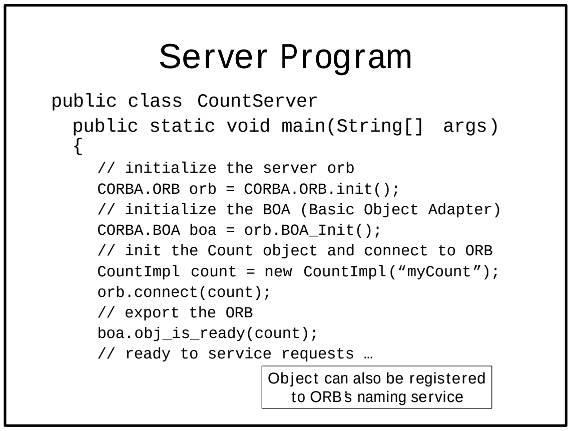
Java Count Implementation (generated by idl2java)

```
public class CountImpl extends _sk_Count
                             implements Count{
   private int sum;

   public CountImpl(String name) {
        super(name); }

   public int sum() throws CORBA.SystemException {
        //implement attribute reader }
   public void sum(int val) throws CORBA.SystemException {
        // implement attribute writer }

   public int increment() throws CORBA.SystemException {
        // implement operation }
}
```



Java Count Implementation (modified by programmer)

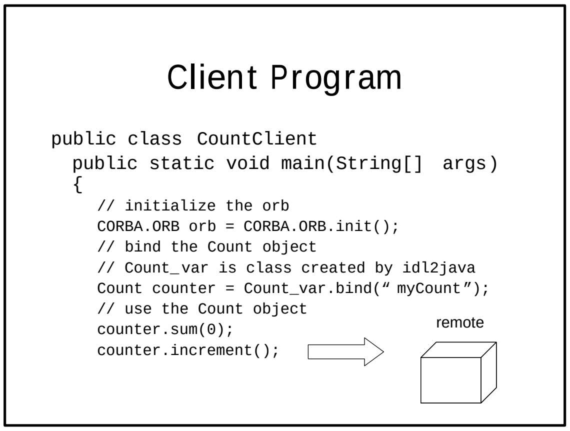
```
public class CountImpl extends _sk_Count
                              implements Count{
  private int sum;

  public CountImpl(String name) {
      super(name); sum = 0; }

  public int sum() throws CORBA.SystemException {
      return sum; }
  public void sum(int val) throws CORBA.SystemException {
      sum = val; }

  public int increment() throws CORBA.SystemException {
      sum++; return sum; }
}
```



Server Program

```
public class CountServer
  public static void main(String[] args)
  {
    // initialize the server orb
    CORBA.ORB orb = CORBA.ORB.init();
    // initialize the BOA (Basic Object Adapter)
    CORBA.BOA boa = orb.BOA_Init();
    // init the Count object and connect to ORB
    CountImpl count = new CountImpl(“myCount”);
    orb.connect(count);
    // export the ORB
    boa.obj_is_ready(count);
    // ready to service requests …
```

Object can also be registered to ORB’s naming service



Client Program

```
public class CountClient
  public static void main(String[] args)
  {
    // initialize the orb
    CORBA.ORB orb = CORBA.ORB.init();
    // bind the Count object
    // Count_var is class created by idl2java
    Count counter = Count_var.bind(“ myCount”);
    // use the Count object
    counter.sum(0);
    counter.increment();
```