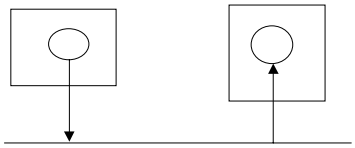
Distributed Programming

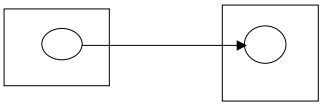
•low level: sending data among distributed computations



•network is visible (to the programmer)

•programmer must deal with many details

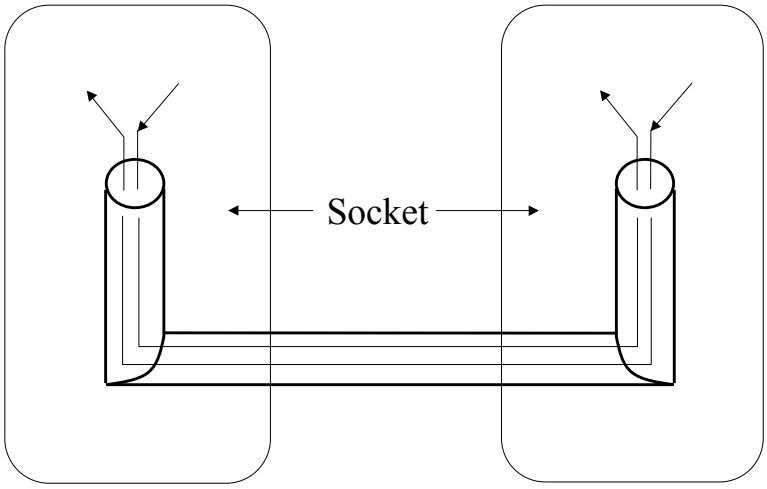
•higher level: supporting invocations among distributed computations



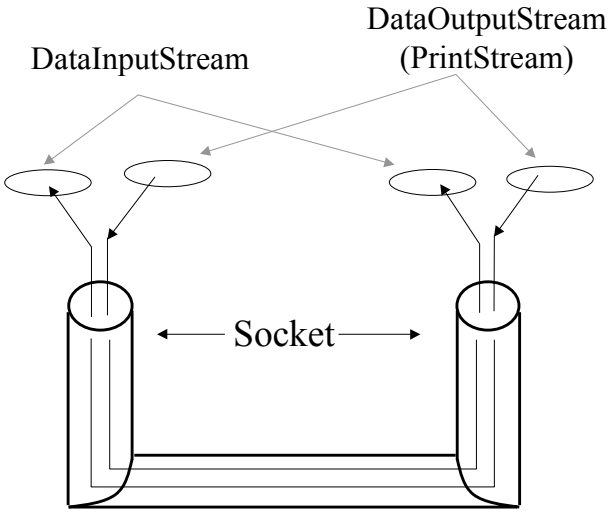
•network is invisible (to the programmer)

•programmer focuses on application

Distributed Data Communication



Distributed Data Communication Java Classes



Basic Socket Usage

client

```
// Establish Socket Connection

Socket cs;
int portno = 5678;


cs = new Socket(``server", portno);


//Establish Data Streams
clin = new DataInputStream(
        cs.getInputStream());


clout = new PrintStream(
        cs.getOutputStream());
```

server

```
// Establish Socket Connection
ServerSocket ss;
Socket sin;
int portno = 5678;
ss = new ServerSocket(portno);
Socket sin = s.accept();

// Establish Data Streams
srout = new PrintStream(
        sin.getOutputStream());

srin = new DataInputStream)
        sin.getInputStream());
```

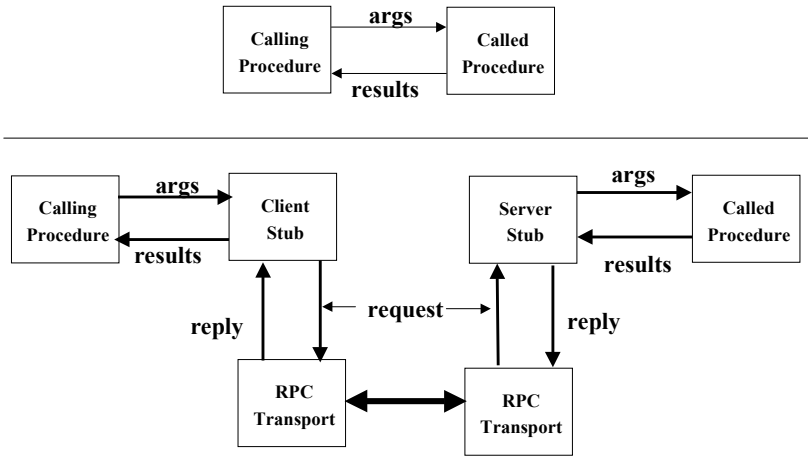
Client Side Code

```
class SocketTest
{
        public static void main( String[] args)
        {       try
                {       Socket t = new Socket(``java.sun.com", 13);
                        DataInputStream is =
                                new DataInputStream(t.getInputStream());
                        boolean more = true;
                        while( more )
                        {       String str = is.readLine();
                                if (str == null) more = false;
                                else
                                        System.out.println(str);
                        }
                }
                catch (IOException e) { System.out.println(``Error" + e); }
        }
}
```


Server Side Code

```
class EchoServer
{       public static void main( String[] args)
        {       try
                {       ServerSocket s = new ServerSocket(8189);
                        Socket = incoming = s.accept();
                        DataInputStream in =
                                new DataInputStream(incoming.getInputStream());
                        PrintStream out =
                                new PrintStream(incoming.getOutputStream());
                        System.out.println( ``Hello! Enter BYE to exit. \r");
                        boolean done = false;
                        while (!done)
                        {       String str = in.readLine();
                                if (str == null)done = true;
                                else
                                {       out.println(``Echo: `` + str + ``\r");
                                        if (str.trim().equals(``BYE"))
                                        done = true;
                                }
                        }
                        incoming.close();
                }
                catch (Exception e) { System.out.println(e); }
        }
}
```

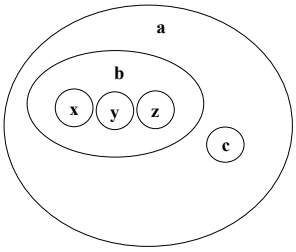
Remote Procedure Call



Remote Procedure Call Issues

- generating stubs
- serialization or arguments and return values
- heterogeneity of data representations
- locating servers in a distributed environment
- authentication of called and calling procedures
- semantics of invocation
  (at-most-once, at-least-once)

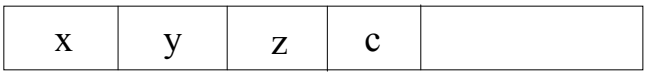
Serialization



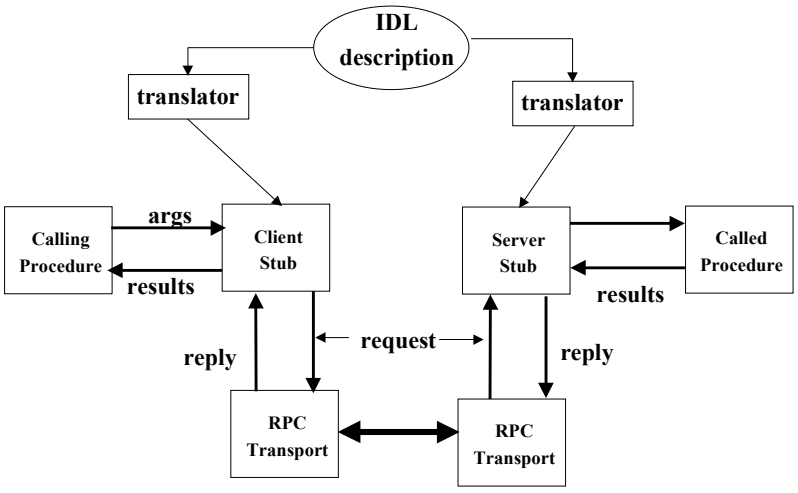
Issues:

- how to represent base types (i.e. int)
- how to represent structured types (arrays)
- how to deal with references (pointers)
- how to treat duplicated objects

transforming a typed, highly structured object into a stream of bytes.



Interface Definition Language



Simple IDL Example

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

From: Ole Arthur Bernsen

IDL Elements

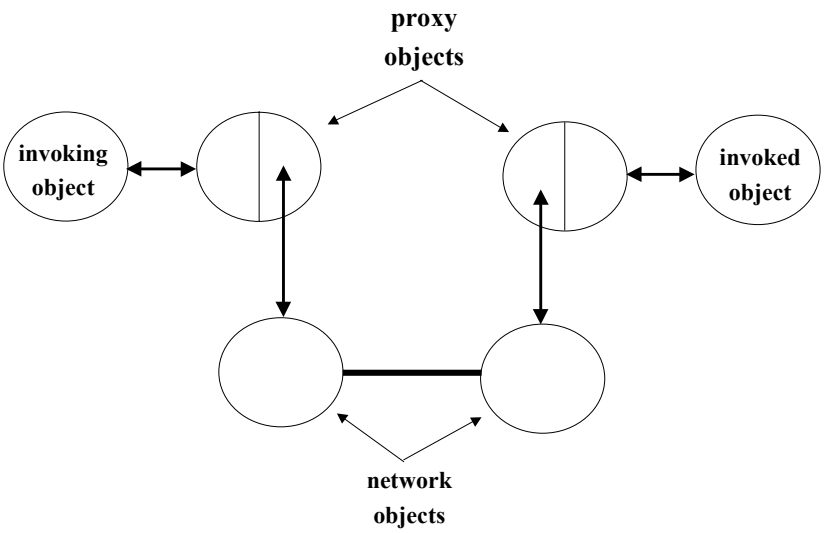
```
module modulename {
  exception exceptionName { [type pname]* };
  typedef type newtype;

  interface newInterface {
   oneway type fname(in type pname1);
   attribute newtype;
  };

  interface newInterface2 : newInterface {
   type fname2 (out newInterface pname3) raises exceptionName;
  };
};
```

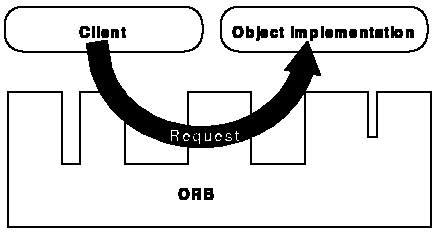
From: Ole Arthur Bernsen

Remote Object Systems



Corba

Goal: interoperability among application components
- •written in different programming languages
- •executing on heterogeneous architectures
- •communicating over different networks.

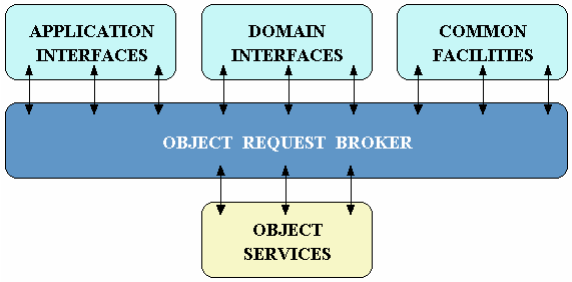


Corba: Common Object Request Broker Architecture

ORB: Object Request Broker

From: Object Mangagement Group

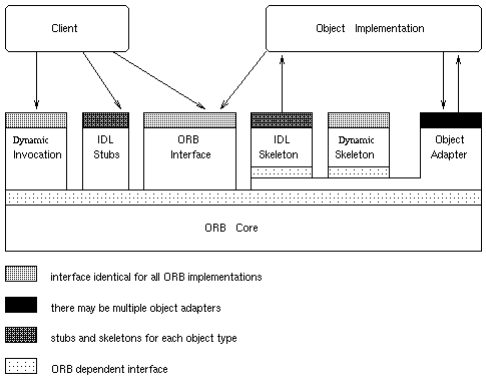
Role of the Object Request Broker



•**Application interfaces**: interfaces for a specific application
•**Domain interfaces**: interfaces shared across applications in a given application domain (publishing)
•**Common Facilities**: generic services that might be needed in several domains (document structure)
•**Object Services**: commonly needed across all applications (e.g., naming, trading)

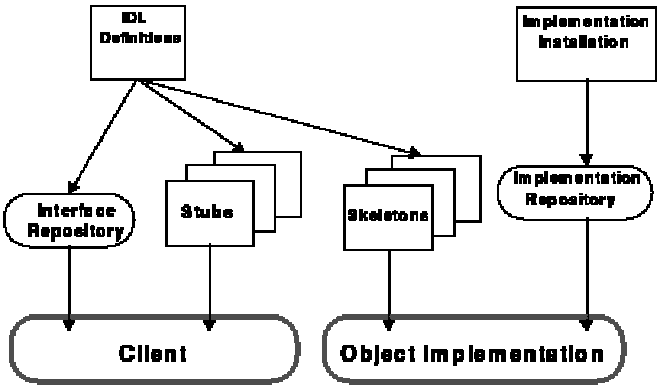
From Doug Schmidt

Elements of Corba



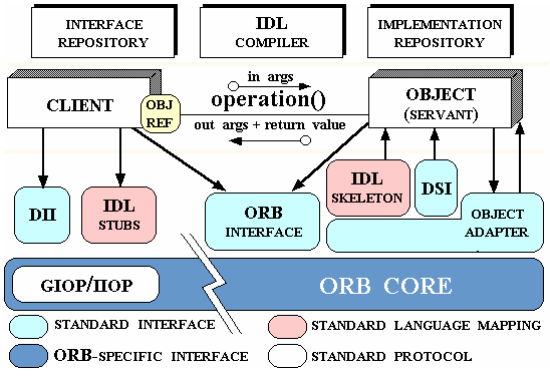
From: Kate Keahey (kksiazek@cs.indiana.edu)

Role of IDL in Corba



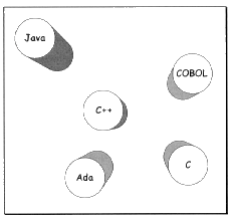
From: Object Mangagement Group

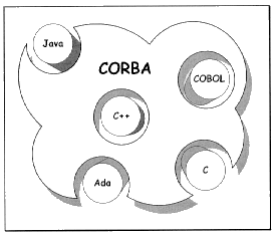
Elements of Corba



From Doug Schmidt

Corba and Java





Corba is still needed to fill in the gaps between Java and system developed in other languages.