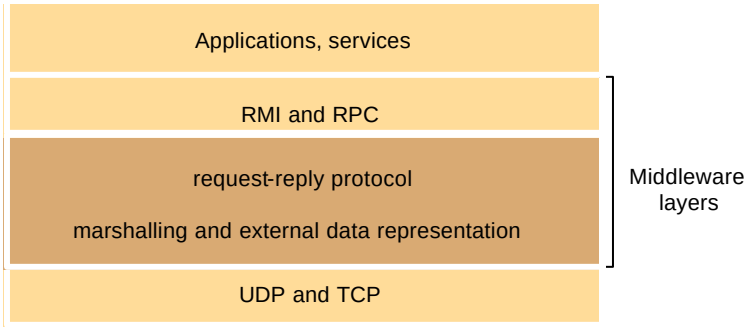
RPC & RMI

Concurrent & Distributed Software

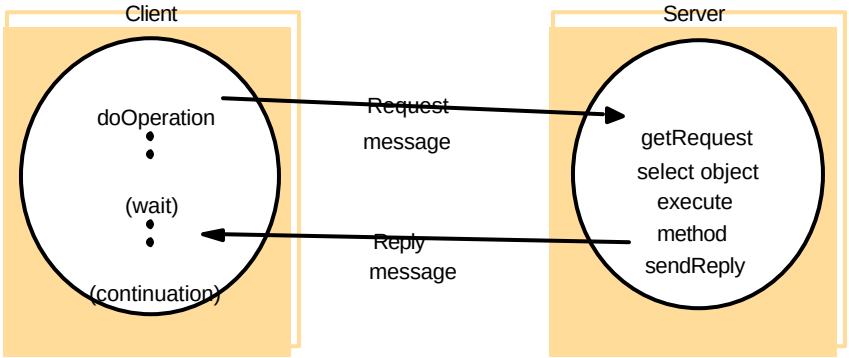
Motivation

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
    - No longer considered a good thing
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

Middleware layers



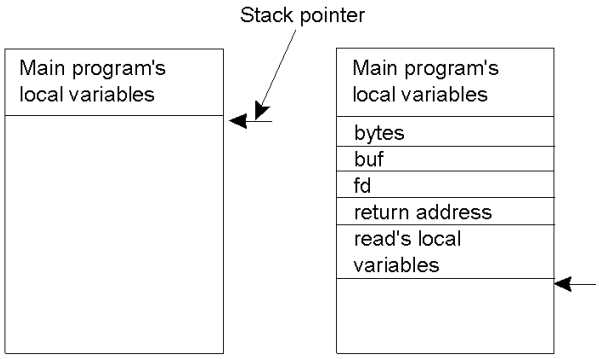
Request-reply communication



Conventional Procedure Call

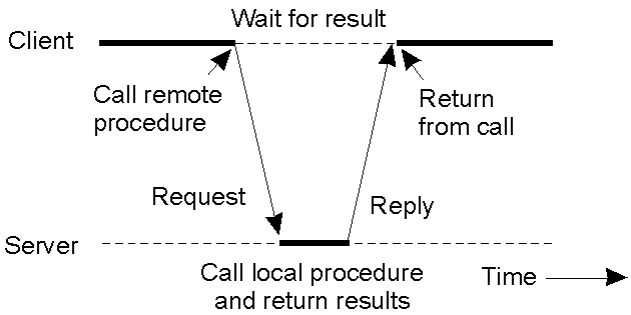
a) Parameter passing in a local procedure call: the stack before the call to read(fd,buf,bytes)

b) The stack while the called procedure is active



Remote Procedure Call

- Principle of RPC between a client and server program.



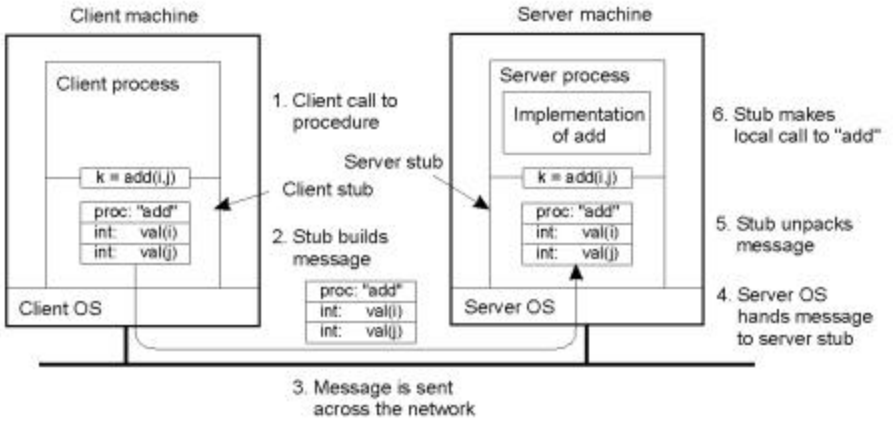
Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and *marshalls* the parameters.
- The server-side stub receives this message, unpacks the marshalled parameters, and peforms the procedure on the server.

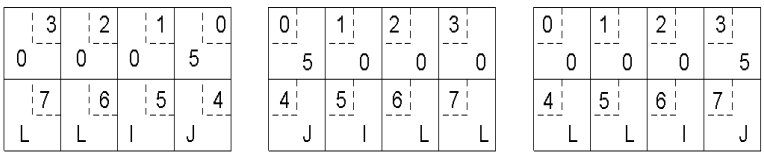
Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

Passing Value Parameters (1)



Passing Value Parameters (2)



(a) (b) (c)

a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

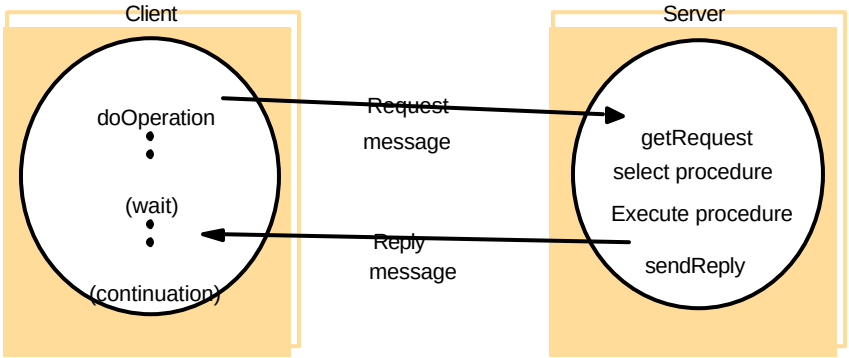
```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)

foobar's local variables	
	x
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

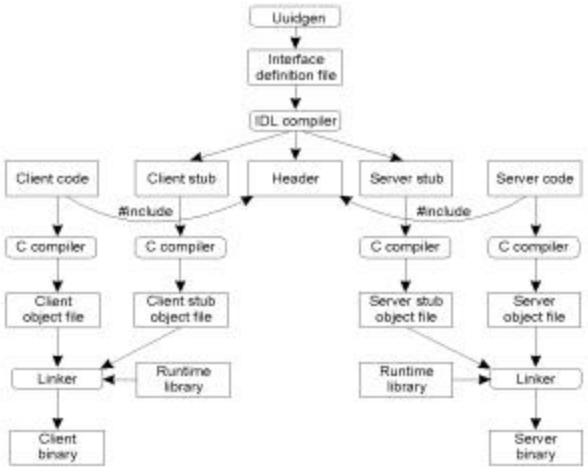
(b)

Request-reply communication



Writing a Client and a Server

The steps in writing a client and a server in DCE RPC (SUN RPC is similar)



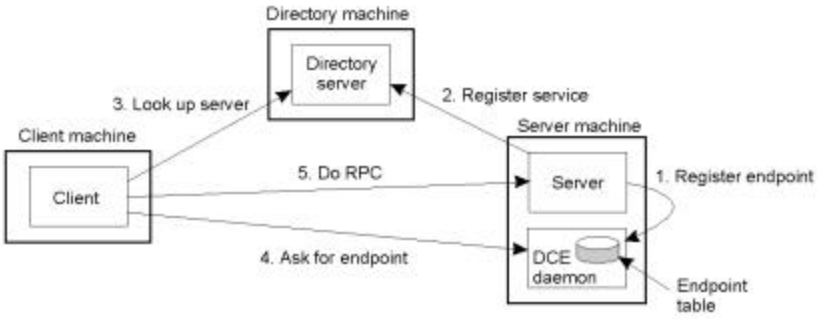
Files interface in Sun XDR

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
  version VERSION {
    void WRITE(writeargs)=1;      1
    Data READ(readargs)=2;        2
  }=2;
} = 9999;
```

Binding a Client to a Server



RMI

- RMI = RPC + Object-orientation
  - Java RMI
  - CORBA
    - Middleware that is language-independent
  - Microsoft DCOM/COM+
  - SOAP
    - RMI on top of HTTP

Interfaces in distributed systems

- Programs organized as a set of modules that communicate with one another via procedure calls/method invocations
- Explicit interfaces defined for each module in order to control interactions between modules
- In distributed systems, modules can be in different processes
- A remote interface specifies the methods of an object that are available for invocation by objects in other processes defining the types of the input and output arguments of each of them

CORBA IDL example

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

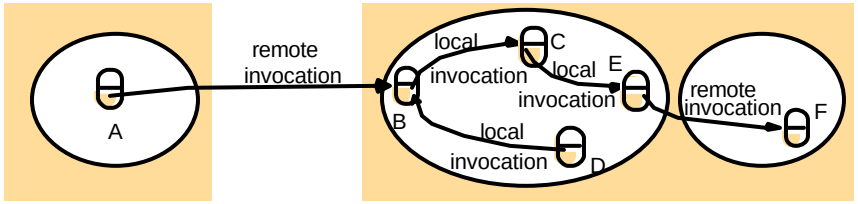
Object model

- Object references
  - Objects accessed via object references
  - Object references can be assigned to variables, passed as arguments and returned as results
- Interfaces
  - Provides a signature of a set of methods (types of arguments, return values and exceptions) without specifying their implementations
- Actions (invocations)
- Exceptions
- Garbage Collection

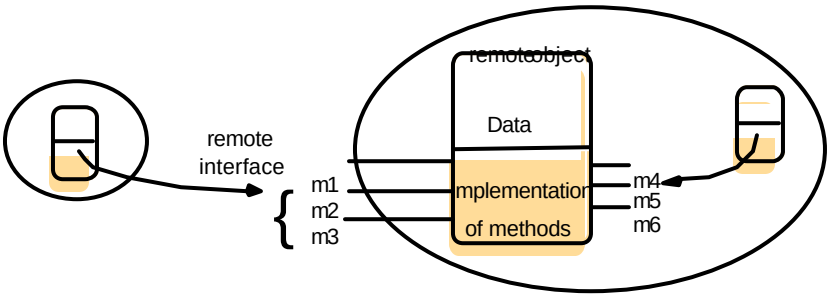
Distributed Objects

- Remote object references
  - An identifier that can be used throughout a distributed system to refer to a particular remote object
- Remote interfaces
  - CORBA provides an interface definition language (IDL) for specifying a remote interface
  - JAVA RMI: Java interface that extends *Remote* interface
- Actions: remote invocations
- Remote Exceptions may arise for reasons such as partial failure or message loss
- Distributed Garbage Collection: cooperation between local garbage collectors needed

Remote and local method invocations



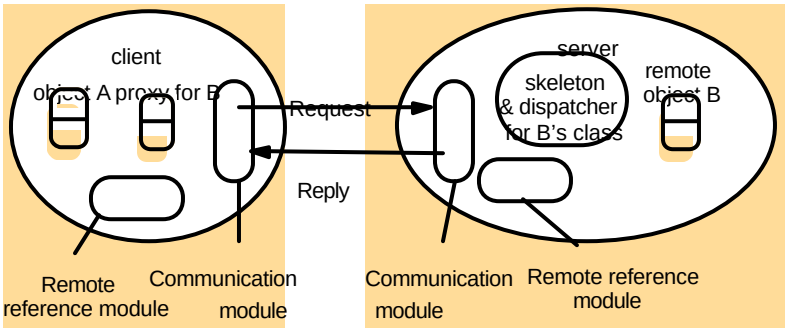
A remote object and its remote interface



RMI Programming

- RMI software
  - Generated by IDL compiler
  - Proxy
    - Behaves like remote object to clients (invoker)
    - Marshals arguments, forwards message to remote object, unmarshals results, returns results to client
  - Skeleton
    - Server side stub;
    - Unmarshals arguments, invokes method, marshals results and sends to sending proxy's method
  - Dispatcher
    - Receives the request message from communication module, passes on the message to the appropriate method in the skeleton
- Server and Client programs

The role of proxy and skeleton in remote method invocation



RMI Programming

- Binder
  - Client programs need a means of obtaining a remote object reference
  - Binder is a service that maintains a mapping from textual names to remote object references
  - Servers need to register the services they are exporting with the binder
  - Java RMI registry, CORBA Naming service
- Server threads
  - Several choices: thread per object, thread per invocation
  - Remote method invocations must allow for concurrent execution

RMI systems

- CORBA – language independent
- DCOM - Microsoft
- Java RMI
- SOAP (Simple Object Access Protocol)
  - HTTP is request-reply protocol
  - XML for data representation

Java RMI

- Features
  - Integrated with Java language + libraries
    - Security, write once run anywhere, multithreaded
    - Object orientation
  - Can pass "behavior"
    - Mobile code
    - Not possible in CORBA, traditional RPC systems
  - Distributed Garbage Collection
  - *Remoteness of objects **intentionally** not transparent*

Remote Interfaces, Objects, and Methods

- Objects become remote by implementing a remote interface
  - A remote interface extends the interface java.rmi.Remote
  - Each method of the interface declares java.rmi.RemoteException in its `throws` clause in addition to any application-specific clauses

Creating distributed applications using RMI

1. Define the remote interfaces
2. Implement the remote objects
3. Implement the client (can be done anytime after remote interfaces have been defined)
4. Register the remote object in the name server registry
5. Generate the stub and client using *rmic*
6. Start the registry
7. Start the server
8. Run the client

Java Remote interfaces *Shape* and *ShapeList*

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject  getAllState() throws RemoteException;          1
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException; 2
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

The *Naming* class of Java RMI registry

void rebind (String name, Remote obj)
This method is used by a server to register the identifier of a remote object by name, as shown in Figure 15.13, line 3.

void bind (String name, Remote obj)
This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)
This method removes a binding.

Remote lookup(String name)
This method is used by clients to look up a remote object by name, as shown in Figure 15.15 line 1. A remote object reference is returned.

String [] list()
This method returns an array of Strings containing the names bound in the registry.

Java class *ShapeListServer* with *main* method

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            ShapeList aShapeList = new ShapeListServant();          1
                Naming.rebind("Shape List", aShapeList );           2
            System.out.println("ShapeList server ready");
            }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
    }
}
```

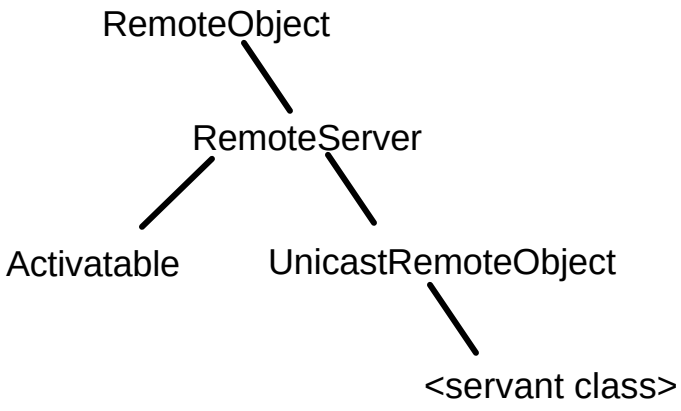
Java class *ShapeListServant* implements interface *ShapeList*

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList {
     private Vector theList;          // contains the list of Shapes          1
     private int version;
    public ShapeListServant()throws RemoteException{...}
    public Shape newShape(GraphicalObject g) throws RemoteException {        2
        version++;
            Shape s = new ShapeServant( g, version);                          3
            theList.addElement(s);
            return s;
    }
    public  Vector allShapes()throws RemoteException{...}
    public int getVersion() throws RemoteException { ... }
}
```


Java client of *ShapeList*

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
   public static void main(String args[]){
     System.setSecurityManager(new RMISecurityManager());
     ShapeList aShapeList = null;
     try{
          aShapeList  = (ShapeList) Naming.lookup("//bruno.ShapeList") ;      1
          Vector sList = aShapeList.allShapes();                              2
     } catch(RemoteException e) {System.out.println(e.getMessage());
     }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
   }
}
```

Classes supporting Java RMI

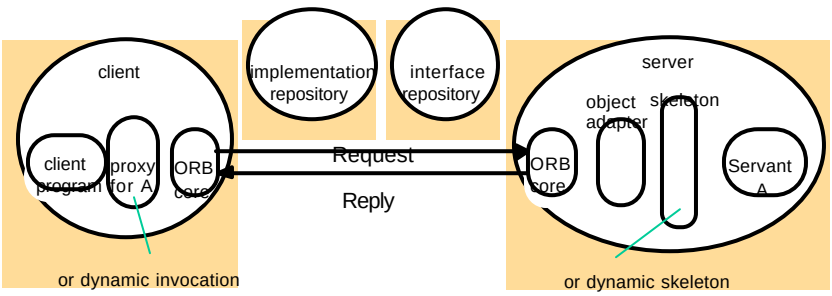


Advanced Techniques

- Passing behavior
  - See Java RMI tutorial track example
- Callbacks
- Activation

CORBA

The main components of the CORBA architecture



IDL interfaces Shape and ShapeList

```
struct Rectangle{                                   1
    long width;
    long height;
    long x;
    long y;
} ;

struct GraphicalObject{                             2
    string type;
    Rectangle enclosing;
    boolean isFilled;
};

interface Shape {                                                          3
    long getVersion() ;
    GraphicalObject getAllState() ;          // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;                                         4
interface ShapeList {                                                      5
    exception FullException{ };                                            6
    Shape newShape(in GraphicalObject g) raises (FullException);           7
    All allShapes();                         // returns sequence of remote object references  8
    long getVersion() ;
};
```


Java interface *ShapeList* generated by *idltojava* from CORBA interface *ShapeList*

```
public interface ShapeList extends org.omg.CORBA.Object {
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;
    Shape[] allShapes();
    int getVersion();
}
```

ShapeListServant class of the Java server program for CORBA interface *ShapeList*

```
import org.omg.CORBA.*;
class ShapeListServant extends _ShapeListImplBase {
    ORB theOrb;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(ORB orb){
        theOrb  = orb;
        // initialize the other instance variables
    }
    public Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException {   1
        version++;
        Shape s = new ShapeServant( g, version);
        if(n >=100) throw new ShapeListPackage.FullException();
        theList[n++] = s;                                                              2
            theOrb.connect(s);
            return s;
    }
    public  Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}
```


Java class *ShapeListServer*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListServer {
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);                                  1
            ShapeListServant shapeRef = new ShapeListServant(orb);           2
            orb.connect(shapeRef);                                           3
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");               4
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", "");           5
            NameComponent path[] = {nc};                                     6
            ncRef.rebind(path, shapeRef);                                    7
            java.lang.Object sync = new java.lang.Object();
            synchronized (sync) {   sync.wait();}
        } catch (Exception e) { ... }
    }
}
```

Java client program for CORBA interfaces *Shape* and *ShapeList*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListClient{
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);                          1
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", "");
            NameComponent path [] = { nc };
            ShapeList shapeListRef =
                ShapeListHelper.narrow(ncRef.resolve(path));         2
            Shape[] sList = shapeListRef.allShapes();                3
            GraphicalObject g = sList[0].getAllState();              4
        } catch(org.omg.CORBA.SystemException e) {...}
    }
```


IDL module Whiteboard

```
module Whiteboard {
    struct Rectangle{
    ...} ;
    struct GraphicalObject {
    ...};
    interface Shape {
    ...};
    typedef sequence <Shape, 100> All;
    interface ShapeList {
    ...};
};
```

IDL constructed types

Type	*Examples*	*Use*
sequence	*typedef sequence <Shape, 100> All;* *typedef sequence <Shape> All* bounded and unbounded sequences of *Shapes*	Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified.
string	*String name;* *typedef string<8> SmallString;* unboundedand bounded sequences of characters	Defines a sequences of characters, terminated by the null character. An upper bound on the length may be specified.
array	*typedef octet uniqueId[12];* *typedef GraphicalObject GO[10][8]*	Defines a type for a multi-dimensional fixed-length sequence of elements of a specified IDL type.

IDL constructed types cont'd

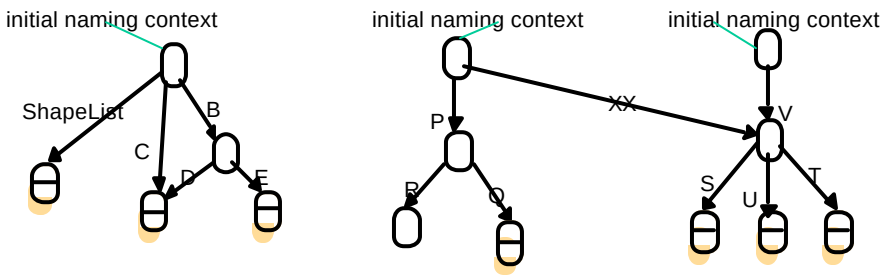
~~*Type*~~	~~*Examples*~~	~~*Use*~~
record	*struct GraphicalObject {* *string type;* *Rectangle enclosing;* *boolean isFilled;* *};*	Defines a type for a record containing a group of related entities. *Structs* are passed by value in arguments and results.
enumerated	*enum Rand* *(Exp, Number, Name);*	The enumerated type in IDL maps a type name onto a small set of integer values.
union	*union Exp switch (Rand) {* *case Exp: string vote;* *case Number: long n;* *case Name: string s;* *};*	The IDL discriminated union allows one of a given set of types to be passed as an argument. The header is parameterized by an *enum* which specifies which member is in use.

CORBA interoperable object references

IOR format

IDL interface type name	Protocol and address details			Object key	
interface repository identifier	IIOP	host domain name	port number	adapter name	object name

Naming graph in CORBA Naming Service



Part of the CORBA Naming Service NamingContext interface in IDL

struct NameComponent { string id; string kind; };

typedef sequence <NameComponent> Name;

interface NamingContext {

void bind (in Name n, in Object obj);
binds the given name and remote object reference in my context.

void unbind (in Name n);
removes an existing binding with the given name.

void bind_new_context(in Name n);
creates a new naming context and binds it to a given name in my context.

Object resolve (in Name n);
looks up the name in my context and returns its remote object reference.

void list (in unsigned long how_many, out BindingList bl, out BindingIterator bi);
returns the names in the bindings in my context.

};

Readings

- Coulouris – Chapters 5, 6, 17
- WWW (see links on class web page)
  - Java RMI tutorial on web
  - "A Young Persons Guide to SOAP"
  - CORBA web documents at OMG web site