

D: A Language Framework for Distributed Programming

Cristina Videira Lopes

PhD Thesis, College of Computer Science, Northeastern University. November 1997.

© Copyright 1997 Xerox Corporation. All rights reserved.

Appendix D. DJ Library Classes

```
//
// Copyright Xerox Corporation, 1997. All rights reserved.
//
// DArgument.java
//
// This class describes DArguments passed in remote operations. DArguments
// consist of an object (possibly null) and a traversal (possibly null).
// This class implements only the two marshaling methods that are called
// from the RMI run-time: writeExternal and readExternal.
//

import java.lang.reflect.*;
import java.io.*;

final public class DArgument implements Externalizable {
    public Object obj;
    Traversal trav;

    public DArgument() {super();}
    public DArgument(Object o, Traversal t) {
        obj = o; trav = t;
    }

    public void writeExternal(ObjectOutput s) {
        Trace.println("IN DArgument.writeExternal");
        try {
            s.writeObject(trav);
            if (trav == null) { // can be a DObject or not. Deep copy for DObjects.
                s.writeObject(obj);
            }
            else { // it's a DObject for sure
                if (obj == null) { // send a null class
                    s.writeObject(null);
                }
                else {
                    // must send the class name, so that readExternal knows what
                    // object to instantiate.
                    String classname = obj.getClass().getName();
                    s.writeObject(classname);
                    ((DObject)obj)._d_writeExternal(s, trav);
                }
            }
        }
        catch (IOException e) {
            System.err.println("Error in DArgument.writeExternal\n" + e.toString());
        }
        Trace.println("OUT DArgument.writeExternal");
    }

    public void readExternal(ObjectInput s) {
        Trace.println("IN DArgument.readExternal");
        try {
            trav = (Traversal)s.readObject();
            if (trav == null) // can be a DObject or not
                obj = s.readObject();
        }
    }
}
```

```
else { // it's a DObject
    // get the actual traversal object
    trav =(Traversal)Class.forName(trav.remoteinterface).//oops, must break
        getDeclaredField(trav.name).get(null);
    String classname = (String)s.readObject();

    if (classname != null) {
        obj = Class.forName(classname).newInstance();
        ((DObject)obj)._d_readExternal(s, trav);
    }
    else {
        obj = null;
    }
}
} catch (Exception e) {
    System.err.println("Error in DArgument.readExternal\n" + e.toString());
}
Trace.println("OUT DArgument.writeExternal");
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// DInvalidRemoteOperation.java  
//  
public class DInvalidRemoteOperation extends Exception {  
    DInvalidRemoteOperation() {  
        super();  
    }  
}
```

```

//
// Copyright Xerox Corporation, 1997. All rights reserved.
//
// DJNaming.java
//
// This class is the DJ interface to Java's Naming class.
// On bind, it sends portal (P) references to the Name Server.
// On lookup, it gets the portal (P) references from the name server, and
// instantiates the two proxies (two levels) that are necessary.
//
// NOTE: this bind service is, in fact, the rebind service of Java.
//

import java.rmi.*;
import java.net.*;
import java.lang.reflect.*;

public class DJNaming{
    private final static String stubTailString="P_Stub";

    public static void bind(String name, DObject obj)
    throws InvalidRemoteObjectException, RemoteException, MalformedURLException{
        Remote remoteObj = null;
        try {
            remoteObj = (Remote)(obj.getClass().getField("_p").get(obj));
        }
        catch (IllegalAccessException e) {
            System.err.println("There has been a serious error."+
                "Error Code: DJNRP");
            System.exit(9);
        } catch(NoSuchFieldException e){
            throw new InvalidRemoteObjectException
                (obj + " is not a valid remotable object\n");
        }
        Naming.rebind(name, remoteObj);
    }

    public static DObject lookup(String name)
    throws InvalidRemoteObjectException, RemoteException, NotBoundException,
        MalformedURLException, java.rmi.UnknownHostException{

        Remote remoteObject = Naming.lookup(name);
        String className = getTheClassName(remoteObject);
        DObject theObject = null;

        try{

            Class theClass = (Class.forName(className));
            Class ppClass = (Class.forName(className + "PP"));
            theObject = (DObject)theClass.newInstance();
            Object ppObject = ppClass.newInstance();
            (theClass.getField("_rself")).set(theObject, ppObject);
            (ppClass.getField("rself")).set(ppObject, remoteObject);

        } catch(ClassNotFoundException e){

            System.err.println("There has been a serious error."+
                "Error Code: DNLUPCNF");
            System.exit(4);
        }catch(InstantiationException e){
            throw new InvalidRemoteObjectException("Could not lookup object " +
                name + "\n"+
                "Probably because I could not find a 0-arity constructor\n"+
                " for "+className);
        }
    }
}

```

```
    }catch(IllegalAccessException e){
      System.err.println("There has been a serious error." +
        "Error Code: DNLUPIA");
      System.exit(4);
    }catch(NoSuchFieldException e){
      System.err.println("There has been a serious error."+
        "Error Code: DNLUNSF");
      System.exit(4);
    }
    return theObject;
  }
}

private static String getTheClassName(Object obj)
throws InvalidRemoteObjectException{
  String className = ((obj.getClass()).getName());
  if (className.endsWith(stubTailString)) {
    return
      className.substring(0,className.length()-stubTailString.length());
  } else {
    throw new InvalidRemoteObjectException (obj +" not valid remote object"
      + "Error Code: DNGRC" +
      className);
  }
}
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// DObject.java  
//  
// The interface that is common to ALL classes that pass through the  
// RIDL weaver.  
//  
  
import java.io.*;  
  
public interface DObject extends Externalizable {  
    void _d_readExternal(ObjectInput in, Traversal t);  
    void _d_writeExternal(ObjectOutput out, Traversal t);  
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// DPartCutter.java  
//  
// Implemented by IncompleteClass and by UnknownIncompleteClass.  
// DPartCutters are used in the marshaling methods of DObjects.  
//  
  
public interface DPartCutter {  
    boolean bypassPart(String name);  
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// IncompleteClass.java  
//  
// IncompleteClass stores which parts of the class are bypassed during  
// marshaling. The class is identified by "name", and the parts are  
// stored in the Vector of strings "bypass".  
//  
  
import java.util.*;  
  
final public class IncompleteClass implements DPartCutter {  
    public String name;  
    public Vector bypass = new Vector (4, 4);  
  
    public IncompleteClass(String n) { name = n; }  
  
    // called during the set up of Traversals.  
    public void bypass (String part) {  
        bypass.addElement(part);  
    }  
  
    // called from the marshling methods during marshaling.  
    public boolean bypassPart(String pname) {  
        String p = null;  
        for (Enumeration e = bypass.elements() ; e.hasMoreElements() ; ) {  
            p = (String)e.nextElement();  
            if (p.equals(pname)) {  
                return true;  
            }  
        }  
        return false;  
    }  
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// InvalidRemoteObjectException.java  
//  
  
import java.rmi.*;  
  
public class InvalidRemoteObjectException extends RemoteException{  
    public InvalidRemoteObjectException(){  
        super();  
    }  
  
    InvalidRemoteObjectException(String str){  
        super(str);  
    }  
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// MethState.java  
//  
// This is the only class in the DJlib that is used to support COOL.  
// It is used to capture the execution state of methods of JCore objects.  
//  
  
import java.util.Vector;  
  
final public class MethState {  
    public int depth = 0; // to handle re-entrance  
    private Vector tl = new Vector(5);  
  
    final public boolean isBusyByOther() {  
        if (depth > 0 && !tl.contains(Thread.currentThread()))  
            return true;  
        else return false;  
    }  
    final public void in() {  
        depth++;  
        tl.addElement(Thread.currentThread());  
    }  
    final public void out() {  
        depth--;  
        tl.removeElement(Thread.currentThread());  
    }  
}
```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// Trace.java  
//  
public class Trace {  
    static boolean TRACE = true;  
  
    public static void traceON() { TRACE = true; }  
  
    public static void traceOFF() { TRACE = false; }  
  
    public static void println(String str) {  
        if (TRACE)  
            print(str + "\n");  
    }  
  
    public static void print(String str) {  
        if (TRACE) {  
            System.out.print(str);  
            System.out.flush();  
            try { Thread.sleep(100); }  
            catch (InterruptedException e) {}  
        }  
    }  
}
```

```

//
// Copyright Xerox Corporation, 1997. All rights reserved.
//
// Traversal.java
//
// This class represents traversal directives. Traversal directives
// are attached to remote interfaces (stored in remoteinterface), and
// during the translation process they are given a name (stored in name).
// They store the classes that are incomplete, in a Vector of
// IncompleteClasses.
//

import java.io.*;
import java.util.*;

final public class Traversal implements Serializable {
    public String remoteinterface;
    public String name;
    public Vector classes = new Vector(4,4);

    private static final DPartCutter Unknown = new UnknownIncompleteClass();

    public Traversal(String n, String ri) {
        remoteinterface = ri;
        name = n;
    }

    public void incompleteClass (IncompleteClass c) {
        classes.addElement(c);
    }

    public DPartCutter isIncompleteClass(String cname) {
        for (Enumeration e = classes.elements() ; e.hasMoreElements() ; ) {
            IncompleteClass c = (IncompleteClass)e.nextElement();
            if (c.name.equals(cname)) {
                return c;
            }
        }
        return Unknown;
    }

    private void writeObject(ObjectOutputStream s) {
        try {
            Trace.println("In Traversal.writeObject");
            s.writeObject(remoteinterface);
            s.writeObject(name);
        } catch (IOException e) {
            System.err.println("Error in packing Traversal.\n" + e.toString());
        }
        Trace.println("Out Traversal.writeObject");
    }

    private void readObject(ObjectInputStream s) {
        try {
            Trace.println("In Traversal.readObject");
            remoteinterface = (String)s.readObject();
            name = (String)s.readObject();
        } catch (Exception e) {
            System.err.println("Error in unpacking Traversal.\n" + e.toString());
        }
    }
}

```

```
//  
// Copyright Xerox Corporation, 1997. All rights reserved.  
//  
// UnknownIncompleteClass.java  
//  
// When a Traversal is asked whether a certain class is incomplete or not,  
// and it finds that that class is not mentioned, then it returns an  
// instance of this class. This class basically says that no bypass  
// should be done (all parts should be copied).  
//  
  
final public class UnknownIncompleteClass implements DPartCutter {  
    public boolean bypassPart(String name) { return false; }  
}
```

