Title: **DISTRIBUTED COMPUTING USING JINI -Example**

Author: Elango Sundaram

Abstract: The following discussion covers an example JINI deployment. The classes involved and an overall implementation method is described. The example code and relevant information is available in the appendix.

1.Overview:

The example system consists of a JINI Look up Service, JINI enabled Service Provider and a JINI enabled client. A HTTP Server can be made as a central code base for downloading code dynamically. All individual operations i.e., compiling code, running rmic (for generating stub classes for the server code), starting HTTP server, starting JINI Look Up Service, Starting RMID, Starting RMI Registry, Starting JINI Service and executing JINI Client operations can be tasked to shell scripts. The example architecture for the implementation is designed such that the services could be located by clients through the look up service and that *any node* could act as a client. This is pictorially represented as follows:

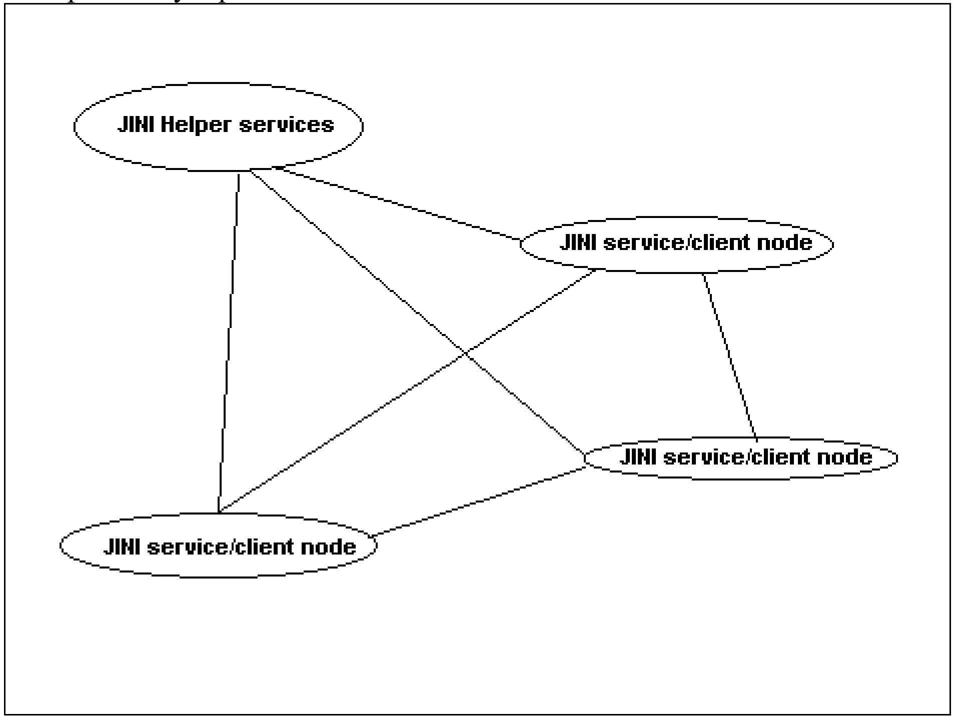

Fig 1 JINI Example

The example implementation testing has been done as follows: the HTTP Server, Look up Service and RMID were run in machine A, JINI service and RMIREGISTRY in machine B and JINI client in machine C. This can be represented as follows:

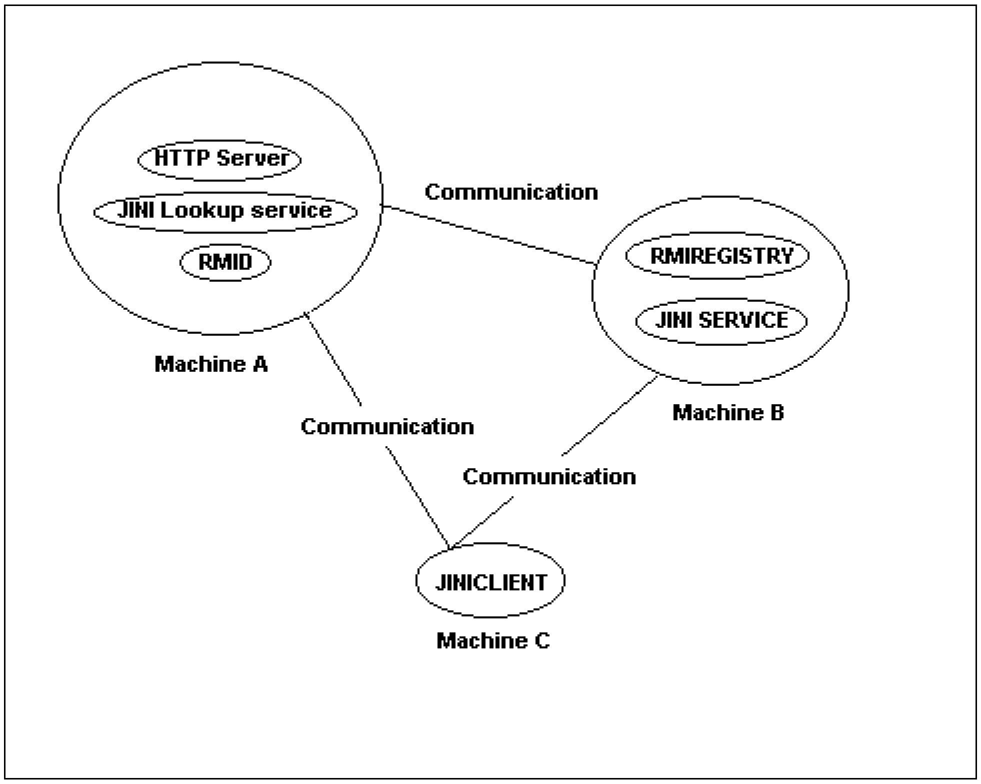

Fig 2 JINI Implementation Specifics

2. Example Details:

The example has a JINI enabled Server and a client (which could be also called a JINI peer). The client and server share a common interface, so as to enable them to have a common understanding of the services provided and the service requested.

A HTTP server can be used to serve up the required class/jar files. A look up service can be used to locate and register services. The server, once started, does a multicast discovery to find the lookup service and registers itself. The client when it starts up, finds the look up server and asks for a service. Once the service is found, the client uses the service proxy to communicate with the service. A monitor functionality is used here so that instead of waiting for random, static time period for the discovery process (this is because the discovery process can take variable time based on network traffic, distance between nodes, network bandwidth etc) to take place, the client can get notified as soon as the discovery is done. The class diagram for the implementation is as follows:

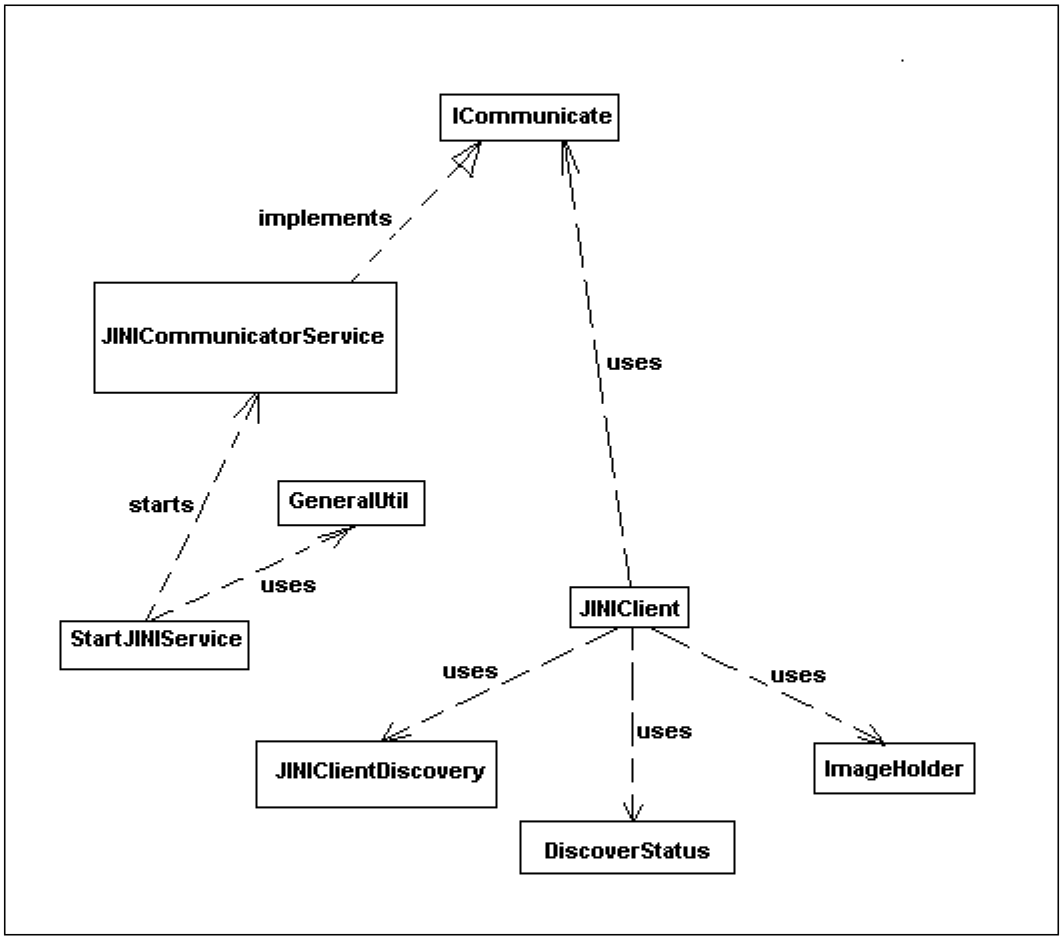

Fig 3 JINI Example Class Diagram

A StartJINIService class is used to actually start up the JINI Service. This is simply a starter class and the JINICommunicatorService class does the actual work of locating the look up server and registering the service. This class implements the ICommunicate interface which defines the methods offered by the service. The JINICommunicatorService implements the UnicastRemoteObject inteface, which satisfies a condition to be a RMI service provider. The StartJINIService class also uses the GeneralUtil to find the IP address of the start up hardware and dynamically bind the service to that address. This eliminates the need for us to manually configure the IP address for the service for each hardware used.

The JINIClient is the class that starts up the client part of the implementation (could also be thought of as a participant peer). The JINIClient uses the JINIClientDiscovery class in order to discover the Look up server and obtain the reference for service. The JINIClient also uses a special class called the DiscoveryStatus. The discovery Status implements the wait notify mechanism. This facility is used to find out exactly (and as soon as) the discovery process is over. This avoids the need to simply sleep the discovery thread to wait for a random amount of time and hope that discovery would have been done. The DiscoveryStatus class could also be configured to *time out* after a configured period of time. The ImageHolder is a simple holder class for holding the Image objects.

The operations involved in the JINI service are as follows. The StartJINIService loads a security manager if none is present. A Java policy file is used to set the security policy for the system. A new Entry object is created with the name of the JINI Service. The JINICommunicatorService class is instantiated. The service name is bound using a name chosen and then the service stub object is obtained. After doing so the JINI JoinManager class (A class that is a part of reference implementation) is used to join the service to the look up service.

The operations involved in the JINI client are as follows. The JINIClient loads a security manager if none is present. We also create the message that we want to send statically so that no time is wasted in creating the test objects when we actually try to send them. A new DiscoveryStatus object is created so that we get immediate notification of service discovery. The JINIClientService is then called up. This class implements the Discovery Listener and hence gets notification of discovery. Once the service is found we get a reference for the ICommunicate interface. This interface is used to call methods one the service provider directly from this point onwards.

3. Conclusion:

The JINI technology is a good candidate for scalable, reliable and low maintenance distributed computing technologies. JINI can play a key role in providing distributed computing functionality where data collection and representation are geographically separated and vast. Some of the interesting advantages of JINI are; having to do less administration after the system is developed and ease of modification of functionality. JINI also promotes object-oriented development in which many programmers are conversant.

4. Appendix:
4.1.Classes Common to Both Service and client:
4.1.1 ICommunicate

```
import java.rmi.*;
import javax.swing.ImageIcon;
/**
  * <BR>
  * This is an interface that will be shared by the jini
  * service proviser and the sercive requester.
  * @author esundara
  */

public interface ICommunicate extends Remote{



/**
 * This method will echo back your string
 */
public String talkBackString(String content) throws RemoteException;

/**
 * This method will echo back your ImageHolder.
 */
public ImageHolder talkBackImage(ImageHolder image) throws
RemoteException;

}
```
4.1.2 Bare Bones Log Class:

```
public class Log {
/**
  * Simple Log Util..
  * @author esundara
  */

/** Turn off Log */
public static  boolean doLog = false;

/** Simpe out */
public static void log(String message) {

if(doLog) {
System.out.println(message);
}

}
}
```

4.2 JINI Service Related Classes:

4.2.1 GeneralUtil:

```
import java.net.*;
/**
  * This class provides some basic utility functions.
  * @author esundara
  */

 public class GeneralUtil {

 /**
 * @return The IP Address Of the Host Machine
 * @return null on some problem with finding Host IP
 */
 public static String getHostAddress() {

        InetAddress iAddr = null;
        String myHostIP="NO_IP";
        try {
            iAddr = InetAddress.getLocalHost();
            myHostIP = iAddr.getHostAddress();
        } catch (java.net.UnknownHostException e) {
            Log.log("ERROR: GeneralUtil: CAN'T get the host IP");
        }
        if (myHostIP != null) {
            Log.log("HostIP: "+myHostIP);
        }
        else Log.log("ERROR:GeneralUtil: HostIP is Null");
        return myHostIP;

 }

 /**
 * @return The HostName Of the Host Machine
 * @return null on some problem with finding Host IP
 */
 public static String getHostName() {

        InetAddress iAddr = null;
        String myHostName="NO_HOST";
        try {
            iAddr = InetAddress.getLocalHost();
            myHostName = iAddr.getHostName();
        } catch (Exception e) {
            Log.log("ERROR: GeneralUtil: CAN'T get the host Name"+
e.getClass().getName());
        }
        if (myHostName != null) {
            Log.log("HostName: "+myHostName);
        }
        else Log.log("ERROR:GeneralUtil: HostName is Null");
        return myHostName;

 }


 }
```

4.2.2 JINICommunicatorService:

```
import java.rmi.*;
import java.io.*;
import java.rmi.server.*;
import net.jini.lookup.*;
import net.jini.core.lookup.*;
import net.jini.core.entry.*;
import net.jini.lookup.entry.*;
import net.jini.lease.*;
import net.jini.discovery.*;
import javax.swing.ImageIcon;

/**
  * This class provides the service implementation for JINI
  * @author esundara
  */


public class JINICommunicatorService extends UnicastRemoteObject
implements ICommunicate,ServiceIDListener,Serializable{

/** The service ID of the service */
public ServiceID serviceID = null;


/* Create a service */
public JINICommunicatorService() throws RemoteException {
      super();
}
/** Create a service at specified port*/

public JINICommunicatorService(int port) throws RemoteException {
      super(port);
}

/** Implement the echo message */
public String talkBackString(String content) throws RemoteException {

      Log.log("Content Obtained");
      Log.log(content);
      return content;
}
/** Implement the return Image */
public ImageHolder talkBackImage(ImageHolder icon) throws
RemoteException {
      Log.log("Image Obtained");
      return icon;

}

public void serviceIDNotify(ServiceID serviceID) {

      this.serviceID = serviceID;
      Log.log("The ServiceID Obtained");
      Log.log(serviceID.toString());


}

/** Get the Service ID*/
public ServiceID getServiceID() {

      return this.serviceID;
}


}
```

4.2.3 StartJINIService:

```
import java.rmi.*;
import java.io.*;
import java.rmi.server.*;
import net.jini.lookup.JoinManager;
import net.jini.core.entry.Entry;
import net.jini.lookup.entry.Name;
import net.jini.discovery.LookupDiscovery;
import net.jini.lease.LeaseRenewalManager;

/**
  * This class will start the JINI service..
  * @author esundara
  */
```

```
public class StartJINIService {
/** Name of the service */
public static String J_COMM_SERV = "JINICommunicatorService";

/** Name/Address of machine*/
public static String serviceAddress=null;
public static String bindName =null;
static {
      serviceAddress = GeneralUtil.getHostName();
      bindName = "rmi://"+serviceAddress+"/"+J_COMM_SERV;
}


/** Serice start */
public static void main(String argv[]) {
      try {
            Log.log("Begin to Start JINI Service..");
            System.setSecurityManager(new RMISecurityManager());
            Entry[] attribs = new Entry[1];
            attribs[0] = new Name(J_COMM_SERV);
            Log.log("JINICommunicatorService: Host is:
"+serviceAddress);
            Log.log("JINICommunicatorService: Service Bind Name is: "+
bindName);
            JINICommunicatorService jCs = new
JINICommunicatorService();
            Log.log("New Communicator Service Object Created");
            Naming.rebind(bindName,jCs);
            Log.log(J_COMM_SERV+" Bound");
            Object serviceStub = Naming.lookup(bindName);
            Log.log("Obtained Service Stub thru Look Up ..");
            JoinManager jM = new
JoinManager(serviceStub,attribs,jCs,new
LookupDiscovery(LookupDiscovery.ALL_GROUPS),new LeaseRenewalManager());


      }
      catch(Exception e) {
            Log.log("ERROR:
JINICommunicatorService:"+e.getClass().getName());
            e.printStackTrace();
      }



}


}
```

4.3 JINI Client Related Classes:

4.3.1 JINIClient:

```
import net.jini.core.discovery.LookupLocator;

import net.jini.core.lookup.*;
import net.jini.discovery.*;
import javax.swing.ImageIcon;
import java.rmi.*;
/**
  * This class will be the entry point for client.
  * The client discovers the JINI Service and will
  * transport objects.
  * @author esundara
  */
public class JINIClient  {

/** Client entry point */
public static void main (String[] args) {
      //Make your Image before its used..
     makeImage();
     // Make Your Message Before its used..
     makeMessage();
     // Load Security Manager if not present..
     initSecurity();
     // Mark the beginning time of operations..
     long beginTime=System.currentTimeMillis();
     // Count the number of successful operations
     long successCounter=0;
     // Loop Over and send Messages or images ..
     for(int i =0; i < 1; i++) {
      if(sendImage()) {
            successCounter++;
      }
      else
        break;

     }
     long endTime=System.currentTimeMillis();
     System.out.println("No.Of Successes = " + successCounter);
     System.out.println("Total Time =" + (endTime-beginTime));
     System.out.println("Average Round trip Message Time  =" +
(float)(endTime-beginTime)/(float)(successCounter) +" millisecs");
     System.out.println("Average Message Time  =" + (float)(endTime-
beginTime)/(float)((successCounter)*2) +" millisecs");

 }

  /**
   * This method will discover the Server and send it message
   * will return status of operation
   * @return true Successful communication
   * @return false Unsuccessful commmunication
   */
  static boolean sendMessage() {
      try {

            DiscoveryStatus dStat = new DiscoveryStatus();
            JINIClientDiscovery jD = new JINIClientDiscovery(dStat);
                  LookupDiscovery lkpDiscovery = new
LookupDiscovery(LookupDiscovery.NO_GROUPS);
                  Log.log("New LookupDiscovery Created.. ");
                  lkpDiscovery.addDiscoveryListener(jD);
                  Log.log("New ServiceFind AddiscoveredEventd TO 
LookupDiscovery");
                lkpDiscovery.setGroups(LookupDiscovery.ALL_GROUPS);
                  Log.log("The Status is "+ dStat.getStatus());
                  if(! (dStat.getStatus()) ) {
                        Log.log("The Service Could Not be Obtained in 
"+ dStat.WAIT_TIME_OUT +"millisecs");
                        return false;
                  }

            LookupLocator lookupLocator = jD.getLookupLocator();

            ICommunicate icom = jD.getComunicatorInterface();
            Log.log("The JINI Service is at "+lookupLocator.getHost()+" 
port "+lookupLocator.getPort());

            String message= makeMessage();
            String returnMessage = null;
            // Communicate...
            returnMessage  = icom.talkBackString(message);
            if(returnMessage != null){
                  Log.log(returnMessage);
                  dStat=null;
                  jD=null;
                  lkpDiscovery=null;
                  icom=null;
                  lookupLocator=null;
                  return true;
            }
            else
            {
                  Log.log("MESSAGE TRANSMISSION FAILED");
                  dStat=null;
                  jD=null;
                  lkpDiscovery=null;
                  icom=null;
                  lookupLocator=null;
                  return false;
            }



      }
      catch(Exception e) {
            Log.log("ERROR: 
JINIClient.."+e.getClass().getName()+e.getMessage());

            return false;
      }


  }
  /**
   * This method will discover the Server and send it image.
   * will return status of operation
   * @return true Successful communication
   * @return false Unsuccessful commmunication
   */
  static boolean sendImage() {
      try {
            DiscoveryStatus dStat = new DiscoveryStatus();
            JINIClientDiscovery jD = new JINIClientDiscovery(dStat);

                  LookupDiscovery lkpDiscovery = new
LookupDiscovery(LookupDiscovery.ALL_GROUPS);
                  Log.log("New LookupDiscovery Created.. ");

                  lkpDiscovery.addDiscoveryListener(jD);
                  Log.log("New ServiceFind AddiscoveredEventd TO
LookupDiscovery");

                  Log.log("The Status is "+ dStat.getStatus());
                  if(! (dStat.getStatus()) ) {
                        Log.log("The Service Could Not be Obtained in
"+ dStat.WAIT_TIME_OUT +"millisecs");
                        return false;
                  }


            LookupLocator lookupLocator = jD.getLookupLocator();

            ICommunicate icom = jD.getComunicatorInterface();
            Log.log("The JINI Service is at "+lookupLocator.getHost()+"
port "+lookupLocator.getPort());
            // communicate
            ImageHolder iReturn  = icom.talkBackImage(makeImage());
            if(iReturn != null){
                  Log.log("Image Received Back");
                  return true;
            }
            else
            {
                  Log.log("Image TRANSMISSION FAILED");
                  return false;
            }

      }
      catch(Exception e) {
            Log.log("ERROR:
JINIClient.."+e.getClass().getName()+e.getMessage());

            return false;
      }


  }

 private static String msg =null;
 /** Create the Message */
 private static String makeMessage() {
      if( msg != null) {

            return msg;
      }

      StringBuffer sbf = new StringBuffer();
      for(int i=0;i< 1000;i++) {
            sbf.append("HELLO");
      }
      msg =  sbf.toString();

      return msg;
 }

 private static ImageHolder ico =null;
 /** Create the Image */
 private static ImageHolder makeImage() {
      if( ico != null) {

            return ico;

      }

      else {
            ico = new ImageHolder();
            ico.setImg(new ImageIcon("jinilogo.gif"));
      }
      return ico;
 }


/** Load the RMI security Manager if it is nor there already */
private  static void initSecurity() {
      try{
            if(System.getSecurityManager() == null)   {
                  Log.log("No Security Mgr. Loading RMISMgr.. ");
                  System.setSecurityManager(new RMISecurityManager());
            }


       }catch (Exception e) {
               Log.log("JiniClient Exception:" +
e+e.getClass().getName()+e.getMessage());
               e.printStackTrace();
         }


}


 }
```

4.3.2 JINIClientDiscovery:

```
import net.jini.core.lookup.*;
import net.jini.core.discovery.LookupLocator;
import net.jini.discovery.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import net.jini.core.entry.*;
import net.jini.lookup.entry.*;
import net.jini.core.lookup.*;
/**
  * This class will do the actual discovery of service..
  * @author esundara
  */

public class JINIClientDiscovery implements DiscoveryListener {


/** The expected interface */
private ICommunicate icom =null;
/** The Locator of service */
private LookupLocator lookupLocator =null;
/** Monitor*/
DiscoveryStatus dStat = null;

/** Discover Service */
public JINIClientDiscovery(DiscoveryStatus dStat) {
      this.dStat=dStat;

}

/**
 * Can get you the service interface if the
 * Server is found ..
 */
public ICommunicate getComunicatorInterface() {
      return this.icom;

}

/** Gets the locator */
public LookupLocator getLookupLocator() {

      return this.lookupLocator;
}



/** This is called when service is discovered */
public void discovered(DiscoveryEvent discoveredEvent) {
      Log.log("Discovered Called....");
        try {
             ServiceRegistrar[] registrars =
discoveredEvent.getRegistrars();
           for (int i = 0; i < registrars.length; ++i) {
                   Log.log("Total No. Of Services: "+
registrars.length+" This is "+i);
                 ServiceID id = registrars[i].getServiceID();
                 Entry[] attribs = new Entry[1];
             attribs[0] = new Name("JINICommunicatorService");
             ServiceTemplate stl = new
ServiceTemplate(null,null,attribs);
             this.lookupLocator = registrars[i].getLocator();
                 Object o = registrars[i].lookup(stl);
             Log.log("The object O is of Type
"+o.getClass().getName());
             if (o instanceof ICommunicate) {
                  Log.log("This Stub implements ICommunicate");
             }
             else {

                  Log.log("This Stub Does Not implement ICommunicate");
             }
             try{

                  this.icom =(ICommunicate) o;
                  Log.log("Remote Interface Obtained");

             }catch(Exception ex){
                  Log.log("JINIClientDiscovery Exception
"+ex.getClass().getName()+ ex.getMessage());

             }


             }
            dStat.setStatus(true);
         }
         catch (Exception e) {
               Log.log("JINIClient Exception:" + e);
         }
}

/** This is called when a service is discarded */
public void discarded(DiscoveryEvent discEvent) {

      Log.log("DISCARD: Discovery Event Called..");
}


}
```

4.3.3 ImageHolder:

```
import javax.swing.ImageIcon;
/**
* This class is a simple holder class for the images.
  * @author esundara
  */
public class ImageHolder implements java.io.Serializable {

/** The icon object*/
private ImageIcon ico = null;

/**
 * Set the image
 */
public void setImg(ImageIcon i){
      ico = i;
}
/**
```

```
 * Get the image object
 */
public ImageIcon getImg() {
      return ico;
}


}
```

4.3.4 DiscoveryStatus:

```
public class DiscoveryStatus{

/**
  * This class provides mechanisms for getting an
  * immediate response to discouvery by using
  * java monitor mechanism
  * @author esundara
  */

/** This boolean will be set as soon as discovery is made*/
private boolean DISCOVERED =false;

/** Time Out Period in milliseconds*/
public long WAIT_TIME_OUT=20000;

/**
 * Calling this method will make it wait
 * till discovery is made
 */
public synchronized boolean getStatus() {
      try {
            while(DISCOVERED==false){
                  wait(WAIT_TIME_OUT);
            }

      }
      catch(Exception e) {
            Log.log("Discovery Status "+e.getClass().getName()+
e.getMessage());
      }
      return this.DISCOVERED;

}

/**
 * This method will notify once a discovery is made
 */
public synchronized void setStatus(boolean stat) {
      try {
            this.DISCOVERED=stat;
            notifyAll();
      }
      catch(Exception e) {
            Log.log("Discovery Status "+e.getClass().getName()+
e.getMessage());
      }
}


}
```

4.4 Some Related unix Scriplets:

4.4.1. Compile JINI Service Related Classes:

```
#!/bin/sh
# Compile Service Source Code and make it to a Jar File

JINI_LIB=jars
SERVER_DIR=serverclasses
javac  -d $SERVER_DIR -classpath ./:$SERVER_DIR:$JINI_LIB/jini-
core.jar:$JINI_LIB/jini-ext.jar JINICommunicatorService.java
StartJINIService.java ICommunicate.java Log.java ImageHolder.java
cd $SERVER_DIR
jar -cvf jiniserver-cl.jar *.class
mv jiniserver-cl.jar ../jars
cd ..
```

4.4.2. Compile JINI Client Related Classes:

```
#!/bin/sh
# Compile
# Compile Client Source Code and make it to a Jar File

JINI_LIB=jars
CLIENT_DIR=clientclasses
javac  -d $CLIENT_DIR -classpath ./:$CLIENT_DIR:$JINI_LIB/jini-
core.jar:$JINI_LIB/jini-ext.jar ICommunicate.java JINIClient.java
JINIClientDiscovery.java DiscoveryStatus.java Log.java ImageHolder.java
cd $CLIENT_DIR
jar -cvf jiniclient.jar *.class
mv jiniclient.jar ../jars
cd ..
```

4.4.3 Starting the webserver:

```
echo 'SERVICE 1 : Web Server '

rm -rf *log

JINI_LIB=jars
java -jar $JINI_LIB/tools.jar -port 20000 -dir $JINI_LIB -trees -
verbose
```

4.4.4. Starting RMI Daemon:

```
echo 'SERVICE 2 : RMID'
rmid -log jini_rmid_log -J-Djava.security.policy=myPolicyFile
```

4.4.5 Starting JINI LookUp Server:

```
echo 'JINI Look Up Service'
```

```
JINI_LIB=jars
java  -jar $JINI_LIB/reggie.jar http://myHTTPServer.com:20000/reggie-
dl.jar myPolicyFile jini_lus_log public
```

4.4.5. Starting Rmi Registry

```
#!/bin/sh
echo $CLASSPATH
echo "RMIREGISTRY"
rmiregistry
```

4.4.7. Start JINI Service

```
#!/bin/sh
# Starts JINI Communicator Service

JINI_LIB=jars
S_L=$JINI_LIB/jini-core.jar:$JINI_LIB/jini-ext.jar
S_L=$S_L:/pkg/j2sdkee1.2/lib/j2ee.jar
S_L=$S_L:$JINI_LIB/jiniserver-cl-all.jar
echo "$S_L"
java -Djava.rmi.server.useCodebaseOnly=true -
Djava.security.policy=myPolicyFile -
Djava.rmi.server.codebase=http://myHTTPServer.com:20000/jiniserver-cl-
all.jar -classpath $S_L StartJINIService
```

4.4.8. JINI Client

```
#!/bin/sh
# JINI Client

JINI_LIB=jars
C_L=$JINI_LIB/jiniserver-cl.jar:$JINI_LIB/jiniclient.jar:
$JINI_LIB/jini-core.jar:$JINI_LIB/jini-ext.jar

java -Djava.rmi.server.useCodebaseOnly=true -Djava.rmi.server.codebase=
http://myHTTPServer.com:20000/jiniserver-cl-all.jar -
Djava.security.policy=myPolicyFile -classpath $C_L JINIClient
```

Author: Elango Sundaram
www.geocities.com/esundara