Devoluciones de llamada en RMI (Callbacks)

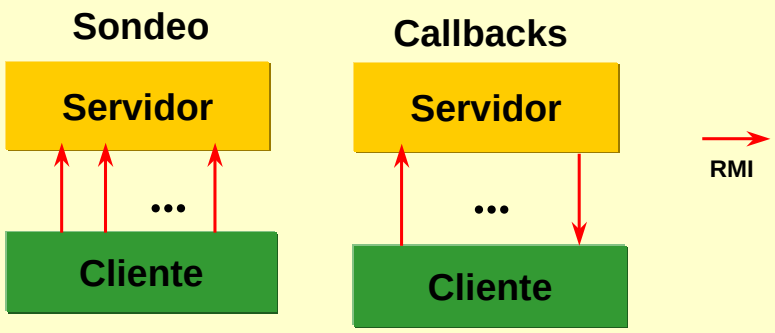
Sistemas Distribuidos

(Capítulo 8 de Distributed Computing de M. L. Liu)

Devolución de llamadas, introducción

- Cliente-servidor clásico
  - servidor pasivo
  - Tipo “pull”.
- “Push/pull”, es interesante en:
  - Monitorización
  - Juegos
  - Subastas
  - Votación/encuesta
  - Chat
  - Tablero de noticias
  - Groupware

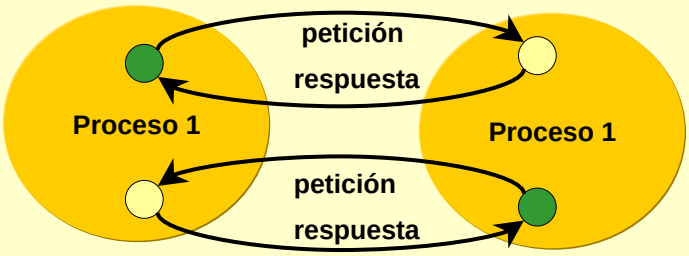
Sondeo vs. Callbacks



- Sin “callbacks”, el cliente debe sondear continuamente el servidor hasta que se produce cierta respuesta.
- Con callbacks el cliente es notificado por el servidor

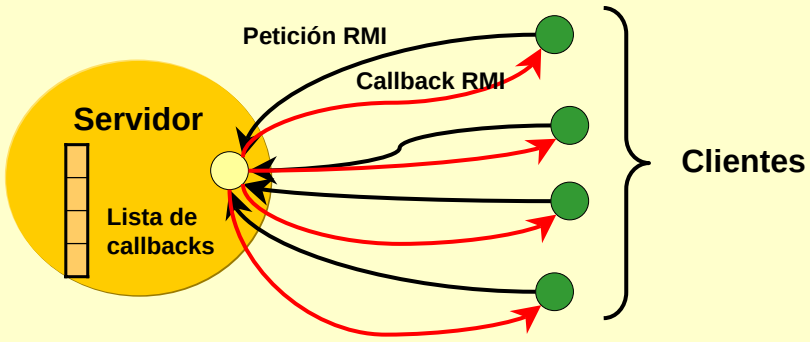
Comunicaciones en dos sentidos

- Algunas aplicaciones precisan que ambos lados inicien IPCs.
- Esto es posible mediante unpar de sockets para comunicación duplex.

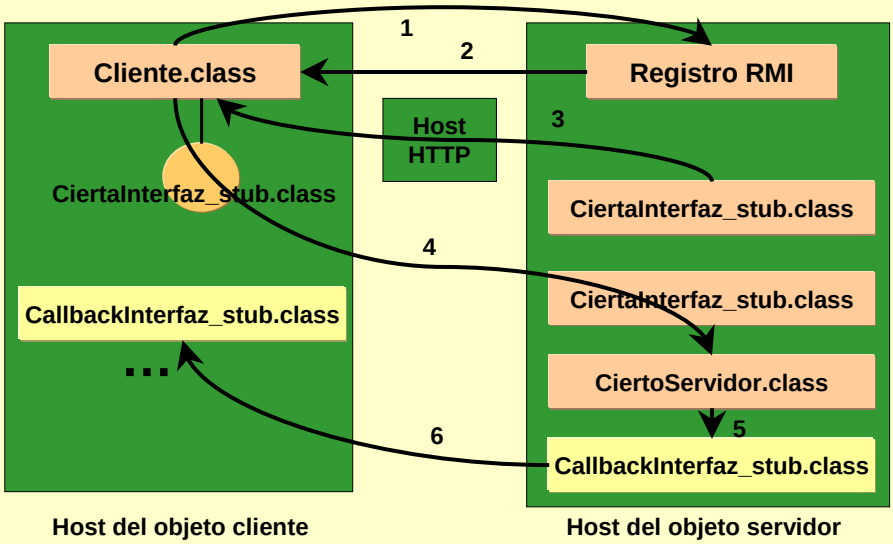


Callbacks RMI

- El cliente inscribe un objeto remoto para recibir una invocación remota.
  - Al ocurrir cierto evento, el servidor realiza una devolución de llamada a cada cliente interesado.



Interacciones C-S en los callbacks



Despliegue

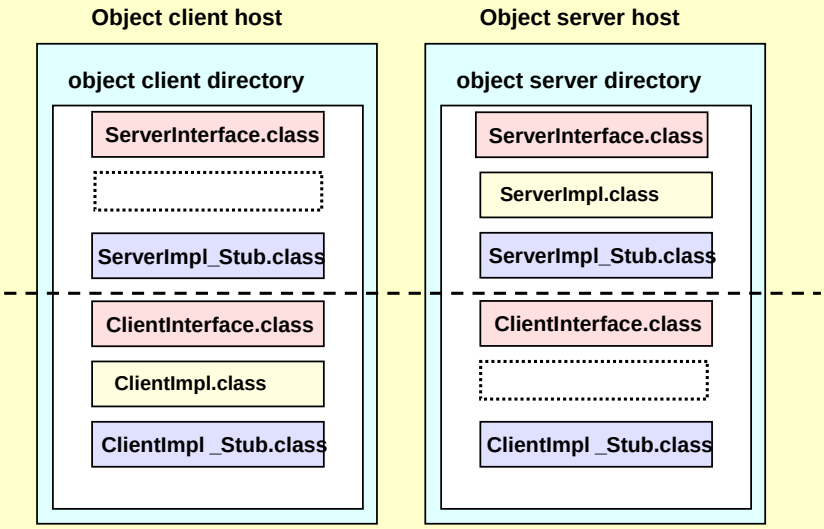


Diagrama de Clases

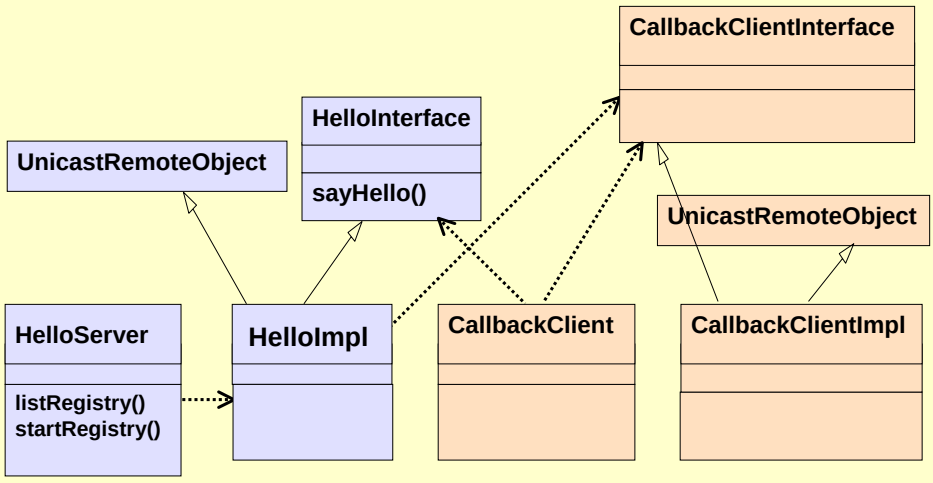
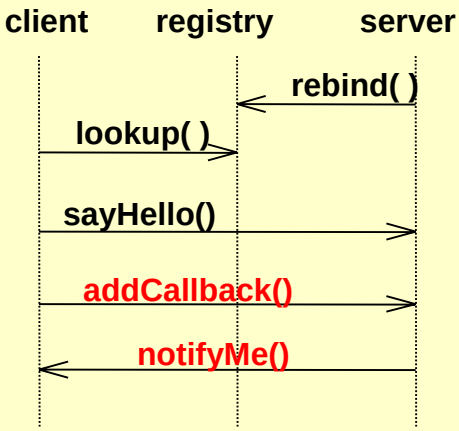


Diagrama de secuencia



Interfaz de callback

- El servidor ofrece un método remoto para que el cliente registre sus callbacks
- Hay que diseñar una interfaz remota para el callback
- La interfaz debe incluir un método que será invocado en el callback desde el servidor.
- El cliente deberá ser una subclase de RemoteObject e implementará la interfaz de callback.
- El cliente se registrará frente a la clase remota para ser rellamado (usualmente en el main).
- El servidor invoca el método remoto del cliente en caso de aparecer el evento indicado.

El servidor “Hola” con callbacks

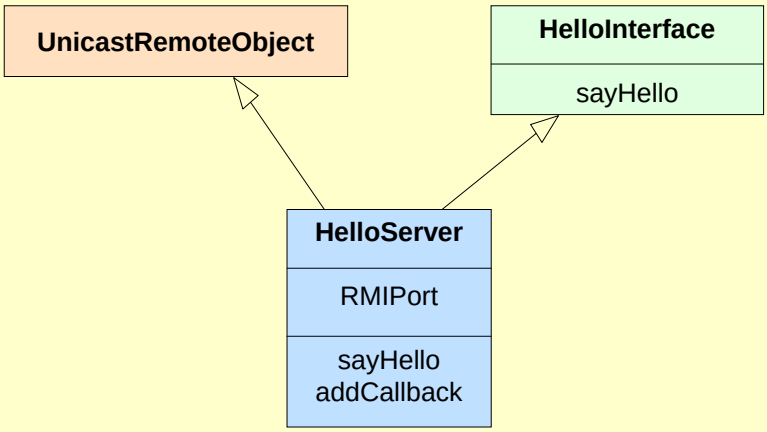


Diagrama de Clases de HelloServer

El servidor “Hola” con callbacks

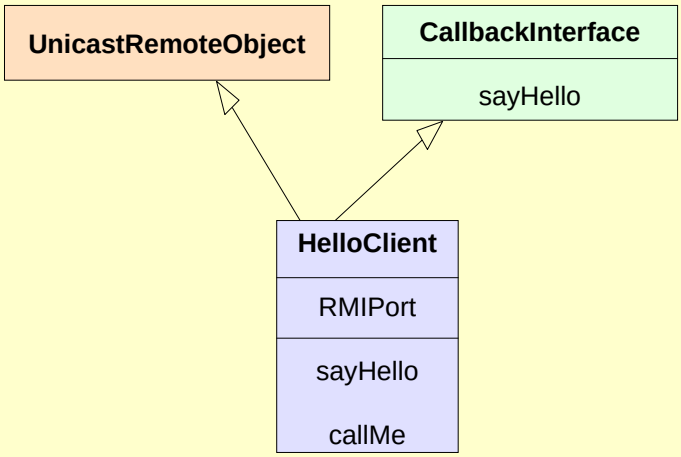


Diagrama de Clases de HelloClient

El servidor “Hola” con callbacks

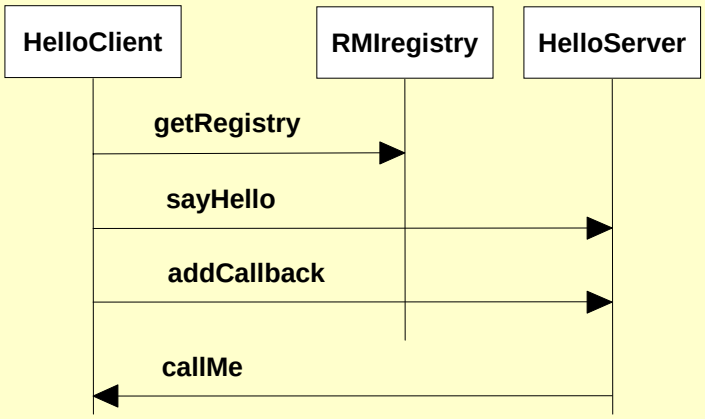


Diagrama de eventos para la aplicación Hello

Interfaz remota del servidor

```
import java.rmi.*;

/* CallbackServerInterface.java,    @author M. L. Liu */
public interface CallbackServerInterface extends Remote {
  // Método remoto usual
  public String sayHello( )    throws java.rmi.RemoteException;

  // Método de registro del callback
  public void registerForCallback( CallbackClientInterface callbackClientObject )
                                   throws java.rmi.RemoteException;

  // Método de desregistro del callback
  public void unregisterForCallback( CallbackClientInterface callbackClientObject )
                                   throws java.rmi.RemoteException;
}
```

Implementación del servidor remoto

```
/* importaciones */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.net.*;
import java.io.*;

/* callbackServer.java, @author M. L. Liu */
```


Implementación del servidor remoto

```
public class CallbackServer  {

 public static void main(String args[ ]) {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  String portNum, registryURL;
  try{  System.out.println( "Enter the RMIregistry port number:");
        portNum  =  (br.readLine()).trim();
        int RMIPortNum  =  Integer.parseInt(portNum);
        startRegistry( RMIPortNum );
        CallbackServerImpl exportedObj  =  new CallbackServerImpl();
        registryURL = "rmi://localhost:" + portNum + "/callback";
        Naming.rebind(registryURL, exportedObj);
        System.out.println("Callback Server ready.");
  }  catch (Exception re) {  System.out.println("Exception: HelloServer.main: " + re);  }
 } // end main
```

Implementación del servidor remoto

```
// Arranca un registro RMI local si no existiera, en cierto puerto.
private static void startRegistry(int RMIPortNum) throws
    RemoteException{
   try {
     Registry registry =  LocateRegistry.getRegistry(RMIPortNum);
     registry.list( );    // Lanza una excepción si no existe el registro
   } catch (RemoteException e) {
     // No hay un registro válido en el puerto.
     Registry registry =  LocateRegistry.createRegistry(RMIPortNum);
   }
 } // end startRegistry
} // end class callbackServer
```


Implementación del sirviente

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;

/* CallbackServerImpl.java,  @author M. L. Liu */
public class CallbackServerImpl extends UnicastRemoteObject
                                implements CallbackServerInterface {
   private Vector clientList;

  public CallbackServerImpl() throws RemoteException {
     super( );
     clientList = new Vector();
  }
```

Implementación del sirviente

```
public String sayHello( )   throws java.rmi.RemoteException
   {     return("hello");     }

 public synchronized void registerForCallback(
                              CallbackClientInterface
   callbackClientObject)
                              throws java.rmi.RemoteException{
   if (!(clientList.contains(callbackClientObject))) {
      clientList.addElement(callbackClientObject);
      System.out.println("Registered new client ");
      doCallbacks();
    } // end if
  }
```


Implementación del sirviente

```
// Desregistra el cliente
public synchronized void unregisterForCallback(
   CallbackClientInterface callbackClientObject)
   throws java.rmi.RemoteException{
   if (clientList.removeElement(callbackClientObject)) {
     System.out.println("Unregistered client ");
   } else {
      System.out.println(
          "unregister: El cliente no estaba registrado.");
   }
 }
```

Implementación del sirviente

```
private synchronized void doCallbacks( )
                    throws java.rmi.RemoteException{
  System.out.println( "*************\n" + "Callbacks iniciados ---");
  for (int i = 0; i < clientList.size(); i++){
    System.out.println("doing "+ i +"-th callback\n");
    // convert the vector object to a callback object
    CallbackClientInterface nextClient =
        (CallbackClientInterface)clientList.elementAt(i);
   // invoke the callback method
    nextClient.notifyMe("Number of registered clients=" +
  clientList.size());
  } // end for
   System.out.println("**************\n" + “Callbacks completados---");
 } // doCallbacks
}// end CallbackServerImpl class
```


Interfaz del callback del cliente

```
import java.rmi.*;

/* CallbackClientInterface.java,  @author M. L. Liu */

public interface CallbackClientInterface
                    extends java.rmi.Remote{
   // Es invocado por un callback del servidor
   public String notifyMe(String message)
                    throws java.rmi.RemoteException;
} // end interface
```

Implementación del cliente

```
import java.io.*;
import java.rmi.*;
/* CallbackClient.java, @author M. L. Liu */
public class CallbackClient {
  public static void main(String args[ ]) {
    try {
      int RMIPort;
      String hostName;
      BufferedReader br = new BufferedReader(  new InputStreamReader(System.in));
      System.out.println("Enter the RMIRegistry host namer:");
      hostName = br.readLine( );
      System.out.println("Enter the RMIregistry port number:");
      String portNum = br.readLine( );
      RMIPort = Integer.parseInt(portNum);
      System.out.println( "Enter how many seconds to stay registered:");
      String timeDuration = br.readLine( );
      int time = Integer.parseInt(timeDuration);
```


Implementación del cliente

```
    String registryURL =  "rmi://" + hostName + portNum + "/callback";
    CallbackServerInterface h = (CallbackServerInterface)Naming.lookup(registryURL);
    System.out.println("Lookup completado\nEl servidor dijo:"+h.sayHello());
    CallbackClientInterface callbackObj =  new CallbackClientImpl();
    h.registerForCallback(callbackObj);
    System.out.println("Registered for callback.");
    try {  Thread.sleep(time * 1000);  }
    catch (InterruptedException ex){ // sleep over  }
    h.unregisterForCallback(callbackObj);
    System.out.println("Desregistrado para callback.");
   } catch (Exception e) {
    System.out.println( "Excepcion en CallbackClient: " + e);
   } // end catch
  } //end main
}//end class
```

Implementación del callback

```
import java.rmi.*;
import java.rmi.server.*;
/* CallbackClientImpl, @author M. L. Liu  */
public class CallbackClientImpl extends UnicastRemoteObject
                    implements CallbackClientInterface {
  public CallbackClientImpl() throws RemoteException { super( ); }
  public String notifyMe(String message){
    String returnMessage = "Call back recibido: " + message;
    System.out.println(returnMessage);
    return returnMessage;
  }
}// end CallbackClientImpl class
```


Archivo de política

```
grant {
  permission java.net.SocketPermission "*:1099",  "connect, accept, resolve";
// Permite contactar con el registro de cualquier host

  permission java.net.SocketPermission "*:1024-65535",  "connect, accept, resolve";
// Permite a los clientes RMI conectar por red con los puertos públicos
// de cualquier host. Podemos poner en estos puertos el registro

  permission java.net.SocketPermission "localhost:1099", "connect, resolve";
// Esta dirección es de un host concreto (cambiar)
  permission java.net.SocketPermission "129.65.242.5:1024-", "connect, accept";

  permission java.net.SocketPermission "*:80",  "connect";
// Permite conexión con el web en cualquier host
};
```