Sesión 5:

Tema 8: Objetos Distribuidos e Invocación Remota. Java RMI

SISTEMAS DISTRIBUIDOS

GRADO EN INGENIERÍA INFORMÁTICA

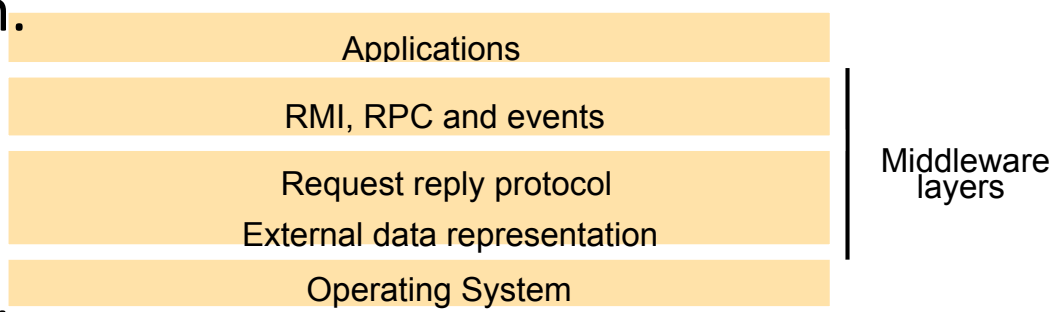
UNED

Introducción (1/2)

- Modelos de programación de aplicaciones distribuidas.
- Extensiones de modelos clásicos:
  - Llamada a procedimiento remoto (RPC).
  - Invocación a métodos remotos (RMI).
  - Basado en eventos.
- Hoy trataremos:
  - La comunicaciópn entre objetos distribuidos.
  - Diseño e implementación de RMI.
  - Java RMI

Introducción (2/2)

- Middleware. Capa software que nos proporciona:
  - Transparencia frente a la ubicación.
  - Protocolos de comunicación.
  - Hardware.
  - Sistemas Operativos.
  - Lenguajes de Programación.



- Interfaces. Conjunto de métodos, eventos y propiedades que expone una clase o módulo, permitiendo modificar la implementación sin afectar a los usuarios de la clase o módulo.
- Interfaces de servicio e Interfaces remotas.
- Lenguajes de definición de interfaces.

Comunicación entre objetos remotos RMI (1/8)

- El modelo de objetos.
  - Referencias a objetos.
  - Interfaces.
  - Acciones.
  - Excepciones. throw/catch
  - Compactación automática de memoria.
- Objetos Distribuidos.
  - Estado de un objeto.
  - Estado de un sistema distribuido.
  - Arquitectura cliente-servidor.

Comunicación entre objetos remotos RMI (2/8)

- El modelo de objetos distribuido.
  - Extensión del modelo de objetos.
  - Objeto Remoto: Puede recibir invocación de métodos remotos, que son aquellas realizadas desde objetos en otros procesos.
  - Conceptos fundamentales para este modelo:
    - Referencia a objeto remoto.
    - Interfaz remota.
    - Acciones en un sistema distribuido.
    - Compactación de memoria en un sistema distribuido.
    - Excepciones.

Comunicación entre objetos remotos RMI (3/8)

- Cuestiones de diseño para RMI.
  - Semántica Invocación remota frente a Invocación local.
    - Pudiera ser. Se ejecuta una vez o ninguna.
    - Al menos una vez. Puede haberse ejecutado varias veces. Operaciones idempotentes.
    - Máximo una vez. Se recibe o el resultado o una excepción. Usada en Java RMI.
  - Nivel de transparencia.
    - ¿Invocaciones locales iguales a RMI?
    - RMI más vulnerables a fallos y mayor latencia.
    - Java RMI: misma sintaxis pero implementan la interfaz Remote y lanzan RemoteExceptions

Comunicación entre objetos remotos RMI (4/8)

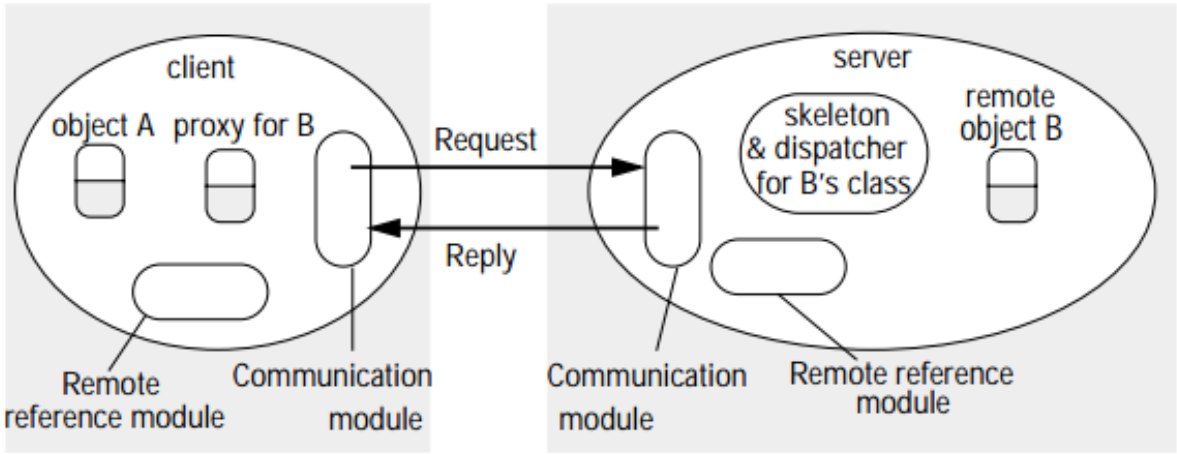
- Implementación de RMI.
- Módulo de comunicación. En cada proceso cliente y servidor.
  - Mantienen la semantica de invocación.
  - El del servidor, en respuesta al identificador de objeto remoto en la petición, selecciona el distribuidor para la clase del objeto invocado, devolviendo la referencia obtenida del módulo de referencia remota.
- Módulo de referencia remota.
  - En cada proceso será responsable de traducir las referencias entre objetos locales y remotos.
  - Usa una tabla de objetos remotos con los objetos remotos del proceso y los proxys locales.

Comunicación entre objetos remotos RMI (5/8)

- Software RMI. Capa entre los objetos de nivel aplicación y los módulos de comunicación y referencia remota.

-Proxy. Hace transparente al cliente la RMI.

-Distribuidor. Selecciona el método apropiado del esqueleto del objeto remoto.



-Esqueleto. Clase que implementa los métodos de la interfaz remota.

- Generación de las clases para cada proxy, distribuidor y esqueleto.
  - Automáticamente mediante un compilador de interfaces.

Comunicación entre objetos remotos RMI (6/8)

- Programas cliente y servidor.
  - Servidor: Distribuidores, esqueletos y clases servidoras.

    Inicialización en clases serv.: Al menos crear uno de los objetos remotos, y registrarlo en un enlazador.
  - Cliente: Proxy de cada objeto remoto al que invoque.
  - Métodos factoria. Las interfaces no pueden tener constructores. Los objetos remotos se crean en la seccion de inicialización o a través de metodos factoria incluidos en la interfaz remota.
- El enlazador.
  - Servicio que facilita las referencia a objetos remotos. Servidores registran y clientes busca los objetos remotos. En Java RMIRegistry.

Comunicación entre objetos remotos RMI (7/8)

- Hilos del servidor. Cada ejecución de una invocación remota correrá en un hilo o hebra separada.
- Activación de objetos remotos.
  - Objeto remoto podrá estar Áctivo o Pasivo.
  - Objeto pasivo: Implementación y estado empaquetado.
  - Proceso activador: Registrár y activar los objetos pasivos, y controlar los objetos activados.
- Almacenes de objetos persistentes.
  - Existencia garantizada cuando esta pasivo. Persistent Java
  - Contendrá grandes cantidades de objetos en disco.
- Ubicación de objetos. Referencia y dirección de Objeto Remoto.
  - Servicio de localización. A partir de referencias.

Comunicación entre objetos remotos RMI (8/8)

- Compactación automática de memoria.
  - Recuperar la memoria cuando no existan referencias a un objeto local o remoto.
  - Algoritmo distribuido basado en recuento de referencias:
    - El servidor controla el nº de proxys existentes
    - Si no existen puede liberar la memoria mientras no existan referencia locales.
    - Semantica de invocación 'al menos una vez' y periodos de concesión renovables por los clientes.
    - Java RMI. Concesiones con 'Lease'.

Eventos y Notificaciones (1/3)

- Objetos reaccionan de forma asincrona a notificaciones asociadas a eventos originados en objetos remotos.
- Paradigma Publica-Subscribe.
- Sistemas distribuidos basados en eventos:
  - Heterogéneos. Sólo se requiere que los objetos generadores de eventos publiquen los tipos de eventos que ofrecen y que los otros objetos se subscriban a los eventos y proporciones una interfaz para recibir las notificaciones.
  - Asíncronos. Notificaciones asíncronas. Desacoplados.

Eventos y Notificaciones (2/3)

- Participantes en notificación de eventos distribuida.
  - Son:
    - Objeto de interés.
    - Evento.
    - Notificación.
    - Suscriptor.
    - Objetos observadores.
    - Anunciante.
  - Semántica de reparto. En función de los requisitos.
  - Reglas para los observadores. Encaminamiento, Filtrado de notificaciones, patrones de eventos y buzones.

Eventos y Notificaciones (3/3)

- Especificación de Eventos Distribuidos de Jini
  - Objetos involucrados:
    - Generadores de eventos.
    - Oyentes de eventos remotos.
    - Eventos remotos.
    - Agentes terceros.
  - Interfaces y clases:
    - RemoteEventListener. Método notify(RemoteEvent).
    - RemoteEvent. Evento generador, tipo de evento, nº de secuencia, objeto empaquetado.
    - EventGenerator. Método register para suscriptores.
  - Agentes Terceros.

Caso de Estudio: Java RMI (1/)

- Extensión del modelos de objetos de Java que da soporte a objetos distribuidos.
  - Misma sintaxis de invocación.
  - Misma comprobación de tipos.
  - El objeto remoto implementa la interfaz *Remote.*
  - El objeto que invoca recibe *RemoteExceptions.*
  - Entorno concurrente.
- Interfaces remotas en Java.
  - Heredan de java.rmi.Remote
  - Los métodos remotos lanzan RemoteExceptions

Caso de Estudio: Java RMI (1/)

- Paso de parámetros y resultados.
  - O son tipos primitivos o implementan la interfaz *java.io.Serializable.*
  - *Paso de objetos remotos: referencia al objeto*
  - *Paso de objetos no remotos: por valor*
- *Decarga de las clases.*
  - *El código de clases pasadas por valor y de los proxys se descarga cuando es necesario.*
  - *Descarga en ambos sentidos:*
    - *Servidor-> Clientes*
    - *Cliente -> Servidor*

Caso de Estudio: Java RMI (1/)

- *RMIregistry. Enlazador para Java RMI*
  - *Uno por computador.*
  - *URLs y referencias a objetos remotos.*
  - *Nombre obj. Remoto: //servidor:puerto/nombreObjeto*
  - *No controla todo el sistema, hay que conocer en que computador esta el objeto remoto.*
  - *Clase Naming:*
    - *void rebind(String nombre, Remote obj)*
    - *void bind(String nombre, Remote obj)*
    - *void unbind(String nombre, Remote obj)*
    - *Remote lookup(String nombre)*
    - *String[] list()*

Caso de Estudio: Java RMI (1/)

- *Construcción de programas clientes y servidores*
  - *Servidor*

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

Inicialización del servidor

Clase Sirviente que implementa la interfaz remota

Interfaz Remota

Enlazamos la instancia del servidor a un nombre

Gestor de Seguridad

Caso de Estudio: Java RMI (1/)

- *Construcción de programas clientes y servidores*
  - *Interfaces Remotas*

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

public interface Shape extends Remote {
      int getVersion() throws RemoteException;
      GraphicalObject  getAllState() throws RemoteException;
}

public interface ShapeList extends Remote {
      Shape newShape(GraphicalObject g) throws RemoteException;
      Vector allShapes() throws RemoteException;
      int getVersion() throws RemoteException;
}
```

Heredan de Remote

Lanzan excepciones RemoteException

Argumento debe ser Serializable

Método Factoria

Caso de Estudio: Java RMI (1/)

- *Construcción de programas clientes y servidores*
  - *Implementación de la Interfaz Remota*

```
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
     private int version;
    public ShapeListServant()throws RemoteException{...}
    public Shape newShape(GraphicalObject g) throws RemoteException {
          version++;
               Shape s = new ShapeServant( g, version);
               theList.addElement(s);
               return s;
    }
    public  Vector allShapes()throws RemoteException{...}
    public int getVersion() throws RemoteException { ... }
}
```

Heredan UnicastRemoteObject

Implementa la Interfaz Remota

Lanzan excepciones RemoteException

Caso de Estudio: Java RMI (1/)

- *Construcción de programas clientes y servidores*
  - *Cliente*

```
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")      ;
          Vector sList = aShapeList.allShapes();
     } catch(RemoteException e) {System.out.println(e.getMessage());
     }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```

Gestor de Seguridad

Pedimos la referencia al objeto remoto al enlazador

Invocamos a un método del objeto remoto

Debemos tratar la excepciones RemoteException que puede lanzarnos el objeto remoto

Caso de Estudio: Java RMI (1/)

- *Devolución de llamada*
  - *Idea: En lugar de los clientes consultar repetidas veces al servidor, este les informará únicamente si ha ocurrido un evento.*
  - *Implementación:*
    - *Cliente crea objeto remoto retrollamada.*
    - *Servidor proporciona método para registrar los objetos retrollamada. Lista de objetos retrollamada.*
    - *Cuando se produzca evento, el servidor usará el método retrollamada de los clientes registrados.*
    - *Concesiones.*
    - *Registro y desregistro.*

Caso de Estudio: Java RMI (1/)

- *Diseño en implementación de Java RMI*
  - *Empleo de Reflexión.*
    - *Distribuidor generico.*
    - *Proxys generados con rmic a partir de las clases compiladas del servidor.*
    - *Cliente envía Method con argumentos y referencia al método. El distribuidor desempaqueta y ejecuta method.invoke(objeto, argumentos) con el objeto local referenciado y los argumentos recibidos.*

Caso de Estudio: Java RMI (1/)

- *Clases de Java que dan soporte a RMI.*
  - *Sirvientes extienden o heredan de UnicastRemoteObject.*
  - *RemoteServer presenta versiones abstractas de los métodos necesarios para los servidores remotos.*
  - *Actibatable: clase abstracta que proporciona objetos activables cuando son invocados.*
  - *RemoteObject variable con referencia al objeto remoto con los siguentes métodos:*
    - *equals*
    - *toString*
    - *readObjetc, writeObject*