# Práctica 1: Proceso Básico de Desarrollo con RMI
## (Será realizado en la Sala de Computo)
## Grupo B

La empresa Centrales Eléctricas de Occidente (CEO) de Popayán, en asociación con la Universidad del Cauca (Grupos I+D GAI e IDIS) requiere implementar un sistema AMI basado en SAA (Sistema de Amarres Automático). SAA fue desarrollado por el grupo GAI del Depto de Automática y Control de la FIET. El objetivo de esta asociación es implementar un sistema AMI-SAA, que permita utilizar la red eléctrica de CEO-Popayán para la transmisión de datos usando los dispositivos desarrollados en SAA, con el fin de monitorear, gestionar y controlar de manera más eficiente el suministro de energía eléctrica en la ciudad de Popayán.

El sistema AMI-SAA está compuesto por los siguientes nodos:

PLC_TU: Son nodos instalados junto a los medidores inteligentes, y es responsable de transmitir información de consumo y configuración a los nodos PLC_MMS. La transmisión de información se realiza utilizando la red de distribución (uso del protocolo PLC)

PLC_MMS: Este tipo de nodo se encuentra instalado cerca a los transformadores de la red de distribución. Son encargados de recibir la información de consumo y configuración de los PLC_TU.

Gestor de la red SAA(GRSAA): Mediante GPRS (General Packet Radio Service), estos nodos se encargan de recibir la información consumo de energía recogido de un sector de la ciudad a través de los nodos PLC_MMS. Con esta información se realiza la facturación de consumo y se realizan estadísticas del consumo energético.

Se requiere implementar una aplicación distribuida utilizado el modelo de Java RMI de Oracle para GNU/Linux, que permita implementar la parte inteligente de los nodos que conforman el sistema AMI-SAA. Cuando el sistema AMI-SAA se pone en marcha, un operador de la empresa CEO se desplaza a las residencias y mediante un menú, registra el dispositivo PLC_TU en los nodos PLC_MMS.

Para llevar a cabo esta operación mediante un menú de texto podrá abrir sesión (ver Figura 1) y si las credenciales son correctas se debe mostrar los siguientes datos del operador (Id, Nombre completo y usuario) y posteriormente se desplegará un menú de texto Operador (ver Figura 2), mediante el cual podrá registrar los dispositivos PLC_TU.

```
=== Menu sesion===
 1. Abrir sesion
 2. Salir
 ====================
Digite opcion: 1
```

*Figura 1. Menú sesión*

```
=== Menu operador ===
1. Registrar dispositivo
2. Consultar dispositivo
3. Salir.
======================
Seleccione una opcion:
```

*Figura 2. Menú operador*

Para ingresar a la aplicación como operador CE0 debe ingresar id, usuario y clave. Las funcionalidades del Menú operador, para esta práctica, no se deben implementar. Si el usuario elige las opciones 1 o 2, se debe desplegar el mensaje:

**En construcción!**

Si al estar en el Menú sesión (ver Figura 1) se ingresan las credenciales incorrectas, se desplegará un Menú usuario (ver Figura 3), mediante el cual solo se podrá consultar los datos de un dispositivo que este ya registrado.:

```
=== Menu usuario ===
1. Consultar dispositivo
2. Salir.
======================
```

*Figura 3. Menú usuario*

Las funcionalidades del Menú usuario, para esta práctica, no se deben implementar. Si el usuario elige las opciones 1, se debe desplegar el mensaje:

**En construcción!**

## Desarrollo de la aplicación

La aplicación debe implementarse usando RMI, mediante el lenguaje de programación Java en un sistema operativo Windows. En la Figura 4 se observa un diagrama de contexto donde se muestran los nodos y usuarios que interactúan con esta aplicación. Las operaciones abrir_sesion() y consultar_usuario(), deben ser implementadas en un servidor denominado plc_mms. Cada vez que se envíe una petición desde el nodo plc_tu y se reciba una petición en el nodo plc_mms, **se deben crear ecos por medio de un mensaje en pantalla** en el cual se describe cual función se está solicitando o ejecutando.

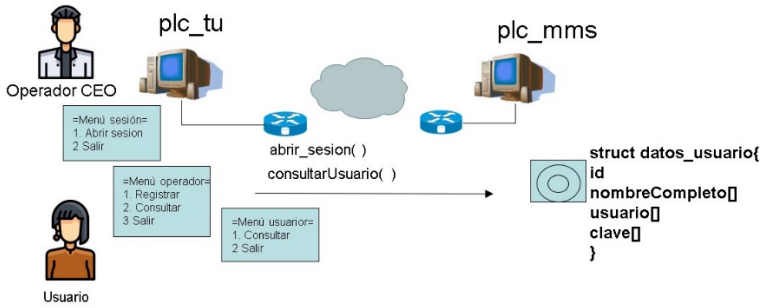


*Figura 4. Diagrama de red enriquecido del sistema*

Tener en cuenta: La solución de este ejercicio debe ser comprimida en formato rar y ser enviada a la plataforma. El nombre del archivo comprimido debe seguir el siguiente formato lsd_rmi_p1_gX.rar. Donde gX, corresponde al nombre del grupo asignado en este curso. En la carpeta se debe almacenar el archivo roles.txt donde se especificar el role de cada integrante.

Antes de iniciar, estructurar una carpeta de trabajo donde se ubicarán los archivos fuente (carpeta 'src/') y los archivos binarios (bytecode) (carpeta 'bin/'). El nombre de la carpeta de trabajo debe coincidir con el nombre del archivo comprimido. La estructura de la carpeta de trabajo es mostrada en la Figura 5.

```
lsd_rmi_p1_gX/
├── bin
└── src
    ├── plc_mms
    │   ├── dto
    │   ├── sop_rmi
    │   │   └── GestionUsuariosInt.java
    │   └── utilidades
    │       ├── UtilidadesConsola.java
    │       └── UtilidadesRegistroS.java
    ├── plc_tu
    │   └── utilidades
    │       ├── UtilidadesConsola.java
    │       └── UtilidadesRegistroC.java
    └── roles.txt
```

*Figura 5. Estructura del directorio de trabajo*

## 1.- Diseñar e implementar los componentes de la interfaz remota

Los archivos GestionUsuariosInt.java y GestionUsuariosImpl.java deben estar ubicados en la carpeta plc_mms/sop_rmi/.

a) **Definiendo la interfaz remota**: Editar el archivo GestionUsuariosInt.java y declarar 2 métodos que podrán ser invocados por el proceso cliente, como se muestran en la Figura 6.

```
package plc_mms.sop_rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

import plc_mms.dto.Usuario_DTO;

//Hereda de la clase Remote, lo cual la convierte en interfaz remota
public interface GestionUsuariosInt extends Remote {
    // Definicion del primer método remoto
    public int abrirSesion(Usuario_DTO objUsuario) throws RemoteException;
    // Definicion del segundo método remoto
    public Usuario_DTO consultarUsuario(int identificacion) throws RemoteException;
}
```

*Figura 6. Interface GestionUsuariosInt.java*

1.b) **Implementando la Interfaz Remota**: Editar el archivo GestionUsuariosImpl.java correspondiente al código del objeto servidor. En el archivo implementar los métodos de la interfaz remota GestionUsuariosInt.java. En la implementación de la clase GestionUsuariosImpl, esta clase hereda de la clase UnicastRemoteObject e implementa la interface GestionUsuariosInt como se muestra en la Figura 7. También es necesario que los métodos implementados en la clase puedan lanzar la excepción RemoteException.

```
/*Clase que implementa la interface remota GestionUsuariosInt*/
public class GestionUsuariosImpl extends UnicastRemoteObject implements GestionUsuariosInt {
    private Usuario_DTO usuario;
    private int sesionOPER;

    public GestionUsuariosImpl() throws RemoteException {…

    @Override
    public int abrirSesion(Usuario_DTO objUsuario) throws RemoteException {…

    @Override
    public Usuario_DTO consultarUsuario(int id) throws RemoteException {…

    int buscarAdmin(Usuario_DTO obj) {…
}
```

*Figura 7. Declaración de la clase GestionUsuariosImpl.java*

1.c) **Implementar la clase que encapsula los datos del usuario**: Editar un archivo denominado Usuario_DTO.java, asignar los atributos de un usuario, crear el constructor y los métodos get. Como se muestra en Figura 8:

```java
public class Usuario_DTO implements Serializable {
    private int id;
    private String nombreCompleto;
    private String usuario;
    private String clave;

    public Usuario_DTO(int id, String nombreCompleto, String usuario, String clave) {
        this.id = id;
        this.nombreCompleto = nombreCompleto;
        this.usuario = usuario;
        this.clave = clave;
    }

    public int getId() {…
    public void setId(int id) {…
    public String getNombreCompleto() {…
    public void setNombreCompleto(String nombreCompleto) {…
    public String getUsuario() {…
    public void setUsuario(String usuario) {…
    public String getClave() {…
    public void setClave(String clave) {…
}
```

*Figura 8. Declaración de la clase Usuario_DTO*

## 2. Crear el código del cliente

Nota: Se debe ubicar en la carpeta 'src/'.

2.a Escribir el código del cliente de objetos: Editar un archivo denominado ClienteDeObjetos.java, (basarse en el código fuente del demo1), e incluir el código correspondiente que permita implementar las actividades indicadas en el enunciado del problema. El archivo ClienteDeObjetos.java debe ubicarse en la carpeta plc_tu/.

En el código del cliente se debe obtener una referencia remota (es decir, una referencia que corresponda a un objeto remoto) asociada al servicio, para luego simplemente invocar de forma convencional sus métodos, aunque teniendo en cuenta que pueden generar la excepción RemoteException. Un ejemplo de cómo puede ser obtenida la referencia remota del objeto se puede ver en la Figura 9, y el método que permite consultar la referencia remota al rmiregistry es mostrado en la Figura 10. El nombre del servicio a consultar debe ser GesUsuarios.

```java
public class ClienteDeObjetos {

    private static GestionUsuariosInt objRemoto;

    Run | Debug
    public static void main(String[] args) {
        int numPuertoRMIRegistry = 0;
        String direccionIpRMIRegistry = "";

        System.out.println(x:"Cual es el la direcciOn ip donde se encuentra  el rmiregistry ");
        direccionIpRMIRegistry = plc_tu.utilidades.UtilidadesConsola.leerCadena();
        System.out.println(x:"Cual es el nUmero de puerto por el cual escucha el rmiregistry ");
        numPuertoRMIRegistry = plc_tu.utilidades.UtilidadesConsola.leerEntero();

        objRemoto = (GestionUsuariosInt) UtilidadesRegistroC.obtenerObjRemoto(direccionIpRMIRegistry,
                numPuertoRMIRegistry,
                identificadorObjetoRemoto:"GesUsuarios");
        MenuPrincipal();

    }
```

*Figura 9. Ejemplo para obtener del rmiregistry una referencia remota del servicio.*

```java
public class UtilidadesRegistroC {
    public static Remote obtenerObjRemoto(String dirIPNS, int puertoNS, String identificadorObjetoRemoto) {
        Remote objetoObtenido = null;
        String URLRegistro = "rmi://" + dirIPNS + ":" + puertoNS + "/" + identificadorObjetoRemoto;
        try {
            objetoObtenido = Naming.lookup(URLRegistro);
        } catch (NotBoundException e) {
            System.out.println(x:"Error, objeto remoto no localizado");
        } catch (MalformedURLException e) {
            System.out.println(x:"Error, url inválida");
        } catch (RemoteException e) {
            System.out.println("Excepcion en obtencion del objeto remoto" + e);
        }
        return objetoObtenido;
    }

}
```

*Figura 10. Ejemplo del método que permite obtener del rmiregistry una referencia remota del servicio*

En la Figura 10 el programa usa el método estático lookup() para obtener del rmiregistry una referencia remota del servicio. Los argumentos de entrada del método main() corresponden a: el nombre de la maquina donde se encuentra el proceso rmiregistry y el puerto donde está escuchando el proceso rmiregistry.

Observe el uso de cast para adaptar la referencia devuelta por lookup(), que corresponde a la interfaz Remote, al tipo de interfaz concreto (GestionUsuariosInt). Una vez obtenida la referencia remota, la invocación del método es convencional, requiriendo el tratamiento de las excepciones que puede generar.

- Compilar el código fuente del objeto de implementación (GestionUsuariosImpl.java) con la herramienta javac (Compilador java).

```
javac -d ../bin plc_mms/sop_rmi/*.java
```

3.a. Implementar el servidor de objetos (SdO): Editar un archivo denominado ServidorDeObjetos.java, (basarse en el código fuente demo01) e incluir el código correspondiente que permita instanciar el objeto servidor y registrarlo en el N_S, un ejemplo de cómo puede ser registrado un objeto en el n_s puede observarse en la Figura 11, en la cual los argumentos de entrada del método main() corresponden a: el nombre de la maquina donde se encuentra el proceso rmiregistry y el puerto donde está escuchando el proceso rmiregistry. El método mostrado en la Figura 11 se apoya de los métodos que permiten crear una referencia remota al n_s, el cual es mostrado en la Figura 12, y registrar un objeto remoto en el n_s como es el mostrado en la Figura 13. El nombre del objeto remoto a registrar debe ser GesUsuarios.

```
public class ServidorDeObjetos {
    Run | Debug
    public static void main(String args[]) throws RemoteException {

        int numPuertoRMIRegistry = 0;
        String direccionIpRMIRegistry = "";

        System.out.println(x:"Cual es el la direcciOn ip donde se encuentra  el rmiregistry ");
        direccionIpRMIRegistry = UtilidadesConsola.leerCadena();
        System.out.println(x:"Cual es el nUmero de puerto por el cual escucha el rmiregistry ");
        numPuertoRMIRegistry = UtilidadesConsola.leerEntero();

        GestionUsuariosImpl objRemoto = new GestionUsuariosImpl();// se leasigna el puerto de escucha del objeto remoto

        try {
            UtilidadesRegistroS.arrancarNS(numPuertoRMIRegistry);
            UtilidadesRegistroS.RegistrarObjetoRemoto(objRemoto, direccionIpRMIRegistry, numPuertoRMIRegistry,
                    identificadorObjetoRemoto:"GesUsuarios");
        } catch (Exception e) {
            System.err.println("No fue posible arrancar el NS o Registrar el objeto remoto" + e.getMessage());
        }
    }
}
```

*Figura 11. Implementación de la clase ServidorDeObjetos que permite instanciar un objeto de la clase GestionUsuariosImpl y registrarlo en el n_s.*

```java
public static void arrancarNS(int numPuertoNS) throws RemoteException {
    try {

        Registry registro = LocateRegistry.getRegistry(numPuertoNS);
        String[] vector = registro.list();
        System.out.println(x:"nombres de objetos remotos registrados ");
        for (String IDObjetoRemoto : vector) {
            System.out.println("nombre : " + IDObjetoRemoto);
        }
        System.out.println("El rmiRegistry se ha obtenido y se encuentra escuchando en el puerto: " + numPuertoNS);

    } catch (RemoteException e) {
        System.out.println("El rmiRegistry no se localizó en el puerto: " + numPuertoNS);

        Registry registro = LocateRegistry.createRegistry(numPuertoNS);
        System.out.println("El registro se ha creado en el puerto: " + numPuertoNS);
    }

}
```

*Figura 12. Método que permite crear una referencia a un n_s que este activo, o crear un n_s si no hay ninguno activo.*

```java
public static void RegistrarObjetoRemoto(Remote objetoRemoto, String dirIPNS, int numPuertoNS,
        String identificadorObjetoRemoto) {
    String UrlRegistro = "rmi://" + dirIPNS + ":" + numPuertoNS + "/" + identificadorObjetoRemoto;
    try {
        Naming.rebind(UrlRegistro, objetoRemoto);
        System.out.println("Se realizó el registro del objeto remoto en el ns ubicado en la dirección: " + dirIPNS
                + " y " + "puerto" + numPuertoNS);
    } catch (RemoteException e) {
        System.out.println(x:"Error en el registro del objeto remoto");

    } catch (MalformedURLException e) {
        System.out.println(x:"Error, url inválida");
    }

}
```

*Figura 13. Método que permite registrar un objeto remoto en el n_s.*

La parte principal de este programa descrita en la clase ServidorDeObjetos, está incluida en la sentencia try y consiste en crear un objeto de la clase que implementa el servicio remoto y darle de alta en el rmiregistry usando el método estático rebind() que permite especificar la operación usando un formato de tipo URL.

## 3 b. Compilar las clases del servidor y cliente

- Compilar las clases del Servidor de Objetos y el Cliente de Objetos con la herramienta javac.

Nota: Se debe ubiar la carpeta 'src/'.
Para compilar el servidor de objetos (SdO):

```
javac -d ../bin plc_mms/*.java
```

Para compilar el cliente de objetos (CdO):

```
javac -d ../bin plc_tu/*.java
```

4.- Lanzar la aplicación.

Nota: Se debe ubicar la carpeta 'bin/'.

a. Ejecutar el servidor de objetos (SdO):
Comando general:

```
java Nombre_clase_servidor_obj
```

Por ejemplo, si la clase pertenece al paquete 'plc_mms':

```
java plc_mms.ServidorDeObjetos
```

b. Ejecutar el cliente de objetos (CdO):
Comando general:

```
java Nombre_clase_cliente_obj
```

Por ejemplo, si la clase pertenece al paquete 'cliente':

```
java plc_tu.ClienteDeObjetos
```

Todos los archivos entregados deben estar ubicados en las carpetas ***plc_tu/*** y ***plc_mms/*** respectivamente. Los archivos fuentes no deben estar vinculados con ningún IDE de desarrollo. El servidor de objetos y el cliente de objetos deben inicialmente solicitar los siguientes parámetros el nombre del equipo o dirección ip y puerto donde se encuentra ejecutándose el proceso rmiregistry.