Práctica 2: DEFINICIÓN DE INTERFACES IDL

Práctica 2. (Será realizada en la Sala de Computo)

El objetivo de esta práctica es crear una completa aplicación basada en CORBA utilizando el lenguaje de definición de interfaces (IDL) y el compilador de java JIDL para generar los stubs y skeleton. En esta práctica se requiere implementar una aplicación distribuida basada en Java IDL, la cual permita a un usuario registrar un paciente, consultar los datos de un paciente registrado en una habitación particular (en el Servidor de Alertas) y reportar el registro a un Servidor de Notificaciones. Los datos a registrar corresponden a: nombre del paciente, apellido del paciente, número de la habitación del paciente y edad del paciente. Estas operaciones serán controladas por el cliente mediante un Menú, tal como se muestra la Figura 1.

========== Menú ==========

1. *Registrar paciente*
2. *Consultar paciente*
3. *Salir*

========== ==========

Digite opción:

Figura 1: Menú de la aplicación

La primera opción permite registrar todos los datos de un paciente. En el lado del Servidor de Alertas, los pacientes serán almacenados en un arrrayList, debe enviarse al Servidor de Notificaciones una notificación del éxito o fracaso del registro de un paciente. La máxima cantidad de pacientes a registrar será 5.

En las Figuras 2 y 3, se muestran ejemplos de los mensajes de notificación de éxito y fracaso que se deben mostrar en el Servidor de Notificaciones.

```
======== Notificación ==========
Se registro el paciente NNN
en la habitación ## EXITOSAMENTE
================================
```

Figura 2: Mensaje caso exitoso

```
======== Notificación ==========
FALLO el registro del paciente NNN
en la habitación ##
================================
```

Figura 3: Mensaje caso no exitóso

En la Figura 4, se muestra un diagrama de red con los métodos que puede invocar el cliente de objetos.

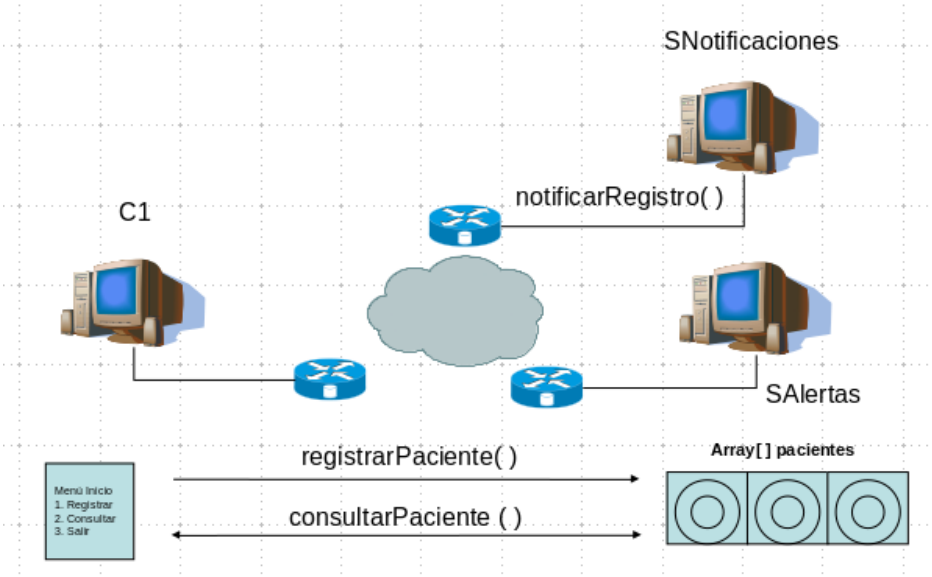

Figura 4: Diagrama de contexto

Restricciones

En la implementación del servidor se deberá validar que no existan dos pacientes con la misma habitación. La estructura de datos utilizada para almacenar los pacientes será un arrayList.

Tener en cuenta: La solución de este ejercicio debe ser comprimida en formato rar y enviada a la plataforma. El nombre del archivo comprimido debe seguir el siguiente formato lsd_corba_p2_apellidoN1_apellidoN2.rar. Donde apellidoN1 corresponde al primer apellido de uno de los integrantes y apellidoN2 corresponde al primer apellido del segundo integrante del grupo. En la carpeta se deben especificar el rol de cada integrante.

Organizar los archivos de acuerdo a la estructura de directorios que se muestra en la Figura 5

```
lsd_corba_p2_ape1N_ape2N/
├── bin
└── src
    ├── cliente
    ├── servidorDeAlertas
    │   ├── dto
    │   ├── servidor
    │   └── sop_corba
    └── servidorDeNotificaciones
        ├── dto
        ├── servidor
        └── sop_corba
```

Figura 5. Estructura del directorio de trabajo.

1. Guía de compilación y ejecución.

El primer paso para crear aplicaciones basadas en CORBA es especificar las operaciones remotas utilizando el lenguaje de definición de interfaces (IDL), el cual puede ser mapeado a una variedad de lenguajes de programación.

La interface IDL para el servidor de Alertas es la siguiente:

```
module sop_corba{

  interface GestionPacientes{
          struct pacienteDTO{
                  string nombre;
                  string apellido;
                  long numeroHabitacion;
                  long edad;
          };

    void registrarPaciente(in pacienteDTO objPacienteo,out boolean resultado);
    boolean buscarPaciente(in long numeroHabitacion,out pacienteDTO objPacienteResultado);
   };
 };
```

La interface IDL para el servidor de Notificaciones es la siguiente:

```
module sop_corba{

  interface GestionNotificaciones{
struct ClsMensajeNotificacionDTO{
                  string nombre;
                  string apellido;
                  long numeroHabitacion;
                  long estado;
          };

    void notificarRegistro(in ClsMensajeNotificacionDTO objNotificacion);
   };
 };
```

Compilar la interface idl de los servidores de Alertas y Notificaciones utilizando los siguientes comandos:
Servidor de Alertas

idlj -fallTIE -pkgPrefix sop_corba servidorDeAlertas alertas.idl

Servidor de Notificaciones

idlj -fallTIE -pkgPrefix sop_corba servidorDeNotificaciones notificaciones.idl

El compilador jidl de java permite convertir una interface definida en lenguaje IDL a correspondientes interfaces java, clase y métodos, los cuales pueden ser utilizados para implementar el código del cliente y servidor.

Este comando generará varios archivos. Revisar el contenido del directorio de trabajo desde donde ejecuto este comando. Un nuevo subdirectorio será creado, debido a que el módulo 'sop_corba' será mapeado como un paquete.

La opción - **fallTIE** genera los archivos de soporte que son utilizados para implementar el Modo TIE. De esta manera una clase GestionPacientesPOATie.java es generada. El constructor de esta clase toma un delegado. Se debe proporcionar la implementación de una clase delegada.. Además genera el paquete **GestionPacientesPackage, el cual contiene clases que proveen operaciones para leer y escribir sobre argumentos de las operaciones remotas.**

2. Utilizar y completar las plantillas del cliente y el servidor

Utilizar las plantillas que se encuentran almacenadas en el sitio destinado al curso. Crear los archivos fuente para el lado cliente ClienteDeObjetos.java, para el lado servidor de alertas ServidorDeObjetos.java y GestionPacientesImpl.java, y para el lado servidor de notificaciones ServidorDeObjetos.java y GestionNotificacionesImpl.java, teniendo en cuenta que:

ClienteDeObjetos.java: Implementa la lógica de negocio del cliente, consulta una referencia del servant y llama las operaciones del objeto remoto a través de un menú.

GestionPacientesImpl.java: Se implementa el objeto corba en Modo Delegación, implemeta la interfaz **GestionPacientesOperations.java**. Implementa la lógica de las operaciones definidas en la interface IDL. En la clase debe crear un método adicional que permita obtener la referencia remota del servidor de notificaciones. En la figura 5 se muestra un ejemplo de un método para registrar el paciente y en la figura 6 la implementación del método que permite obtener la referencia remota del servidor de notificaciones

```
@Override
public void registrarPaciente(pacienteDTO objPacienteo, BooleanHolder resultado) {
    System.out.println("invocando al método registrar paciente");
    resultado.value=listaPacientes.add(objPacienteo);
    ClsMensajeNotificacionDTO objMensaje= new ClsMensajeNotificacionDTO(objPacienteo.nombre, objPacienteo.apellido, objPacienteo.numeroHabitacion, 1);
    referenciaNotificaciones.notificarRegistro(objMensaje);
}
```

Figura 5. Ejemplo del método registrar paciente

```
public void obtenerLaReferenciaRemotaDelServidorDeNotificaciones(String direccionIPNS, String puertoNS)
{
    try {
        String[] vec = new String[4];
        vec[0] = "-ORBInitialPort";
        vec[1] = direccionIPNS;
        vec[2] = "-ORBInitialPort";
        vec[3] = puertoNS;

        // crea e inicia el ORB
        ORB orb = ORB.init(vec, null);

        // obtiene la base del naming context
        org.omg.CORBA.Object objRef= orb.resolve_initial_references("NameService");
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // *** Resuelve la referencia del objeto en el N_S ***
        String name = "objNotificaciones";
        referenciaNotificaciones = GestionNotificacionesHelper.narrow(ncRef.resolve_str(name));
```

Figura 6. Método que permite obtener la referencia remota del servidor de notificaciones

GestionNotificacionesImpl.java: Se implementa el objeto corba en Modo Delegación, implemeta la interfaz **GestionNotificacionesOperations.java**. Implementa la lógica de las operaciones definidas en la interface IDL.

ServidorDeObjetos.java: Implementa la lógica para crear el servant y registrarlo en el ns. En la figura 7 se muestra un ejemplo de la implementación del servidor de objetos. En color rojo se encuentra la invocación del método que permite obtener la referencia remota del servidor de notificaciones y en color verde el paso del servant al POA.

```
ORB orb = ORB.init(vec, null);

System.out.println("2. Obtiene la referencia al poa raiz, por medio del orb ");
org.omg.CORBA.Object objPOA = null;
objPOA=orb.resolve_initial_references("RootPOA");
POA rootPOA = POAHelper.narrow(objPOA);
System.out.println("3. Activa el POAManager");
rootPOA.the_POAManager().activate();

System.out.println("4. Crea el objeto servant");
GestionPacientesImpl ObjServant = new GestionPacientesImpl();
ObjServant.obtenerLaReferenciaRemotaDelServidorDeNotificaciones(vec[1], vec[3]);
System.out.println("5. Crea el objeto tie y se registra una referencia al objeto servant mediante el contructor");
GestionPacientesPOATie objTIE= new GestionPacientesPOATie(ObjServant);
System.out.println("6. Obtiene la referencia al orb ");
GestionPacientes referenciaRemota = objTIE._this(orb);

System.out.println("7. Obtiene una referencia al servicio de nombrado por medio del orb");
org.omg.CORBA.Object objRefNameService = orb.resolve_initial_references("NameService");
System.out.println("8. Convierte la ref genérica a ref de NamingContextExt");
NamingContextExt refContextoNombrado = NamingContextExtHelper.narrow(objRefNameService);
System.out.println("9.Construir un contexto de nombres que identifica al servant");

String name = "objPaciente";
NameComponent path[] = refContextoNombrado.to_name( name );

System.out.println("10.Realiza el binding de la referencia de objeto en el N_S");
refContextoNombrado.rebind(path, referenciaRemota);

System.out.println("El Servidor esta listo y esperando ...");
orb.run();
```

Figura 7. Ejemplo de la implementación del servidor de objetos.

Distribuir estos archivos fuente en los subdirectorios pertinentes de src/. Para registrar el servant del lado servidor de alertas y consultarlo se utilizara como nombre "**objPaciente**". Para registrar el servant del lado servidor de notificaciones y consultarlo se utilizara como nombre "**objNotificaciones**".

3. Compilar los códigos fuente

Compilar los códigos fuente del ítem 2 y los stubs y skeleton que fueron generados por el precompilador idlj en el paso 1, por medio de los siguientes comandos:

Para compilar los soportes:
Servidor de alertas:
javac -d ../bin servidorDeAlertas/sop_corba/*.java
Servidor de notificaciones:
javac -d ../bin servidorDeNotificaciones/sop_corba/*.java

Para compilar los fuentes del servidor:

Servidor de alertas:
javac -d ../bin servidorDeAlertas/servidor/*.java

Servidor de notificaciones:
javac -d ../bin servidorDeNotificaciones/servidor/*.java

Para compilar los fuentes del lado cliente:
javac -d ../bin cliente/*.java

4. Lanzar el Object Request Broker Daemon (ORBD)

Antes de correr el cliente y servidor, se debe correr el ORBD, el cual es un servicio de nombrado. Un servicio de nombres de CORBA es un servicio que permite a una referencia de un objeto CORBA asociarla con un nombre. El nombre del enlace se puede almacenar en el servicio de nombres, y un cliente puede proporcionar el nombre para obtener la referencia al objeto deseado.

a. Lanzar el orbd mediante el comando:

orbd –ORBInitialHost dir_IP -ORBInitialPort N_Puerto

b. Ubicarse en el directorio bin/. Lanzar el Servidor de Notificaciones mediante el comando:

java servidorDeNotificaciones.Servidor -ORBInitialHost dir_IP -ORBInitialPort N_Puerto

c. Ubicarse en el directorio bin/. Lanzar el Servidor de Alertas mediante el comando:

java servidorDeAlertas.Servidor -ORBInitialHost dir_IP -ORBInitialPort N_Puerto

dir_IP y N_Puerto son la dirección ip y puerto donde se encuentra escuchando el servicio orbd.

d. Ubicarse en el directorio bin/. Lanzar el Cliente mediante el comando:

java cliente.Cliente -ORBInitialHost dir_IP -ORBInitialPort N_Puerto

dir_IP y N_Puerto son la dirección ip y puerto donde se encuentra escuchando el servicio orbd.