Práctica 3: CALLBACKS en CORBA

Ejercicio (Será realizado en la Sala de Computo)

Diseñar un programa que permita enviar los datos básicos de un paciente y el indicador de su frecuencia cardiaca a un servidor de alertas, el cual analizara si la frecuencia cardiaca se encuentra fuera de un rango normal. Cada vez que se genere una alerta, el servidor de alertas debe enviar un mensaje de alarma a la habitación del paciente, para informar a los acompañantes el cambio de estado en los signos vitales del paciente. El mensaje a mostrar en la habitación del paciente es el siguiente:

> El paciente que se encuentra en la habitación [Número] presenta una frecuencia cardiaca de [Indicador de la frecuencia] la cual está fuera del rango normal

El sistema debe gestionar la información utilizando el modelo de CORBA e implementándolo en lenguaje java. El rango normal de la frecuencia cardiaca por edad se muestra a continuación:

FRECUENCIA CARDIACA		
Grupo	**Edad**	**Latidos por minuto**
Adolescente	**13 – 16 años**	**70-80**
Adulto	**16 años y más**	**60-80**

En el cliente se debe crear un menú, tal como muestra la Figura 1:

```
=== Gestión anteproyectos ===
1. Registrar usuario
2. Salir
============================
```

Figura 1. Contenido del menú

La primera opción permite registrar al paciente y enviar los indicadores: no de habitación, edad, y frecuencia cardiaca al servidor.

Para observar el funcionamiento del programa debe ejecutar el ServidorDeObjetos y varios ClientesDeObjetos.

Desarrollo de la aplicación

La aplicación debe implementarse usando Java IDL, en un sistema operativo Linux o Windows. En la siguiente figura se observa un diagrama de contexto que muestra las operaciones que puede realizar el cliente. Las operaciones debe ser implementadas en un servidor, cada vez que se realice una petición en el cliente y se reciba una petición en el servidor, se debe crear un eco por medio de un mensaje en

pantalla en el cual se describe cual método se está invocando o ejecutando

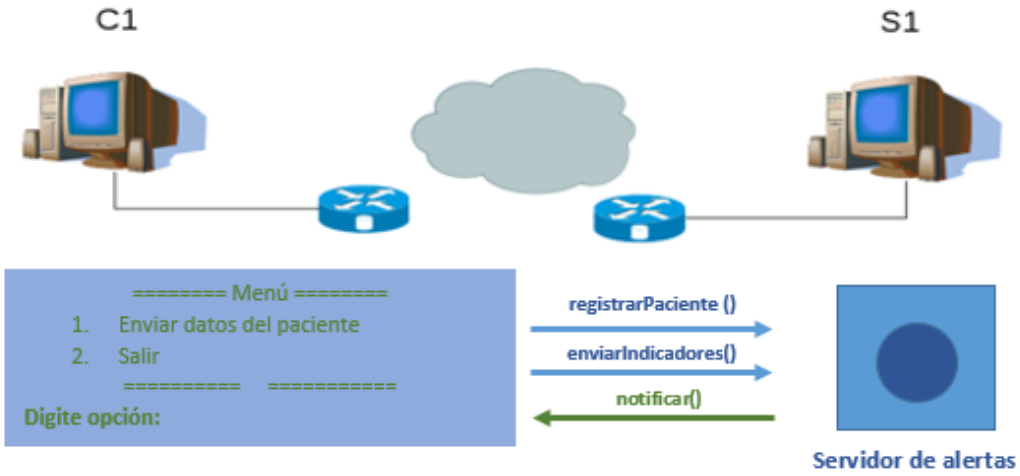

Figura 2. Diagrama de contexto de la aplicación

Antes de iniciar el ejercicio se debe estructurar un directorio de trabajo donde se ubicaran los archivos fuente (directorio 'src') y los archivos binarios (bytecode) (directorio bin) ver Figura 3.

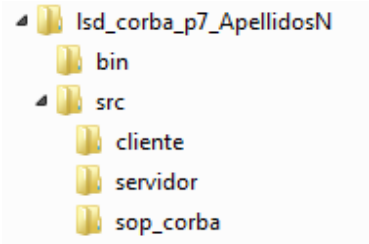

Figura 3. Estructura de directorios de la aplicación

1. **Diseñar e implementar los componentes de la interfaz**

Para la implementación de esta aplicación el estudiante debe definir tres paquetes de trabajo: *cliente*, *servidor* y *sop_corba*. En dichos paquetes debe distribuir de manera adecuada los diferentes archivos que se indica en esta guía.

1a , 1b-Definiendo la interfaz remota del cliente y servidor: Editar el archivo g*estionusuarios*.idl. En el se incluye la definición de las interfaces *PacienteCllbckInt y GestionAlertasInt, y el DTO PacienteDTO,*

```
module sop_corba{
   interface PacienteCllbckInt{
         string notificar(in long numeroHabitacion,in long frecCardiaca);
   };
   struct PacienteDTO{
            string nombre;
            string apellido;
            long numeroHabitacion;
            long edad;
            PacienteCllbckInt pacbck;
   };
   interface GestionAlertasInt{
       boolean registrarPaciente(in PacienteDTO pacientedto);
       boolean enviarIndicadores(in long numHabitacion,in long frecuenciaCardiaca) ;
   };
};
```

1.c Implementando la Interfaz UsuarioCllbck:
Crear y editar el archivo *PacienteCllbckImpl*.java donde se deben implementar el método notificar (ver Figura 3), el cual permite notificar a un paciente que la frecuencia cardiaca esta por fuera del rango normal.

```
package cliente;

Import sop_corba.*;

public class PacienteCllbckImpl extends PacienteCllbckIntPOA{

}
```

1.d Implementando la Interfaz Servidor Remota:

Crear y editar el archivo *GestiónAlertasImpl.java* donde se deben implementar los métodos especificados en la interfaz remota *GestionAlertasInt* (ver Figura 4). En esta clase se debe implementar:

El método *registrarUsuario*(): Almacena el objeto PacienteDTO, dentro del cual se almacena la referencias del objeto callback en un arreglo o vector.
El método *enviarIndicadores*(): Permite enviar los indicadores número de habitación y frecuencia al servidor de alertas.

```
package cliente;

Import sop_corba.*;

public class GestionAlertasImpl extends  GestionAlertasIntPOA{

}
```

Proporcionar una clase servidora y una clase cliente, basado en las plantillas publicadas en el sitio del curso. Organizar los archivos en el directorio src (subdirectorios cliente/ y servidor/). El nombre del servicio debe ser registrado con el nombre de '**ges-alertas**'. Tener en cuenta que en la lógica del cliente se debe incorporar la creación de un POA, con el fin que este genere la referencia remota del objeto callback. En la figura 4 se muestra la creación del objeto callback del lado del cliente, generación de su referencia mediante un POA y consulta al ns del objeto remoto denominado "ges-alertas"

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

// obtiene el contexto de nombrado raiz de Name Service
   org.omg.CORBA.Object ref=orb.resolve_initial_references("NameService");
// Usa NamingContextExt
   NamingContextExt ncref = NamingContextExtHelper.narrow(ref);

  POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
  rootPOA.the_POAManager().activate();
// instancia el servant
  PacienteCllbckImpl pacbck = new PacienteCllbckImpl();
  // obtiene la referencia del rootpoa & activate el POAManager
  org.omg.CORBA.Object ref1 = rootPOA.servant_to_reference(pacbck);
  PacienteCllbckInt href1 = PacienteCllbckIntHelper.narrow(ref1);

// *** Resuelve la referencia del objeto en el N_S ***
  String name = "ges-alertas";
  refObjetoRemoto = GestionAlertasIntHelper.narrow(ncref.resolve_str(name));
```

Figura 4. Descripción de la creación del objeto callback del lado del cliente, generación de su referencia mediane un POA y consulta al ns del objeto remoto denominado "ges-alertas"

2. Generar los stubs y compilar los códigos fuentes y

2.a) Generar los soportes CORBA: Compilar la interfaz idl:

```
idlj -fall alertas.idl
```

Este comando generará varios archivos. Chequear el contenido del directorio de trabajo src/sop_corba/. En el subdirectorio sop_corba estarán ubicados los fuentes generados por idlj, debido a que el módulo 'sop_corba' será mapeado como un paquete.

2.b. Compilar los códigos fuente;

- Compilar los códigos fuente ubicados en los directorios cliente/ y servidor/; y los códigos fuente generados en el item anterior por el precompilador idlj utilizando los siguientes comandos:

```
javac -d ../bin cliente/*.java
javac -d ../bin servidor/*.java
javac -d ../bin sop_corba/*.java
```

2. Ejecutar la aplicación.

1. Lanzar el orbd, el cual es un demonio conteniendo un Servicio de Nombrado Transiente, un Servicio de Nombrado Persistente, un Servicio Bootstrap, y un Gestor de Servidor.

```
orbd –ORBInitialHost dir_IP -ORBInitialPort N_Puerto
```

La opción -ORBInitialPort específica el puerto en el cual el N_S debería ser iniciado (Se recomienda escoger un valor mayor a 1024).

La opción -ORBInitialHost específica la máquina (especificar un dirección IP o nombre de red de la máquina) en la cual el N_S esta ejecutándose.

2. Lanzar el Servidor:

```
java servidor.ServidorDeObjetos –ORBInitialHost dir_IP -ORBInitialPort N_Puerto
```

3. Lanzar el Cliente:

```
java cliente.ClienteDeObjetos –ORBInitialHost dir_IP -ORBInitialPort N_Puerto
```

```
Nota: Se deben ejecutar 3 clientes, en el siguiente orden:
a) 2 clientes se registran y no reciben las notificaciones
b) 1 cliente se registra y recibe las notificaciones
```