



**UNIVERSIDAD DEL CAUCA**  
**FACULTAD DE INGENIERIA ELECTRÓNICA Y TELECOMUNICACIONES**  
**PROGRAMA DE INGENIERÍA DE SISTEMAS**

---

## **PRACTICA DE LABORATORIO No. 4**

### **Patrón de arquitectura: Cliente-Servidor**

#### **OBJETIVO**

- Construir aplicaciones mediante el patrón Cliente-Servidor.

#### **INTRODUCCIÓN**

La arquitectura cliente-servidor (C/S) es un modelo de diseño de software desde el punto de vista de componentes y conectores en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados componentes **servidores(S)**, y los demandantes, llamados componentes **clientes(C)**. Un cliente realiza **peticiones** a otro programa, el servidor, quien le da respuesta. Esta comunicación es posible ya que éstos componentes se comunican a través de un conector de tipo petición-respuesta (/) basado en algún protocolo de comunicación normalmente conocido como el *middleware*. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadores.

Una explicación completa del patrón está en el libro Software Architecture In Practice de Len Bass, página 241 (El libro está en la plataforma virtual):

#### Client-Server Pattern

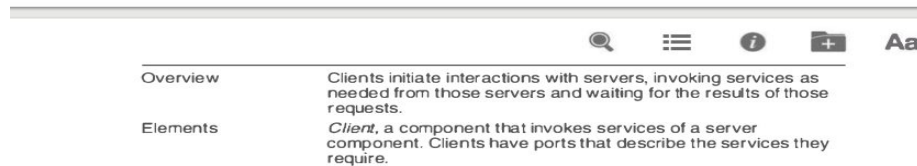
**Context:** There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.

**Problem:** By managing a set of shared resources and services, we can promote modifiability and reuse by factoring out common services and having to modify these in a single location, or a small number of locations. We want to improve scalability and availability by centralizing the control of these resources and services, while distributing the resources themselves across multiple physical servers.

**Solution:** Clients interact by requesting services of servers, which provide a set of services. Some components may act as both clients and servers. There may be one central server or multiple distributed ones.

The [client-server pattern](#) solution is summarized in [Table 13.5](#); the component types are *clients* and *servers*; the principal connector type for the [client-server pattern](#) is a data connector driven by a request/reply protocol used for invoking services.

Table 13.5. [Client-Server Pattern Solution](#)



Overview	Clients initiate interactions with servers, invoking services as needed from those servers and waiting for the results of those requests.
Elements	<i>Client</i> , a component that invokes services of a server component. Clients have ports that describe the services they require.

## REQUISITOS

Antes de abordar el tema de Cliente/Servidor, se requieren conocer dos temas fundamentales: **Hilos y Sockets**.

Los **hilos** (en inglés Thread) son un trozo de código que puede ser ejecutado al mismo tiempo que otro. Esto nos permite construir aplicaciones multitarea. En esencia, la multitarea nos permite ejecutar varios procesos a la vez; es decir, de forma concurrente y por tanto nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes. El siguiente tutorial explica dos formas de crear hilos en java: [Tutorial de hilos](#).

Por otro lado, los **sockets** son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor) Java por medio de la librería *java.net* nos provee dos clases: *Socket* para implementar la conexión desde el lado del cliente y *ServerSocket* que nos permitirá manipular la conexión desde el lado del servidor. El [siguiente tutorial](#) explica el funcionamiento básico de sockets. Finalmente, el [siguiente tutorial](#) crea un chat con Sockets e Hilos.

Una vez comprendidos los temas de hilos y sockets ya puedes empezar con el taller.

## APLICACIÓN DE EJEMPLO: GESTION DE CLIENTES DE UNA AGENCIA DE VIAJES

Para explicar el funcionamiento del patrón Cliente-Servidor se va a construir una sencilla aplicación en Java que permite gestionar clientes (agregar, editar, eliminar y consultar) de una agencia de viajes (Ver la Figura siguiente).

Ver Clientes

Id	Nombres	Apellidos	Dirección	Celular	Email	Sexo
98000001	Andrea Maria	Sanchez	Calle 14 No 11-1...	3145878752	andrea@hotmail....	Femenino
98393281	Wilson Libardo	Pantoja Yepez	Santa Barbara	3148556447	wpantoja@gmail....	Masculino

Cerrar

Clientes

Este ejercicio es cliente/servidor y utiliza un miniframework MVC.  
La aplicación cliente se conecta al servidor (Registraduría) mediante Sockets.  
Si el id del cliente no está en la bd de la agencia, la solicita a la registraduría.  
La aplicación, además, tiene persistencia mediante Jdbc.  
En la bd de la registraduría están las cédulas desde 98000001 hasta 98000010.

Número de identificación:

98000002

Nombres:

Libardo

Apellidos:

Pantoja

Dirección (opcional):

Santa Barbar Popayan

Celular (opcional):

3141257845

Email (opcional):

libardo@gmail.com

Sexo:

Masculino

Los datos del cliente fueron consultados en la registraduría.

Grabar

Eliminar

Cerrar

El cliente es la aplicación **AgenciaViajes** y el servidor es la aplicación **Registraduría**. Varias agencias de viajes pueden conectarse a un servidor de la Registraduría y consultar clientes. El cliente y servidor pueden ejecutarse en máquinas distintas, o en la misma máquina (localhost).

La aplicación tiene las siguientes características:

1. El cliente y el servidor se comunican a través del protocolo TCP mediante Sockets.

2. La aplicación maneja datos persistentes.
3. El Servidor devuelve datos a los clientes mediante objetos serializados en formato [Json](#).
4. La aplicación es multihilo, cuando un cliente se conecta al servidor, la petición se atiende en su propio hilo de ejecución.

El usuario teclea un número de identificación del cliente, tan pronto el cursor deja el foco de la caja de texto (Focus Lost), el sistema busca la identificación en la base de datos de la agencia de viajes. Si la encuentra carga los demás datos y activa los botones Grabar (editar cliente) y Eliminar. Si no lo encuentra, hace una conexión al servidor de la registraduría para consultar el cliente, si lo encuentra devuelve sus datos. En el caso que no encuentre la identificación, se activa el botón Grabar (agregar cliente), que permitirá agregar el nuevo usuario al sistema.

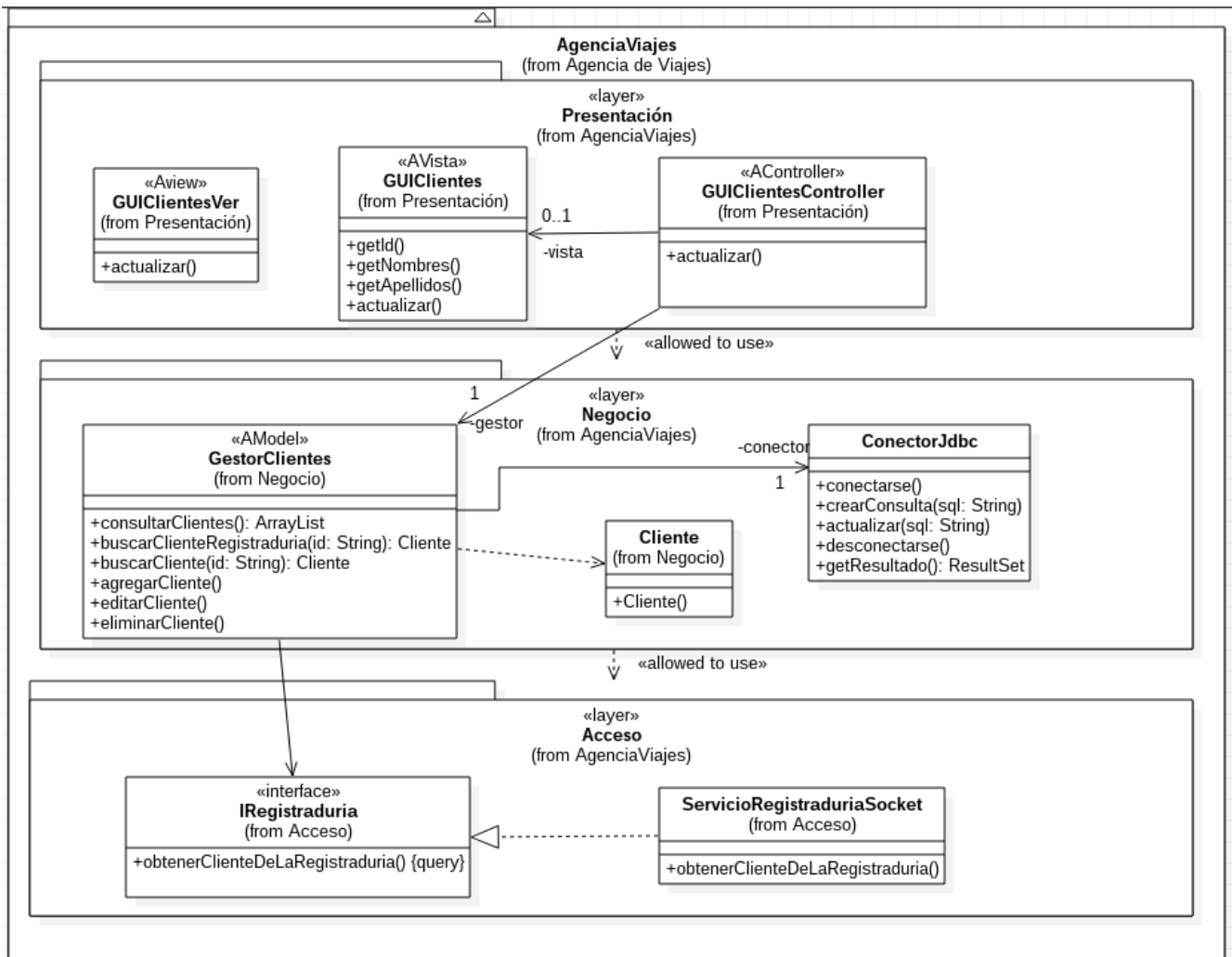
Antes de abrir la aplicación, debe hacer el siguiente cambio en el código fuente. Editar el archivo `agenciaviajes.negocio.ConectorJdbc` y cambiar la ubicación de la base de datos:

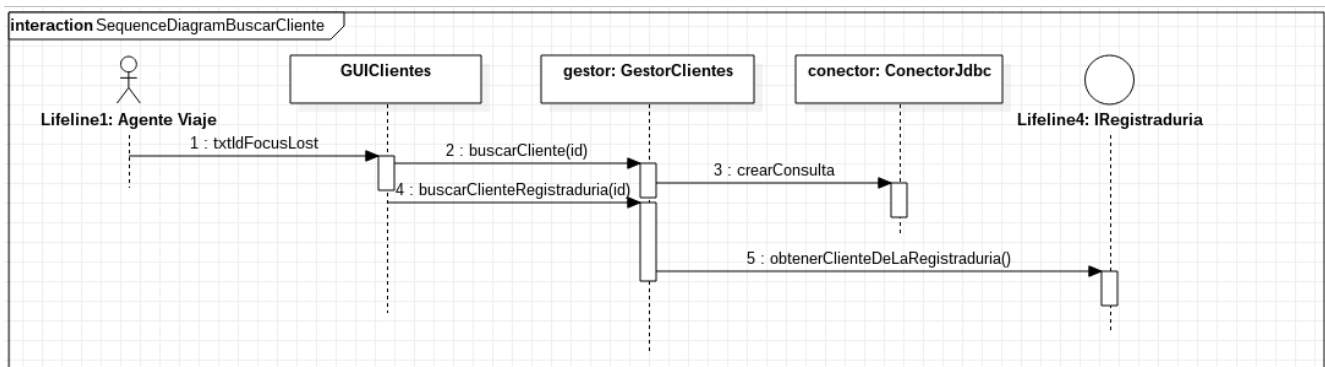
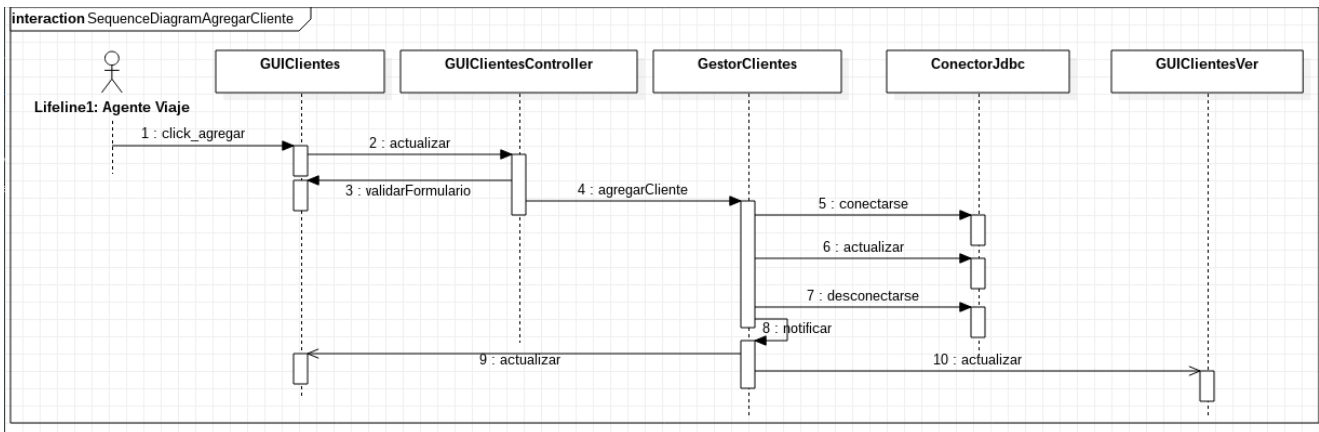
```
//Cambie la URL de su bd local, por ejemplo, si tiene Windows, sería algo como:  
private final String URL = "jdbc:hsqldb:file:C:\\Taller4-cliente-servidor\\AgenciaViajes\\bd\\clientes;hsqldb.lock_file=false";
```

La aplicación utiliza el motor de bases de datos [hsqldb](#), que es un gestor de base de datos escrito en java y que es fácil de configurar. La base de datos hace parte del código fuente de la aplicación, no requiere instalación y configuración por separado.

Para correr el programa, primero ejecute el servidor (Registraduría) y luego el cliente (AgenciaViajes).

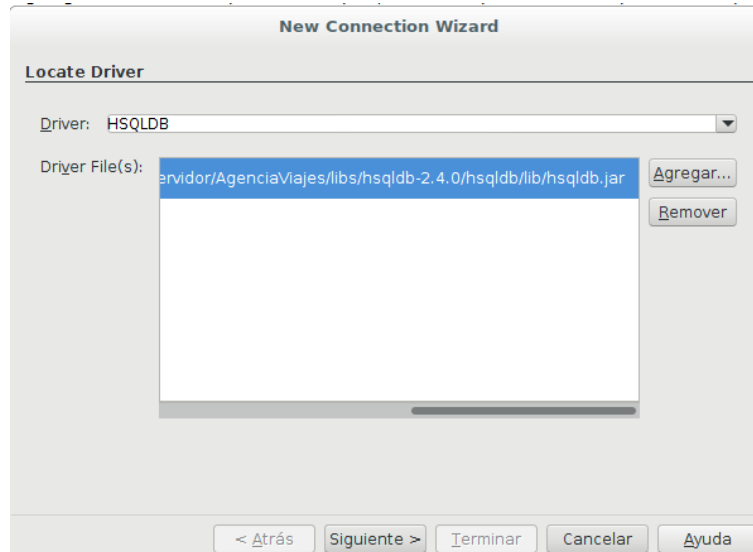
A continuación los diagramas de clases y de secuencia de la aplicación.



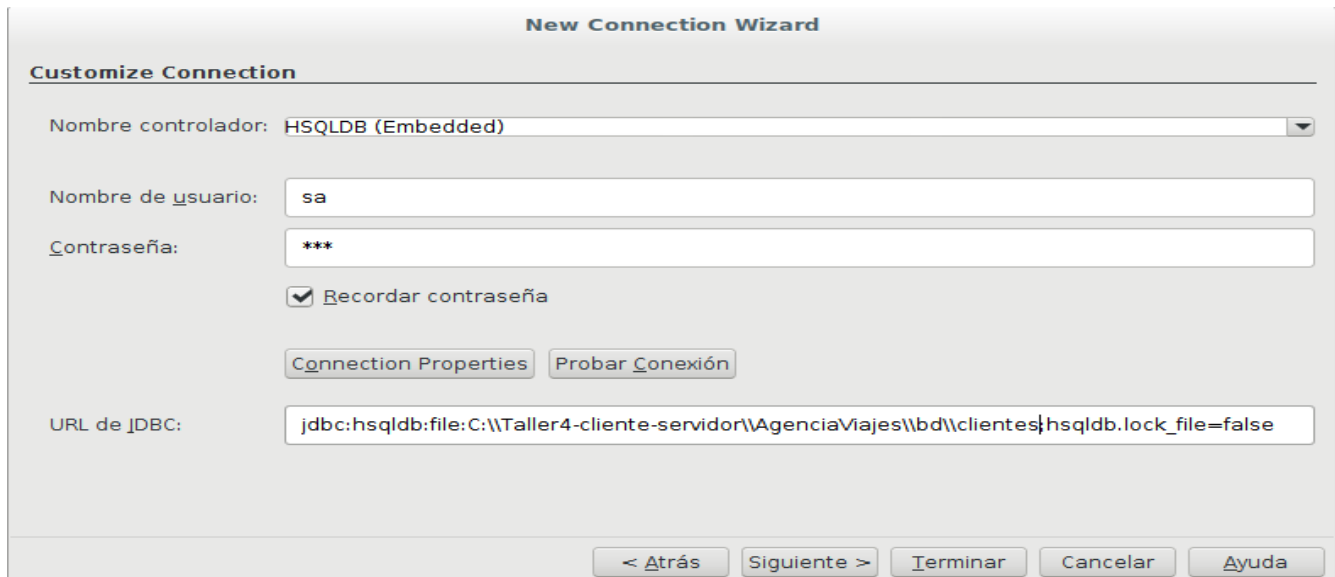


Opcionalmente, si desea hacer cambios en la base de datos, desde Netbeans se puede accederla. Siga los siguientes pasos:

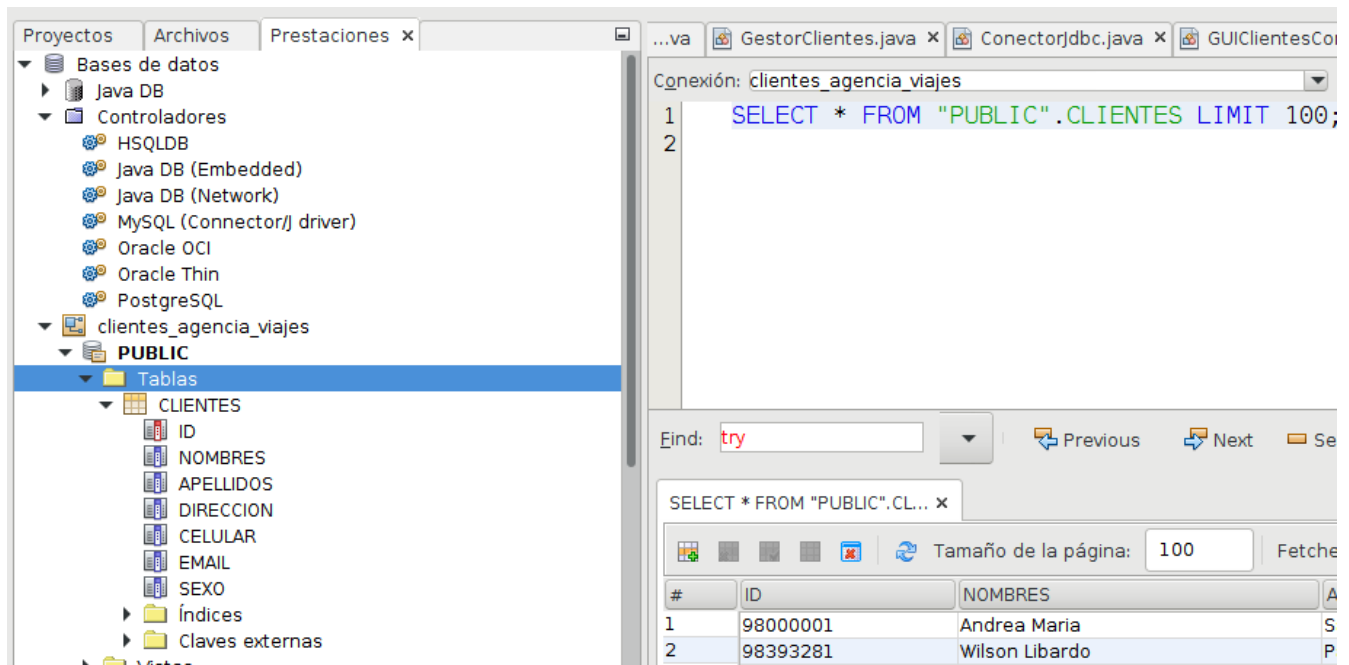
1. Agregar el Controlador HSQLDB. En la pestaña Prestaciones, haga click derecho en el ícono Controladores, y elija Nuevo Controlador. Siga el asistente y utilice el controlador `hsqldb.jar` que se está en la subcarpeta `libs/hsqldb-2.4.0/hsqldb/lib/` del proyecto de AgenciaViajes.
2. Una vez configurado el controlador debe crear la conexión a la base de datos. En la pestaña Prestaciones, haga click derecho en el ícono Bases de datos, elija Nueva Conexión de Bases de datos y presione siguiente. En la siguiente ventana debe elegir el driver y la ruta del driver:



Al darle siguiente debe colocar el nombre del controlador, nombre de usuario (sa) y Contraseña (123) y la URL de JDBC (Acorde a la carpeta donde esté su aplicación AgenciaViajes) tal como lo muestra la Figura:



Puede darle clic en Probar Conexión para asegurarse que todo esté correcto. De ahí en adelante siguiente hasta terminar (El nombre de la conexión es cualquier cadena, por ejemplo, clientes\_agencia\_viajes). Si todo está correcto, al darle clic derecho en la conexión creada y darle la opción Conectar, le debe cargar la base de datos similar a la siguiente Figura:



## CONCLUSIÓN

Se ha trabajado una ejemplo del patrón Cliente-Servidor que a la vez usa el patrón MVC. Es el momento de extender la aplicación. Para ello, ejecute la aplicación, comprenda el código fuente apoyándose en los modelos. No está de más que ejecute las pruebas unitarias y de integración: `GestorClientesTest.java`.

## ¿QUÉ SE DEBE ENTREGAR?

1. Agregue una vista más a la aplicación. Puede ser la que permite graficar la cantidad de hombres y mujeres. Esta vista debe ser similar a `GUIClientesVer.java`. Debe ser notificada tan pronto el modelo cambie de estado. Suba la aplicación a github.
2. Extienda la aplicación de tal forma que en el momento de grabar un Cliente en la aplicación local AgenciaViajes, también se grave una copia del cliente en un Servidor Central. Para ello, debe invocar a un nuevo servicio.
3. Investigue las ventajas y desventajas del patrón Cliente/Servidor en términos de desempeño, escalabilidad, seguridad y disponibilidad.



Trabajar en grupos de dos personas. Cree un documento en formato PDF con los integrantes, la URL del repositorio de la aplicación y el punto teórico. Finalmente, suba el archivo en la tarea respectiva.