Universidad de las Fuerzas Armadas – ESPE

Departamento de Ciencias de la Computación

Análisis y Diseño

Integrantes: Isaac Escobar

Eduardo Mortensen

Diego Ponce

NRC: 14571

Patrón Cliente-Servidor

Definición del patrón

El patrón cliente-servidor es un modelo arquitectónico de software que organiza la comunicación entre dos componentes principales: el cliente, que solicita servicios o datos, y el servidor, que responde a dichas solicitudes y ejecuta las operaciones requeridas. En este modelo, el servidor representa una máquina o programa centralizado que almacena recursos y contiene la lógica de negocio, mientras que el cliente es una aplicación o dispositivo que interactúa directamente con el usuario.

La comunicación entre cliente y servidor se basa en un protocolo de solicitud-respuesta, como HTTP/HTTPS para servicios web o SQL en bases de datos. Este enfoque permite centralizar el procesamiento y almacenamiento de datos, mientras que el cliente se limita a gestionar la interfaz de usuario y enviar solicitudes específicas. Al separar responsabilidades, el patrón cliente-servidor facilita la escalabilidad, el mantenimiento, la seguridad y el control de acceso centralizado.

Aplicación en la industria financiera

El modelo cliente-servidor ha sido fundamental en la evolución de los sistemas financieros, especialmente en la banca. Desde los inicios de la banca digital, los cajeros automáticos (ATM) y terminales en sucursales (clientes) han enviado solicitudes a servidores centrales que contienen la información de las cuentas bancarias. Estos servidores responden con datos como saldos o historiales de transacciones, permitiendo operaciones en tiempo real.

Hoy en día, las aplicaciones de banca en línea, tanto web como móviles, continúan utilizando este modelo. El cliente (la aplicación del usuario) solicita servicios como la visualización del saldo o los últimos movimientos, mientras que el servidor procesa la petición, accede a la base de datos bancaria y devuelve la información requerida.

Este modelo ofrece ventajas como:
- Escalabilidad mediante el despliegue de múltiples servidores.
- Seguridad centralizada para proteger los datos financieros.
- Integridad de datos al contar con una única fuente confiable de información.

También se aplica en otros procesos financieros, como plataformas de pago, sistemas CRM bancarios y cajeros automáticos remotos.

Ejemplos comunes del uso del patrón en banca:

- Cajeros automáticos (ATM): Envían el número de cuenta al servidor, que responde con el saldo o autoriza transacciones.

- Banca en línea: Las aplicaciones móviles o web permiten al usuario ingresar peticiones que son procesadas por el servidor bancario.

- Sistemas internos de oficinas bancarias: Terminales y sistemas de atención acceden a los servidores para consultar y actualizar información de cuentas.

Este modelo garantiza el acceso seguro, consistente y auditable a los datos financieros y facilita la implementación de funcionalidades avanzadas como cifrado, autenticación y generación de reportes.

Ejemplo en Java: Consulta de Saldo Bancario

A continuación, se presenta un ejemplo simple en Java que simula una interacción cliente-servidor para consultar el saldo de una cuenta bancaria utilizando sockets TCP.

Servidor.java

```
import java.net.*;
import java.io.*;

public class Servidor {
```

```
    public static void main(String[] args) {
        int puerto = 5000;
        try (ServerSocket servidor = new ServerSocket(puerto)) {
            System.out.println("Servidor escuchando en el puerto " + puerto);
            Socket cliente = servidor.accept();
            System.out.println("Cliente conectado: " + cliente.getInetAddress());

            BufferedReader entrada = new BufferedReader(new
InputStreamReader(cliente.getInputStream()));
            PrintWriter salida = new PrintWriter(cliente.getOutputStream(), true);

            String cuenta = entrada.readLine();
            System.out.println("Petición recibida para la cuenta: " + cuenta);

            String respuestaSaldo = "El saldo de la cuenta " + cuenta + " es $1,234.56";
            salida.println(respuestaSaldo);
            System.out.println("Respuesta enviada al cliente.");

            cliente.close();
            System.out.println("Conexión cerrada.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Cliente.java

```
import java.net.*;
import java.io.*;

public class Cliente {
    public static void main(String[] args) {
        String host = "localhost";
        int puerto = 5000;
        try (Socket socket = new Socket(host, puerto)) {
            System.out.println("Conectado al servidor");

            BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter salida = new PrintWriter(socket.getOutputStream(), true);

            String cuenta = "123456789";
            System.out.println("Solicitando saldo de la cuenta: " + cuenta);
            salida.println(cuenta);

            String respuesta = entrada.readLine();
            System.out.println("Respuesta del servidor: " + respuesta);
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

El programa implementa una arquitectura cliente-servidor usando sockets TCP. El **servidor** escucha en un puerto esperando conexiones. Cuando un **cliente** se conecta, envía un número de cuenta; el servidor procesa la solicitud (simulando una consulta) y responde con el saldo correspondiente. Luego, ambos cierran la conexión.