Universidad de las Fuerzas Armadas - ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Software

Análisis y Diseño de Software - NRC:23305

Tema:

Grupo: 3

Integrantes:

Ronny Ibarra
Carlos Rivera
Angelo Sanchez

Profesora: Ing. Jenny Ruiz

Taller 2

Subir el documento en formato PDF de la Investigación realizada en los diferentes Patrones de diseño,

1. **Definición del patrón de diseño Cliente-Servidor**

El patrón Cliente-Servidor es una arquitectura de diseño que se utiliza en el análisis y desarrollo de sistemas para organizar los componentes en dos roles principales: el cliente, que realiza solicitudes, y el servidor, que responde a esas solicitudes ofreciendo servicios o recursos.

Este patrón permite separar la lógica del sistema de la interfaz de usuario, facilitando un diseño más claro, modular y mantenible. Uno de sus beneficios clave es la escalabilidad, es decir, la capacidad del sistema para crecer o adaptarse cuando aumenta el número de usuarios o la carga de trabajo, sin afectar negativamente su rendimiento. Esto se logra gracias a que los servicios están centralizados en el servidor, lo que permite reforzar su capacidad sin necesidad de modificar los clientes.

Desde el punto de vista del diseño, esta arquitectura permite modelar sistemas distribuidos, donde los procesos pueden ejecutarse en distintos entornos, manteniendo la eficiencia y la organización del sistema.

El patrón Cliente-Servidor como se dijo es un estilo arquitectónico donde el sistema se divide en dos componentes principales:

- **Cliente:** Solicita servicios o recursos.
- **Servidor:** Proporciona servicios o recursos solicitados por uno o varios clientes.

> “La arquitectura cliente-servidor permite que los servicios estén disponibles en un servidor remoto accesible mediante una red”

2. Aplicación en la industria del patrón Cliente-Servidor

El patrón cliente-servidor se utiliza ampliamente en el desarrollo de sistemas, sobre todo cuando se requiere que múltiples usuarios accedan a un servicio centralizado desde distintos dispositivos. Esta arquitectura permite dividir claramente las funciones, facilitando tanto el desarrollo como el mantenimiento del software.

En mi análisis, he notado que este patrón es común en:

- Aplicaciones web y móviles, donde el cliente (navegador o app) se comunica con un servidor que gestiona la lógica del sistema.
- Sistemas empresariales y bancarios, que requieren seguridad y control centralizado de datos.
- Servicios de bases de datos, donde los clientes realizan operaciones sobre un servidor SQL.

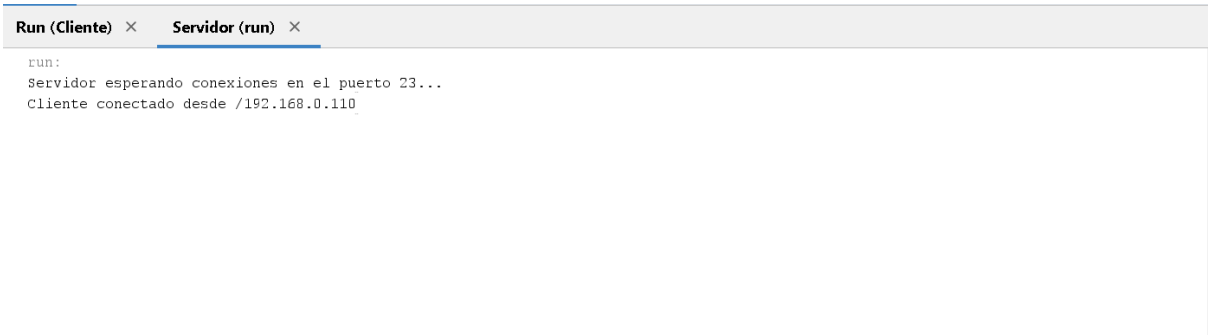
Desde la perspectiva práctica, las ventajas que ofrece según Sommerville son:

- Reutilización de servicios, permitiendo que varios clientes accedan al mismo recurso o lógica.
- Mantenimiento centralizado, lo que reduce costos y errores al actualizar solo el servidor.
- Escalabilidad, ya que se puede atender a más usuarios agregando recursos al servidor sin rediseñar el sistema completo.

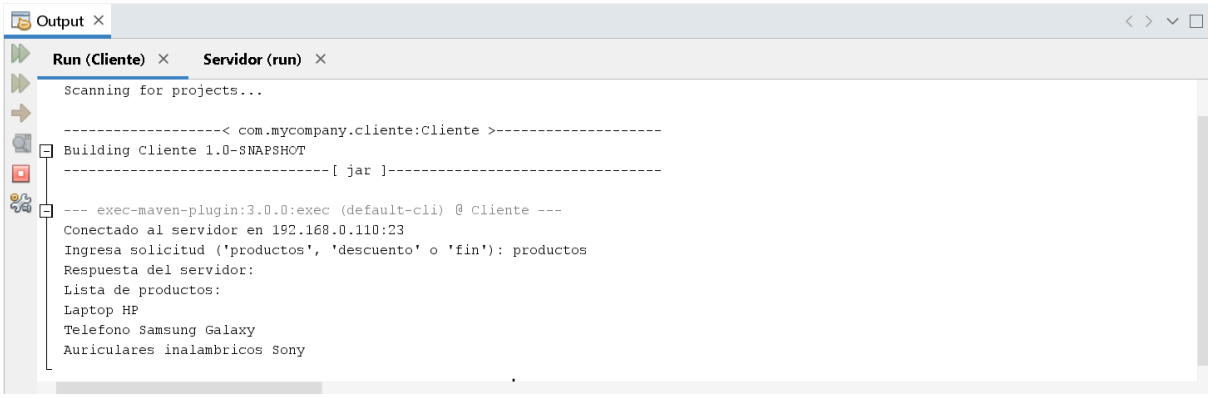
3. Ejemplo con código fuente IDE OO

Servidor:

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package servidor;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;


public class Servidor {

    private static List<String> productos = new ArrayList<>();
    private static List<String> descuentos = new ArrayList<>();

    // Inicialización estática de productos y descuentos
    static {
        productos.add("Laptop HP");
        productos.add("Telefono Samsung Galaxy");
        productos.add("Auriculares inalambricos Sony");

        descuentos.add("Descuento del 10% en todos los productos.");
        descuentos.add("Hoy, todos los clientes reciben un cupón de descuento especial.");
        descuentos.add("Compra dos productos y obtén el tercero gratis.");
    }

    public static void main(String[] args) {
        int puerto = 23;

        try {
            // Crear un socket para el servidor en el puerto especificado
            ServerSocket servidor = new ServerSocket(puerto);
            System.out.println("Servidor esperando conexiones en el puerto " + puerto + "...");

            // Esperar conexiones continuamente
            while (true) {
                // Aceptar la conexión entrante desde un cliente
                Socket cliente = servidor.accept();
                System.out.println("Cliente conectado desde " + cliente.getInetAddress());

                // Obtener flujos de entrada y salida del socket
                DataInputStream entrada = new DataInputStream(cliente.getInputStream());
                DataOutputStream salida = new DataOutputStream(cliente.getOutputStream());

                // Variable para controlar la conexión con el cliente
                boolean clienteConectado = true;

                // Manejar las solicitudes del cliente
                while (clienteConectado) {
                    // Leer la solicitud del cliente
                    String solicitud = entrada.readUTF();
                    String respuesta = "";

                    // Procesar la solicitud del cliente
                    if (solicitud.equals("productos")) {
                        respuesta = obtenerListaDeProductos();
                    } else if (solicitud.equals("descuento")) {
                        respuesta = obtenerDescuentoAleatorio();
                    } else if (solicitud.equals("fin")) {
                        // Si la solicitud es 'fin', desconectar al cliente
                        clienteConectado = false;
                    } else {
                        // Si la solicitud no es válida, enviar un mensaje de error al cliente
                        respuesta = "Solicitud no válida. Debe ingresar 'productos', 'descuento' o 'fin'.";
                    }

                    // Enviar la respuesta al cliente
                    salida.writeUTF(respuesta);
                }

                // Cerrar flujos y el socket cuando el cliente se desconecta
                entrada.close();
                salida.close();
                cliente.close();
            }
        } catch (IOException e) {
            // Manejar excepciones de entrada/salida
            e.printStackTrace();
        }
    }

    // Función para obtener la lista de productos como una cadena formateada
    private static String obtenerListaDeProductos() {
        StringBuilder listaProductos = new StringBuilder("Lista de productos:\n");
        for (String producto : productos) {
            listaProductos.append(producto).append("\n");
        }
        return listaProductos.toString();
    }

    // Función para obtener un descuento aleatorio de la lista de descuentos
    private static String obtenerDescuentoAleatorio() {
        Random random = new Random();
        int indice = random.nextInt(descuentos.size());
        return descuentos.get(indice);
    }
}
```


Run (Cliente)
Servidor (run)
run:
Servidor esperando conexiones en el puerto 23...
Cliente conectado desde /192.168.0.110

Cliente:


Output
Run (Cliente)
Servidor (run)
Scanning for projects...
------------------< com.mycompany.cliente:Cliente >-------------------
Building Cliente 1.0-SNAPSHOT
--------------------------------[jar]---------------------------------
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Cliente ---
Conectado al servidor en 192.168.0.110:23
Ingresa solicitud ('productos', 'descuento' o 'fin'): productos
Respuesta del servidor:
Lista de productos:
Laptop HP
Telefono Samsung Galaxy
Auriculares inalambricos Sony

```java
package com.mycompany.cliente;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

public class Cliente {
    private int puerto;
    private String ip;

    public Cliente(String ip, int puerto) {
        this.ip = ip;
        this.puerto = puerto;

        try (
            Socket cliente = new Socket(ip, puerto);
            DataInputStream entrada = new DataInputStream(cliente.getInputStream());
            DataOutputStream salida = new DataOutputStream(cliente.getOutputStream());
            Scanner scanner = new Scanner(System.in);
        ) {
            System.out.println("Conectado al servidor en " + ip + ":" + puerto);

            boolean conectado = true;
            while (conectado) {
                System.out.print("Ingresa solicitud ('productos', 'descuento' o 'fin'): ");
                String solicitud = scanner.nextLine();

                // Enviar solicitud al servidor
                salida.writeUTF(solicitud);

                if ("fin".equalsIgnoreCase(solicitud)) {
                    conectado = false;
                    System.out.println("Conexión finalizada.");
                } else {
                    // Leer y mostrar respuesta del servidor
                    String respuesta = entrada.readUTF();
                    System.out.println("Respuesta del servidor:\n" + respuesta);
                }
            }

        } catch (IOException e) {
            System.out.println("Error de conexión con el servidor:");
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        // IP corregida sin espacios
        new Cliente("192.168.0.110", 23);
    }
}
```