Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

División de Tecnologías para la Integración CiberHumana

Ingeniería en Computación

Ingeniería de Software

D07 - CB224 - 210896

7. Diseño Arquitectónico

Profesor: Morales Ramírez Thelma Isabel

Alumno: Juárez Rubio Alan Yahir

Código: 218517809

Correo: alan.juarez5178@alumnos.udg.mx

16 de octubre de 2024

Índice

7. Diseño Arquitectónico

1. Introducción

El diseño arquitectónico nos permite definir cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global de esté. Es el enlace entre el diseño y la ingeniería de requerimientos, que identifica los principales componentes estructurales en un sistema y la relación entre ellos.

1.1. Conceptos de Arquitectura de Software

Responde a cada una de las preguntas propuestas:

1. ¿Qué es la arquitectura de software y por qué es importante en el desarrollo de aplicaciones?

   La **arquitectura del software** es la manera esencial en que los componentes de un sistemas se encuentran conectados entre sí o la organización fundamental de un sistema.

   La **arquitectura de software** es importante debido a que nos permite identificar cada uno de los componentes del sistema que serán esenciales para el éxito o fallo de nuestro sistema y el desarollo del sistema para servir y proteger dicho componentes esenciales.

2. ¿Qué es un patrón de diseño arquitectónico y por qué son importantes en el desarrollo de software? Proporciona ejemplos de patrones de diseño comunes.

   Un **patrón arquitectónico** es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares al patrón de diseño de software, pero tienen un alcance más amplio.

   Los patrones arquitectónicos son importantes debido a que ofrecen soluciones a problemas de arquitectura de software, definiendo elementos y relaciones entre ellos, junto con restricciones sobre su uso.

   Algunos ejemplos de diseño arquitectónico son:

   - **Cliente-Servidor**: Este patrón comprende dos partes, uno o varios servidores y múltiples clientes. Mientras los usuarios, a través de un sistema, solicitan servicios, recursos y demás al servidor, el servidor se encarga de suministrar dichas peticiones a dichos usuarios.
   - **Capas**: Se puede utilizar para estructurar programas que se pueden descomponer en

grupos de subtareas, cada una de las cuales se encuentra en un nivel particular de abstracción. Cada capa proporciona servicios a la siguiente capa superior.

- **MVC (Modelo Vista Controlador)**: Este modelo divide una aplicación interactiva en 3 partes:
  - **Modelo**: Contiene la funcionalidad y los datos básicos
  - **Vista**: Muestra la información al usuario
  - **Controlador**: Maneja la entrada del usuario

3. ¿Qué significa el principio de "separación de preocupaciones" en la arquitectura de software? ¿Cómo se logra esto en la práctica?

   El principio de Separación de preocupaciones (SoC, por sus siglas en inglés) es una guía de diseño de software que se enfoca en separar los aspectos diferentes de un sistema en diferentes componentes para que cada uno se ocupe de una tarea específica y no haya mezcla de responsabilidades.

4. Habla sobre la diferencia entre un "framework" y una "biblioteca" en el contexto de la arquitectura de software. ¿Cómo pueden ayudar en el desarrollo de aplicaciones?

   - **Framework**: Establece la estructura y el flujo de control de una aplicación. El desarrollador sigue sus reglas y el framework llama al código que este escribe (inversión de control). Es una solución completa, con convenciones predefinidas que aceleran el desarrollo, pero con menos flexibilidad.

     Los **frameworks** proporcionan estructura y buenas prácticas, facilitando aplicaciones organizadas y rápidas de crear.

   - **Biblioteca**: A diferencia de un *framework*, la **biblioteca** es más flexible y modular. El desarrollador tiene el control total y decide cuándo usarla. Proporciona funciones específicas sin dictar la estructura general de la aplicación.

     Las **bibliotecas** permiten solucionar problemas puntuales con más control y personalización.

1.2. Estilos Arquitectónicos

Para la resolución de los ejercicios que siguen a continuación sólo considere los estilos arquitectónicos: Basado en Capas, Cliente-Servidor, Pipes and Filters, Basado en Eventos, Peer-to-Peer.

Para cada una de las siguientes situaciones, discuta qué estilo o estilos arquitectónicos resultan aplicables y justifica brevemente porque:

1. La aplicación necesita procesar la entrada en forma de flujo continuo (stream) y producir una salida también en forma de flujo continuo.

   Considero que el estilo adecuado es *pipes and filters* deido este ha sido diseñado para el procesamiento de datos en flujo continuo. Cada filtro procesa los datos y los pasa al siguiente, permitiendo una cadena de procesamiento eficiente donde la entrada fluye a través de una serie de transformaciones, hasta generar la salida.

2. Hay una gran cantidad de usuarios que se conectarán al sistema a través de una red.

   Desde mi punto de vista, el estilo que se adecúa a este caso es el *cliente-servidor* principalmente porque permite a múltiples clientes (usuarios) conectarse a un servidor a través de internet para que este le suministre los servicios y recursos solicitados or el cliente.

3. La aplicación necesita ser ejecutada en diferentes plataformas.

   Si fuera un aplicación web (e.g. caso anterior) diría que cliente-servidor debido a que estas aplicaciones pueden ser ejecutadas en múltiples sistemas. Sin embargo, como no es el caso, considero que el estilo *basado en capas* es más óptimo, especialmente porque puede separar la aplicación en capas, tales como presentación y lógica del sistema y, por ende, tiende a hacer más sencillo la portabilidad de una aplicación.

4. El sistema debe tener un alto performance tanto en tiempo como en espacio (memoria).

   El estilo que considero que es el más adecuado es el *Peer-to-Peer*. Este estilo permite distribuir la carga de procesamiento y memoria de manera uniforme.

5. El sistema debe ser capaz de ser ajustado para alcanzar el mejor uso de los recursos de software y hardware, los cuales pueden ser cambiados en el futuro.

   El diseño *basado en eventos* considero el más óptimo para este caso, ya que este permite que los componentes del sistema sean activas solamente cuando ocurre un determinado evento, lo cual permite un mejor aprovechamiento de los recursos del sistema.

6. Un componente clave del sistema es la solución de un problema recurrente en el dominio de aplicación. Se desea reusar el componente en sistemas similares.

   Pienso que el estilo *basado en capas* es la mejor solución a este caso debido a que permite

separar el sistema en diferentes capas, lo cual facilita la reutilización de componentes.

1.3. Modelo Estilo Arquitectónico

Para cada uno de los problemas descritos a continuación, modele el estilo arquitectónico solicitado. Para cada caso analice las ventajas y desventajas del modelo, e indique si podría utilizar otro estilo arquitectónico (o combinación de estilos). En caso afirmativo, justifique adecuadamente su decisión y realice además el modelo alternativo.

1.3.1. Tubos y Filtros

Se desea contar con un sistema capaz de mejorar la calidad del sonido de escuchas telefónicas, grabación a distancia de conversaciones, etc. El sonido será digitalizado y a partir de allí se lo someterá a diversas transformaciones para mejorar su calidad. Las transformaciones posibles son:

- Dividir los sonidos según su frecuencia, lo que produce varias bandas de sonido diferentes que pueden ser analizadas separadamente o vueltas a reunir.
- Disminuir la velocidad con que se reproduce el sonido de una cantidad fijada dinámicamente por el usuario.
- Eliminar todos los sonidos de una frecuencia dada por el usuario.
- Aumentar y disminuir el volumen en una cantidad fijada por el usuario.

El usuario puede determinar en cualquier paso si desea escuchar el resultado. Cada vez que se realiza una mejora a la calidad del sonido, es el usuario quien decide qué operación de las disponibles desea realizar.

Modelado El sistema se puede estructurar como una cadena de **filtros** que procesan los datos de sonido (la entrada). Cada filtro aplica una de las transformaciones posibles, y los resultados pueden pasar a otro filtro o ser escuchados por el usuario. Un posible flujo sería:

1. **Filtro 1**: Dividir los sonidos según su frecuencia.
2. **Filtro 2**: Disminuir la velocidad del sonido.
3. **Filtro 3**: Eliminar sonidos de una frecuencia específica.
4. **Filtro 4**: Aumentar o disminuir el volumen.

Los datos fluyen a través de estos filtros secuencialmente, y el usuario puede decidir en cualquier punto si desea escuchar el sonido procesado hasta ese momento.

Ventajas

- **Modularidad**: Cada operación de mejora de sonido es independiente (filtros separados). Esto facilita la adición, eliminación o modificación de filtros sin alterar el resto del sistema.
- **Reutilización**: Los filtros pueden ser reutilizados en otros sistemas que necesiten manipulación de sonido.
- **Flexibilidad**: El usuario puede elegir qué operaciones aplicar y en qué orden, lo que se adapta bien a las necesidades específicas del proceso.

Desventajas

- **Orden fijo de procesamiento**: Aunque el usuario puede elegir qué filtros aplicar, el flujo de procesamiento sigue siendo secuencial, lo que puede limitar la capacidad de reordenar dinámicamente las operaciones.
- **Latencia**: Cada filtro introduce un retraso en el procesamiento. Si el sonido pasa por muchos filtros, podría generar una latencia notable en el resultado final.

1.3.2. Sistemas en Capas o Estratificados

Los pacientes de una guardia de terapia intensiva son monitoreados por dispositivos electrónicos análogos adosados a sus cuerpos. Los dispositivos miden los signos vitales de los pacientes. Hay un sensor para el ritmo cardíaco (activo, una señal para cada latido del corazón), presión sanguínea (pasivo) y la temperatura (pasivo). Se necesita un programa que lea los signos vitales con una frecuencia especificada para cada paciente. Los valores leídos serán comparados con rangos que serán reportadas con mensajes de alarma en el monitor del box de enfermería. También debe mostrarse un mensaje apropiado si falla un dispositivo. Además, se requiere almacenar los datos y proveer reportes estadísticos.

Modelado El sistema puede estructurarse en capas para separar las diferentes responsabilidades del monitoreo de pacientes. Un modelo de capas podría ser el siguiente:

1. **Capa de Sensores**: Gestiona la lectura de datos de los dispositivos (ritmo cardíaco, presión sanguínea, temperatura) en intervalos de tiempo específicos para cada paciente.
2. **Capa de Procesamiento de Señales**: Compara los valores leídos con los rangos normales y determina si es necesario generar una alarma.

3. **Capa de Almacenamiento y Reportes**: Almacena los datos recogidos y genera reportes estadísticos.
4. **Capa de Presentación**: Muestra los resultados y las alarmas en los monitores del personal de enfermería, incluyendo fallas de dispositivos.

Ventajas

- **Separación de responsabilidades**: Cada capa tiene una responsabilidad clara, lo que facilita el mantenimiento y la evolución del sistema.
- **Modularidad**: Las capas pueden modificarse o mejorarse independientemente. Por ejemplo, se puede cambiar la forma en que se gestionan los sensores sin afectar la capa de presentación.
- **Escalabilidad**: Nuevas funciones, como la incorporación de más sensores o diferentes tipos de alarmas, se pueden integrar fácilmente agregando o ajustando capas.

Desventajas

- **Desempeño**: La comunicación entre capas puede introducir una sobrecarga en términos de tiempo y uso de recursos.
- **Dependencia entre capas**: Si una capa falla, podría afectar negativamente a las demás, especialmente si no está bien desacoplada.

Referencias

Cruz, L. (2011). *Ingeniería de Software*. Pearson Education, Estado de México, México, 9th edition.

huaman, W. C. (2018). Los 10 patrones comunes de arquitectura de software. https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b. Consultado el 16 de octubre de 2024.

Pitaliya, S. (2021). Understanding Software Architecture: A Complete Guide. https://sarrahpitaliya.medium.com/understanding-software-architecture-a-complete-guide-cb8f05900603. Consultado el 16 de octubre de 2024.

sofka (s.f.). Separation of concerns. https://sofka-practices.github.io/principios/base/separation_of_concerns/. Consultado el 16 de octubre de 2024.