Introducción a la arquitectura de software

Ramón Camacho

1.Introducción

La arquitectura de software es crucial para el diseño de sistemas complejos. Este artículo introduce estilos arquitectónicos comunes y su combinación en diseños heterogéneos. Se presentan seis estudios de caso que ilustran cómo las representaciones arquitectónicas pueden mejorar la comprensión de sistemas complejos. También se discuten problemas pendientes y direcciones de investigación prometedoras en el campo.

2.De los lenguajes de programación a la arquitectura de software

Una caracterización del progreso en los lenguajes y herramientas de programación han sido los aumentos regulares en el nivel de abstracción. Para ponernos en contexto, a continuación, observaremos el desarrollo histórico de las técnicas de abstracción en las ciencias de la computación.

2.1 Lenguajes de programación de alto nivel

En la década de 1950, el software estaba escrito directamente en lenguaje máquina. La inserción de una nueva funcionalidad requería una comprobación manual de todo el programa. Unas de las primeras abstracciones del software fue sustituir símbolos simples por códigos de operación. Esta abstracción de mayor nivel permitió desarrollar programas más sofisticados y surgieron patrones en el uso de los datos.

2.2 Tipos de datos abstractos

A finales de la década de los 60's los buenos programadores compartían una intuición sobre el software: si se tiene una estructura de datos correcta, el resto del desarrollo será más fácil. La conversión de una intuición a una teoría implicaba la comprensión: la estructura del software, especificaciones, problemas lingüísticos, integridad del resultado, normas para la combinación de tipos y ocultación de información.

2.3 Arquitectura de Software

Los buenos diseñadores de sistemas de software ahora reconocen organizaciones de sistemas útiles. Con el tiempo, muchas organizaciones se han ido desarrollando de manera informal pero ahora forman parte del vocabulario de los diseñadores de software. Por ejemplo, en algunas de las descripciones de arquitecturas de software, se menciona: modelo cliente-servidor, capas de abstracción, enfoque distribuido y orientado a objetos, entre otras.

A continuación, se verá más a detalle estas arquitecturas de software comunes.

3.Estilos arquitectónicos comunes

Ahora examinamos algunos de estos estilos arquitectónicos representativos y ampliamente utilizados.

Para contextualizar, es necesario reconocer adaptar un marco en común para las arquitecturas, las cuales están formadas de componentes, junto con una interacción entre estos componentes, los conectores.

Un estilo arquitectónico entonces define una familia de tales componentes e interacciones en términos de un patrón de organización estructural.

3.1Tuberías y filtros

El estilo arquitectónico de tuberías y filtros se basa en componentes que tienen entradas y salidas para leer y producir flujos de datos. Los filtros hacen transformaciones locales y producen salidas antes de que se consuma la entrada, sin compartir información con otros filtros. Este estilo permite una fácil comprensión y reutilización, pero no es bueno para aplicaciones interactivas. En este estilo arquitectónico a los componentes se les conoce como "*filtros*" y a las conexiones como "*tuberías*".

3.2 Abstracción de datos y organización orientada a objetos

En este estilo, las representaciones de datos y sus operaciones primitivas asociadas se encapsulan en un tipo de datos u objeto abstracto. Aquí los componentes son objetos, los cuales son responsables de preservar la integridad de su representación, la representación está oculta de otros objetos. Esto da pie a una propiedad agradable, la cual es poder afectar la implementación del objeto sin afectar a los clientes.

Lo más significativo es que para que un objeto interactúe con otro debe conocer la identidad de ese otro objeto por tanto cada vez que la identidad de un objeto cambia, es necesario modificar todos los demás objetos que lo invocan explícitamente, el cual puede presentarse como una desventaja.

3.3 Invocación implícita basada en eventos

En este caso los datos son más abstractos que en el ejemplo anterior. La principal idea de esta arquitectura es que, en vez de hacer una invocación de un procedimiento directamente, el componente puede anunciar uno o más eventos. Los eventos son invocados implícitamente a medida que los datos son modificados. La principal invariante de esta arquitectura es que los anunciantes de eventos no saben cuál componente será afectado por esos eventos anunciados. Esta solución ofrece la posibilidad de añadir módulos tan fáciles como sólo registrarlos para ser invocados en los eventos de datos en cambio. Una de sus principales desventajas es que los componentes ceden el control del cálculo al sistema.

3.4 Sistema por capas

Aquí, la organización es jerárquica, cada capa provee de un servicio a la capa que está sobre ella y actúa como cliente respecto a la capa que está debajo. Las conexiones están definidas por protocolos que determinan como las capas tienen que interactuar. Unas de sus características a favor, es que permiten el diseño basado en incrementar el nivel de abstracción, además, permiten la mejora, ya que cambios en su funcionamiento como mucho puede afectar a dos capas. Por último, el reúso también está presente ya que las capas pueden ser fácilmente intercambiables. Una de sus desventajas notables es la caída de rendimiento.

3.5 Repositorios

En este estilo hay dos tipos de componentes: Una estructura central de datos que representa el estado actual, y una colección independiente de componentes que operan en el archivo principal de datos. Usualmente consta de tres partes más representativas. Primero, fuente de conocimiento: Paquetes separados de conocimiento. Segundo, estructura de datos “blackboard”: Datos de estado de resolución de problemas. Por último, control: Las fuentes de conocimiento responden de manera oportunista cuando los cambios en la pizarra los hagan aplicables.

3.6 Intérpretes controlados por tablas

En una organización de interprete una máquina virtual está producida en el software. El intérprete incluye el seudoprograma que se está interpretando y el propio motor de interpretación. Un intérprete generalmente consta de cuatro componentes: una maquina de interpretación, una memoria que contiene el seudocódigo a ser interpretado, una representación del estado de control del motor de interpretación y una representación del estado actual del programa a ser simulado.

3.7 Otras familias de arquitecturas

Existen otros numerosos estilos de arquitecturas y patrones, por lo cual no todas son mencionadas. Sim embargo, una categoría importante es: Procesos distribuidos. Los procesos distribuidos en un conjunto de arquitecturas están enfocados a la construcción de sistemas de multi procesos. Una forma común de arquitectura de sistema distribuido es una organización de "cliente-servidor". En estos sistemas un servidor representa un proceso que brinda servicios a otros procesos (clientes).

3.8 Arquitecturas heterogéneas

Las arquitecturas heterogéneas son aquellas arquitecturas que no se basan en una estructura pura de otra, sino que es una combinación de diferentes organizaciones arquitectónicas, la cual permiten sistemas más complejos y acordes a las necesidades del sistema.