Introducción a la arquitectura de software

Gisela Guadalupe Rocha Astorga

A medida que aumenta el tamaño y la complejidad de los sistemas de software, el problema de diseño va más allá de los algoritmos y las estructuras de datos de la computación, es este el nivel de diseño de la arquitectura de software. Una característica del del progreso en los lenguajes y herramientas de programación han sido los aumentos en el nivel de abstracción de los bloques de construcción de los diseñadores de software. En la década de 1950 el software estaba escrito en lenguaje de máquina; los programadores decidieron colocar instrucciones y datos de manera individual en la memoria de la computadora, entonces, se reconoció que el diseño de la memoria y la actualización de las referencias podrían automatizarse. El resultado fueron ensambladores simbólicos, la sustitución de símbolos simples por códigos de operación de máquinas, direcciones de máquina aún por definir y secuencias de instrucciones fue quizás la primera forma de abstracción en software. Entonces quedó claro que ciertos patrones de ejecución eran comúnmente útiles; y estas ideas fueron capturadas en los primeros lenguajes de alto nivel. El progreso en el diseño del lenguaje continuó con la introducción de módulos para proporcionar protección para procedimientos y estructuras de datos relacionados, A finales de la década de 1960, los buenos programadores compartían la idea de que, si obtienes las estructuras de datos correctas, el esfuerzo hará que el desarrollo del resto del programa sea mucho más fácil, y así como ellos, los buenos diseñadores de sistemas de software buscaban reconocer organizaciones de sistemas útiles. Muchas organizaciones se han desarrollado con el tiempo, y ahora son parte del vocabulario de los diseñadores de sistemas de software.

Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural. Es decir, determina el vocabulario de componentes y conectores que se pueden usar en instancias de ese estilo, junto con las restricciones de cómo se pueden combinar. Hablaremos de los distintos estilos a continuación. En un estilo de tubería y filtro, cada componente tiene un conjunto de entradas y un conjunto de salidas. Un componente lee flujos de datos en sus entradas y produce flujos de datos en sus salidas, entregando una instancia completa del resultado en un orden estándar. Estos sistemas tienen una serie de buenas propiedades: permiten comprender el comportamiento general de entrada/salida de un sistema, apoyan la reutilización, se pueden mantener y mejorar fácilmente, permiten el análisis de rendimiento y punto muerto y admiten la ejecución concurrente.

Por otro lado, tenemos el estilo de abstracción de datos y organización orientada a objetos, en el cual las representaciones de datos y sus operaciones asociadas se encapsulan en un tipo de datos u objeto abstracto. Estos objetos interactúan a través de invocaciones de funciones y procedimientos. Dos aspectos importantes de este estilo son: un objeto es responsable de preservar la integridad de su representación y que la representación está oculta de otros objetos. Este estilo tiene muchas buenas propiedades, entre ellas como ya conocemos destaca que debido a que un objeto oculta su representación a sus clientes, es posible cambiar la implementación sin afectar a esos clientes.

Adicionando a los estilos, tenemos la invocación implícita basada en eventos. Tradicionalmente, en un sistema en el que las interfaces de componentes proporcionan una colección de procedimientos y funciones, los componentes interactúan entre sí invocando explícitamente esas rutinas. Sin embargo, ha habido un interés considerable en la invocación implícita. La idea detrás de esta es que en lugar de invocar un procedimiento directamente, un componente puede anunciar uno o más eventos. Los componentes en un estilo de invocación

implícita son módulos cuyas interfaces proporcionan tanto una colección de procedimientos como un conjunto de eventos. El principal problema es que los anunciantes de eventos no saben qué componentes se verán afectados por esos eventos. Entre los beneficios de la invocación implícita se encuentran: que proporciona un fuerte soporte para la reutilización y que facilita la evolución del sistema. Pero la principal desventaja de la invocación implícita es que los componentes renuncian al control sobre el cálculo realizado por el sistema.

Además de los diseños anteriores, tenemos los sistemas en capas; un sistema en capas se organiza jerárquicamente, cada capa proporciona servicio a la capa superior y sirve como cliente a la capa inferior. Por lo tanto, en estos sistemas, los componentes implementan una máquina virtual en alguna capa de la jerarquía. Los sistemas en capas tienen varias propiedades deseables entre ellos que, apoyan el diseño basado en niveles crecientes de abstracción, apoyan la mejora, y apoyan la reutilización. Aunque tienen desventajas, principalmente que no todos los sistemas se estructuran fácilmente en capas.

También contamos con el estilo de repositorio. En un estilo de repositorio hay dos tipos muy distintos de componentes; una estructura de datos central que representa el estado actual, y una colección de componentes independientes operan en el almacén de datos central. El modelo de pizarrón generalmente se presenta con tres partes principales: las fuentes de conocimiento, la estructura de datos de pizarrón y control.

Ahora hablaremos del diseño de intérpretes controlados por tablas. En una organización de intérpretes, una máquina virtual se produce en software. Un intérprete incluye el pseudoprograma que se está interpretando y el propio motor de interpretación. El pseudoprograma incluye el propio programa y el motor de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución.

Hay muchos otros estilos y patrones arquitectónicos. Algunos están muy extendidos y otros son específicos de dominios particulares. Una categoría importante es la de procesos distribuidos en la cual, los sistemas distribuidos han desarrollado una serie de organizaciones comunes para sistemas multiproceso. Una forma común de arquitectura de sistema distribuido es una organización "cliente-servidor". En estos sistemas, un servidor representa un proceso que proporciona servicios a otros procesos, pero, el servidor no conoce de antemano las identidades o el número de clientes que tendrán acceso a él en tiempo de ejecución. Por otra parte, tenemos las organizaciones principales de programas en donde el programa principal actúa como el controlador de las subrutinas, proporcionando típicamente un bucle de control. También se cuenta con arquitecturas de software específicas de dominio las cuales proporcionan una estructura organizativa adaptada a una familia de aplicaciones. Al especializar la arquitectura al dominio, es posible aumentar el poder descriptivo de las estructuras. Otra categoría serían los sistemas de transición de estado, estos sistemas se definen en términos de un conjunto de estados y un conjunto de transiciones con nombre que mueven un sistema de un estado a otro. Y, por último, tenemos los sistemas de control de proceso cuyos se caracterizan aproximadamente como un bucle de retroalimentación.

Hemos estado hablando principalmente de estilos arquitectónicos puros, aunque la mayoría de los sistemas suelen implicar alguna combinación de varios estilos, lo que da lugar a arquitecturas heterogéneas. Hay diferentes maneras en que los estilos arquitectónicos se pueden combinar. Una forma es a través de la jerarquía, Una segunda forma de combinar estilos es permitir que un solo componente use una mezcla de conectores arquitectónicos. Una tercera forma de combinar estilos es elaborar completamente un nivel de descripción arquitectónica en un estilo arquitectónico completamente diferente.