PRACTICA ARQUITECTONICA Y PATRONES

Hay muchas formas de diseñar mal y solo algunas formas de hacerlo bien.

Las tácticas suelen utilizar una única estructura o mecanismo computacional, y están destinadas a abordar una única fuerza arquitectónica. Por esta razón, brindan un control más preciso a un arquitecto al tomar decisiones de diseño que los patrones, que generalmente combinan múltiples decisiones de diseño en un paquete. Las tácticas son los "bloques de construcción" del diseño, a partir de los cuales la arquitectura se crean patrones. Las tácticas son átomos y los patrones son moléculas. La mayoría de los patrones consisten en (se construyen a partir de) varias tácticas diferentes. Por eso decimos que los patrones empaquetan tácticas.

PATRONES DE ARQUITECTURA

Un patrón arquitectónico establece una relación entre:

■ **Un contexto.** Una situación común y recurrente en el mundo que da lugar a un problema.

■ **Un problema.** El problema, apropiadamente generalizado, que surge en el contexto dado. La descripción del patrón describe el problema y sus variantes, y describe cualquier fuerza complementaria u opuesta. La descripción del problema a menudo incluye atributos de calidad que deben cumplirse.

■ **Una solución**. Una resolución arquitectónica exitosa del problema, apropiadamente abstraída. La solución describe las estructuras arquitectónicas que resuelven el problema, incluido cómo equilibrar las muchas fuerzas en el trabajo. La solución describirá las responsabilidades y las relaciones estáticas entre los elementos (utilizando una estructura de módulo), o describirá el comportamiento en tiempo de ejecución de los elementos y la interacción entre ellos (diseñar un componente y conector o una estructura de asignación). La solución para un patrón está determinada y descrita por:

■ Un conjunto de tipos de elementos (por ejemplo, repositorios de datos, procesos y objetos)

■ Un conjunto de mecanismos de interacción o conectores (por ejemplo, método llamadas, eventos o bus de mensajes)

■ Un diseño topológico de los componentes

■ Un conjunto de restricciones semánticas que cubren la topología, el comportamiento de los elementos y Mecanismos de interacción.

La descripción de la solución también debe dejar claro qué atributos de calidad son proporcionada por las configuraciones estáticas y en tiempo de ejecución de los elementos. Este formulario de {contexto, problema, solución} constituye una plantilla para documentar un patrón.
Los sistemas complejos exhiben múltiples patrones a la vez.

CATEGORIZACION DE PATRONES

Los patrones se pueden categorizar por el tipo dominante de elementos que mostrar: **patrones de módulo** mostrar módulos, **patrones de componente y conector (C&C)** muestran componentes y conectores, y los **patrones de asignación** muestran una combinación de elementos de software (módulos, componentes, conectores) y elementos no software. La mayoría de los patrones publicados son patrones de C&C, pero hay patrones de módulos y patrones de asignación también. Comenzaremos con el abuelo de los patrones de módulo, el patrón en capas.

Patrones de módulo

Patrón en capas -LAYERED

Contexto: Todos los sistemas complejos experimentan la necesidad de desarrollar y evolucionar partes del sistema de forma independiente. Por esta razón los desarrolladores del sistema necesitan una separación de preocupaciones clara y bien documentada, de modo que los módulos del sistema se puede desarrollar y mantener de forma independiente.

Problema: El software debe estar segmentado de tal manera que los módulos se puede desarrollar y evolucionar por separado con poca interacción entre las partes, apoyando la portabilidad, modificabilidad y reutilización.

Solución: Para lograr esta separación de preocupaciones, el patrón en capas divide el software en unidades llamadas capas. Cada capa es una agrupación de módulos que ofrece una conjunto cohesivo de servicios. Existen restricciones en la relación de uso permitido entre las capas: las relaciones deben ser unidireccionales. Las capas dividen completamente un conjunto de software, y cada partición se expone a través de una interfaz pública.
Las capas se crean para interactuar de acuerdo con una estricta relación de ordenamiento. Si (A, B)
Es en esta relación, decimos que a la implementación de la capa A se le permite utilizar cualquiera de las facilidades públicas provistas por la capa B. En algunos casos, módulos en una capa podría ser necesario utilizar módulos directamente en una capa inferior no adyacente; normalmente solo se permiten los usos de la siguiente capa inferior. Este caso de software en una capa superior que utiliza módulos en una capa inferior no adyacente se denomina **puente de capas**. Si se producen muchos casos de puenteo de capas, es posible que el sistema no cumpla con los objetivos de portabilidad y modificabilidad que el uso de capas estrictas ayuda a lograr. No se permiten usos ascendentes en este patrón.
Por supuesto, nada de esto es gratis. Alguien debe diseñar y construir las capas, que a menudo pueden agregar costos iniciales y complejidad a un sistema. Además, si las capas no están diseñadas correctamente, en realidad pueden interferir, al no proporcionar las abstracciones de nivel inferior que necesitan los programadores de niveles superiores. Y
las capas siempre agregan una penalización de rendimiento a un sistema. Si se hace una llamada a un función en la capa superior, esto puede tener que atravesar muchas capas inferiores antes de ser ejecutado por el hardware. Cada una de estas capas agrega algo de sobrecarga, como mínimo en forma de cambio de contexto.

La tabla 13.1 resume la solución del patrón en capas.
Las capas casi siempre se dibujan como una pila de cajas. El permiso de uso
La relación se denota por adyacencia geométrica y se lee de arriba hacia abajo, como en Figura 13.1

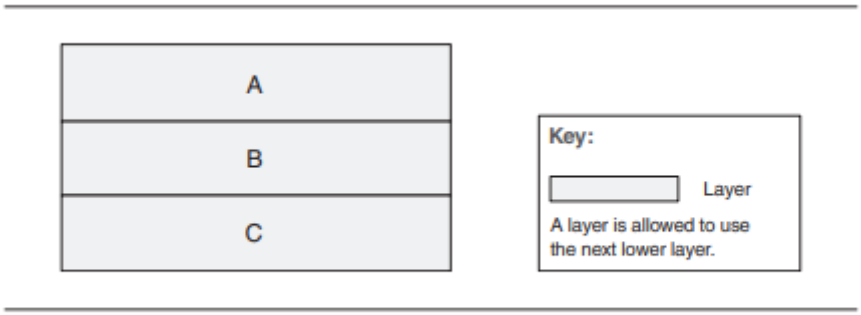

FIGURE 13.1 Stack-of-boxes notation for layered designs

Solución de patrón en capas

Visión general

El patrón de capas define capas (agrupaciones de módulos que ofrecen un conjunto cohesivo de servicios) y un uso permitido unidireccional relación entre las capas. El patrón generalmente se muestra gráficamente apilando cajas que representan capas una encima de la otra.

Elementos

Capa, una especie de módulo. La descripción de una capa debe definir qué módulos contiene la capa y una caracterización del conjunto cohesivo de servicios que proporciona la capa.

Relaciones

Permitido para usar, que es una especialización de relación de dependiencia. El diseño debe definir cuál es el uso de la capa las reglas son (por ejemplo, "una capa puede utilizar cualquier capa inferior" o "una capa se permite usar solo la capa inmediatamente debajo de ella ”) y cualquier excepciones permitidas.

Restricciones

■ Cada pieza de software se asigna exactamente a una capa.
■ Hay al menos dos capas (pero normalmente hay tres o más).
■ Las relaciones de uso permitido no deben ser circulares (es decir, una menor capa no puede usar una capa superior).

Debilidades

■ La adición de capas agrega un costo inicial y complejidad a un sistema.
■ Las capas contribuyen a reducir el rendimiento

Algunos puntos más finos de capas

Una arquitectura en capas es uno de los pocos lugares donde las conexiones entre los componentes se pueden mostrar por adyacencia, y donde "arriba" y "abajo" importa. Si pone la Figura 13.1 boca abajo de modo que C esté arriba, esto representan un diseño completamente diferente. Diagramas que usan flechas entre las casillas para denotar relaciones conservan su significado semántico sin importar el orientación.

El patrón de capas es uno de los patrones más utilizados en ingeniería de software, pero a menudo me sorprende la cantidad de personas que aún obtienen está mal. Primero, es imposible mirar una pila de cajas y saber si el puente entre capas está permitido o no. Es decir, ¿puede una capa usar cualquier capa inferior, o simplemente el siguiente más bajo? Es lo más fácil del mundo resolver esto; todos

El arquitecto debe incluir la respuesta en la clave de la nota del diagrama (algo que recomendamos para todos los diagramas). Por ejemplo, considere el patrón en capas presentado en la Figura 13.2 en la página siguiente.

Pero todavía me sorprende que pocos arquitectos se molesten en hacer esto.

Y si no es así, sus diagramas de capas son ambiguos.

En segundo lugar, cualquier conjunto antiguo de cajas apiladas una encima de la otra no constituyen una arquitectura en capas. Por ejemplo, mire el diseño que se muestra en la Figura 13.3, que usa flechas en lugar de adyacencia para indicar las relaciones entre los cuadros. Aquí, todo está permitido para usarlo todo. Esta definitivamente no es una arquitectura en capas. La razón es que si la capa A se reemplaza por una versión diferente, la capa C (que la usa en esta figura) bien podría tener que cambiar. No queremos que nuestra capa de máquina virtual cambia cada vez que cambia nuestra capa de aplicación. Pero todavía me sorprende cuántas personas llaman "capas" a una pila de cajas alineadas entre sí (o cree que las capas son iguales que las capas en una arquitectura de varios niveles).

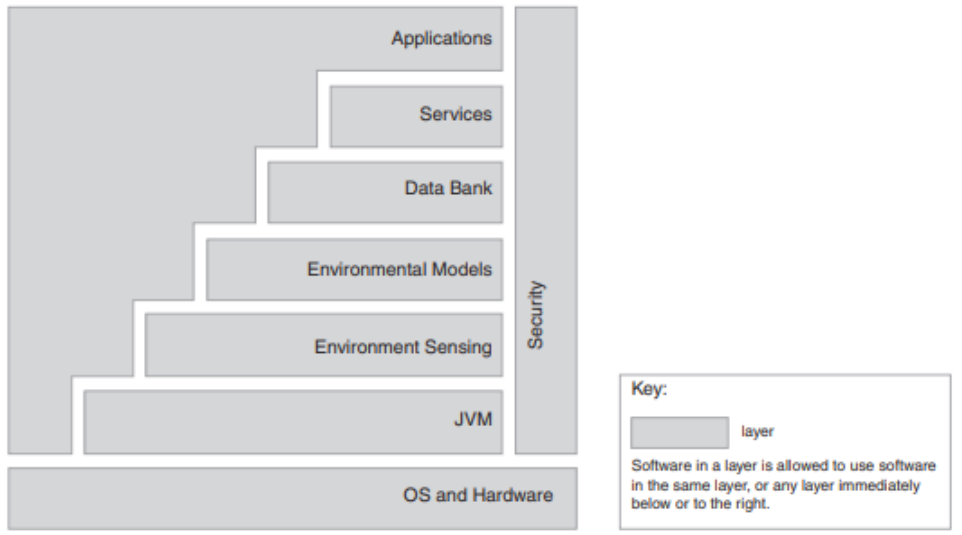

FIGURE 13.2 A simple layer diagram, with a simple key answering the uses question

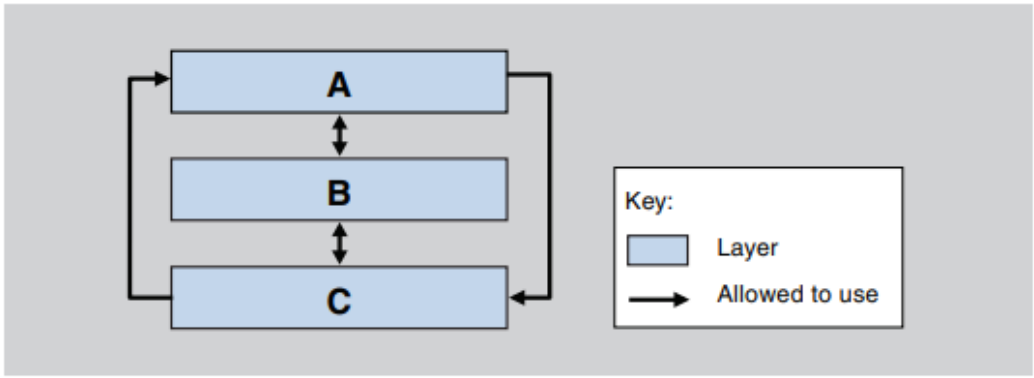

FIGURE 13.3 A wolf in layer's clothing

En tercer lugar, muchas arquitecturas que pretenden tener capas se ven algo como la Figura 13.4. Este diagrama probablemente significa que los módulos en A, B o C puede usar módulos en D, pero sin una clave que nos diga con seguridad, podría significar cualquier cosa. Los "sidecars" como este suelen contener utilidades comunes (a veces importados), como controladores de errores, protocolos de comunicación o bases de datos o mecanismos de acceso. Este tipo de diagrama tiene sentido solo en el caso donde no se permite el puenteo de capas en la pila principal. De lo contrario, D podría simplemente convertirse en la capa más inferior de la pila principal, y el "sidecar" la geometría sería innecesaria. Pero todavía me sorprende la frecuencia con la que veo este diseño no tiene explicación.

A veces, las capas se dividen en segmentos que denotan un grano más fino. descomposición de los módulos. A veces esto ocurre cuando un preexistente conjunto de unidades, como los módulos importados, comparten el mismo uso permitido relación. Cuando esto sucede, debe especificar qué reglas de uso están en efecto entre los segmentos. Son posibles muchas reglas de uso, pero deben hacerse explícito. En la Figura 13.5, las capas superior e inferior están segmentadas. Los segmentos de la capa superior no pueden usarse entre sí, pero los segmentos de la capa inferior sí lo son. Si dibuja el mismo diagrama sin las flechas, será más difícil diferenciar las diferentes reglas de uso dentro de capas segmentadas. Los diagramas en capas son a menudo una fuente de datos ocultos. ambigüedad porque el diagrama no hace explícito el uso permitido relaciones.

Finalmente, el punto más importante sobre las capas es que una capa no tiene permitido usar cualquier capa por encima de ella. Un módulo "usa" otro módulo cuando depende de la respuesta que reciba. Pero se permite hacer una capa hacia arriba llamadas, siempre que no espere una respuesta de ellos. Así es como el El esquema común de manejo de errores de las devoluciones de llamada funciona. Un programa en la capa A llama a un programa en una capa inferior B, y los parámetros incluyen un puntero a un programa de manejo de errores en A que la capa inferior debe llamar en caso de error. El software en B hace la llamada al programa en A, pero

no le importa en al menos lo que hace. Al no depender en modo alguno de los contenidos de A, B está aislado de los cambios en A.

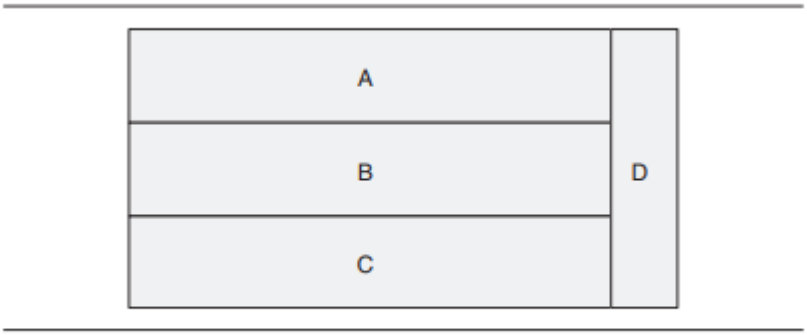

FIGURE 13.4 Layers with a "sidecar"

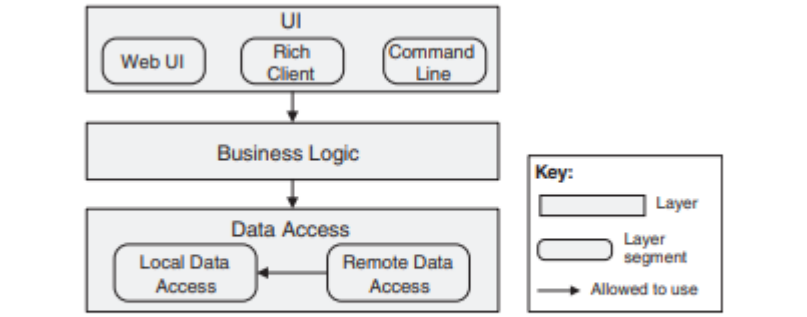

FIGURE 13.5 Layered design with segmented layers

Patrón de tubería y filtro -Pipe and Filter Pattern

Contexto: se requieren muchos sistemas para transformar flujos de elementos de datos discretos, de entrada a salida. Muchos tipos de transformaciones ocurren repetidamente en la práctica, por lo que es deseable crearlos como piezas independientes y reutilizables.

Problema: Dichos sistemas deben dividirse en componentes reutilizables, débilmente acoplados, con mecanismos de interacción genéricos y simples. De esta manera pueden ser combinados de forma flexible entre sí. Los componentes, siendo genéricos y vagamente acoplados, se reutilizan fácilmente. Los componentes, al ser independientes, pueden ejecutarse en paralelo.

Solución: se caracteriza el patrón de interacción en el patrón de tubería y filtro mediante sucesivas transformaciones de flujos de datos. Los datos llegan a la entrada de un filtro puerto (s), se transforma y luego se pasa a través de su puerto (s) de salida a través de una tubería para el siguiente filtro. Un solo filtro puede consumir datos de, o producir datos para, uno o más puertos.

Hay varias debilidades asociadas con el patrón de tubería y filtro.
Por ejemplo, este patrón no suele ser una buena opción para un sistema interactivo, ya que no permite los ciclos (que son importantes para los comentarios de los usuarios). Además, tener grandes cantidades de filtros independientes puede agregar cantidades sustanciales de sobrecarga, porque cada filtro se ejecuta como su propio proceso o subproceso. Además, los sistemas de tuberías y filtros pueden no ser apropiados para cálculos de larga duración, sin la adición de alguna forma de funcionalidad de punto de control / restauración, ya que la falla de cualquier filtro (o tubería) puede hacer que toda la tubería falle.
La solución del patrón de tubería y filtro se resume en la tabla 13.4.
Las tuberías almacenan datos en búfer durante la comunicación. Debido a esta propiedad, los filtros pueden ejecutar de forma **asincrónica y concurrente**. Además, un filtro normalmente no conoce la identidad de sus filtros aguas arriba o aguas abajo. Por esta razón, pipeline Los sistemas de tubería y filtro tienen la propiedad de que el cálculo general puede ser tratada como la composición funcional de los cálculos de los filtros, por lo que más fácil para el arquitecto razonar sobre el comportamiento de un extremo a otro.
Los sistemas de transformación de datos suelen estar estructurados como conductos y filtros, con cada filtro responsable de una parte de la transformación general de la entrada datos. El procesamiento independiente en cada paso admite la reutilización, paralelización y razonamiento simplificado sobre el comportamiento general. A menudo, tales sistemas constituyen el front end de aplicaciones de procesamiento de señales. Estos sistemas reciben datos de sensores en un conjunto de filtros iniciales; cada uno de estos filtros comprime los datos y realiza procesamiento (como suavizado). Los filtros descendentes reducen aún más los datos y hacen síntesis a través de los datos derivados de diferentes sensores. El filtro final normalmente pasa sus datos a una aplicación, por ejemplo, proporcionando información para el modelado o herramientas de visualización.

Otros sistemas que utilizan tuberías y filtros incluyen los construidos con tuberías UNIX, la arquitectura de procesamiento de solicitudes del servidor web Apache, el mapa-reducir patrón (presentado más adelante en este capítulo), Yahoo! Tuberías para procesar feeds RSS, muchos motores de flujo de trabajo y muchos sistemas de cálculo científico que tienen que procesar y analizar grandes flujos de datos capturados. La figura 13.8 muestra un UML diagrama de un sistema de tubería y filtro.

Tabla 13.4 Solución de patrón de tubería y filtro

Visión general

Los datos se transforman de las entradas externas de un sistema a sus salidas a través de una serie de transformaciones realizadas por sus filtros conectado por tuberías.

Elementos

Filtro, que es un componente que transforma los datos leídos en su entrada puerto (s) a los datos escritos en sus puertos de salida. Los filtros se pueden ejecutar simultáneamente entre sí. Los filtros pueden transformar datos de forma incremental; es decir, pueden comenzar a producir resultados tan pronto como comiencen a procesar los insumos. Las características importantes incluyen tasas de procesamiento, formatos de datos de entrada / salida y la transformación ejecutada por el filtro.
Pipe, que es un conector que transmite datos desde los puertos de salida de un filtro a los puertos de entrada de otro filtro. Una tubería tiene una única fuente para su entrada y un único objetivo para su salida. Una tubería conserva la secuencia de elementos de datos y no altera el paso de los datos .Las características importantes incluyen el tamaño del búfer, el protocolo de interacción, la velocidad de transmisión y el formato de los datos que pasan a través de una tubería.

Relaciones

La relación de apego asocia la salida de filtros con la entrada de tuberías y viceversa.

Restricciones

Las tuberías conectan los puertos de salida del filtro a los puertos de entrada del filtro. Los filtros conectados deben estar de acuerdo con el tipo de datos que se transmiten en el tubo de conexión.
Las especializaciones del patrón pueden restringir la asociación de componentes a un gráfico acíclico o una secuencia lineal, a veces llamado tubería.
Otras especializaciones pueden prescribir que los componentes tengan ciertos puertos con nombre, como los puertos stdin, stdout y stderr de UNIX filtros.

Debilidades

El patrón de tubería y filtro generalmente no es una buena opción para un sistema interactivo.
Tener una gran cantidad de filtros independientes puede agregar cantidades de gastos generales computacionales.
Los sistemas de tuberías y filtros pueden no ser apropiados para operaciones de larga duración.

cálculos.

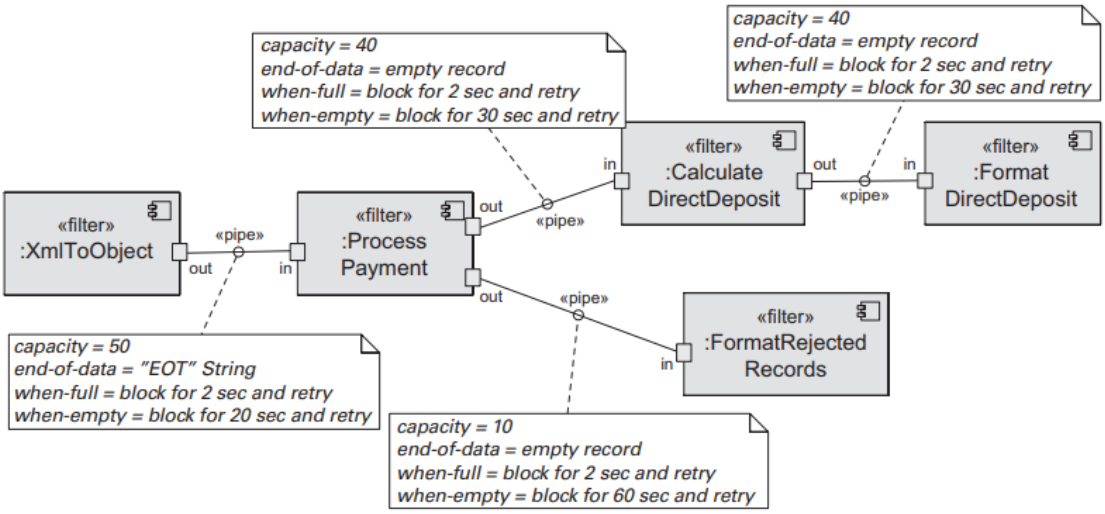

FIGURE 13.8 A UML diagram of a pipe-and-filter-based system