Diseño (3)

Alejandro Teruel

17 de mayo 2013

Contenido: La arquitectura y los patrones de diseño de software. Arquitectura por capas

Niveles de diseño

Hasta ahora hemos visto cómo aplicar los principios de acoplamiento, cohesión, encapsulamiento y ocultamiento de información al diseño de métodos y clases.

Adicionalmente introdujimos los diagramas de clase como una notación que puede usarse para registrar el diseño estructural de un sistema en términos de sus clases y las relaciones entre las clases.

Un software mediano puede tener decenas y hasta centenas de clases -es obvio que para entender tal estructura tenemos que ir agrupando o encapsulando las clases por afinidad de algún tipo.

Nuevamente podemos aplicar, recursivamente los principios de acoplamiento y cohesión para tales agrupaciones.

Una pregunta clave es si, así como las clases sugieron del encapsulamiento de rutinas (renombradas como métodos) cuya afinidad resultó ser la de operar sobre una agrupación de variables, ¿surge algún otro concepto al agrupar clases?

Algunos desarrolladores de software han venido identificando ciertos patrones en esos agrupamientos. Algunos patrones surgen cuando varias clases colaboran para un objetivo emergente mientras que otros, de mayor nivel parecen surgir bajo una visión que se ha denominada "arquitectónica".

La metáfora arquitectónica aplicada al software se toma por analogía con la relación entre arquitectura e ingeniería civil. El arquitecto concibe la metáfora , la apariencia, la *visión del todo* del artefacto y la ingeniería toma tal concepción como el punto de partida para traducir la metáfora a la realidad de pilotes, vigas, columnas, tuberías, cableados, para anclar la apariencia en un fundamento, para cerciorarse que la *visión del todo* pervade las partes, para que los patrones previstos de uso sean atendidas apropiadamente por las capacidades de funcionamiento del artefacto.

Christopher Alexander identificó y formalizó la idea de *patrones* en la disciplina de Arquitectura (*The Timeless Way of Building*). Expuso cómo ciertos componentes arquitectónicos tienen una función que obedecen a necesidades psicológicas. Por ejemplo, una buena arquitectura requiere de un espacio de transición entre el exterior y el interior, un portal de vislumbramiento y alumbramiento de mundos diferentes. Ese patrón arquitectónico, en la arquitectura criolla clásica se llama *zaguán;* en otros mundos arquitectónicos recibe nombres como *foyer*, vestíbulo o antesala. Cuando corresponde a la transición al mundo privado, personal, íntimo (a diferencia de cuando corresponde a una transición a un mundo corporativo, de hotel u oficinas) concede como mínimo ecos de pasadizos, túneles, canales -el espacio se encoge, se oscurece, susurra. Para Alexander cada patrón arquitectónico o urbanístico describe:

- Un problema que se presenta una y otra vez en el dominio de interés;

- El núcleo o esencia de lo que constituye una posible solución a ese problema, de manera que puedes aprovechar esa esencia de un millón de formas diferentes para diseñar una solución.

La esencia del vestíbulo es el mismo en todas las casas en que se encuentre, pero en todas se manifiesta de manera diferente.

Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides[1] se inspiraron en las ideas de Christopher para presentar un primer grupo de patrones de diseño para el software en su libro *Design Patterns: Elements of Reusable Object-Oriented Software* (1995). Para ellos, todo patrón puede describirse en términos de trece elementos esenciales, de los que destacaremos cuatro:

- El nombre dado al patrón;
- El contexto del problema en que es aplicable el patrón. Esto constituye la motivación del patrón y a veces se describen las fuerzas que inciden en la necesidad de una solución;
- Los elementos que se interrelacionan para proporcionar una solucuón y cómo lo hacen (el diseño, las relaciones, las responsabilidades y la forma en que colaboran los elementos);
- Las consecuencias (resultados y compromisos) de aplicar el patrón.

Arquitectura de Capas

*[***Preguntar qué conocen de arquitectura de capas (en Sistemas de Operación...]*

En un software con arquitectura de capas, se agrupan las clases por "capas". Usualmente las capas más internas corresponden a menores niveles de abstracción (están más cerca del hardware), mientras que las capas más externas corresponden a mayores niveles de abstracción o cercanía al dominio del usuario final.

Entre las arquitecturas por capas más conocidas se encuentran:

- El modelo ISO OSI de 7 capas para protocolos de redes de computadores,
- La biblioteca gráfica OpenGL y
- La arquitectura de la mayor parte de los sistemas de operación actuales.
- Arquitecturas de dos, tres y cuatro capas para Sistemas de Información. En estos casos las capas suelen recibir, en Inglés, el nombre de *tiers* y no *layers*.

A título de ejemplo consideremos los niveles de abstracción representados en tres de las siete capas del modelo OSI:

- La capa 1 (la más interna) corresponde al nivel *físico*. Sus objetos sólo saben que transmiten y

1 En un curioso arranque humorístico, este grupo de autores recibió el mote de *Gang of Four* a raíz de la publicación del libro pionero *Design Patterns* (Addison-Wesley, 1995) en posible alusión indirecta a la facción política china formada por cuatro altos dirigentes comunistas cuyo juicio y condena impactaron a la opinión pública en 1981 y posible alusión directa al grupo de rock post-punk del mismo nombre que se cita como influencia importante para grupos como *REM, Red Hot Chili Peppers* y *Nirvana* (http://en.wikipedia.org/wiki/Gang_of_Four_%28band%29). El epíteto sobrevive en el imaginario político-periodístico de habla inglesa como puede deducirse de la lectura de : https://en.wikipedia.org/wiki/Gang_of_Four_%28disambiguation%29

reciben un flujo de bits por conexiones físicas punto a punto. Pueden hacer uso de algunas características de estas conexiones, pueden por ejemplo variar la velocidad de la transmisión o manipular parámetros que correspondan a la especificidad del canal físico (canal inalámbrico, fibra óptica, par trenzado, etc.)

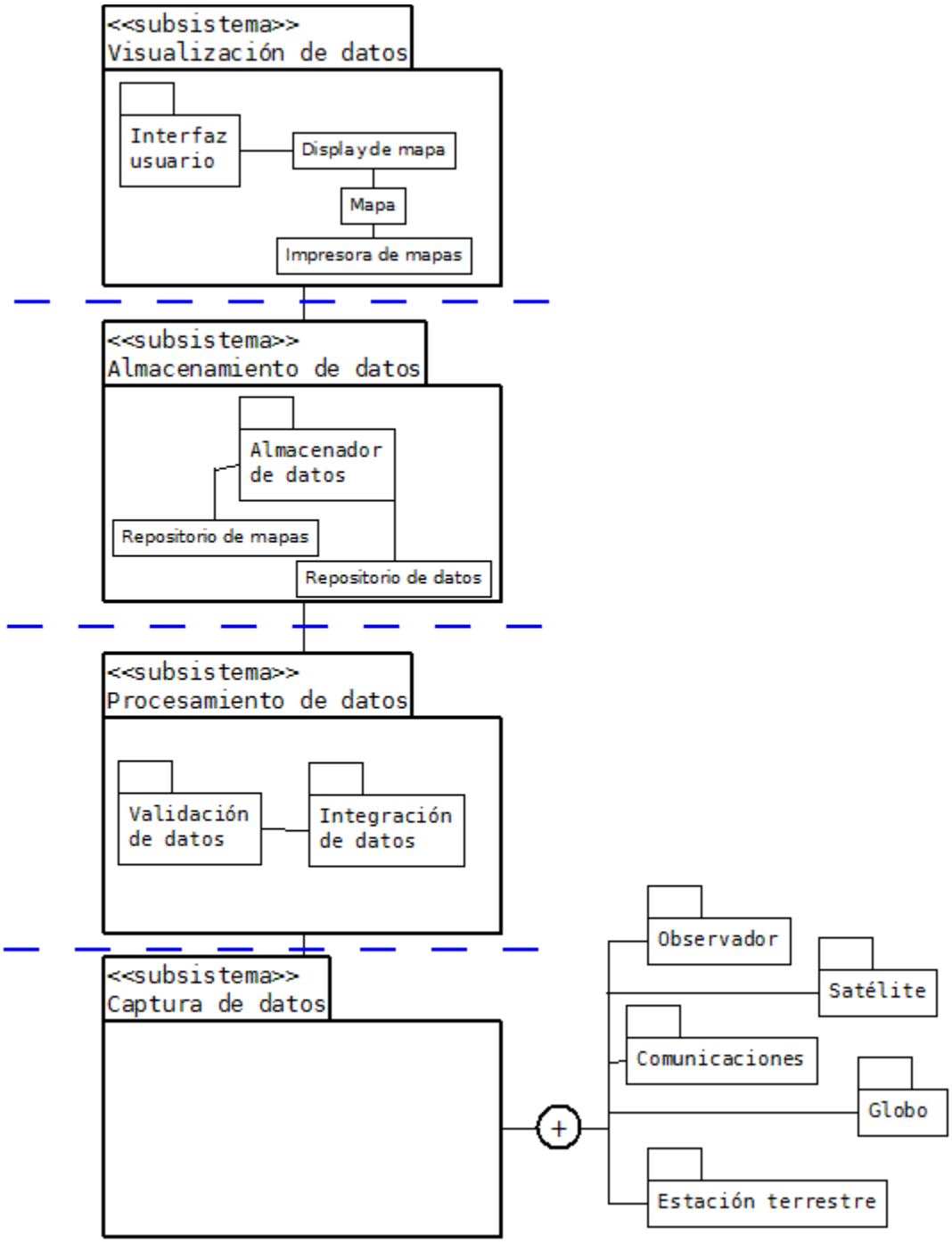
- La capa 2 corresponde al nivel de *enlace de de datos.* Abstraen los flujos de bits a *secuencias codificadas de bits.* Los códigos permiten detectar y corregir bits que se corrompen en la transmisión y recepción que ocurre a nivel de la capa 1.
- La capa 3 corresponde al nivel de la red. Los objetos de este nivel saben de *rutas* y por ende puede escoger enviar un paquete de bits de un nodo de la red a otro por una ruta o por otra. Incluso, si un nodo intermedio está caído, un nodo adyacente que esté enterado de esta falla puede re-rutear los paquetes para evitar que queden esperando en un punto hasta que se reestablezca la comunicación. También pueden encargarse de escoger la mejor ruta para el paquete de acuerdo con la información que manejen sobre la congestión y el tráfico en la red.

Así por ejemplo, a nivel más abstracto, el usuario maneja la noción (abstracta) de un correo electrónico. En el nivel de la aplicación se manejan conceptos como cuerpo de un mensaje, la dirección electrónica del destinario (pepito@xxx.com), la fecha de envío. Estos elementos terminan convertidos en bits que se transmiten por canales físicos hasta llegar al nodo donde se hace el proceso inverso y se reconstruye y entrega el mensaje enviado al destinatario.

Una arquitectura de capas aplicada a clases propone una partición de las clases de un software. Cada conjunto de la partición se considera un encapsulamiento por nivel de abstracción conceptual[2]. En términos de los niveles de cohesión que hemos manejado, cada capa presenta cohesión lógica como mínimo, pero puede presentar cohesión temporal o hasta comunicacional. Nótese que el criterio primordial de agrupamiento es por afinidad de nivel de abstracción; el acoplamiento no juega un papel definitorio, en principio, en la definición de las fronteras de las cápsulas.

Sommerville presenta una arquitectura de capas para un sistema meteorológico que abarca desde la captura de los datos obtenidos en diversos dispositivos,su almacenamiento, procesamiento y la visualización de esos mapas en forma de mapas:

2 El concepto que caracteriza o resume la abstracción encapsulada por la capa es lo que hemos denominado su concepto *emergente* aunque más que emerger, *dirige* o *determina* el diseño de la capa.



El diagrama es un ejemplo del diagrama de paquetes UML. Un *paquete* UML es un contenedor que encapsula artefactos UML: puede contener otros paquetes, clases o casos de uso, entre otros. Ayuda a organizar diagramas grandes para que sean más manejables e inteligibles. El sistema presentado por Sommerville muestra una arquitectura de cuatro capas, donde cada capa *Captura de datos, Procesamiento de datos, Almacenamiento de datos* y *Visualización de datos* está representada en el diagrama por un paquete estereotipado como <<*subsistema*>>. Note como cada capa está vinculada sólo a su capa adyacente. Adicionalmente se muestra la organización de cada capa en términos de (sub)paquetes y, en el caso de las capas *Visualización de datos* y *Alemacenamiento de datos,* clases.

Esencia del patrón

- Nombre del patrón: Capas (o arquitectura de capas);
- Problema: La necesidad de agrupar clases en subsistemas con cierta cohesión.
- Solución: Agrupar las clases por nivel de abstracción conceptual.
- Consecuencias: Hay que diseñar la filosofía de cómo y cuando se pasa de un nivel de abstracción a otro sin caer en excesivos cambios en niveles de abstracción y escogiendo un nivel de granularidad apropiada para que la abstracción sea significativa y tenga fuerza conceptual propia.

Variantes de la arquitectura de capas

Las variantes de las arquitecturas de capas comienzan cuando empiezan a tomarse en cuenta criterios adicionales a la cohesión, tales como lo son los criterios de ocultamiento de información y acoplamiento débil:

- Una arquitectura por capas puede adicionalmente ocultar la estructura de las abstracciones que maneja. En tal caso se requiere algún ente que traduzca de una abstracción a otra cuando se intente comunicar de un objeto de una capa a un objeto de otra capa -el flujo de bits de la capa 1 necesita traducirse a una secuencia codificada de bits al ser pasada a la capa 2.
- Podemos reducir el acoplamiento de una capa si exigimos que los objetos de una capa sólo se pueden comunicar con objetos de una capa *adyacente*;
- También reducimos el acoplamiento si creamos *interfaces* entre las capas, obligando, o por lo menos estimulando, a los desarrolladores a ajustarse a un diseño en el que los objetos de una capa se comunican a través de la interfaz con la capa adyacente y no directamente con cualquier objeto de esa capa adyacente. Este uso de interfaces reduce considerablemente el esfuerzo requerido para cambiar la estructura interna de una capa.

Otras posibles consecuencias y características de las arquitecturas por capas

1. La arquitectura de capas también requiere que se defina una filosofía de diseño coherente respecto al control particularmente en lo relacionado al tratamiento de fallas. Cuando un objeto de una clase detecta una falla, le solicita a la capa inferior un retrabajo o le avisa a la capa superior para que ésta, con una visión más abstracta, y presumiblemente, más global, de lo que significa la falla tome acciones para informar, contener o superar la falla.
2. Uno de los puntos álgidos de una arquitectura de capas, pueden ser los costos de la transmisión y traducción para pasar de una capa a otra. También puede producirse una *cascada* de solicitudes de servicios de las capas superiores a las capas inferiores, a medida que la solicitud de la capa superior va traduciéndose en requisitos a las capas inferiores. En líneas generales la eficiencia puede disminuir a medida que se incremente el número de capas y la flexibilidad de la arquitectura puede aumentar con el número de capas. El número y *granularidad* de las capas es un elemento importante a tomar en el diseño de una arquitectura de capas.
3. La arquitectura puede apuntar a que las capas más internas ofrezcan menos servicios. En la

práctica, en ciertos contextos, se ha encontrado que diseñar una *pirámide invertida de reúso* contribuye a una mayor reutilización de las capas inferiores.

4. Si una capa termina incorporando muchas clases, puede volver a aplicarse la idea de esta arquitectura para identificar subcapas. En otros casos, ante la amenaza de proliferación de *capas horizontales*, se ha preferido introducir la noción de *capas verticales.* Las (sub)capas verticales suelen basarse en una noción de particionamiento de los servicios ofrecidos por una capa.
5. ¿Cuánta información pasar de una capa a otra?
   La comunicación entre capas puede ocasionalmente requerir incluir información variable. Si adoptamos modelos puros tipo *push* ("empujar"), en muchos casos podemos terminar por enviar información de más a la estrictamente requerida para un caso particular -afectando así el nivel de acoplamiento entre las capas. En el modelo contrario, "halado" (*pull* o por demanda), la capa a que se requiere el servicio solicita mayor información de la capa llamada. Para que este modelo prospere la capa a que se solicita el servicio tiene que saber a quién dirigirse para pedir información adicional -necesidad que también afecta el acoplamiento. En un capítulo posterior veremos como el patrón Editorial-Suscriptor (Publish-Subscribe) ayuda a mantener el acoplamiento en niveles mínimos.

Una reflexión final sobre arquitecturas de software

Una arquitectura de software explicita una estructura *subyacente* de ciertos componentes claves y sus interrelaciones. Larry Best proporciona un excelente ejemplo cuando trae a colación software de gestión para el cobro a morosos y software de gestión de despacho para reparaciones. A primer golpe de vista las dos aplicaciones son muy diferentes. En la gestión de cobros a morosos, el personal de oficina se comunica con personas que no están al día en sus pagos. En el despacho de cuadrillas de reparación, el personal técnico se apertrecha con las herramientas y repuestos apropiados y se trasladan a los sitios donde deben llevar a cabo las reparaciones. Sin embargo, bajo las diferencias de los dos software se encuentran una cantidad de elementos comunes centrados en el flujo de tareas. Ambos software se benefician de estructurarse como aplicaciones centradas en flujos de tareas donde el sistema asigna a uno o más trabajadores a encargarse de tareas de un grafo de tareas de acuerdo con una prioridad determinada basada en habilidades y competencias específicas, asiste al trabajador a completar las tareas, le hace el seguimiento a los resultados y provee una variedad de otros servicios de soporte.

La estructura común a ambas aplicaciones es una arquitectura orientadas a flujos de tareas, donde destacan objetos tales como tareas, colas de tareas, recursos, herramientas, habilidades, trabajador, grupo de trabajo, calendario y plan de trabajo.

La esencia de una arquitectura de software se encuentra en la comprensión de los aspectos comunes a varias aplicaciones; la comprensión arquitectónica es un factor crucial en la reutilización efectiva de componentes.

Lecturas recomendadas

[Necesitan ser actualizadas] Las dos referencias clásicas al tema de arquitectura de software son:

- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommelad, Michael Stal: *A System of Patterns: Pattern-Oriented Software Architecture.* Wiley, 1996. La Sección 2.2 (*From Mud to Structures)* contiene una interesante introducción al tema de arquitectura por capas. Algunos de los ejemplos ya están un poco viejos...
- Mary Shaw, David Garlan: *Software Architecture: Perspectives on an emerging discipline.* Prentice-Hall, 1996.

[Necesita ser actualizado] Una referencia a la arquitectura de capas para sistemas de información:

- Alejandro Teruel: *Arquitectura de Capas.* Última actualización 21 de mayo 2001. http://ldc.usb.ve/~teruel/ci3715/clases/arqCapas.html

[A ser considerado en el futuro para su posible inclusión en el curso]

- Larry Best: *What is Architectural Software Development?* Portand Pattern Repository, 1995. http://c2.com/ppr/ams.html Consultado 03/05/2013 Se mencionan algunas otras posibles arquitecturas de software de apoyo empresarial.

El ejemplo de la arquitectura de capas para un software meteorológico se tomó de:

- Ian Sommerville: *Software Engineering.* 8° edición. Addison-Wesley, 2007. La sección 14.2 ilustra el proceso de diseño orientado a objetos.

y la explicación de la notación UML utilizada en este capítulo puede encontrarse en:

- Russ Miles, Kim Hamilton: *Learning UML 2.0.* O´Reilly, 2006. El capítulo 13 (*Organizing your model: Packages*) introduce la noción de paquetes en UML.