Introducción

El diseño de la arquitectura trata de entender cómo se debe organizar un sistema y cómo se tiene que diseñar la estructura global de ese sistema. El diseño de la arquitectura es la primera etapa en el proceso de diseño del software. Es el enlace crucial entre el diseño y la ingeniería de requisitos, ya que identifica los principales componentes estructurales de un sistema y la relación entre ellos. El resultado de esta etapa es un modelo arquitectónico que describe la forma en que el sistema se organiza como un conjunto de componentes de comunicación.

La arquitectura de software es importante porque afecta al rendimiento y, a la capacidad de distribución y mantenimiento de un sistema (Bosch, 2000). Como afirma Bosch, los componentes individuales implementan los requisitos funcionales del sistema. Los requisitos no funcionales dependen de la arquitectura del sistema, es decir, de la forma en que dichos componentes se organizan y comunican. En muchos sistemas, los requisitos no funcionales están también influenciados por componentes individuales, pero no hay duda de que la arquitectura del sistema es la influencia dominante.

Decisiones estructurales

El diseño de la arquitectura de software es un proceso creativo en el cual se diseña una organización del sistema que cubrirá los requisitos funcionales y no funcionales de éste. Puesto que se trata de un proceso creativo, las actividades del proceso dependen del tipo de sistema que se va a desarrollar, los antecedentes y la experiencia del arquitecto del sistema, así como de los requisitos específicos del sistema. Por lo tanto, es útil pensar en el diseño de la arquitectura como un conjunto de decisiones a tomar en vez de cómo una secuencia de actividades.

Durante el proceso de diseño de la arquitectura, los arquitectos del sistema deben tomar algunas decisiones estructurales que afectarán profundamente al sistema y al proceso de desarrollo. Partiendo del conocimiento y de la experiencia, se deben considerar las siguientes preguntas fundamentales sobre el sistema:

1. ¿Cómo se va a dividir e sistema en componentes?
2. ¿Cómo deben interactuar los componentes?
3. ¿Cuál va a ser la interfaz de cada componente?
4. ¿Qué estilo arquitectónico se va a utilizar?

Cada sistema de software es único, pero los sistemas en el mismo dominio de aplicación tienen normalmente arquitecturas similares que reflejan los conceptos fundamentales de dicho dominio.

Debido a la estrecha relación entre los requerimientos no funcionales y la arquitectura de software, el estilo y la estructura arquitectónica particulares que se elijan para un sistema dependerán de los requisitos no funcionales (ver transparencia 4).

Existe un conflicto potencial entre algunas arquitecturas. Por ejemplo, utilizar componentes grandes mejora el rendimiento, y usar componentes pequeños aumenta la facilidad de mantenimiento. Si tanto el rendimiento como la facilidad de mantenimiento son requisitos importantes del sistema, entonces debe encontrarse algún compromiso. Esto en ocasiones se logra usando diferentes patrones arquitectónicos para distintas partes del sistema.

Herramientas de representación

En esta sección, abordamos las posibilidades de UML para modelar la arquitectura del sistema. Dicho modelado presenta dos aspectos:

- El modelado de la arquitectura del software y su estructuración en componentes (sistemas) y su interrelación (interfaces).
- El modelado de la arquitectura material y el reparto físico de los programas.

Un componente es una caja negra que ofrece servicios de software, que se describen por una o varias interfaces del componente.

Una interfaz es una clase abstracta que especifica un conjunto de características públicas. La idea clave detrás de las interfaces es separar la especificación de la funcionalidad (la interfaz) de su implementación. Una interfaz no se puede instanciar.

Los componentes también pueden depender de otros componentes para llevar a cabo los servicios que ofrecen. Esta dependencia se expresa en forma de una interfaz necesaria que describe los servicios deseados.

El modelado de los componentes y sus relaciones se describe mediante el diagrama de componentes.

El modelado de la arquitectura material describe los nodos y sus vínculos e incluye la localización de los elementos software dentro de los nodos en su forma física. La descripción se efectúa mediante el diagrama de despliegue.

El diagrama de componentes

Un componente es una unidad de software que proporciona una serie de servicios a través de una o varias interfaces. Se trata de una caja negra cuyo contenido queda fuera del interés de los clientes. Está totalmente encapsulado. La definición de los componentes recuerda a la definición de clases que implementan una o varias interfaces. Una clase que implementa una o varias interfaces es un componente. Por el contrario, un componente no es necesariamente una clase. En los componentes, las interfaces se pueden implementar con lenguajes de programación no orientados a objetos como puede ser el lenguaje C.

Para realizar operaciones internas, los componentes pueden depender de otros y, en este caso, se convierten en clientes de esos otros componentes. Como cualquier cliente de un componente, no conocen su estructura interna. Por lo tanto, sólo dependen de las interfaces de los componentes de los cuales son clientes. Estas interfaces reciben el nombre de *interfaces*

necesarias del componente *cliente*. La interfaces que describen los servicios proporcionados por un componente se denominan *interfaces suministradas*.

Las interfaces necesarias se representan mediante un semicírculo y los componentes dentro de un rectángulo con el estereotipo <<component>>. Este estereotipo se puede reemplazar por el icono del componente (ver transparencia 6)

También es posible usar la relación de realización para las interfaces suministradas y la relación de dependencia para las interfaces necesarias (ver transparencia 8). Esta representación alternativa, presenta la ventaja de detallar las firmas de los métodos contenidas en las interfaces.

En la orientación a objetos, la arquitectura del software de un sistema se construye mediante un conjunto de componentes vinculados por interfaces suministradas e interfaces necesarias. El diagrama de componentes describe esta arquitectura.

Los componentes son posiblemente el elemento UML más estereotipado. Esto es porque un componente se puede utilizar para representar muchos tipos diferentes de elementos. UML 2 proporciona un pequeño conjunto de estereotipos estándar de componentes (ver transparencia 7).

El diagrama de despliegue

En UML el despliegue es el proceso de asignar artefactos a nodos, o instancias de artefactos a instancias de nodo.

El diagrama de despliegue especifica el hardware físico sobre el que el sistema software se ejecutará y también especifica cómo el software se despliega en ese hardware.

El diagrama de despliegue transforma la arquitectura del software creada en diseño en la arquitectura física que lo ejecuta.

Un nodo representa un tipo de recurso computacional sobre el que se pueden desplegar los artefactos para su ejecución. Para nodos, existen dos estereotipos estándar (ver transparencia 9). La notación gráfica para nodos e instancias de nodos se muestra en esa misma transparencia

Los nodos se pueden anidar en nodos. Una asociación entre nodos representa un canal de comunicación a través del cual se puede pasar información.

En la forma de instancia, las instancias de nodo representan dispositivos físicos reales o instancias de entornos de ejecución que se ejecutan sobre esos dispositivos.

Un artefacto representa la especificación de un elemento concreto del mundo real. Los artefactos se despliegan en nodos. Algunos ejemplos de artefactos son: archivos fuentes, archivos ejecutables, scripts, tablas de base de datos, documentos y salidas del proceso de desarrollo (p. e., un modelo UML).

Una instancia de artefacto representa una instancia específica de un artefacto determinado, por ejemplo, una copia específica de un archivo ejecutable desplegado en una máquina determinada. Las instancias de artefacto se despliegan en instancias de nodo.

Un artefacto puede proporcionar la manifestación física para cualquier tipo de elemento UML. Normalmente, muestran uno o más componentes. Los artefactos se etiquetan con el estereotipo <<artifact>> y pueden tener un icono de artefacto situado en su esquina superior derecha. Los artefactos pueden depender de otros artefactos.

Todo artefacto tiene un nombre de archivo en su especificación que indica la ubicación física del artefacto. Por ejemplo, este nombre de archivo podría especificar una URL en la que se encontrara la copia maestra del artefacto. Las instancias de artefacto tienen nombre de archivo que apunta a la ubicación física de la instancia.

En la transparencia 10, se lista algunos estereotipos estándar de artefactos, proporcionados por UML, que representan diferentes tipos de archivos, y se muestra la notación gráfica de artefactos e instancias de artefactos.

En la transparencia 11 se muestran los dos tipos de relaciones de dependencia básicas entre los distintos elementos de un diagrama de despliegue. También se muestran ejemplos de estas relaciones.

Estilos arquitectónicos

La idea de los patrones arquitectónicos como forma de presentar, compartir y reutilizar el conocimiento sobre los sistemas software se usa ampliamente en la actualidad. El origen de esto fue la publicación de un libro sobre patrones de diseño orientados a objetos (Gamma *et al*., 1995). Los patrones arquitectónicos se propusieron en la década de 1990 con el nombre de "estilos arquitectónicos" (Shaw y Garlan, 1996).

Un patrón arquitectónico se puede considerar como una descripción abstracta, que se ensayó y puso a prueba en diferentes sistemas y entornos. De este modo, un patrón arquitectónico debe describir una organización de sistema que ha tenido éxito en sistemas previos. Debe incluir información sobre cuándo es y cuándo no es adecuado usar dicho patrón, así como sobre sus fortalezas y debilidades.

Arquitectura MVC

El muy conocido patrón Modelo-Vista-Controlador (MVC), es el soporte del manejo de la interacción en muchos sistemas basados en la Web.

En las transparencias 13 y 15 se presentan los modelos gráficos de la arquitectura asociada con el patrón MVC. En ellas se muestra la arquitectura desde diferentes vistas: la transparencia 13 muestra una vista conceptual; en tanto la transparencia 15 ilustra una posible arquitectura en tiempo de operación, cuando este patrón se usa para el manejo de la interacción en un sistema basado en Web.

Los conceptos de separación e independencia son fundamentales para el diseño de la arquitectura porque permiten localizar cambios. El patrón MVC separa elementos de un sistema, permitiéndoles cambiar de forma independiente. Por ejemplo, agregar una nueva vista o cambiar una vista existente puede hacerse sin modificar los datos subyacentes en el modelo.

Arquitectura en capas

El patrón de arquitectura en capas es otra forma de lograr separación e independencia (ver transparencia 16). Aquí la funcionalidad del sistema está organizada en capas separadas, y cada una se apoya sólo en las facilidades y los servicios ofrecidos por la capa inmediatamente debajo de ella.
Este enfoque en capas soporta el desarrollo incremental de sistemas. Conforme se desarrolla una capa, algunos de los servicios proporcionados por esta capa deben quedar a disposición de los usuarios. La arquitectura es cambiable y portátil. En tanto la interfaz no varíe, una capa puede sustituirse por otra equivalente. Más aún, cuando las interfaces de capa cambian o se agregan nuevas facilidades a una capa, solo resulta afectada la capa adyacente.

En la transparencia 16 se muestra un ejemplo de una arquitectura en capas con cuatro capas. La capa inferior incluye software de soporte al sistema, por lo general soporte de base de datos y sistema operativo. La siguiente capa es la de la aplicación, que incluye los componentes relacionados con la funcionalidad de la aplicación, así como los componentes de utilidad que usan otros componentes de aplicación. La tercera capa se relaciona con la gestión de la interfaz de usuario y con brindar autentificación y autorización al usuario, mientras que la capa superior proporciona facilidades de interfaz de usuario. El número de capas es arbitrario. Cualquiera de las capas en el ejemplo podría dividirse en dos o más capas.

En la transparencia 17 se muestra un ejemplo de cómo puede aplicarse este patrón de arquitectura en capas a un sistema de biblioteca llamado LIBSYS, que permite el acceso electrónico controlado a material con derechos de autor de un conjunto de bibliotecas universitarias. Tiene una arquitectura de cinco capas y, en la capa inferior, están las bases de datos individuales de cada biblioteca.

Arquitectura de repositorio

Los patrones de arquitectura en capas y MVC son ejemplos de patrones en que la vista presentada es la organización conceptual de un sistema. El patrón de repositorio, describe cómo comparten datos un conjunto de componentes en interacción.

La mayoría de los sistemas que usan grandes cantidades de datos se organizan sobre una base de datos o un repositorio compartido. Por lo tanto, este modelo es adecuado para aplicaciones en las que un componente genere datos y otro los use. Los ejemplos de este tipo de sistema incluyen sistemas de información administrativa, sistemas CAD y entornos de desarrollo interactivo para software.

En la transparencia 18 se presenta una situación en la que puede usarse un repositorio. Este diagrama muestra una herramienta CASE que incluye diferentes herramientas para soportar desarrollo dirigido por modelo. En este caso, el repositorio puede ser un entorno controlado por versión que hace un seguimiento de los cambios al software y permite regresar a versiones anteriores.

Organizar herramientas alrededor de un repositorio es una forma eficiente de compartir grandes cantidades de datos. No hay necesidad de transmitir explícitamente datos de un componente a otro. Sin embargo, los componentes deben operar en torno a un modelo de repositorio de datos acordado. Inevitablemente, éste es un compromiso entre las necesidades específicas de cada herramienta y sería difícil o imposible integrar nuevos componentes si sus modelos de datos no se ajustan al esquema acordado.

En el ejemplo de la transparencia 18, el repositorio es pasivo, y el control es responsabilidad de los componentes que usan el repositorio.

Arquitectura cliente-servidor

El patrón de repositorio se interesa por la estructura estática de un sistema sin mostrar su organización en tiempo de operación. El patrón cliente-servidor ilustra una organización en tiempo de operación de uso muy común para sistemas distribuidos-

Un sistema que sigue el patrón cliente-servidor se organiza como un conjunto de servicios y servidores asociados, y de clientes que acceden y usan servicios. Los componentes básicos de este modelo son:

1. *Un conjunto de servidores que ofrecen servicios a otros componentes*. Ejemplos de estos incluyen servidores de impresión; servidores de archivo que brindan servicios de administración de archivos, y un servidor compilador, que proporciona servicios de compilación de lenguajes de programación.
2. *Un conjunto de clientes que solicitan los servicios que ofrecen los servidores*. Habrá usualmente varias instancias de un programa cliente que se ejecuten de manera concurrente en diferentes computadoras.
3. *Una red que permite a los clientes acceder a dichos servicios*. La mayoría de los sistemas cliente-servidor se implementan como sistemas distribuidos, conectados mediante protocolos de Internet.

Las arquitecturas cliente-servidor se consideran a menudo como arquitecturas de sistemas distribuidos; sin embargo, el modelo lógico de servicios independientes que opera en servidores separados puede implementarse en una sola computadora. De nuevo, un beneficio importante es la separación e independencia. Los servicios y servidores pueden cambiar sin afectar otras partes del sistema.

Es posible que los clientes deban conocer los nombres de los servidores disponibles, así como los servicios que proporcionan. Sin embargo, los servidores no necesitan conocer la identidad de los clientes o cuántos clientes acceden a sus servicios. Los clientes acceden a los servicios que proporciona un servidor, a través de llamada a procedimiento remoto usando un protocolo solicitud-respuesta, como el protocolo http utilizado en la www. En esencia, un cliente realiza una petición a un servidor y espera hasta que recibe respuesta.

En la transparencia 20 se presenta un ejemplo de sistema que se basa en el modelo cliente-servidor. Se trata de un sistema multiusuario basado en la Web, para ofrecer un repertorio de películas y fotografías. En este sistema, varios servidores manejan y despliegan los diferentes tipos de medios. El

catálogo debe manejar una variedad de consultas y ofrecer vínculos hacia el sistema de información Web, que incluye datos acerca de las películas y los videos, así como un sistema de comercio electrónico que soporte la venta de fotografías, películas y videos. El programa cliente es simplemente una interfaz de usuario integrada, construida mediante un navegador Web, para acceder a dichos servicios.

La ventaja más importante del modelo cliente-servidor es que es una arquitectura distribuida, que puede usarse de manera efectiva en sistemas en red con distintos procesadores distribuidos. Es fácil agregar un nuevo servidor e integrarlo al resto del sistema, o bien actualizar de manera clara servidores sin afectar otras partes del sistema.