Capítulo XIII

Diagramas de Componentes y de Despliegue

Diagramas de componentes y de despliegue

Tabla de contenido

1.- Componentes.....201
1.1.- Interfaces.....202
1.2.- Utilización de los diagramas de componentes.....204
2.- Diagramas de paquete.....205
3.- Diagramas de despliegue.....205
3.1.- Arquitecturas de varios niveles.....205
3.2.- Nodos.....207
4.- Combinación de las herramientas.....208

Diagramas de componentes y de despliegue

1.- Componentes

Un componente de software es una parte física de un sistema, como puede ser un módulo, una base de datos, un programa ejecutable, una biblioteca de programas, etc. Puede considerarse que un componente es la materialización de una o más clases. En efecto, las clases son conceptos –constituyen una abstracción de un conjunto de atributos y operaciones- que se implementan o materializan en los componentes.

Existen tres grandes grupos o tipos de componentes:

- Componentes de distribución

  Son los componentes que conforman un sistema, como los programas ejecutables, los DLL, controles ActiveX, Java Beans, etc.

- Componentes de trabajo

  Son los componentes con los que se crean los componentes de distribución, como los programas fuente, las bases de datos, etc.

- Componentes de ejecución

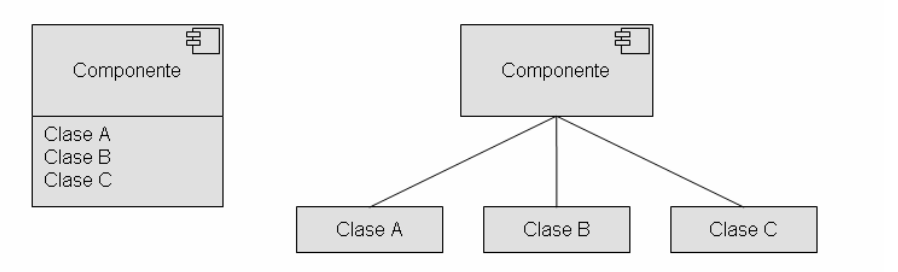
  Son los componentes que, en el transcurso de la ejecución de un sistema, se crean en forma dinámica, como los índices que crean los motores de búsqueda, como resultado de alguna consulta.

En un diagrama de componentes, un componente se representa con un rectángulo en el que se inscribe su nombre y en el que se muestran dos pequeños rectángulos en su lado izquierdo. También pueden utilizarse los símbolos que se muestran en la figura.



Muchas veces, para claridad del modelo, el nombre del componente se precede del nombre del "paquete" –módulo, aplicación o sistema- al cual pertenece el componente.

Las clases que implementa un componente pueden indicarse inscribiendo sus nombres en el rectángulo que representa al componente o mostrando las relaciones de dependencia con dichas clases.



1.1.- Interfaces

En el capítulo IX, en el que revisamos los diagramas de clase, se introdujo el concepto de interfaz, señalando que es "una clase que sólo contiene operaciones", en la que se incluyen operaciones que son comunes a varias clases.

En el caso de los componentes, el concepto es similar, añadiendo que la ejecución de un componente sólo puede ser hecha a través de su interfaz. Esto es, un componente presenta su interfaz para que otros componentes puedan utilizar sus operaciones y, sólo a través de su interfaz, podrá hacerse uso de sus operaciones.

Para un componente dado, existen dos tipos de interfaces: los interfaces provistos o requeridos y los interfaces utilizados.

Con el fin de tener una comprensión más clara de lo que es una interfaz, consideremos un ejemplo muy sencillo. Supongamos que hemos abierto la aplicación excel en nuestro microcomputador; en nuestra pantalla aparece el formato de filas y columnas que utiliza excel. Sabemos que "detrás de la pantalla", en la memoria de nuestro microcomputador está alojado un componente que hace los cálculos –llamémoslo excel/programa-, de acuerdo a los datos y comandos que recibe del formato que aparece en pantalla –llamémoslo excel/ pantalla -. Este último componente sólo realizará sus operaciones si se le indican los datos y las funciones establecidas por su sintaxis a través de excel/pantalla. Si nosotros -considerándonos como otro componente del sistema- deseamos que excel realice algún cálculo, tenemos que utilizar

ese formato y esa sintaxis, de lo contrario no obtendremos los resultados que buscamos. En la terminología de UML, diremos que excel/programa provee o realiza la interfaz excel/pantalla y que nosotros excel/usuario utilizamos esa interfaz.

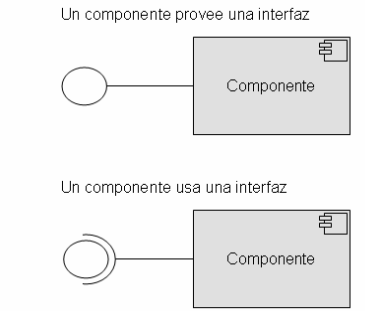
El concepto de interfaz es una de las bases fundamentales para la reutilización de objetos, pues un objeto puede ser reemplazado por otro si ambos tienen la misma interfaz.

En el capítulo arriba citado, mencionábamos el ejemplo de un sistema de recursos humanos en el que la clase *Vacación* corresponde a los períodos de vacación tomados por un empleado, que contiene la operación de *Cantidad-de-días-hábiles*. Mencionábamos también que esa operación es similar a la que se realiza en la clase *Interés-Devengado* en un sistema de finanzas. Veíamos que esta operación de cálculo de los días hábiles transcurridos entre dos fechas muy bien puede ser compartida –reutilizada- en esos dos sistemas o en otros sistemas o módulos. Ahora bien, para que esa operación pueda ser compartida, los módulos que la invoquen tienen que "hablar" con ella, utilizando las "mismas palabras" –en este caso, dos fechas, una de inicio del período y otra del final- o dicho de otro modo, *Vacación* puede ser reemplazado por *Interés-Devengado*, o por cualquier otro, si tiene la misma interfaz.

Expresando lo anterior en palabras mucho más simples, a *Cantidad-de-días-hábiles* le da lo mismo trabajar para *Vacación* que para *Interés-Devengado*, siempre y cuando se dirijan a ella con las mismas "palabras". Es decir, la operación de *Cantidad-de-días-hábiles* puede ser reutilizada por cualquier sistema, siempre y cuando se invoque en la misma forma.

Así pues, podemos entender que, dado que cada componente tiene una interfaz, que constituye su "puerta de entrada" frente al mundo de los componentes, cualquier sistema puede utilizar un componente determinado, accediendo a él a través de su interfaz. Dicho en los términos de la OMG, "un componente es una unidad sustituible que puede ser reemplazado en tiempo de diseño o en tiempo de ejecución por un componente que ofrezca una funcionalidad equivalente, basada en la compatibilidad de sus interfaces".

Las interfaces pueden representarse de varias formas, como vemos en la gráfica.



La relación entre un componente y una interfaz se denomina realización y está representada por la línea que une la interfaz y el componente.

La OMG, en sus documentos sobre UML, nos señala que:

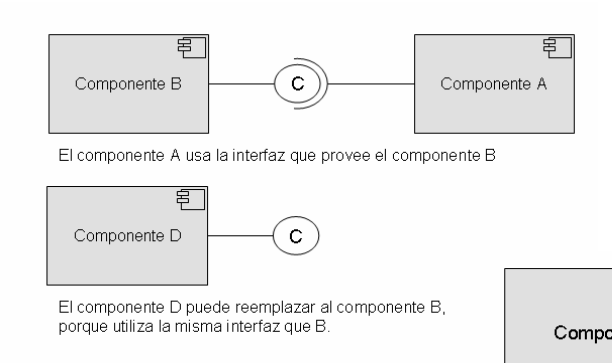
> "Una interfaz es un tipo de clasificador que representa la declaración de un conjunto coherente de características y obligaciones públicas. Una interfaz especifica un contrato; cualquier instancia de un clasificador que utilice la interfaz debe satisfacer ese contrato. Las obligaciones que se pueden asociar a una interfaz están dadas en la forma de varias clases de restricciones (tales como las pre y poscondiciones) o de especificaciones de protocolo, que pueden establecer restricciones de orden en las interacciones con la interfaz.
>
> Puesto que las interfaces son declaraciones, no son instanciables. Por el contrario, la especificación de una interfaz es puesta en ejecución por una instancia de un clasificador, lo que significa que el clasificador instanciable presenta una fachada pública que se conforma de acuerdo con la especificación de la interfaz. Obsérvese que un clasificador dado puede poner en ejecución más de una interfaz y que una interfaz se puede poner en ejecución por un número de diversos clasificadores."

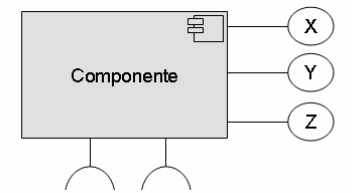
1.2.- Utilización de los diagramas de componentes

Los diagramas de componentes pueden ser utilizados para modelar sistemas de software de cualquier tamaño y complejidad. La herramienta nos permite especificar un componente como unidad modular con interfaces bien definidos, reemplazable dentro de su ambiente.

El concepto de componente encaja dentro de las ideas de desarrollo basado en componentes y estructuración de sistemas basada en componentes, en las cuales un componente se va modelando a través de todo el ciclo de desarrollo y sucesivamente se va refinando hasta llegar a

su implantación y creación de su "run time" –módulo ejecutable-. Un componente puede ser considerado como una unidad autónoma, dentro de un sistema o subsistema, tiene uno o más interfaces -proporcionados o requeridos- y sus "interioridades" permanecen ocultas e inaccesibles, con excepción de la forma que está prevista en sus interfaces.

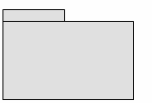




Si los componentes se diseñan de tal forma que puedan ser tratados tan independientemente como sea posible, esos componentes y los subsistemas que ellos conforman, podrán ser reutilizados y sustituidos en forma flexible, conectándolos a través de sus interfaces. Así mismo, una vez instalados, esos componentes pueden ser reimplementados independientemente, cuando sea necesario actualizar las funciones de un sistema en producción.

2.- Diagramas de paquete

Los diagramas de paquete, más que un diagrama constituyen una herramienta para mostrar los elementos que se integran en un sistema, aplicación o módulo.



3.- Diagramas de despliegue

3.1.- Arquitecturas de varios niveles

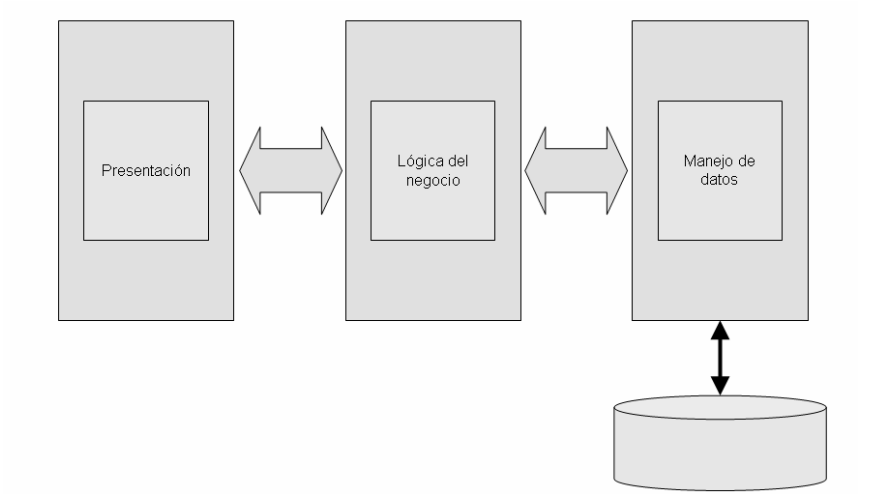
Como hemos mencionado en varias oportunidades, el proceso de desarrollo de sistemas ha evolucionado dramáticamente en los últimos años. En la última década esa evolución se ha visto impulsada por el paradigma de orientación a objetos (OO), a la utilización de Internet y las redes de datos, como medio global de comunicación, y a la unificación de

estándares, tanto de metodologías como de plataformas de desarrollo de software, entre ellas la plataforma para la construcción de aplicaciones web multinivel J2EE (Java 2 Enterprise Edition) y la plataforma Microsoft .NET .

Los sistemas cliente/servidor típicos pertenecen a la categoría de las aplicaciones que operan en dos niveles: la aplicación reside en el la estación de trabajo del usuario, comúnmente llamada el cliente, mientras que la base de datos se encuentra alojada en un servidor de datos. En este tipo de aplicaciones, podemos observar que el "esfuerzo de computación" realmente lo realiza la estación de trabajo cliente, mientras que el servidor sólo cumple las funciones de almacenamiento y extracción de los datos; esto es, no existe una distribución de la carga de trabajo proporcional a la capacidad de cómputo, pues los clientes, que suelen ser equipos de menor capacidad de cómputo que los servidores, realizan el trabajo más pesado. Adicionalmente, en este tipo de arquitecturas, el mantenimiento de las aplicaciones se vuelve engorroso, ya que cada modificación que se haga en cualquiera de ellas, deberá ser distribuida o trasladada a todos los clientes.

El concepto de arquitecturas de tres niveles viene a solucionar estos problemas, pues en ella se presentan tres grandes niveles:

- El nivel de presentación – ventanas, informes, etc.-
- El nivel de lógica de la aplicación – operaciones, cálculos y reglas que gobiernan el proceso-.
- El nivel de almacenamiento – mecanismos de almacenamiento y manejo de los datos-.



La capa de presentación sólo se encarga de enviar y recibir los datos, formateándolos adecuadamente, mientras que el segundo nivel realiza las operaciones y los cálculos. La tercera capa se encarga de almacenar y servir los datos que requieren las aplicaciones.

La separación entre la capa de presentación y la de lógica permite un mejor aprovechamiento de los recursos de computación y, además, brinda una gran flexibilidad para desarrollar y mantener aplicaciones, ya que se pueden tener múltiples interfaces sin cambiar la lógica de la aplicación o actualizar la lógica de la misma, sin modificar las interfaces –sin que afecte al usuario-.

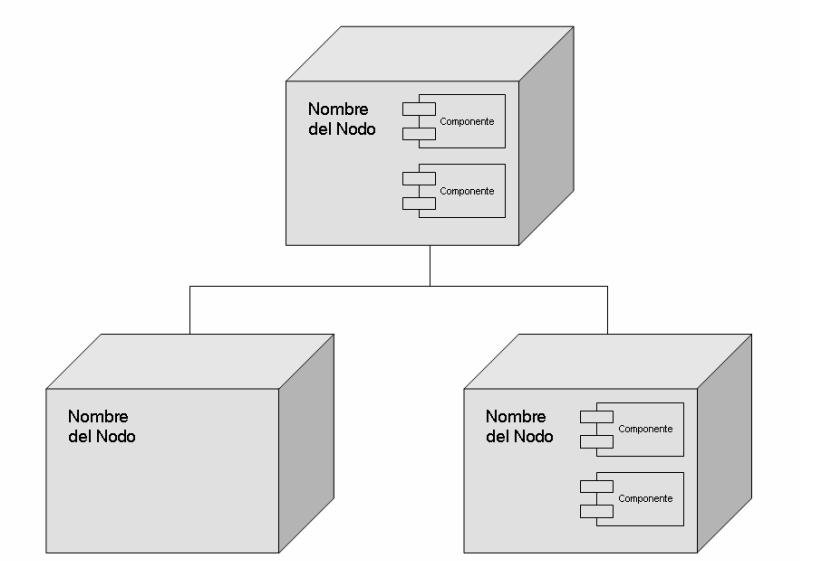
Aun cuando sólo se mencionan tres niveles, esta arquitectura se denomina multinivel, porque prácticamente cada uno de los tres niveles que hemos citado, puede descomponerse en otros niveles. Por ejemplo, el nivel de almacenamiento puede tener asociados servicios de replicación, para actualizar un datawarehouse, etc.

En una aplicación web, por ejemplo, tenemos unas estaciones de trabajo que utilizan un navegador – browser-. En este browser, a través de diferentes mecanismos -scripts html, java beans, etc.- la aplicación presenta a los usuarios los diferentes formatos de pantalla –nivel presentación-, que le permiten solicitar o enviar datos a un servidor de aplicaciones, en el que se encuentran unos programas que ejecutan las operaciones requeridas por la aplicación –nivel lógica de la aplicación- y que, a su vez, intercambian datos con un servidor de bases de datos –nivel almacenamiento- encargado de satisfacer sus requerimientos de información.

Sin entrar en muchos más detalles sobre la arquitectura de múltiples niveles, queremos destacar que esta separación de componentes en diferentes nodos o elementos de cómputo requiere que, al realizar el diseño detallado y el desarrollo de una operación, establezcamos qué cosa va en qué lugar y podamos tener a la mano un mapa de cómo fluirán las operaciones y cómo los componentes se comunicarán e interrelacionarán. UML incorpora como solución a esta necesidad, el uso de los diagramas de despliegue.

3.2.- Nodos

Al igual que los componentes los nodos pertenecen al mundo material. Vamos a definir un nodo como un elemento físico, que existe en tiempo de ejecución y representa un recurso computacional que generalmente tiene alguna memoria y, a menudo, capacidad de procesamiento.



Los nodos sirven para modelar la topología del hardware sobre el que se ejecuta el sistema. Un nodo representa normalmente un procesador o un dispositivo sobre el que se pueden desplegar los componentes.

4.- Combinación de las herramientas.

