Unidad V. Diseño y Desarrollo del Sistema
M. en C. Euler Hernández Contreras

Contenido

1. **Diseño de la Interfaz gráfica de usuario**
2. **Diseño del modelo de datos**
   a) Mapeo Objeto-Relacional
   b) Lenguajes de Programación Orientados a Objetos
3. **Vista Física**
   a) Diseño de la Arquitectura Física
4. **Diagrama de distribución**
   a) Vista de desarrollo
   b) Diagrama de Componentes
5. **Diagrama de Paquetes**
6. **Pruebas del Sistema**

Referencia Bibliográfica

1. Booch Grady, El lenguiaje Unificado de modelado, Pearson Educación

2. García Rubio, Félix Oscar. Medición y estimación del software: Técnicas y Métodos para mejorar la calidad y la productividad. AlfaOmega. México 2008. 332 págs. ISBN 9788478978588.

3. Humphrey Watts S. PSP: A Self-Improvement Process for Software Engineers. Addison Wesley. USA 2005. 364 págs. ISN 9780321305497.

4. Kan Stephen H., Metrics and Models in Software Quality Engineering, 2 edition. Addison-Wesley Professional

5. Kimmel Paul. Manual de UML. Mc Graw Hill. España 2006. 280 págs. ISBN 9789701058992

6. Neustadt Ila, Arlow Jim. UML 2. Anaya. España 2006. 608 págs. ISBN 9788441520332

7. Palacio B. Juan. Mirando Alrededor. El día a día en los proyectos Software. Lulu.com. España 2007. 192 págs. ISBN 9781847531339.

8. Piattini Mario, Calvo-Manzano J. Análisis y diseño de aplicaciones informáticas de gestión. Una perspectiva de Ingeniería del Software. Alfaomega. México 2004. 710 págs. ISBN 9701509870

9. Piattini Mario, García Félix. Calidad de Sistemas Informáticos. Alfaomega. México 2007. 388 págs. ISBN 9789701512678

10. Pressman Roger S. Ingeniería del software: Un enfoque Práctico. Mc Graw Hill. México 2005. 900 pags. ISBN 9701054733.

11. Priolo Sebastián. Métodos Ágiles. Users. Argentina 2009. 336 págs. ISBN: 97898134797-1

12. Schach Stephen. Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc Graw Hill. España 2005. 458 págs. ISBM 9789701049822

13. Sommerville Ian. Ingeniería de Software. Addison Wesley. España 2008. 712 págs ISBN 9789702602064.

Diseño de la Interfaz gráfica de usuario

Existe una disciplina enfocada a estudiar la interfaz de usuario; la cual es llamada Interacción Persona-Ordenador (IPO) se conoce en la comunidad internacional como *Human-Computer Interaction* (HCI) o *Computer-Human Interaction* (CHI).

¿Por qué es tan importante estudiar la interfaz de usuario?

Porque es una parte muy importante del éxito o fracaso de una aplicación interactiva. Según los estudios realizados por MYERS a través de una encuesta hecha a desarrolladores, alrededor de un 48% del código de la aplicación está dedicado a la interfaz.

La ACM, *Association for Computer Machinery*, es, posiblemente, la organización internacional de investigadores y profesionales interesados en todos los aspectos de la computación más importante del mundo. Esta asociación tiene un grupo especial de trabajo en temas de IPO denominado SIGCHI, *Special Interest Group in Computer Human Interaction*, el cual propuso la siguiente definición de Interacción Persona-Ordenador:

> *Es la disciplina relacionada con el diseño, evaluación y implementación de sistemas informáticos interactivos para el uso de seres humanos, y con el estudio de los fenómenos más importantes con los que está relacionado.*

El tema principal de esta disciplina está en la interacción y más específicamente en la interacción entre unos o más seres humanos y uno o más ordenadores.

Los *objetivos de la IPO* son desarrollar o mejorar la seguridad, utilidad, efectividad, eficiencia y usabilidad de sistemas que incluyan computadoras. Cuando decimos sistemas no nos referimos tan solo al hardware y al software sino también a todo el entorno.

La *interfaz* es una superficie de contacto entre dos entidades. En la interacción persona-ordenador estas entidades son la persona y el ordenador.

NEGROPONTE en su libro *"Being digital"* nos da una definición muy sencilla:

> *La interfaz es el sitio donde los bits y las personas se encuentran.*

CHI dice que es un lenguaje de entrada para el usuario, un lenguaje de salida para el ordenador y un protocolo para la interacción.

Obviamente, aparte de la interacción física entre usuario y ordenador hemos de añadir un nivel cognitivo referido a que es necesario que el lado

humano comprenda el protocolo de interacción y actúe sobre la interfaz e interprete sus reacciones adecuadamente. Podemos decir por tanto que:

> *La interfaz de usuario de un sistema consiste de aquellos aspectos del sistema con los que el usuario entra en contacto, físicamente, perceptivamente o conceptualmente. Los aspectos del sistema que están escondidos para el usuario se denominan la implementación.*

GERRIT VAN DER VEER nos define la interfaz como el conocimiento que los usuarios pueden y deberían tener para poder utilizar satisfactoriamente el sistema.

La interfaz de usuario es el principal punto de contacto entre el usuario y el ordenador; es la parte del sistema que el usuario ve, oye, toca y con la que se comunica. El usuario interacciona con el ordenador para poder realizar una tarea. Dependiendo de la experiencia del usuario con la interfaz, el sistema puede tener éxito o fallar en ayudar al usuario a realizar la tarea. El tipo de problemas que origina una interfaz de usuario pobre incluye la reducción de la productividad, un tiempo de aprendizaje inaceptable y niveles de errores inaceptables que produce frustración y probablemente el desechar el sistema.

Características de la GUI:

Ventanas	Permiten desplegar información de manera simultánea.
Iconos	Representan diferentes tipos de información.
Menús	Ayuda a seleccionar comandos.
Apuntador	Permite seleccionar un menú o indicar elementos de interés.
Gráficos	Muestran al usuario la información de manera atractiva.

Ventajas de la GUI:

Son relativamente fáciles de aprender y utilizar: Los usuarios sin experiencia pueden aprender a utilizar la interfaz después de una sesión breve de capacitación.

Para interactuar con el sistema, los usuarios cuentan con pantallas múltiples (ventanas): Es posible ir de una tarea a otra sin perder de vista la información generada durante la primera tarea.

Es posible interactuar rápidamente y tener acceso inmediato a cualquier punto de la pantalla.

Principios de diseño de interfaz de usuario:

Familiaridad del usuario: Debe utilizar términos y conceptos que más utilizan el sistema.

Consistencia: Las operaciones comparables se activan de la misma forma.

Mínima sorpresa: El comportamiento del sistema no debe provocar sorpresa a los usuarios.

Recuperabilidad: La interfaz debe incluir mecanismos para permitir a los usuarios recuperarse de los errores.

Guía al usuario: Cuando los errores ocurren, la interfaz debe proveer retroalimentación significativa y características de ayuda sensible al contexto.

Diversidad de usuarios: La interfaz debe proveer características de interacción apropiada para los diferentes tipos de usuarios del sistema.

Schneiderman clasificó, en 1998, en 5 estilos primarios las formas de Interacción con el usuario.

1. *Manipulación directa*: El usuario interactúa directamente con los objetos de la pantalla. Ejemplo: borrar un archivo, un usuario lo puede arrastrar de un bote de basura.
2. *Selección de menús*: En la que el usuario selecciona un comando de una lista de posibilidades (un menú).
3. *Llenado de formularios*: En éste el usuario llena campos. Algunos campos tienen menús y/o botones asociados
4. *Lenguaje de comandos*: El usuario emite comandos especiales y parámetros asociados para indicar al sistema que hacer.
5. *Lenguaje natural*: El usuario emite un comando para borrar un archivo.

Vista Física

UML incluye dos tipos de vistas para representar unidades de implementación: la vista de implementación y la vista de despliegue.

Vista de implementación: Muestra el empaquetado físico de las partes reutilizables del sistema en unidad sustituibles, llamada *componentes*. Una vista de implementación muestra la implementación de los elementos del diseño (tales como clases) mediante componentes, así como sus interfaces y dependencias entre componentes. Los componentes son las piezas reutilizables de alto nivel a partir de las cuales se pueden construir los sistemas. [Diagrama de componentes]

Vista de despliegue: Muestra la disposición física de los recursos de ejecución computacional, tales como computadores y sus interconexiones. Se llama nodos. Durante la ejecución, los nodos pueden contener componentes y objetos. [Diagrama de Despliegue]

Diagrama de Componentes

Es un diagrama que muestra las organizaciones y las dependencias entre tipos de componentes, el cual representa las dependencias entre componentes software incluyendo componentes de código fuente, componentes del código binario, y componentes ejecutables (Figura 5.1).

Un módulo software se puede representar como componente, algunos componentes existen en tiempo de compilación, algunos existen en tiempo de enlace y algunos existen en tiempo de ejecución; otros componentes existen varias de estas ocasiones. Un componente de sólo compilación es aquel que es significativo únicamente en tiempo de compilación. Un componente de ejecución en este caso, sería un programa ejecutable.

Un diagrama de componentes tiene solamente una versión con descriptores, no tiene versión con instancias. Para mostrar las instancias de los componentes, se debe usar un diagrama de despliegue.

Un *componente* es una unidad física de implementación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema. Los componentes bien diseñados no dependen directamente de otros componentes sino de las interfaces que ofrecen los componentes. Los componentes soportan más interfaces y requieren ciertas interfaces de otros componentes.

Una *interfaz* es una lista de operaciones que una pieza de software o de hardware ofrece y puede realizar.

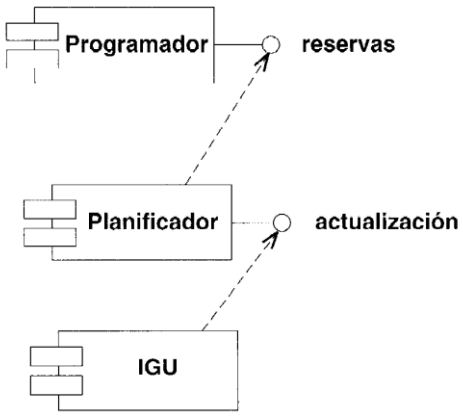

Figura 5.1 Dependencias entre componentes

El uso de las llamadas interfaces permite evitar las dependencias directas entre componentes, facilitando una sustitución más fácil de nuevos componentes.

Como se indica en la Figura 5.2 un componente se representa como un rectángulo con un pequeño icono con dos pestañas en su esquina superior derecha. El nombre del componente aparece en el rectángulo. Un componente puede tener atributos y operaciones, pero éstos a menudo se ocultan en los diagramas.

La relación entre un componente y sus interfaces se puede mostrar de dos formas diferentes. La primera (y más frecuente) consiste en representar la interfaz en su formato icónico abreviado. Una interfaz proporcionada se representa como un semicírculo unido al componente por una línea (un "enchufe"). En ambos casos, el nombre de la interfaz se coloca junto al símbolo. La segunda manera representa la interfaz en su formato expandido, mostrando quizás sus operaciones. El componente que realiza la interfaz se conecta a ella con una relación de realización. El componente que accede a los servicios del otro componente a través de la interfaz se conecta a ella mediante una relación de depenedencia.

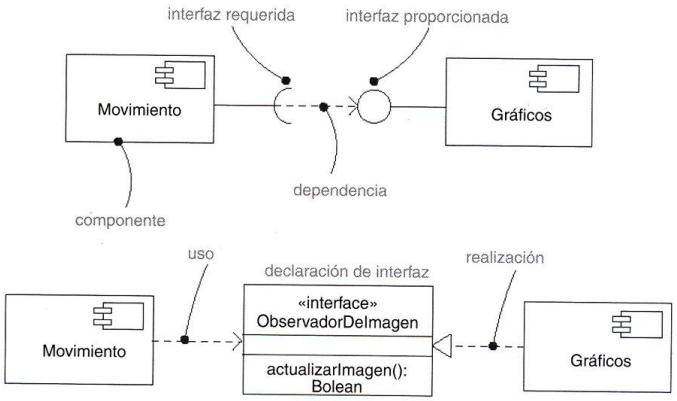

Figura 5.2 Componentes e Interfaces

Para tener más control sobre la implementación pueden utilizarse puertos. Un *puerto* es una ventana explícita dentro de un *componente encapsulado*. En un componente encapsulado, todas las interacciones dentro y fuera del componente pasan a través de los puertos.

Un puerto se representa como un pequeño cuadrado insertado en el borde de un componente (representa un agujero a través de la frontera de encapsulación del componente). Tanto las interfaces requeridas como las proporcionadas pueden enlazarse al puerto. Una interfaz proporcionada represente un servicio que puede ser solicitado a través de ese puerto, mientras que una interfaz requerida representa un servicio que el puerto necesita obtener de algún otro componente.

Cada puerto tiene un nombre, de forma que pueda ser identificado de manera única, dados el componente y el nombre del puerto; el nombre del puerto puede ser utilizado por algunas partes internas del

componente para identificar el puerto al cual enviar y del cual recibir los mensajes. El nombre del componente junto con el nombre del puerto identifican de manera única un puerto específico en un componente específico para que sea utilizado por otros componentes.

La Figura 5.3 muestra el modelo de un componente Vendedor de entradas con puertos, cada puerto tiene un nombre y opcionalmente un tipo que indica la naturaleza del puerto. El componente tiene puertos para la venta de entradas, carga de espectáculos y cobro por tarjeta de crédito.

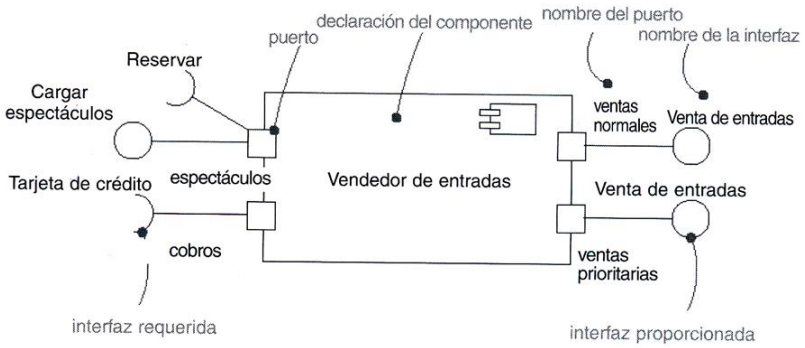

Figura 5.3 Puertos en un componente

Un componente puede implementarse como una pieza de código, pero en los sistemas más grandes lo deseable es poder construir componentes más grandes utilizando componentes menores como bloques de construcción. La estructura interna de un componente está formada por las partes de componen la implementación del componente junto con las conexiones entre ellos.

La Figura 5.4 muestra un componente de un compilador construido a partir de cuatro tipos de partes. Hay un analizador léxico, un analizador sintáctico, un generador de código y de uno a tres optimizadores.

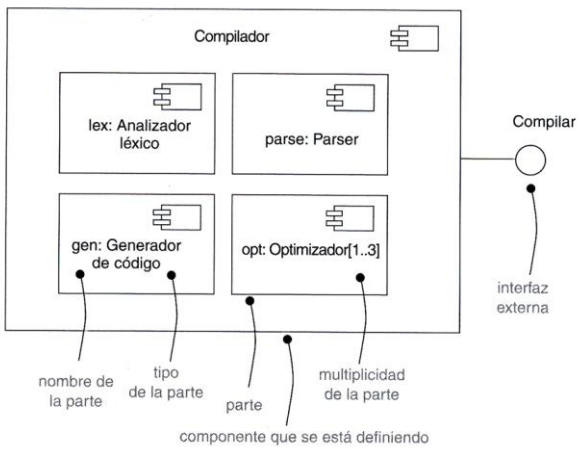

Figura 5.4 Partes dentro de un componente

También se pueden conectar puertos internos a puertos externos del componente global, esto se denomina un conector de delegación, ya que los mensajes sobre el puerto externo son delegados al puerto interno. Esto se representa con una flecha del puerto interno al externo, se pueden ver de dos maneras, según se prefiera; por un lado, el puerto interno es el mismo que el puerto externo, ha sido movido a la frontera y se le ha permitido asomarse al exterior. En segundo enfoque, cualquier mensaje hacia el puerto externo es transmitido inmediatamente hacia el puerto interno y viceversa.

La Figura 5.5 muestra un ejemplo con puertos internos y diferentes tipos de conectores.

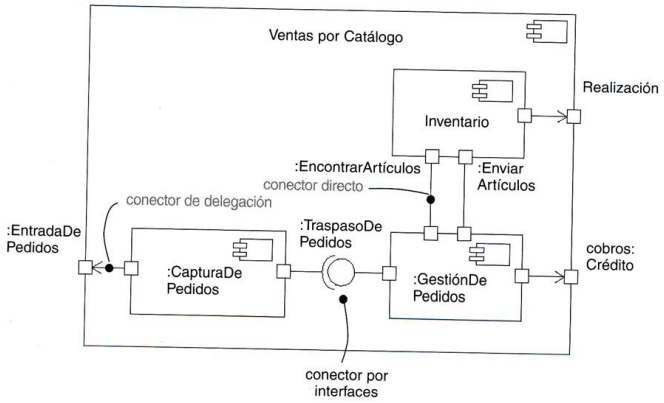

Figura 5.5 Conectores

Las solicitudes externas sobre el puerto *EntradaDePedidos* se delegan hacia el puerto interno del subcomponente *CapturaDePedidos*, este componente a su vez envía su salida al puerto *TraspasoDePedidos* que está conectado con una junta circular al subcomponente *GestiónDePedidos*, éste tipo de conexión implica que no existe un conocimiento especial entre ambos componentes; la salida podría estar conectada a cualquier componente que cumpliese la interfaz *TraspasoDePedidos*. El componente *GestiónDePedidos* se comunica con el componente *Inventario* para encontrar los artículos que quedan en el almacén, esto se representa con un conector directo; como no se muestra ninguna interfaz, se está sugiriendo que la conexión está más fuertemente acoplada. Una vez encontrados los artículos en el almacén, el componente *GestiónDePedidos* accede a un servicio externo *Crédito*; esto se muestra por el conector de delegación hacia el puerto externo denominado cobros, una vez que el servicio externo de *Crédito* responde, el componente *GestiónDePedido* se comunica con un puerto diferente *EnvíoDeArtículos* del componente *Inventario* para preparar el envío del pedido. El componente *Inventario* accede a un servicio externo *Realización* para ejecutar finalmente el envío.

Diagrama de Despliegue

Es aquel diagrama que muestra la configuración de los nodos de proceso y las instancias de componentes y objetos que residen en ellos. Los componentes representan unidades de código en ejecución (porque se han compilado aparte) no aparecen estos diagramas, más bien aparecen en los diagramas de componentes. Un diagrama de despliegue muestra las instancias mientras que un diagrama de componentes muestra la definición de los tipos de componentes por sí mismos.

La vista de despliegue contiene las instancias de los nodos conectados por los enlaces de comunicaciones.

El *diagrama de despliegue* es una red de símbolos de nodos conectados por líneas que muestran las asociaciones de comunicación (ver Figura 5.6). Los símbolos de nodo pueden contener instancias de componentes, indicando que el componente vive o se ejecuta en el nodo. Los símbolos de componente pueden conectar objetos, indicando que el objeto es parte del componente. Los componentes se conectan con otros componentes con flechas discontinuas de dependencia (posiblemente a través de interfaces). Esto indica que un componente utiliza los servicios de otro componente, se puede utilizar un estereotipo para indicar la dependencia exacta, si es necesario.

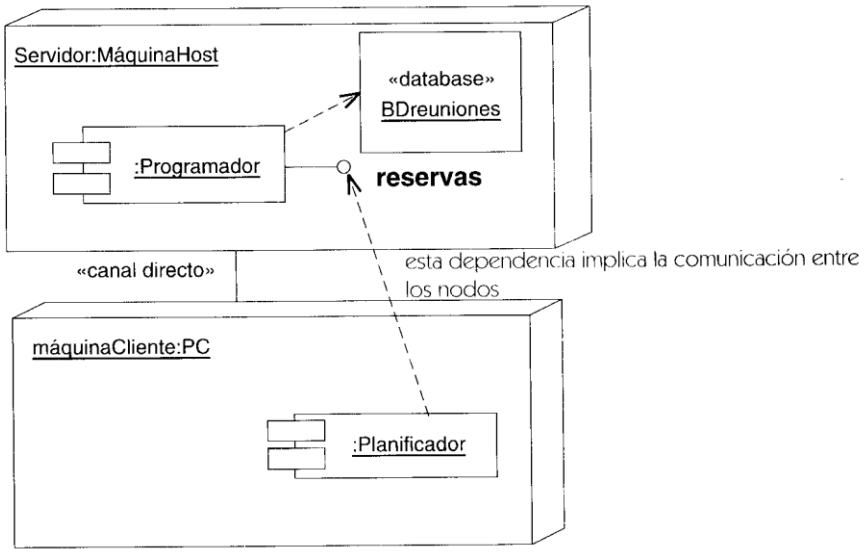

Figura 5.6 Diagrama de Despliegue de un sistema cliente-servidor

UML proporciona una representación gráfica para los nodos, como se muestra en la Figura 5.7, ésta notación canónica permite visualizar un nodo independiente de cualquier hardware específico.

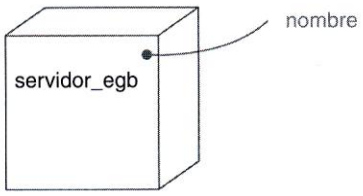

Figura 5.7 Nodos

Un *nodo* es un elemento físico que existe en el tiempo de ejecución y representa un recurso computacional que generalmente, tiene alguna memoria y a menudo, capacidad de procesamiento; gráficamente un nodo se representa como un cubo.

Cada nodo debe tener un nombre que lo distinga del resto de los nodos; un nombre es una cadena de texto y éste solo se denomina *nombre simple*. Un *nombre calificado* consta del nombre del nodo precedido del nombre del paquete en el que se encuentra. Normalmente un nodo se dibuja mostrando solo su nombre, como se ve en la Figura 5.8.

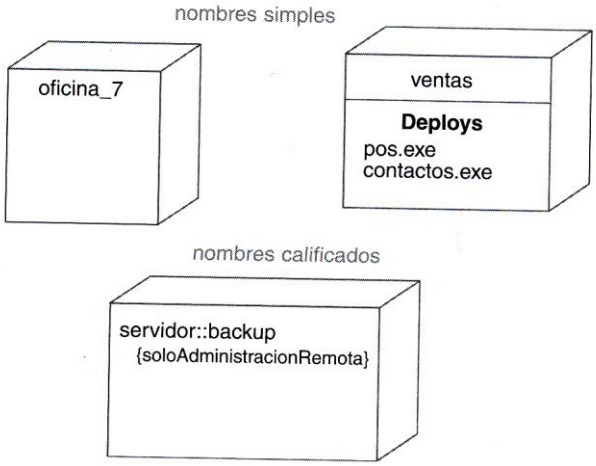

Figura 5.8 Nodos con nombres simples y calificados

En muchos aspectos, los nodos se parecen a los artefactos: ambos tienen nombres, ambos pueden participar en relaciones de dependencia, generalización y asociación; ambos pueden anidarse, ambos pueden tener instancias, ambos pueden participar en interacciones. Sin embargo, hay algunas diferencias significativas entre los nodos y los artefactos:

Los artefactos son los elementos que participan en la ejecución de un sistema, los nodos son los elementos donde se ejecutan los artefactos.

Los artefactos representan el empaquetamiento físico de los elementos lógicos; los nodos representan el despliegue físicos de los artefactos.

La primera diferencia es la más importante. Expresando con sencillez, en los nodos se ejecutan los artefactos; los artefactos son las cosas que se ejecutan en los nodos. La segunda diferencia sugiere una relación entre las clases, artefactos y nodos. En particular, un artefacto es la manifestación de un conjunto de elementos lógicos, tales como las clases y colaboraciones, y un nodo es la localización sobre la que se despliegan los artefactos. Una clase puede manifestarse como uno o más artefactos y a su vez, un artefacto puede desplegarse sobre uno o más nodos. Como se muestra en la Figura 5.9 la relación entre un nodo y el artefacto que despliega puede mostrarse explícitamente a través del anidamiento. La mayoría de las veces no es necesario visualizar esas relaciones gráficamente, pero se indicarán como una parte de la especificación del nodo, por ejemplo usando una tabla.

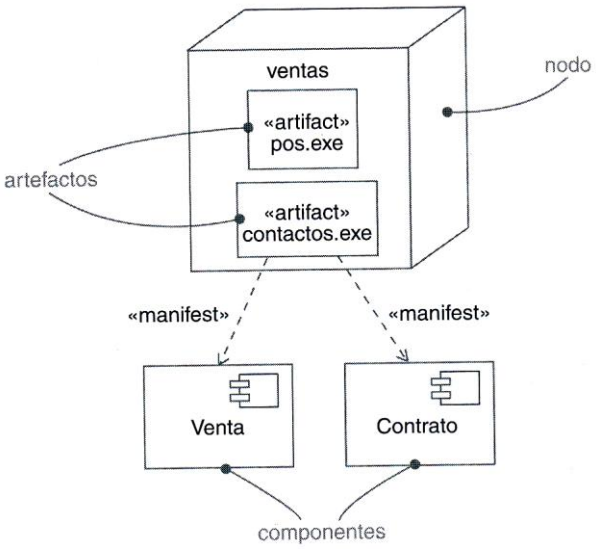

Figura 5.9 Nodos y artefactos

Un conjunto de objetos o artefactos asignados a un nodo como un grupo se denomina *unidad de distribución*.

Los nodos se pueden organizar especificando relaciones de dependencia, generalización y asociación (incluyendo agregación) entre ellos.

El tipo más común de relación entre nodos es la asociación; en este contexto una asociación representa una conexión física entre nodos, como puede ser una conexión Ethernet, una línea en serie o un bus compartido, como se muestra la Figura 5.10, se pueden utilizar asociaciones incluso para modelar conexiones indirectas, tales como un enlace por satélite entre procesadores distintos.

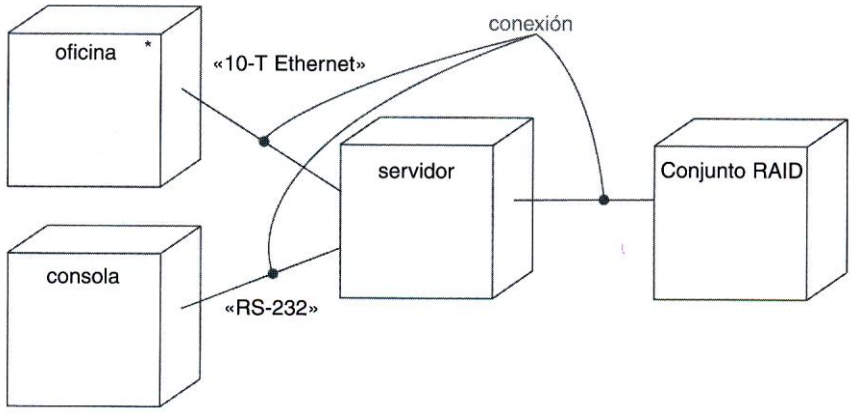

Figura 5.10 Conexiones

Como los nodos son similares a las clases, se dispone de toda la potencia de las asociaciones, esto significa que se pueden incluir roles, multiplicidad y restricciones. Como se muestra en la figura anterior, estas asociaciones se deberán estereotipar si se quieren modelar tipos de conexiones (por ejemplo distinguir entre una conexión Ethernet 10-T y una conexión en serie RS-232).

Con UML, los diagramas de despliegue se utilizan para visualizar los aspectos estáticos de estos nodos físicos y sus relaciones y, para especificar sus detalles para la construcción, como se muestra en la Figura 5.11

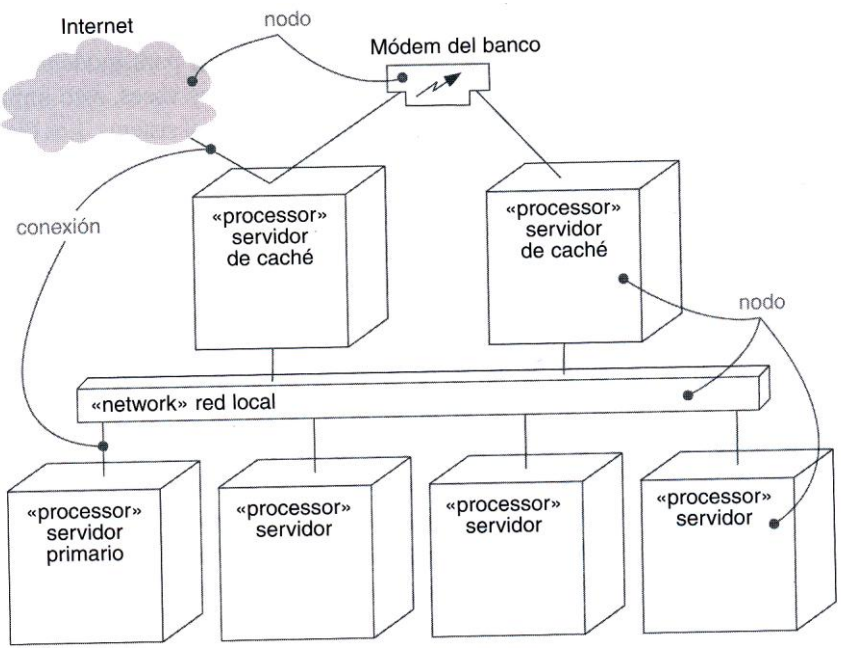

Figura 5.11 Diagrama de Despliegue

Cuando se moldea la vista de despliegue estática de un sistema, normalmente se utilizan los diagramas de despliegue de una de las tres maneras.

1. Para modelar sistemas embebidos

Un sistema embedido es una colección de hardware con gran cantidad de software que interactúa con el mundo físico. Los sistemas embebidos involucran software que controla dispositivos como motores, actuadores y pantallas y que, a su vez, están controlados por estímulos externos tales como entradas de sensores, movimientos y cambios de temperatura. Los diagramas de despliegue se pueden utilizar para modelar los dispositivos y los procesadores que comprenden un sistema embebido (Figura 5.12).

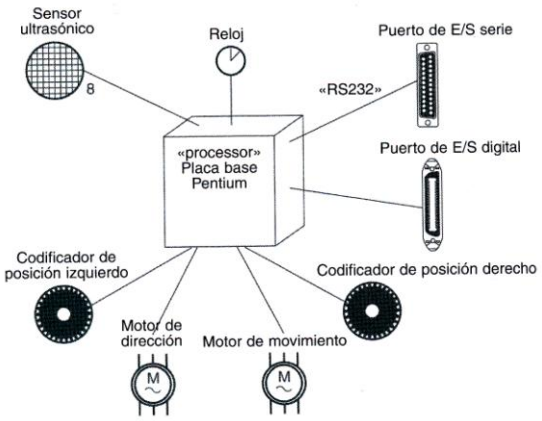

Figura 5.12 Modelado de un sistema embebido

2. Para modelar sistemas cliente/servidor

Un sistema cliente/servidor es una arquitectura muy extendida que se basa en hacer una clara separación de interés entre la interfaz de usuario del sistema (que reside en el cliente) y los datos persistentes del sistema (que residen en el servidor). Los sistemas cliente/servidor son extremo del espectro de los sistemas distribuidos y requieren tomar decisiones sobre la conectividad de red de los clientes a los servidores y sobre la distribución física de los artefactos software a través de los nodos. La topología de tales sistemas se puede modelar mediante diagramas de despliegue (Figura 5.13).

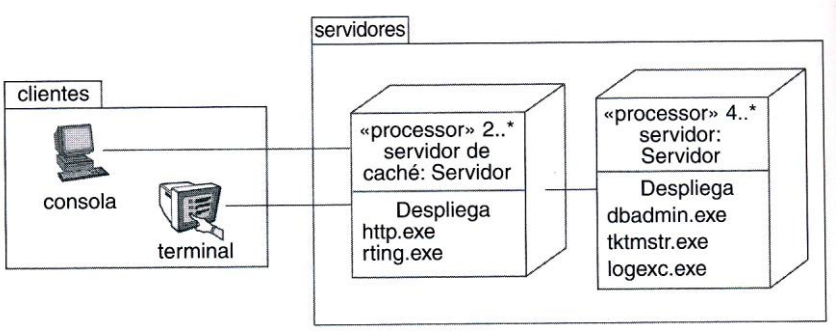

Figura 5.13 Modelado de un sistema cliente/servidor

3. Para modelar sistemas completamente distribuidos

En el otro extremo del espectro de los sistemas distribuidos se encuentran aquellos que son ampliamente, sino totalmente, distribuidos y

que incluyen varios niveles de servidores; tales sistemas contienen a menudo varias versiones de los artefactos software, algunos de los cuales pueden incluso migrar de un nodo de otro. El diseño de tales sistemas requiere tomar decisiones que permitan un cambio continuo de la topología del sistema. Los diagramas de despliegue se pueden utilizar para visualizar la topología actual del sistema y la distribución de artefactos, para razonar sobre el impacto de los cambios en esa topología (Figura 5.14).

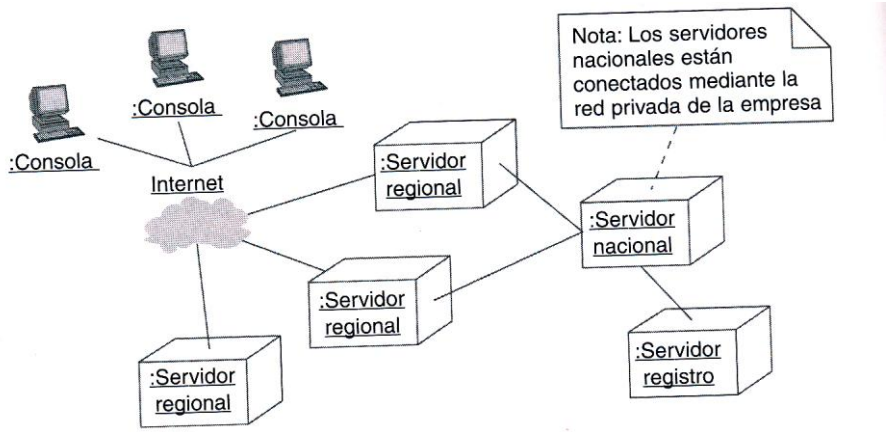

Figura 5.14 Modelado de un sistema completamente distribuido

Diagrama de artefactos

Un *diagrama de artefactos* muestra un conjunto de artefactos y sus relaciones. Gráficamente un diagrama de artefactos es una colección de nodos y arcos.

Con UML los diagramas de artefactos se utilizan para visualizar los aspectos estáticos de estos artefactos físicos y sus relaciones y para especificar sus detalles para la construcción, como se muestra en la Figura 5.15.

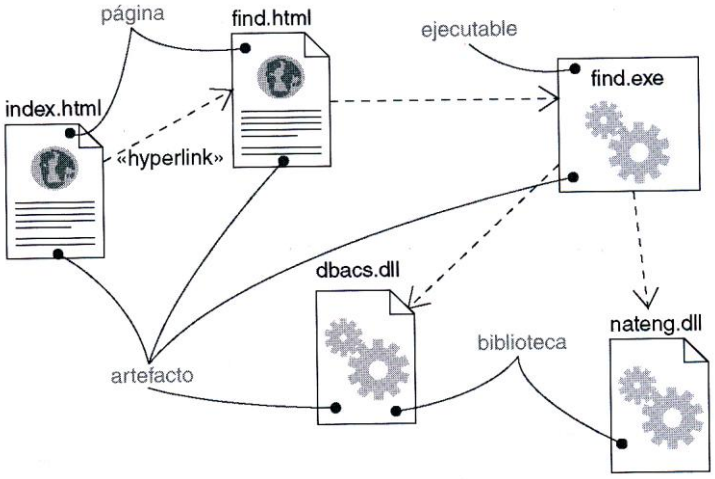

Figura 5.15 Diagrama de artefactos

Cuando se modela la vista de implementación estática de un sistema, normalmente se utilizarán los diagramas de artefactos de una de las cuatro maneras siguientes:

1. Para modelar código fuente.

   Los diagramas de artefactos se pueden utilizar para modelar la configuración de los archivos de código fuente, los cuales representan los artefactos obtenidos como productos del trabajo y para configurar el sistema de gestión de configuraciones (Figura 5.16).

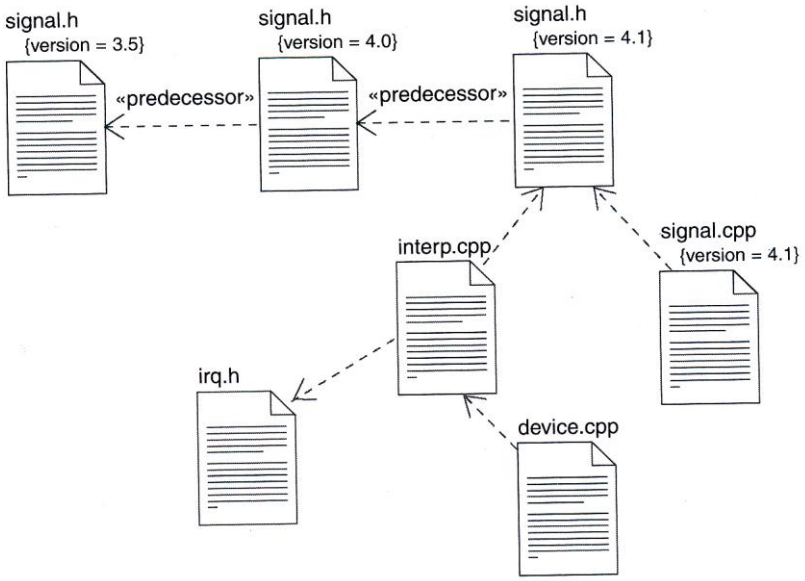

Figura 5.16 Modelado de código fuente

2. Para modelar versiones ejecutables.

   Una versión es un conjunto de artefactos relativamente consistente y completo que se entrega a un usuario interno o externo. En el contexto de los artefactos, una versión se centra en las partes necesarias para entregar un sistema en ejecución; cuando se modela una versión mediante diagramas de artefactos, se están visualizando, especificando y documentando las decisiones acerca de las partes físicas que constituyen el software (es decir, sus artefactos de despliegue, Figura 5.17) .

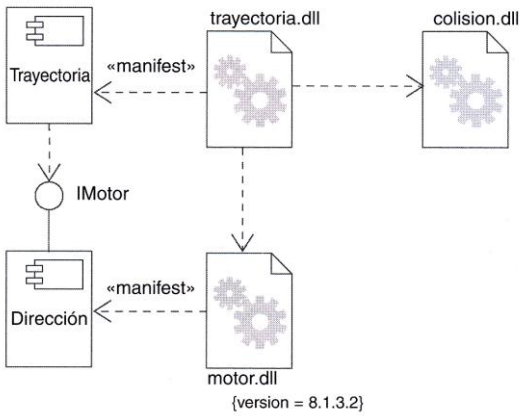

Figura 5.17 Modelado de una versión ejecutable

3. Para modelar bases de datos físicas

   Una base de datos física puede ser vista como la realización concreta de un esquema y que pertenece al mundo de los bits. En efecto los esquemas ofrecen una API para la información persistente; el modelo de una base de datos física representa el almacenamiento de esa información en las tablas de una base de datos relacional o las páginas de una base de datos orientada a objetos. Los diagramas de artefactos se utilizan para representar estos y otros tipos de bases de datos físicas (Figura 5.18).

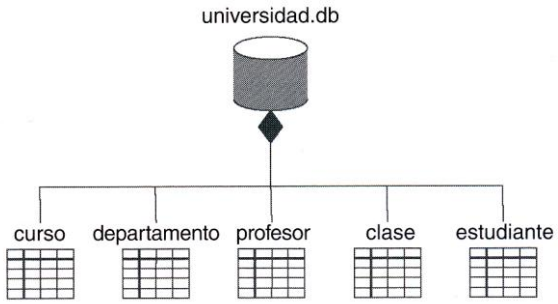

Figura 5.18 Modelado de una base de datos física

4. Para modelar sistemas adaptables.

   Algunos sistemas estáticos; sus artefactos entran en escena, participan en la ejecución y desaparecen. Otros sistemas son más dinámicos e implican agentes móviles o artefactos que migran con el propósito de equilibrar la carga o recuperación de fallos. Los diagramas de artefactos se utilizan junto a algunos de los diagramas de UML para modelar el comportamiento con el fin de representar a estos tipos de sistemas (Figura 5.19).

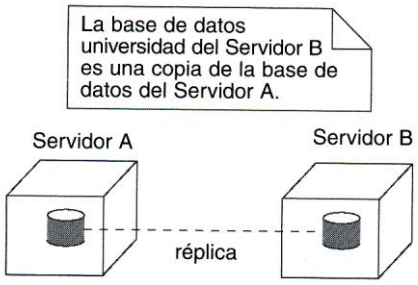

Figura 5.19 Modelado de sistemas adaptables

Diagrama de Paquetes

Un *paquete* es un mecanismo de propósito general para organizar el propio modelo de manera jerárquica. Gráficamente, un paquete se representa como una carpeta de con una pestaña.

Cada paquete ha de tener un nombre que lo distinga de otros paquetes. Un nombre es una cadena de texto. El nombre solo se denomina *nombre simple*; un *nombre calificado* consta del nombre del paquete precedido por el nombre del paquete en el que se encuentra, si es el caso. Para separar los nombres de los paquetes se emplea un signo de puntos duplicado (::). Un paquete se dibuja normalmente mostrando solo su nombre, como se ve en la Figura 5.20 Al igual que las clases, los paquetes se pueden dibujar adornados con valores etiquetados o con apartados adicionales para mostrar sus detalles.

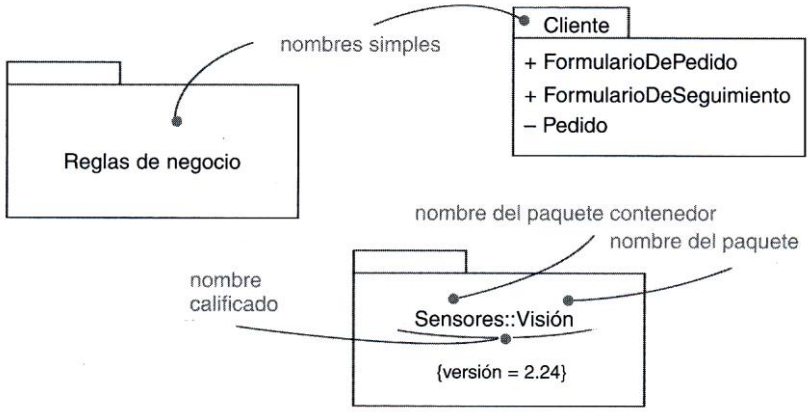

Figura 5.20 Nombres de paquetes simples y nombres calificados

La Figura 5.21 muestra la estructura de paquete de un subsistema para el procesamiento de pedidos. El subsistema en sí se representa mediante un paquete con un estereotipo. Contiene varios paquetes ordinarios. Las dependencias entre paquetes se muestran mediante flechas discontinuas. La figura muestra también algunos paquetes externos de los cuales depende el subsistema, éstos pueden ser componentes estandarizados o bien elementos de una biblioteca.

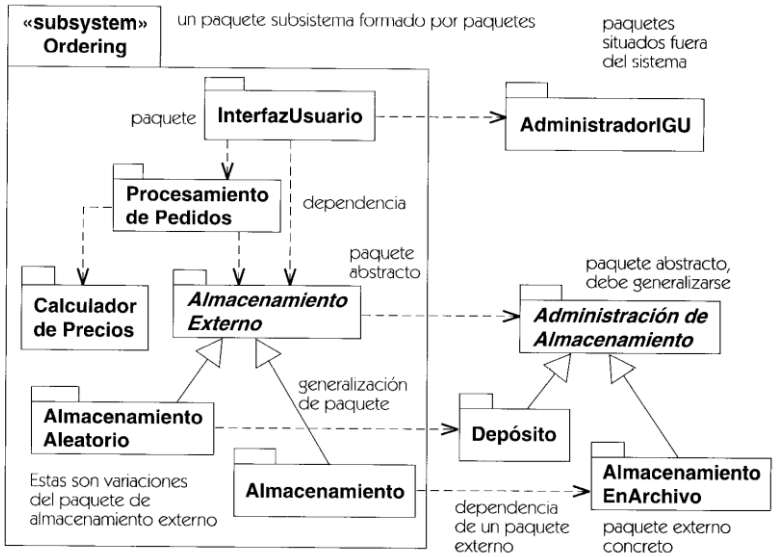

Figura 5.21 Los paquetes y sus relaciones