Polibits

ISSN: 1870-9044

polibits@nlp.cic.ipn.mx

Instituto Politécnico Nacional

México

Acosta Gonzaga, Elizabeth; Álvarez Cedillo, Jesús Antonio; Gordillo Mejía, Abraham

Arquitecturas en n-Capas: Un Sistema Adaptivo

Polibits, núm. 34, 2006, pp. 34-37

Instituto Politécnico Nacional

Distrito Federal, México

Arquitecturas en n-Capas: Un Sistema Adaptivo

M. en C. Elizabeth Acosta Gonzaga
M. en C. Jesús Antonio Álvarez Cedillo
Profesores del CIDETEC-IPN
M. en C. Abraham Gordillo Mejía
Profesor de la UPIICSA-IPN

Los primeros sistemas de cómputo, tal y como los conocimos, se encontraban aislados unos de otros y contaban con sus propios dispositivos propios de E/S, y los programas tenían acceso a la computadora únicamente a través de dichos dispositivos; con la llegada y auge de las redes de comunicaciones el panorama ha cambiado, y ahora es necesario escribir programas que interactúan y/o dependen de otros programas en computadoras lejanas, en un esquena que se conoce como programación distribuida.

Una aplicación distribuida es un sistema compuesto de un conjunto de programas corriendo en múltiples computadoras (hosts); su arquitectura es un esqueleto de programas diferentes, con una descripción de qué programas están corriendo en qué hosts, cuales son sus responsabilidades y qué protocolos determinan la forma en la cual diferentes partes del sistema se comuinican unas con otras.

El concepto de capas (tiers), proporciona una manera conveniente para agrupar diferentes clases de arquitectura. Así, si una aplicación se ejecuta en una computadora, ésta tiene una arquitectura de una capa (one-tiers), si la aplicación se esta ejecutando dos computadoras, por ejemplo, una aplicación web CGI que corre en un navegador (cliente), y en un servidor web, se dice que es una aplicación de dos capas, ya que se tiene un programa cliente y un programa servidor; la diferencia principal entre éstos es que el servidor da respuesta a solicitudes de varios clientes, mientras que un cliente inicia una solicitud de información a un servidor.

Una aplicación de tres capas suma un tercer programa, comúnmente una base de datos, en la cual el servidor almacena información. Este tipo de aplicaciones es una mejora incremental a la arquitectura de dos capas; así, si una solicitud llega de un cliente a un servidor, éste solicita o almacena información en la base de datos, la base regresa información al servidor, y el servidor regresa la información al cliente.

Una arquitectura de múltiples capas (n-tiers), permite que un número ilimitado de programas corra simultáneamente, enviando información de uno a otro, utilizando diferentes protocolos para comunicarse e interactuar simultáneamente; esto permite crear aplicaciones más poderosas que proporcionen diversos servicios a varios clientes.

Sin embargo, esto también representa desventajas; por ejemplo, existe la posibilidad de que se introduzca código maligno, tal como los gusanos (worms), ocasionando problemas en el diseño, implementación y rendimiento. Se han desarrollado nuevas tecnologías que ayudan a combatir esta anomalía, tales como CORBA, EJB, DCOM y RMI; sin embargo, para saltar de tres a n-capas, así como para saltar de una a dos, o de dos a tres, deben tomarse en cuenta diversas consideraciones de seguridad.

Arquitecturas de una capa

Una arquitectura de una capa es un programa simple que no necesita acceso a la red mientras se ejecuta; entre las aplicaciones más comunes de este tipo están los procesadores de texto y los compiladores. Como ya se ha mencionado, un navegador y un servidor web forman parte de un arquitectura de dos capas, pero ¿qué pasa si el navegador web baja un applet java y éste se ejecuta en la computadora?. El applet no accesa a la red mientras se ejecuta, entonces, desde que llega al cliente se convierte en una aplicación de una capa; en estos términos, un programa escrito con JavaScript o VBScript incrustado en código HTML, también puede decirse que es aplicación de una capa.

Las aplicaciones de una capa tienen una gran ventaja: la simplicidad, pues no necesitan del manejo de protocolos, y no se necesita garantizar la sincronización entre datos lejanos, ni el manejo de rutinas de excepción para soportar las fallas inherentes de la red o de un servidor manejando

diferentes versiones de un protocolo o programa. Estos programas representan una gran ventaja en cuanto al rendimiento, ya que los datos no necesitan atravesar la red, esperar su turno en el servidor y entonces regresar, por lo que no sobrecargan la red ni al servidor con tráfico extra.

Arquitecturas de dos capas

Una arquitectura de dos capas en realidad tiene tres partes: un cliente, un servidor y un protocolo, mismo que hace un puente entre las capas del cliente y del servidor. El diseño de dos capas es muy efectivo para aplicaciones de red, así como para programas con interfaz gráfica de usuario (GUI); comúnmente los programas GUI se almacenan del lado del cliente, y la lógica del negocio del lado del servidor, esto permite regenerar y validar datos del usuario en el cliente, donde se tiene una respuesta más rápida; en el proceso, se conservan los tan apreciados recursos de la red y del servidor. De igual forma, la lógica del negocio vive en el servidor, donde está segura, además de hacer uso los recursos de dicho servidor.

Una aplicación de dos capas es un programa cliente/servidor con una GUI como presentación, escrita en un lenguaje de alto nivel tal como Java, C++, etc. En un programa de dos capas se ve la división entre las capas frontales (front) y las capas posteriores (end). En la primera capa, el cliente no necesita preocuparse acerca del almacenamiento de los datos o acerca del procesamiento de múltiples solicitudes; la segunda capa, el servidor, no necesita preocuparse acerca de cómo alimentar datos, o sobre cuestiones relacionadas con la Interfaz Gráfica (UI). Por ejemplo, una aplicación de conversación en línea (Chat) está compuesta por un cliente que acepta entradas del usuario y despliega mensajes, y un servidor que entrega mensajes de un cliente a otro.

Otro ejemplo sería un programa CGI que calcula una hipoteca (puede ser un servlet Java o script Perl). Los datos de entrada se proporcionan a través del protocolo HTTP y específicamente en un formulario HTML, que el usuario llena; la salida es uno o más archivos HTML, pero en realidad, todos los cálculos ocurren en el servidor, y aunque se incorpora un navegador web para presentar datos de salida y aceptar datos de entrada del usuario, toda la ejecución real del programa ocurre en el servidor.

Así, si se escribe un programa para el servidor y no el código que debe ejecutarse en el cliente, entonces, ¿puede decirse que es una aplicación de dos capas?. Esto es altamente discutible, ya que puede decirse que el código HTML es realmente una forma primitiva de código de un programa, y si además se añade un Javascript u otro código del lado del cliente, entonces en verdad se tiene una aplicación de dos capas. Una arquitectura de una capa combina todas las funciones en un solo proceso, mientras que una arquitectura de dos niveles debe separar diversas funciones. En realidad, elegir entre una o dos capas depende de las necesidades de la aplicación.

Arquitecturas de tres capas

Muy probablemente, una aplicación de dos niveles necesitará almacenar datos en un servidor. Tradicionalmente la información se almacena en un sistema de archivos (FileSystem); sin embargo, el riesgo a la integridad de datos se presenta cuando múltiples clientes solicitan simultáneamente al servidor realizar tareas. El sistema de archivos en el mejor de los casos, maneja controles de concurrencia rudimentarios, por lo cual la solución más común es añadir un tercer programa, una base de datos.

Una base de datos especializada almacena, recupera e indexa datos, tal como una arquitectura de dos capas, separa la capa de lógica del negocio de la capa GUI; una arquitectura de tres capas permite separar la lógica del negocio y el acceso de datos, como se muestra en la **Figura 1**.

Con una base de datos es posible proporcionar índices a los mismos, incluir métodos altamente optimizados para su recuperación, y proporcionar replica, respaldos, redundancia, y procedimientos de carga específicos que balancean según las necesidades de la información.

En la actualidad existen sistemas SQL RDBMS, tales como Oracle, Sybase, etc, y algunos tipos tales como OODB (bases de datos orientadas a objetos), ORDB (bases de datos relacionales), etc.

El procedimiento general para usar una base de datos implica, en primer lugar, realizar un esquema que describa los datos para posteriormente escribir consultas que almacenen y recuperen

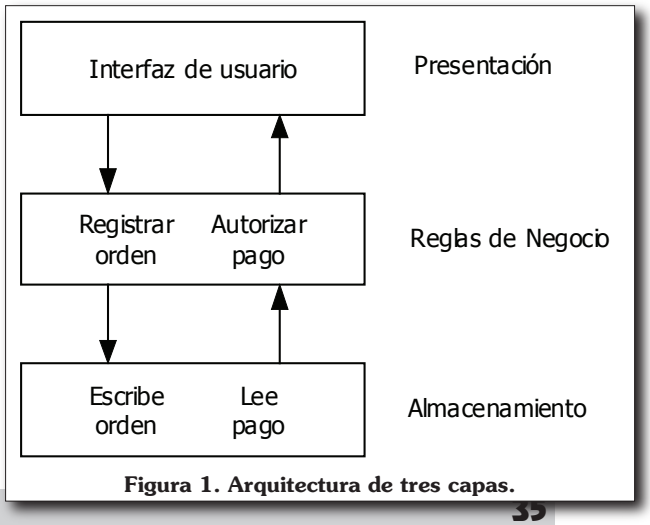

Figura 1. Arquitectura de tres capas.

dichos datos. Existen muchos casos que usan bases de datos; sin embargo, también existen ocasiones en que se puede almacenar una cantidad pequeña de datos en un archivo local simple, en vez de una base de datos relacional. La regla común es: si solo se necesita almacenar datos, y recuperarlos por nombre, un sistema de archivos es suficiente. Pero si se requieren búsquedas sobre la información, entonces es necesaria una base de datos, especialmente si las búsquedas incluyen varios criterios.

Uno de los beneficios de las bases de datos, además del acceso rápido y la fiabilidad, es que múltiples aplicaciones o servicios pueden tener acceso a los mismos datos; éste beneficio está implícito en las aplicaciones de tres o de n-capas. Los procedimientos almacenados (*stored procedure*), son pequeños programas que corren en una base de datos, y puesto que están cerca de los datos, pueden manipularlos; por ejemplo, clasificando, filtrando, transformando, etc. Sin embargo, realizar esto en el lado del servidor es muy costoso.

Existen operaciones que requieren de procedimientos o disparadores almacenados (*triggers*), para funcionar; sin embargo, es fácil utilizarlos de forma incorrecta, es decir si se colocan dentro de la lógica del negocio constituyen un desorden a la estructura de tres capas, y así, en lugar de tener una GUI, lógica y almacenamiento separados, se tiene la lógica del negocio mezclada con el almacenamiento. Además, si el procedimiento se escribe en otro lenguaje diferente al del código en uso, hace que el código del procedimiento sea mucho más difícil de mantener y de eliminar errores.

Por ejemplo, un sitio Web de comercio electrónico incluye un carrito de compras, y al término de las mismas se necesita calcular el costo del envío; entonces, se escribe un procedimiento almacenado que calcule el costo automáticamente por alguna compañía de entrega de paquetería, tal como FedEx, cada vez que se salva un registro de compras (procedimiento conocido como *trigger*); pero si después de seis meses se decide enviar con otra compañía que proporciona un costo más barato dependiendo de la zona del destino, entonces tendrá que escribirse nuevo código, seguramente en SQL. Si se desordenan los datos se puede interferir con la integridad de los mismos, pero si el código original se hubiera escrito directamente en Java, en una capa intermedia, podría extenderse la clase DeliveryMethod.

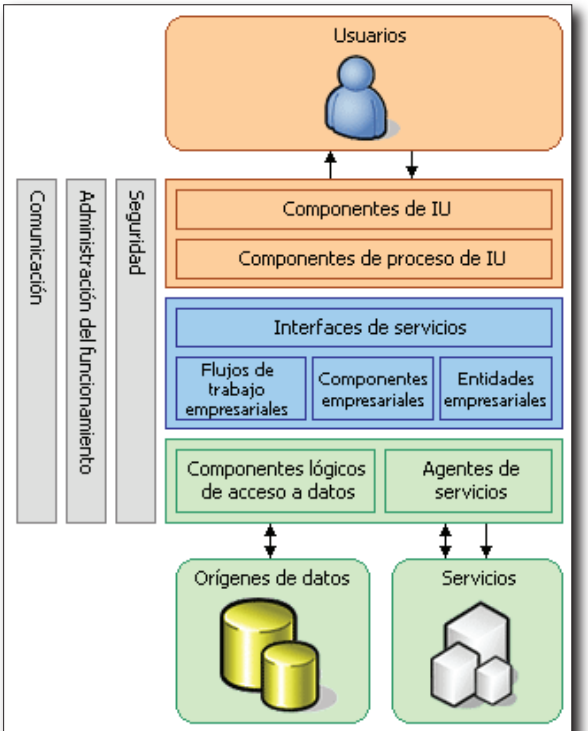

Figura 2. Diseño general de una aplicación con una arquitectura de n-capas.

Arquitecturas de n capas

Cuando se escribe una aplicación, normalmente se diseñan y escogen los objetos comunicándolos a través de mensajes en código de alto nivel. Los protocolos que utilizan los objetos distribuidos manejan lo rudo, los detalles de bajo nivel, tales como los parámetros de ordenamiento, localización de objetos remotos, manejo de transacciones, etc.

Un ejemplo de una aplicación de n-capas es un sistema para almacenamiento de mercancías; en este ambiente múltiples datos alimentan órdenes de compra que llegan de diferentes fuentes, de múltiples bases de datos, y éstas a su vez son consultadas por múltiples clientes que ejecutan aplicaciones especializadas; tiene sentido unir éstos bloques diferentes con el hilo de una arquitectura común de objetos distribuidos, tales como CORBA o EJB, (ver **Figura 2**).

En una arquitectura de n-capas se requiere diseñar objetos realmente reutilizables, que puedan usarse para proyectos futuros. Si los requisitos para un proyecto cambian es necesario reescribir el código; aún más importante es el hecho que, dejando la seguridad que proporciona una arquitectura por capas, se corre el riesgo de diseñar un sistema que sea más complejo que el pensado originalmente. Esto evita el avance, puesto que una decisión descuidada en el diseño puede tener aspectos no considerados. Sin embargo, la transmisión de aplicaciones de objetos distribuidos en CORBA no es tan rápida como aquellas diseñadas con protocolos de socket estándar; CORBA es lento por que necesita ser general, y su gran fortaleza es su mayor debilidad.

Por lo tanto, si lo que se necesita es rapidez el mejor método es hacerlo uno mismo (DIY, *Do It Yourself*). Esto realmente no representa un inconveniente, ya que es aceptable

si el sistema se puede extender añadiendo nuevos módulos, lo que se conoce comúnmente como arquitectura escalable; ésta permite que sus programas se puedan extender sobre múltiples máquinas. Cuando un sistema tiene muchos objetos que interactúan recíprocamente, el número de combinaciones aumenta exponencialmente con el número de objetos, y entonces es recomendable escoger un estándar existente.

En un sistema de tres capas existen millones de accesos a un solo servidor, lo que produce una saturación al mismo; se puede resolver el problema agregando más servidores, a lo que se llama balanceo de carga, (miles de accesos entre varios servidores equivalentes). Cuando se tiene una sola base de datos para todos los servidores, esto no es representa problema si, por ejemplo, un servidor escribe datos y otro servidor necesita leerlos inmediatamente después. Sin embargo, en un arquitectura de n-capas, hay docenas o centenares de objetos que corren simultáneamente, ejecutándose en muchos anfitriones; si el sistema es lento, no es inmediatamente claro que objeto o cual host requiere balancear la carga. Se neesita el análisis a detalle del tráfico de la red y de los archivos de bitácora, para aislar los cuellos de botella; es decir, aumentando el grado de detalle en el diseño del objeto también se limita el rendimiento del sistema.

En un sistema de tres capas prebalanceado, se conoce que hay servidores usando de forma óptima su CPU; si una solicitud llega al sistema, éste se encarga de realizarla. Si es necesario esperar a que la base de datos responda a una consulta (*query*), entonces se generan múltiples hilos o tareas que trabajan sobre otra solicitud; sin embargo, si la cadena de la comunicación tiene que pasar por diferentes máquinas en la red, entonces el servidor original puede estar inactivo, esperando el regreso de una serie de mensajes que están formados en algún objeto remoto sobrecargado en alguna parte de la red. Los CPU de la línea frontal están subutilizados, mientras que el CPU posterior esta sobrecargado, y no existe una forma simple de transferir los ciclos inactivos de una máquina a otra; entonces se debe rediseñar a nivel de objetos para eliminar las ineficiencias.

El surgimiento de la tecnología de componentes distribuidos es la clave de las arquitecturas de n-capas. Estos sistemas de computación utilizan un número variable de componentes individuales que se comunican entre ellos utilizando estándares predefinidos y marcos de comunicación (*frameworks*) como:

CORBA - (*Common Object Request Broker Architecture*) de Object Management Group (OMG).
DNA- (*Distributed interNet Architecture*) de Microsoft (incluye COM/DCOM y COM+ además de MTS, MSMQ, etc.) .
EJB- (*Enterprise Java Beans*) de Sun Microsystems.
XML- (*eXtensible Markup Language*) de World Wide Web Consortium (W3C).

Conclusiónes

La esencia de los modelos de desarrollo por capas es el concepto de *separación*, es decir, mantener cada componente tan separado del contexto global como sea posible. Así, cada capa simplemente es la agrupación de todos los componentes que tienen una funcionalidad común.

Una arquitectura de n-capas forma parte de un proceso revolucionario, actualmente en desarrollo, basado en la aplicación de estas nuevas tecnologías (componentes y estándares de Internet). Estas tecnologías son los bloques para crear Software de Negocio y Sistemas de Información adaptables que ayuden a las empresas a integrar todos sus sistemas de Tecnologías de la Información, mientras que obtienen una ventaja clara en el uso de Internet.

Los componentes distribuidos de una arquitectura de n-capas son una tecnología esencial para crear la siguiente generación de aplicaciones e-business, aplicaciones que son altamente escalables, confiables y que proporcionan un alto rendimiento y una integración sin fallas con los sistemas heredados.

Bibliografía

[1] Jacobson, Ivar, Grady Booch, and James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.* México: Addison-Wesley, 1999.

[2] Kruchten, Philippe. «*Architectural Blueprints—The 4+1 View Model of Software Architecture*». IEEE Software, IEEE. November 1995, pp. 42-50.

[3] Larman, Craig. *UML y Patrones, Introducción al análisis y diseño orientado a objetos.* México: Prentice Hall, 1999.

[4] Wilson, Scott F. *Analyzing Requirements and Defining Solution Architectures*. Redmond: Microsoft Press, 1999.

[5] Fernández Aramayo, David Ricardo. *Arquitectura de Software.* Universidad Tecmilenio, ITESM.