Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Gestión de Datos

Arquitectura de Aplicaciones

Matías N. Leone
Octubre 2008

Índice

Introducción

A lo largo de la cursada se estudian las diversas características de las Bases de Datos.

Pero para entender su aceptación actual en el mercado, es necesario estudiar las características de las aplicaciones que se benefician de su uso.

A lo largo de este apunte estudiaremos las distintas arquitecturas existentes para el desarrollo de aplicaciones, junto las ventajas y desventajas de cada una.

Todos los puntos desarrollados son brindados desde la perspectiva de un desarrollador, arquitecto o diseñador, dejando de lados los aspectos del análisis de la aplicación y el managment del proyecto.

Sistemas de Gestión de Datos

Existen muchos tipos de aplicaciones de software que se desarrollan actualmente en el mercado. En este apunte nos centraremos en las aplicaciones denominadas "Sistemas de Gestión de Datos".

Estas aplicaciones se caracterizan por:

- Administran un gran volumen de datos: del orden de los millones de registros.
- El dominio de la aplicación se centra en el manejo de sus datos.
- Poseen un gran dinamismo en sus datos: muchas altas, bajas y modificaciones a diario.
- Reciben gran cantidad de consultas variadas sobre sus datos.
- Las operaciones a realizar sobre ellos suelen ser simples, en lo que a su aspecto algorítmico se refiere.
- Operaciones comunes:
    - Altas (INSERT)
    - Bajas (DELETE)
    - Modificación (UPDATE)
    - Consultas (SELECT)

Al mencionar que estos sistemas realizan operaciones simples nos referimos a que la mayoría de sus funcionalidades no involucran cálculos físico-matemáticos complejos (ejemplos: inversión de matrices, cálculo de flujo sobre superficies, reconocimiento de patrones por redes neuronales, etc.). Esto no desmerece para nada su desarrollo ni los hace más simple a la hora de implementarlos. Simplemente obligan al diseñador de la aplicación a centrar todos sus esfuerzos en el manejo del gran volumen datos del sistema.

Los Sistemas de Gestión de Datos pueden ser utilizados básicamente por dos tipos de usuarios:

- Usuarios intensivos: son usuarios que utilizan el sistema en forma constante, todos los días, varias veces al día. La productividad y efectividad de su trabajo se encuentra estrechamente relacionada con la interacción del sistema (ejemplo: cajera de supermercado)

- Usuarios casuales: son usuarios que acceden al sistema en forma esporádica, pocas veces al día y que no dependen en forma exclusiva del sistema para seguir operando (ejemplo: persona que compra de libros por Amazon).

Un sistema con tiempos de respuesta grandes puede tornarse terriblemente engorrosa para un usuario intensivo, mientras que un usuario casual podría tolerar esos lapsos sin mayores inconvenientes.

Las decisiones arquitectónicas a tomar a la hora de diseñar la forma de implementar una aplicación deberán tener en cuenta que tipos de usuarios serán los que van a interactuar con el sistema a construir.

Clasificación de arquitecturas

A la hora de diseñar un sistema, desde el punto de vista de su implementación, es necesario tener en cuenta una serie de factores tecnológicos que influyen en gran medida en el éxito o fracaso del proyecto.

A continuación se especifica una clasificación de sistemas, desde el punto de vista de su arquitectura tecnológica.

Cabe aclarar que en la vida real, muchos sistemas quizás no encajen en ninguna categoría, o sea híbridos de varias categorías a la vez.

Por cada categoría mencionada se detallan las ventajas y desventajas existentes, que la hacen mejor o peor para una situación determinada.

Arquitectura Monocapa

Se caracterizan por ser sistemas monousuarios, utilizados por una única persona a la vez, sin que exista concurrencia de acceso a la información

Son comúnmente denominados Desktop, dado que consisten en una aplicación instalada en la PC del usuario, compuesta por un instalador y un ejecutable que permite acceder a la misma.

No posee conectividad. Es un componente completamente independiente que no interactúa con ninguna otra aplicación, salvo con el sistema operativo en el cual fue instalado.

La lógica de negocio está centralizada. Todas las reglas propias del negocio que se implementa se encuentran en un solo lugar, en el código de la aplicación instalada en cada máquina.

En cuanto a la persistencia, los datos de la aplicación se guardan en archivos propios de la aplicación (ejemplo: archivo .doc del Word). Esto permite que la información pueda ser recuperada en un uso posterior de la aplicación. Estos datos se almacenan en cada máquina en la que se encuentre instalado el programa, sin ningún tipo de centralización.

Ventajas

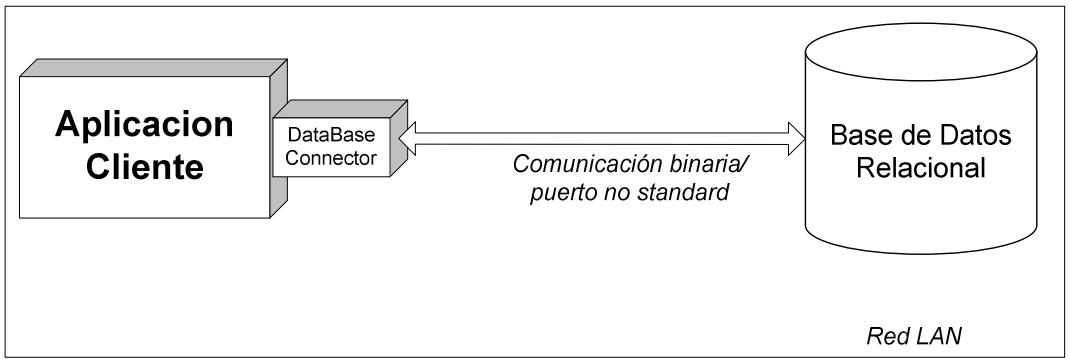
- Es simple de construir. La aplicación esta conformada por un único componente.
- Facilidad en el rastreo de bugs: todos los problemas que existan en la aplicación se encuentran en el único componente que conforma la misma.

- Acceso directo a datos: no se requiere una conexión para acceder a los datos de la aplicación.
- Fácil puesta en marcha: solo es necesario instalar la aplicación en la PC.

Desventajas

- La integración de datos entre aplicaciones de distintas máquinas es complicada. Cada PC que utilice la aplicación almacena los datos de la operatoria realizada en su propio archivo. Para integrar la información de distintas aplicaciones es necesario realizar una consolidación de los archivos persistidos. (ejemplo: juntar dos archivos de Word de dos alumnos que se encuentran realizando un mismo TP para la facultad)
- Para solucionarlo los problemas de integración se puede:
    - Hacer una consolidación de datos a mano.
    - Utilizar procesos batch automáticos, corriendo fuera de hora, que integren todos los datos y verifiquen su integridad.
- Normalmente la aplicación requiere una versión distinta, que se adapte a cada Sistema Operativo, por lo tanto se encuentran acoplados al Sistema Operativo para el cual fueron desarrolladas.

Arquitectura Cliente-Servidor



Están compuestas por dos módulos:

- Aplicación Cliente: componente Desktop, instalado en la PC de cada usuario, de características similares a las mencionadas en la arquitectura anterior.
- Aplicación Servidor: servidor centralizado, con una Base de Datos que hace de repositorio de la aplicación.

La base de datos es un programa corriendo en el servidor, con una IP y puerto de escucha.

La aplicación cliente se conecta a la base de datos directamente, mediante un conector o driver de base de datos. Cada base de datos existente en el mercado provee su driver para los lenguajes de programación mas conocidos. Algunos de los conectores más comunes son:

- JDBC: conector para aplicaciones JAVA
- ADO.NET: conector para aplicaciones .NET
- ODBC: conector a base de datos genérico

La lógica de negocio ahora puede estar distribuida en dos partes:

- En el código de la aplicación Desktop del cliente
- En la base de datos manifestada como:
    - Tablas: las entidades del modelo de datos y sus relaciones con otras tablas conforman gran parte del modelado del negocio.
    - Stored Procedures
    - Views
    - Triggers
    - Constraints

Ventajas

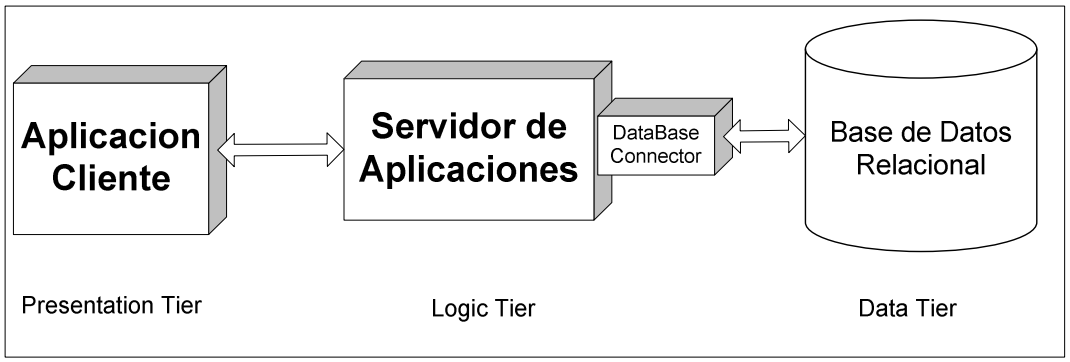
- Al incorporar una base de datos como un módulo de nuestro sistema, ganamos todas las ventajas propias de ella:
    - Concurrencia de consultas
    - Carga de datos simultánea
    - Consulta filtrada y performante sobre gran volumen datos
    - Integridad de los datos almacenados
- Se eliminan los problemas de integración de datos, presentes en la arquitectura anterior. Pueden trabajar varios usuarios a la vez, en forma simultánea, y ambos ver reflejado en forma inmediata, los cambios producidos.
- Se delega gran parte de la complejidad del sistema en la base de datos, la cual es un componente ya programado y probado durante varios años.
- Los usuarios pueden estar en diferentes lugares físicos, y acceder al mismo servidor de base de datos.

Desventajas

- Surgen algunos problemas de acceso o conectividad:
    - La PC del cliente debe tener acceso a la IP del servidor. Esto implica que todas las PC cliente deben encontrarse en la misma red que el servidor. Existen varias formas de lograr solucionar este problema:
        - Todas las PC se encuentran conectadas a la misma red LAN
        - Se accede a través de Internet (Ejemplo: VPN)
    - La comunicación es binaria por un puerto no Standard (distinto del 80, puerto default del protocolo HTTP). Un Proxy suele bloquear este tipo de comunicación por dos motivos:
        - Bloqueo por puertos: las bases de datos no suelen tener sus servicios de escucha en los puertos que normalmente se encuentran abiertos por default.
        - Bloqueo por protocolo: incluso aunque el servicio de la base de datos sea instalado para utilizar el puerto 80, la información que viaja en esta comunicación no pertenece al protocolo HTTP, por lo tanto también puede ser identificada y bloqueada.

- Requiere instalación: la aplicación Cliente, por poseer características similares a las de la arquitectura Monocapa, suele requerir una instalación sobe la PC del usuario. Esto puede generar inconvenientes si el usuario no posee los permisos necesarios (ejemplo: imposibilidad de instalar una aplicación de consulta desde un cyber café, oficina, biblioteca, etc.). Además puede volver engorroso el uso de la aplicación y desalentar a usuarios casuales.
- Al igual que la arquitectura en Monocapa, la aplicación cliente normalmente se encuentra acoplada al sistema operativo.
- Aumenta la complejidad de actualización de la aplicación:
    - Cada vez que surjan cambios o arreglos, será necesario instalar una nueva versión de la aplicación en cada PC. Para superar este problema es se puede optar por alguna de las siguientes alternativas:
        - Crear un módulo de detección de actualizaciones en forma automática (Service Pack, Update Center). Este módulo no debería dejar que el usuario utilice la aplicación por mucho tiempo, con una versión discontinuada. Su implementación puede no ser trivial, dado que hay que tener en cuenta todos los posibles escenarios de actualización que existen.
        - Crear un sitio del cual se pueda descargar la última versión de la aplicación. Corremos el riesgo de que nuestros usuarios prefieran la versión anterior y no instalan la nueva.
        - Enviar personas como instaladores a cada PC para configurar la nueva versión. Esta última opción puede tornarse engorrosa en caso de contar con un gran número de usuarios.
- Aumento de complejidad en rastreo de bugs: cuando surge un error, ahora la causa del mismo puede encontrarse en dos lugares distintos:
    - En la base de datos
    - En la aplicación cliente
- La aplicación se encuentra divida en dos partes, lo cual complica su programación (comparado con la arquitectura anterior)
- La aplicación que el usuario debe instalar en su PC puede terminar ocupando un tamaño significativo, dado que casi todo el código de la aplicación se encuentre en ella. Esto complica aún más la actualización de la aplicación.
- Problemas de seguridad: como el cliente posee en su PC casi todo el código de la aplicación, podría alterarlo para evitar validaciones de dominio (mediante técnicas *cracking* y *decompilación* de código). Además el usuario posee acceso directo a la base de datos. Una mala administración de permisos de usuario podría dejar expuesta toda la información almacenada en la base de datos.

Arquitectura Multicapa



Están compuestas por tres módulos:

- Aplicación cliente. Puede ser de básicamente dos formas:
  - Desktop: un cliente instalable en la PC, de características similares a las de las arquitecturas anteriores.
  - Web: un cliente accedido a través de un Browser de Internet
  - Servidor de aplicaciones (Application Server): aplicación con código que se ejecuta en el servidor.
  - Base de datos

En este modelo de arquitectura es el servidor de aplicaciones, y no la aplicación cliente, la que se conecta a la base de datos. La conexión se realiza a través de un driver de base de datos, de la misma forma en que se realizaba en las arquitecturas anteriores.

La aplicación cliente, en cambio, no accede directamente a los datos, sino que lo hace a través de la aplicación servidor.

Para mencionar las características de esta arquitectura, dividiremos su estudio en dos grandes categorías: aplicaciones con cliente web y aplicaciones con cliente desktop.

Arquitectura Multicapa con cliente Desktop

Este tipo de arquitectura intenta atacar los principales problemas presentados en la arquitectura Cliente-Servidor.

El Cliente sigue siendo una aplicación instalable, pero ahora no accede directamente a los datos, sino que lo hace a través de la aplicación servidor.

El cliente se considera liviano (thin-client) porque el usuario no posee la aplicación entera en su poder, sino solo una cáscara del sistema. Muchas reglas del negocio pueden ser ahora implementadas en la aplicación servidor.

Uno de los temas centrales que surgen en esta arquitectura, es el método de comunicación que se va a utilizar entre la aplicación cliente y la aplicación servidor.

La aplicación cliente ahora no puede aprovechar los beneficios de un driver de base de datos, sino que debe utilizar algún método propio de conexión.

Existen diversas tecnologías para solucionar este problema, dependiendo del lenguaje utilizado:

- Comunicación tradicional a bajo nivel, mediante la utilización de Sockets
- Comunicación mediante RPC (Remote Procedure Call) o RMI (Remote Method Invocation): ambas alternativas presentes en la mayoría de los lenguajes de

programación actuales, proveen mecanismos automáticos para la ejecución de código remoto, junto con la serialización de los parámetros de entrada y salida.
- Mediante el uso de WebServices. La aplicación cliente puede utilizar el protocolo HTTP para intercambiar mensajes con la aplicación servidor, mediante archivos en formato XML.

Ventajas
- La aplicación cliente puede poseer una interfaz gráfica más robusta, interactiva y performante que las que se pueden lograr actualmente con tecnologías Web.
- Como la aplicación servidor es la encargada del acceso a la Base de Datos, la lógica de la pantalla no se encuentra tan acoplada a la fuente de origen de datos.
- La aplicación cliente no posee acceso directo a la base de datos y las validaciones de negocio mas importantes pueden ser programadas en la aplicación servidor, eliminando de esta forma algunos problemas de seguridad presentes en la arquitectura anterior.
- El programa instalado en la PC del usuario no necesita contar con toda la lógica del negocio, por lo tanto suele ser bastante más liviano que en una arquitectura Cliente-Servidor. Gracias a esto, la actualización de versiones de la aplicación es más fácil.

Desventajas:
- La aplicación cliente es más liviana y fácil de actualizar que en la arquitectura Cliente-Servidor, pero sigue siendo necesario desarrollar un módulo de actualización para la instalación de futuras versiones.
- La lógica de la aplicación ahora se encuentra dividida en tres partes. Esto complica la detección y arreglo de fallas.
- Es necesario determinar que librerías de código serán utilizadas desde el cliente y cuáles desde el servidor. Su desarrollo se vuelve más complicado que en una arquitectura en dos capas.
- La interacción de la capa Servidor en los pedidos a la base de datos puede representar un cuello de botella en la performance de la aplicación.
- Si no se utiliza WebServices para la comunicación entre la aplicación cliente y la aplicación servidor, esta arquitectura presenta los mismos problemas de conectividad que la arquitectura Cliente-Servidor.

Arquitectura Multicapa con cliente Web

Las aplicaciones Web intentan eliminar muchos de los problemas de conectividad presentes en las arquitecturas anteriores. Para ello centran sus energías en el desarrollo de aplicaciones que se ejecuten dentro de un Browser de Internet.

Por aplicaciones Web no nos referimos a simples páginas HTML o portales de contenido, sino a aplicaciones enteras y complejas, con mucha lógica de negocio involucrada.

Esta arquitectura es una de las más recientes y se encuentra en constante evolución.

El desarrollo de aplicaciones Web es un tema amplio y completo que por si solo podría requerir varios libros para ser explicado correctamente. Solo nos limitaremos a nombrar sus características más relevantes para nuestro estudio.

Una aplicación Web se compone de tres partes:

- Aplicación cliente ejecutando en un Browser de Internet. El código de la aplicación podrá estar compuesto por diversos elementos, dependiendo de la plataforma de desarrollo elegida. Constantemente surgen nuevos elementos que intentan mejorar la experiencia de las aplicaciones Web. Algunos de los elementos más comunes son:
    - HTML
    - JavaScript
    - Hojas de estilo (CSS)
    - Otros elementos no Standard (plugins):
        - Java (Applets, JavaWebStart)
        - Flash (ActionScript, Flex)
        - ActiveX (COM)
- Application Server. Al tratarse de aplicaciones Web, el servidor debe poder lidiar con el protocolo HTTP. Al igual que en la aplicación cliente, existen muchos servidores dependiendo de la tecnología elegida. Algunos de los más comunes son:
        - Apache (PHP)
        - Tomcat (Java)
        - Internet Information Server (ASP .NET, ASP)
        - ColdFusion (Flash)
- Base de datos

En este tipo de arquitectura, nuestra aplicación cliente no consiste en un instalable junto a un ejecutable, sino que la misma es descargada y ejecutada dentro de un Browser de Internet.

El Browser de Internet fue originalmente creado con el objetivo de acceder a documentos HTML, ubicados en servidores HTTP. Con el tiempo sus prestaciones fueron evolucionando hasta casi permitir hoy en día ejecutar aplicaciones de una magnitud y complejidad similar a las aplicaciones Desktop.

El Browser es un candidato ideal para solucionar muchos de los problemas de conectividad, acceso y acoplamiento, existentes en las otras arquitecturas nombradas. Sus principales características son:

- Existen versiones de Browser para todos los sistemas operativos más utilizados:
    - Windows: Internet Explorer, Firefox, Opera, Safari, Chrome
    - Linux: Firefox, Opera
    - MacOs: Firefox, Safari, Opera
- El protocolo HTTP utilizado por los Browser se considera normalmente de acceso libre.

En las aplicaciones Web nos vemos obligados a utilizar una capa intermedia en nuestra aplicación, que nos provea del acceso a datos, dado que nuestro cliente no puede utilizar un driver de base de datos para conectarse directamente a ésta (esto se debe al esquema de seguridad de los Browsers, denominado Sandbox).

La aplicación cliente se comunica con la aplicación servidor, mediante pedidos del protocolo HTTP, utilizando el puerto 80.

La aplicación servidor se comunica con la base de datos de la misma forma que en la arquitectura de dos capas, a través de un driver de base de datos.

Ventajas

- Eliminan el problema de acceso presente en todas las demás arquitecturas. El puerto 80 y el protocolo HTTP suelen estar permitidos en casi cualquier ambiente. Cualquier usuario con acceso a Internet puede acceder a la aplicación.
- Eliminan los problemas de instalación de la aplicación. El usuario puede acceder a la aplicación desde cualquier PC que tenga un Browser para navegar por Internet.
- Al no requerir instalación, el usuario siempre accede a la última versión disponible de la aplicación. El Browser descarga la última versión disponible de la misma en el Application Server, de esta forma. se evitan los problemas de actualización.
- Al accederse a través de Internet, se eliminan los problemas de conectividad. Las PC clientes no necesitan ser partes de una misma red LAN. Solo es necesario el acceso a Internet.
- Como la aplicación cliente no se instala ni se ejecuta directamente, sino a través del Browser, se eliminan los problemas de acoplamiento al sistema operativo.
- Al no requerir instalación y poder ser accedidas desde cualquier PC con acceso a Internet, suelen ser ágiles para un usuario casual.
- Al igual que en la arquitectura Multicapa con cliente Desktop, el usuario solo posee una parte de la aplicación. Por lo tanto se eliminan muchos problemas de seguridad.

Desventajas

- Si bien las aplicaciones WEB no requieren instalación, es necesario tener el Browser perfectamente configurado. Los siguientes elementos suelen traer conflictos:
  - Permitir la apertura ventanas emergentes (Popups)
  - Permitir la ejecución de JavasScript
  - Permitir la ejecución de Plugins
  - Contar con los Plugins en la versión correcta
  - Tener habilitado un nivel de seguridad que permite acceder a la aplicación
- La performance del protocolo HTTP deja mucho que desear a la hora transferir grandes volúmenes de datos. El protocolo HTTP es un protocolo orientado a caracteres y no binario, con lo cual se añade un gran overhead al tamaño de los mensajes transmitidos.
- Actualmente la performance de ejecución de código JavaScript en los Browser es inferior a la de otros lenguajes existentes: Java, .NET, PHP.
- La calidad de la interfaz gráfica de usuario que se puede lograr, en especial en lo que se refiere a la interacción con componentes gráficos, suele ser inferior a la que es posible lograr mediante una aplicación Desktop.
- Las aplicaciones Web logran desacoplarse del Sistema Operativo pero son dependientes del Browser de Internet sobre el cual ejecutan. Actualmente existen muchas diferencias en la forma en que los Browsers tratan el contenido de las aplicaciones Web. Algunos de los problemas más frecuentes son:

    - No todos los Browsers renderizan HTML de la misma forma, con lo cual una misma aplicación puede visualizarse de formas distintas en Browsers diferentes.
    - JavaScript es interpretado en forma distinta, y con distintos niveles de performance, en cada Browser.
    - Las hojas de estilo (CSS) poseen propiedades que funcionan en algunos Browsers y otros no, y muchas propiedades que funcionan en todos los Browsers son renderizadas de forma distinta.
- Si bien se evitan los problemas de actualización de la aplicación, el cliente descarga el 100% de la aplicación cada vez que accede a la misma. Esto puede volverse perjudicial en aplicaciones grandes.
- Si el usuario no tiene acceso Internet (problemas con el ISP) no se puede acceder a la aplicación. Una aplicación crítica deberá tener en cuenta esta desventaja a la hora de decidir la arquitectura a utilizar.
- Actualmente el tiempo de desarrollo requerido para una aplicación con arquitectura Web suele ser bastante mayor que el requerido para desarrollar la misma aplicación en arquitectura Desktop.
- La cantidad de lenguajes diferentes que conviven en una misma aplicación suele complicar el mantenimiento de la misma.