El Modelo Cliente/Servidor

Ing. Emiliano Marini
www.linuxito.com
Octubre de 2012

El modelo Cliente/Servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Las aplicaciones Clientes realizan peticiones a una o varias aplicaciones Servidores, que deben encontrarse en ejecución para atender dichas demandas.

El modelo Cliente/Servidor permite diversificar el trabajo que realiza cada aplicación, de forma que los Clientes no se sobrecarguen, cosa que ocurriría si ellos mismos desempeñan las funciones que le son proporcionadas de forma directa y transparente. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. Tanto el Cliente como el Servidor son entidades abstractas que pueden residir en la misma máquina o en máquinas diferentes.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

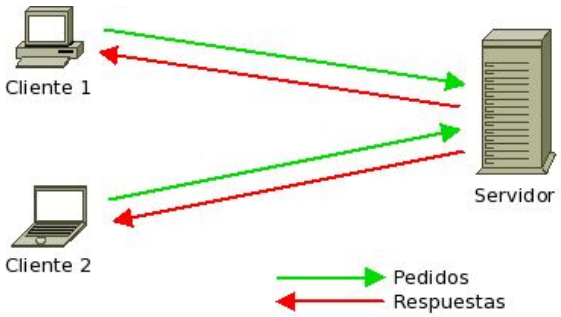

Figura 1. El Modelo Cliente/Servidor.

El Modelo de Servicios

Un modelo es una vista abstracta que establece las definiciones, reglas y relaciones entre las estructuras relacionadas con la aplicación. Sirve de base para el intercambio de ideas durante el desarrollo lógico de la aplicación y determina cómo será la aplicación resultante. Cuando alguien habla de una casa, automáticamente asumimos que ésta tendrá un salón, habitaciones, baños, cocina, etc., sin que se nos diga nada más. Aunque la casa no tuviera salón, el modelo nos serviría como punto de partida para "entender" el concepto "casa" y empezar a discutir

sobre ella. De igual manera, el modelo de una aplicación nos indica lo que hace una aplicación o más exactamente, lo que uno cree que debe hacer la aplicación.
El modelo de servicios establece tres grandes conjuntos de funcionalidades, en cada uno de los cuales se encuadran las distintas tareas en las que se ve involucrado cualquier tipo de proyecto de desarrollo. Dicho modelo establece los siguientes conjuntos que comúnmente denominamos lógicas.

Lógica de Presentación

Esta lógica es la responsable del control de todos los aspectos relacionados con la interacción entre el usuario y la aplicación. Para llevar a cabo esta tarea de control, es necesario conocer qué tipos de usuarios utilizarán la aplicación, qué actividades tienen que realizar y, teniendo en cuenta estos datos, cuáles son los mejores estilos de interfaz para que esos usuarios realicen sus tareas. En esta lógica se engloban todas las tareas que deben ser realizadas por la parte Cliente del modelo general.
Con el fin de independizar (en la medida de lo posible) la interfaz de usuario de las características propias de los procesos, debemos tener presente que la codificación de las tareas asociadas a esta lógica consiste, principalmente, en la llamada a procesos independientes situados en las otras lógicas, cuya ejecución es totalmente transparente.
Si en la capa que implementa la lógica de presentación no incluimos lógica del negocio ni accesos directos a datos, conseguiremos que esta capa sea inmune a los cambios introducidos en los procedimientos de la empresa, así como a los cambios de los sistemas de gestión de datos utilizados.

Lógica de Negocio

Es la lógica de la aplicación que controla la secuencia de acciones y fuerza el cumplimiento de las reglas del negocio propias de cada empresa; además, asegura la integridad de las transacciones de las operaciones necesarias que haya que realizar para que se cumplan dichas reglas. La lógica del negocio también transforma una serie de datos en información útil para el usuario mediante la aplicación de las reglas apropiadas.
El objetivo que debe cumplir esta lógica es el de aislar las reglas del negocio, así como las transformaciones de datos de los consumidores (usuarios y otros componentes de esta misma capa) y de los sistemas de gestión de datos. Este aislamiento tiene las siguientes ventajas:

- Flexibilidad a la hora de decidir cómo y dónde situar el código de esta lógica: en componentes dentro de una aplicación servidora; en procedimientos almacenados, dentro del sistema gestor de datos; o incluso en el cliente.
- La habilidad de colocar distintas interfaces de usuario para un mismo conjunto estándar de reglas de negocio. Por ejemplo, el conjunto de reglas que define las operaciones realizables con los clientes puede implementarse como un solo componente que se ejecuta en un servidor. Los servicios que ofrece este componente pueden utilizarse desde una macro que se ejecute dentro de Microsoft Office, desde una aplicación desarrollada con Visual Basic o desde páginas HTML vistas desde Internet Explorer.

- Facilita el mantenimiento de las reglas del negocio y de su lógica, aislando los cambios de las interfaces de los usuarios y de los datos.
- La habilidad para sustituir el código de estas reglas, de forma que, aunque el conjunto de reglas que se encuentra dentro de un conjunto de servicios del negocio varía de un país a otro, las interfaces de esos servicios pueden permanecer constantes.

Estos procesos, dada su naturaleza, pueden ser cambiantes en cuanto a su construcción, pero no en cuanto a su funcionalidad. Al regirse por directrices empresariales, éstas podrían cambiar atendiendo a razones internas, sin variar necesariamente la funcionalidad que proporcionan.
Tomemos como ejemplo el cálculo de los sueldos de los empleados. Dicho cálculo se basa en algunas normas legales que deben ser respetadas, aunque la compañía también puede introducir cambios que influyen en dicho cálculo. El proceso que realiza el cálculo en cuestión proporciona una funcionalidad concreta y específica: el cálculo del sueldo, pero su codificación dependerá de las directrices de la compañía. Es posible que la compañía establezca un porcentaje de productividad (dependiendo de la categoría del empleado) que influirá en el importe total del sueldo, pero también es perfectamente posible que, en un futuro, introduzca nuevos procedimientos para incentivar, esto supondría un cambio interno en el proceso de cálculo, pero no en su funcionalidad.

Lógica de Datos

En este conjunto entran los procesos encargados de la gestión de los datos propiamente dicha, es decir, los procesos encargados del mantenimiento de los datos, de garantizar las reglas de integridad referencial establecidas, así como de la gestión de las transacciones. Estas tareas son realizadas, generalmente, por un Sistema de Gestión de Bases de Datos Relacionales, como SQL Server, Oracle, MySQL, Informix, etc.

Figura 2. El Modelo de Servicios.

Modelo Cliente/Servidor 2 capas

Uno de los objetivos de las aplicaciones de 2 capas es separar la lógica de acceso a los datos de lo que es la interfaz de usuario y trasladarla al servidor. Habitualmente, se implementan servicios como procedimientos almacenados en el sistema gestor de datos; con esto se

pretende reducir la carga de los clientes y centralizar las operaciones comunes de acceso a los datos. El Sistema Gestor de Datos también suele incorporar la funcionalidad necesaria para trabajar en entornos multiusuarios.
En este modelo intervienen únicamente dos entidades: El Cliente y El Servidor.
El papel de Cliente lo desempeña la aplicación final del usuario, que implementará todas las funciones correspondientes a la lógica de presentación, más algunas de las funciones relacionadas con la lógica del negocio, como pueden ser determinadas validaciones de datos y condiciones de recuperación.
El papel de Servidor lo desempeña el propio SGBD, el cual se ocupará de todas las funciones correspondientes a la lógica de datos, más las restantes funciones correspondientes a la lógica del negocio, mediante la codificación de Procedimientos Almacenados.
Este es el modelo C/S más sencillo y más utilizado habitualmente. En la mayor parte de los casos, el desarrollador de una aplicación de este tipo, desarrolla únicamente la aplicación Cliente y utiliza al propio motor de BD como aplicación servidora, de modo que no se codifica la aplicación Servidora propiamente dicha.
El mantenimiento de las aplicaciones Cliente que utilizan este modelo exige un esfuerzo considerable, dado que las reglas del negocio que son implementadas por sí mismas, provocarán la modificación del código de la aplicación en el caso en que éstas varíen.
Como hemos dicho anteriormente, el resto de las reglas del negocio se implementan en el servidor de datos, que dispone de un recurso propio, denominado Procedimiento Almacenado. Un Procedimiento Almacenado es una porción de código que reside en la Base de Datos, se compila una sola vez y es compartido por todos aquellos usuarios que gocen de los permisos establecidos con relación a su acceso.
El hecho de codificar ciertas reglas del negocio mediante Procedimientos Almacenados, minimiza el impacto que supone la alteración de dichos procedimientos, ya que no siempre es necesaria la re-compilación del código de la aplicación.
Otra consecuencia derivada de dicho modelo es la dependencia de la aplicación final, respecto de la organización de los datos. La aplicación necesita conocer el modelo de datos para poder acceder a él.
Este modelo tiene la desventaja de no ser escalable, pues cada cliente está consumiendo, como mínimo, una conexión con el servidor de datos y, dado que éstas son limitadas, se está restringiendo el número de clientes que pueden coexistir.

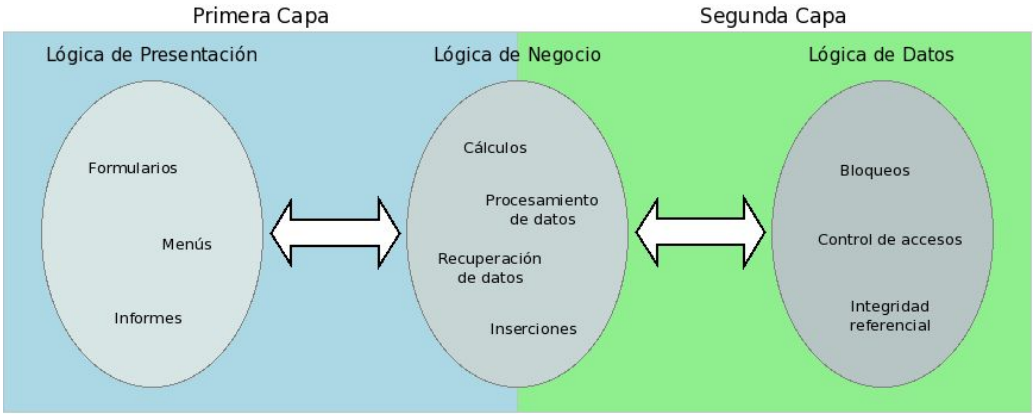

Figura 3. El Modelo Cliente/Servidor de 2 capas.

Modelo Cliente/Servidor multicapa

La arquitectura cliente/servidor genérica tiene dos tipos de nodos en la red: clientes y servidores. Consecuentemente, estas arquitecturas genéricas se refieren a veces como arquitecturas de dos niveles o dos capas.
Algunas redes disponen de tres tipos de nodos:

- Clientes que interactúan con los usuarios finales.
- Servidores de aplicación que procesan los datos para los clientes.
- Servidores de la base de datos que almacenan los datos para los servidores de aplicación.

Esta configuración se llama una arquitectura de tres-capas.
Este modelo aporta una flexibilidad adicional en la construcción de aplicaciones cuando éstas aumentan su complejidad. Influye tanto en el modelo de aplicación (lógicas de presentación, del negocio y de datos) como en la distribución de los servicios. El modelo conceptual de una aplicación establece sus definiciones, reglas y relaciones así como su estructura. Hay partes de la lógica que residen en el cliente, normalmente las que se refieren a la interfaz de usuario, mientras que las del negocio y de datos suelen residir en los servidores, que proporcionan los mecanismos necesarios para el trabajo en entornos multiusuarios. En este tipo de modelo se aplica íntegramente el modelo de servicios ya que, cada una de las capas se corresponde con cada una de las lógicas descritas.
Para llevar a cabo la implementación de un modelo como éste, se hace uso de los mismos recursos que en el modelo de dos capas. En la actualidad las técnicas y lenguajes de programación de servidores han avanzado de tal forma de permitir desarrollos modulares (por ejemplo mediante la arquitectura MVC, Model-Control-View), orientación a objetos y existen innumerables frameworks y librerías para simplificar la tarea de los desarrolladores de software.
Una de las características principales de este modelo reside en la desconexión total entre la lógica de presentación y la lógica de los datos. Las conexiones que se producen, se dan entre las lógicas de presentación y del negocio, y las lógicas del negocio y la de datos. Este modelo hace que la aplicación final sea completamente independiente del origen de los datos que

procesa, tarea que pasa a ser competencia directa del componente especializado.
A pesar de esto, no es necesario que las distintas lógicas residan en máquinas diferentes; en la mayoría de los casos, es perfectamente compatible su implementación en la misma máquina, si bien este diseño no es el más habitual.

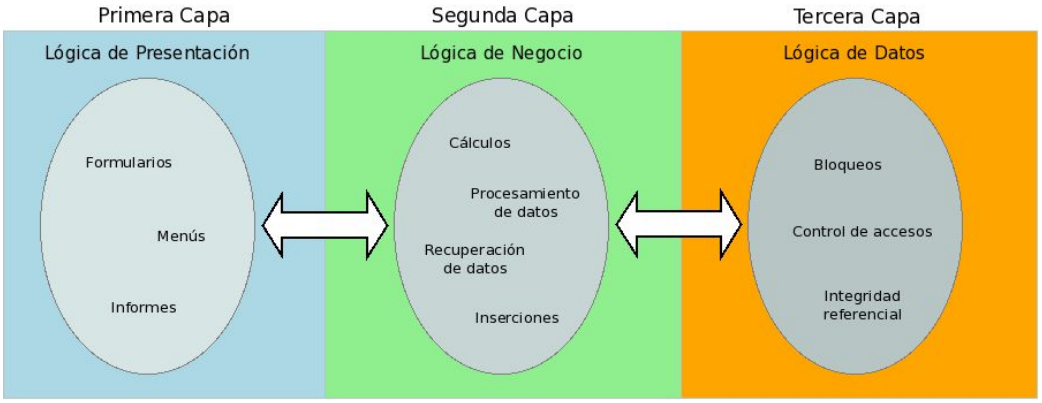

Figura 4. El Modelo Cliente/Servidor multicapa.

Las principales ventajas de este modelo son:

Soporte multi-lenguaje	Los componentes pueden desarrollarse utilizando distintos lenguajes de programación.
Centralización de componentes	Los componentes pueden agruparse y situarse de una manera centralizada, lo que facilita su desarrollo y posterior distribución.
Reparto de carga	Los componentes desarrollados pueden repartirse en varios servidores, mejorando así el rendimiento de la red y la escalablidad.
Accesos más eficientes a los datos	El problema de la limitación de conexiones que admite la base de datos (o para las que se tiene licencia) ahora sólo afecta a los componentes y no a todos los clientes. Además, no es necesario instalar los drivers que se precisan para establecer la conexión con las fuentes de datos, en todos los clientes, sólo se instalarán en aquellos en los que se sitúen los componentes especializados.
Mejor seguridad	Los componentes de la capa intermedia pueden compartir una seguridad centralizada basada en perfiles de usuarios. Es posible asignar o denegar permisos componente a componente o por paquetes. Esto último simplifica su administración.

Con la implementación de clustering de servidores y cloud computing, se agregan más capas al modelo para proveer abstracciones en la implementación de los servicios. Algunos ejemplos son servidores front-end, servidores proxy reverso, balanceadores de carga, etc.

Internet como ejemplo

Internet representa otra forma de arquitectura Cliente/Servidor de 3 capas. Las primeras aplicaciones para Internet no aprovechaban la potencia de las computadoras clientes, ya que los navegadores se limitaban a traducir los archivos HTML que recibían desde los servidores. Los desarrolladores que querían dar una mayor funcionalidad tenían que recurrir a otras tecnologías como Scripts del servidor o CGI.
Las nuevas tecnologías permiten utilizar la potencia de las computadoras clientes ampliando las posibilidades: javascript, HTML5, applets Java, objetos Flash y controles ActiveX son algunos ejemplos. Los navegadores pueden acceder a componentes situados en servidores que implementan reglas del negocio a través de las páginas PHP, ASP, Java, etc. Estas páginas, además de enviar información al cliente, también pasan información desde el cliente a los componentes del negocio situados en la capa intermedia, los cuales asegurarán las reglas del negocio.
Desde el punto de vista de la arquitectura, la lógica de presentación lo implementan las tecnologías en los navegadores cliente, la lógica del negocio la implementan las aplicaciones Web hospedadas en los servidores Web (por ejemplo Apache, IIS o Tomcat) y la lógica de datos la implementan los SGBD (por ejemplo MySQL, SQL Server, Oracle, Informix, Postgres, etc.)

Ventajas y desventajas de cada modelo

Es común encontrarse en entornos corporativos con muchas aplicaciones desarrolladas a medida que fueron implementadas utilizando el modelo Cliente/Servidor tradicional de 2 capas, donde una aplicación monolítica accede directamente al motor de bases de datos (SGBD), tal como se observa en la figura 5.

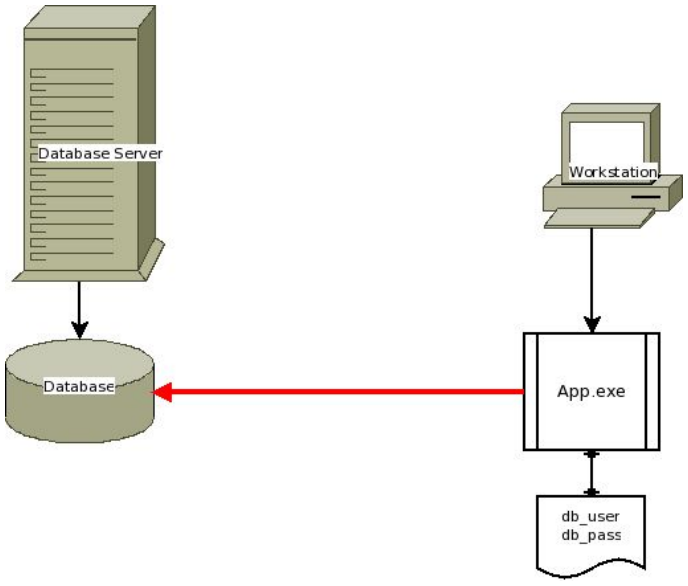

Figura 5. Esquema tradicional de 2 capas.

Esta configuración posee un gran número de desventajas desde el punto de vista de mantenimiento, escalabilidad, confiabilidad y seguridad. Debido a que la aplicación está distribuida en todas las estaciones de trabajo clientes, éstas realizan todo el procesamiento de las capas de presentación y del negocio. Cada vez que se necesita una modificación en la lógica del negocio (y la misma no está cubierta por procedimientos almacenados en la base de datos, stored procedures) es necesario recompilar y redistribuir la aplicación. En lo referente a seguridad, la autenticación de usuarios se hace utilizando usuarios del motor de bases de datos, los cuales se deben ingresar o almacenar en la estación de trabajo cliente. Esto permite, por ejemplo, saltear cualquier control lógico de seguridad o integridad implementado en la aplicación cliente y conectarse directamente al motor de bases de datos para ejecutar consultas arbitrarias de forma directa.

Ventajas:	Desventajas:
● Ninguna (desde el punto de vista del desarrollo puede aducirse que la codificación es más simple pero, debido a la amplia disponibilidad de frameworks y librerías que existen en la actualidad, es falso). ● Tal vez puede aducirse que esta implementación no requiere de un servidor de aplicación. Aunque esto no es ninguna ventaja si se cuenta con una infraestructura virtualizada.	● Pobre escalabilidad: a medida que se incrementan los clientes se debe redistribuir y reconfigurar la aplicación y se deben reconfigurar firewalls, permisos en servidores de BD, asignar nuevas direcciones IP, etc. ● Pobre mantenimiento: si se hacen cambios en el código de la aplicación, se debe redistribuir. ● Pobre confiabilidad: la heterogeneidad de estaciones de trabajo, malware instalado, compatibilidad, etc. puede tener implicancias en la lógica del negocio. ● Pobre seguridad: la autenticación mediante usuarios del motor de bases de datos (a nivel de servicio) presenta un severo riesgo.

A causa de estas desventajas, en algunos entornos Microsoft Windows se ha optado por un esquema un tanto diferente, incorporando un servidor Microsoft Terminal Server. El objetivo claramente consiste en mejorar el mantenimiento, confiabilidad y seguridad de estas aplicaciones implementadas utilizando el esquema clásico de 2 capas.
Los clientes ejecutan la aplicación instalada en un servidor Windows, abriendo una sesión remota restringida (sólo se puede ejecutar la aplicación, no es posible abrir un escritorio remoto), tal como se observa en la figura 6.
Aunque ahora el procesamiento se realiza en un servidor intermedio y se trata de una arquitectura de 3 capas, estas capas no coinciden con el modelo de servicio (como se observa en la figura 4). Esto se debe a que sigue tratándose de una aplicación monolítica que realiza todo el procesamiento de la lógica del negocio y de presentación pues las estaciones de trabajo se convierten en terminales bobas (todo el procesamiento gráfico se realiza en el servidor

Terminal). Por otro lado, desde el punto de vista de seguridad, a pesar de que es mejor que la configuración anterior (ya no es necesario almacenar credenciales del SGBD en las estaciones de trabajo), no es la solución adecuada pues, al autenticar utilizando credenciales de un servidor Windows, se abre todo un nuevo abanico de riesgos y vulnerabilidades.

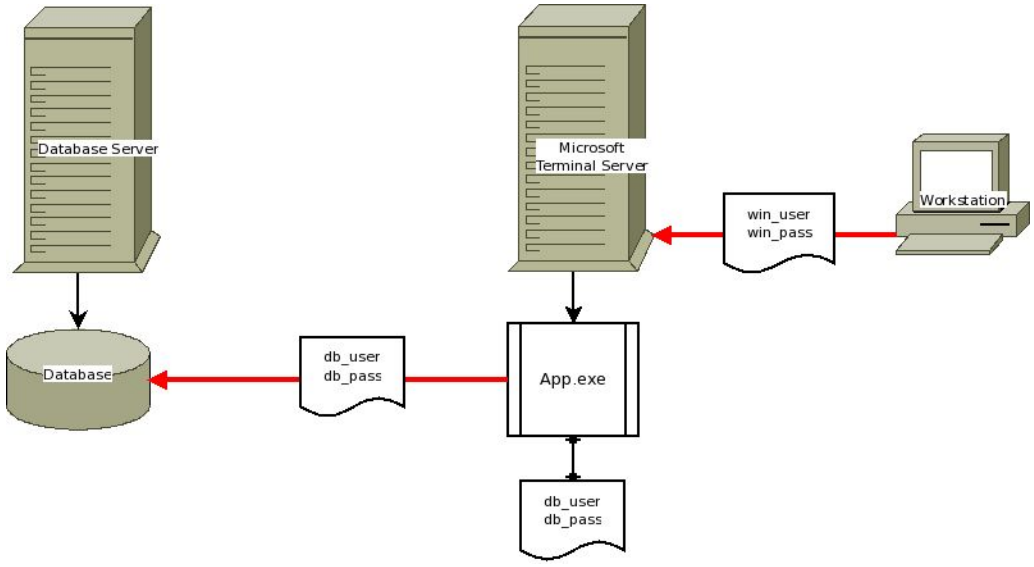

Figura 6. Configuración híbrida de 2/3 capas.

Ventajas:	Desventajas:
● Mantenimiento: no hace falta redistribuir la aplicación si se hacen cambios en el código. ● SGBD: no hace falta utilizar credenciales del motor de bases de datos en las estaciones de trabajo clientes.	● Procesamiento centralizado: todo el procesamiento lo realiza el servidor Microsoft Terminal, lo que rompe con la idea principal del modelo Cliente/Servidor de distribuir el procesamiento. ● Escalabilidad deficiente: a medida que se incrementan los clientes se deben reconfigurar firewalls, permisos en en el servidor Microsoft Terminal, asignar nuevas direcciones IP, etc. Además el procesamiento centralizado limita la escalabilidad respecto a recursos de HW y licencias. ● Seguridad deficiente: la autenticación mediante usuarios de Windows (a nivel SO) introduce nuevas vulnerabilidades.

La configuración deseable para cualquier desarrollo Cliente/Servidor es construir una aplicación en 3 capas que respete el modelo de servicio, tal como se muestra en la figura 4. De esta forma se logra la transparencia entre los datos y la presentación, y se distribuye el procesamiento entre las diferentes partes de la arquitectura. Este modelo es el más escalable

en cuanto a recursos de procesamiento y mantenimiento. Además es el que presenta la mayor fortaleza en cuanto a seguridad y confiabilidad.
Cabe destacar que las capas en esta configuración son abstractas. Las capas de negocio y de datos pueden convivir en el mismo HW. Es decir, los servidores de aplicación y de bases de datos pueden convivir en un mismo equipo (o sistema operativo), en caso de que sea necesario economizar recursos (sin perjuicio de la escalabilidad).
Esta técnica de desarrollo posee una escalabilidad inherente. Puede comenzarse hospedando los servidores que implementan las capas de negocio y datos en un mismo HW, y luego si se incrementa la cantidad de usuarios es posible hospedar los servidores cada uno en su propio HW. Más aún, si a futuro se incrementa notoriamente el número de usuarios de la aplicación es posible implementar cada servidor utilizando un cluster con balanceo de carga.
Así mismo esta técnica no atenta contra la facilidad de codificación, gracias a la amplia variedad de frameworks y librerías disponibles.
Finalmente, cuando se opta por implementar aplicaciones Web, la separación de las capas se obtiene de forma natural (tal como se demostró en la sección anterior para el caso de Internet). Y la ventaja adicional del desarrollo Web es que permite hacer modificaciones en la capa de presentación sin redistribuir software cliente.
La figura 7 muestra la configuración deseable para cualquier desarrollo Cliente/Servidor.

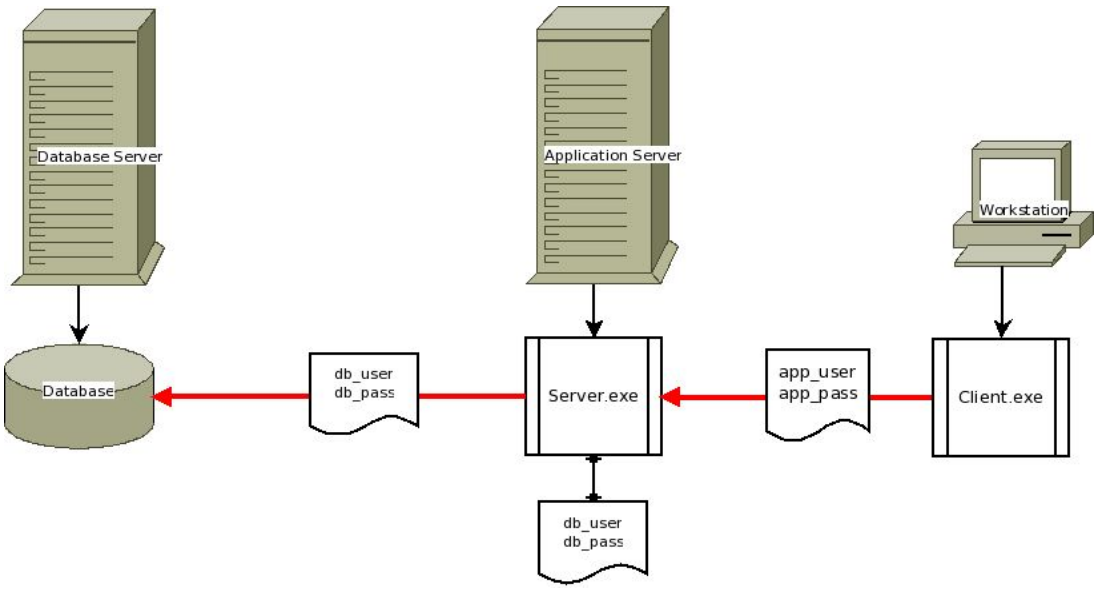

Figura 7. Configuración deseable para cualquier aplicación utilizando el Modelo Cliente/Servidor multicapa.

Ventajas:	Desventajas:
● Mantenimiento: los cambios en la lógica del negocio no afectan a la lógica de presentación, por lo que no es necesario recompilar ni redistribuir código cliente. ● Mayor rendimiento: se logra gracias al procesamiento distribuido, el servidor de	● Complejidad: se incrementa ya que aumenta el número de componentes y se distribuye el procesamiento (en parte atenuado por la variedad de frameworks y

aplicación sólo realiza el procesamiento de la lógica de negocio y deja el procesamiento de la capa de presentación a los clientes. ● Escalabilidad: la separación en capas simplifica el mantenimiento y el procesamiento distribuido permite mejor escalabilidad respecto al HW. ● Seguridad: la autenticación se hace a nivel de aplicación. No se comprometen credenciales de SO o de servicio.	librerías disponibles).