ARQUITECTURAS Y TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES WEB

Andrés Vignaga, Daniel Perovich

Universidad de la República, Facultad de Ingeniería, Instituto de Computación
Montevideo, Uruguay

{avignaga,perovich}@fing.edu.uy

Resumen

En la actualidad, es cada vez más común la aplicación de tecnologías Web en el desarrollo de aplicaciones de porte empresarial. Por su gran cantidad y diferente naturaleza, dichas tecnologías se pueden aplicar tanto para compartir archivos en una intranet, como para desarrollar la interfaz de usuario de un sistema de software. Esta última aplicación de las tecnologías Web es el contexto de este trabajo. Justamente debido a la gran diversidad de opciones, conjuntamente con la dificultad que presenta cada una para su comprensión, es que puede resultar complicado elegir una alternativa, sobre todo para quien no posea un conocimiento profundo en cada tecnología. Este trabajo se centra principalmente en las soluciones provistas por Microsoft y Sun Microsystems. Más que estudiar individualmente cada una de las diferentes tecnologías disponibles, el enfoque seguido es realizar un estudio abstracto de familias de tecnologías que presentan características comunes. El objetivo de esto es comprender la esencia de un modelo tecnológico previo a investigar los detalles de una tecnología particular. Asimismo, se presenta un relevamiento de aquellos aspectos de la arquitectura, incluyendo una clara separación entre arquitectura física y lógica, que definen la infraestructura, de cuya configuración depende la aplicabilidad de un determinado modelo tecnológico.

Palabras clave: Tecnologías Web, Arquitectura de Software, Desarrollo en Capas, Cliente / Servidor, Intranet

Indice

Introducción

El objetivo principal de este informe es la presentación y estudio de las características generales de las tecnologías que brindan *Microsoft* y *Sun Microsystems* para el desarrollo de aplicaciones de tipo empresarial con arquitectura cliente/servidor. Este trabajo no debe ser entendido como una referencia para el uso de las tecnologías tratadas. Por el contrario, debe ser un medio para conocer los detalles particulares de la filosofía de cada una, de forma de brindar elementos de juicio para la toma de decisiones en una realidad particular.

El tratamiento dado en este trabajo a las tecnologías es general, aunque en ciertos casos el enfoque se ajusta a aplicaciones de tipo empresarial. Existe actualmente una fuerte tendencia al uso de tecnologías web en este tipo de aplicaciones. Esta tendencia se debe a que resulta un medio sencillo y conocido, en comparación a otros, para implementar ciertos aspectos de una aplicación con arquitectura cliente/servidor. El uso de tecnologías web en redes privadas, como por ejemplo de área local, hace de éstas lo que se conoce como intranets. Desarrollar para intranets permite la utilización de todas las tecnologías disponibles para el desarrollo para Internet.

Muchas de las tecnologías tratadas en este trabajo sirven para desarrollar tanto para Internet como para intranets, aunque son estudiadas también otro tipo de tecnologías.
Por otra parte, en lo referente al diseño de aplicaciones, el término arquitectura en capas está actualmente muy difundido, aunque su significado no está estandarizado. El desarrollo de aplicaciones de mediano o mayor porte no deben quedar ajenos a este tema. En este trabajo se confronta la arquitectura cliente/servidor con la arquitectura en capas, buscando un relacionamiento entre ambos conceptos. Como se verá, las tecnologías aplicables al desarrollo de una aplicación cambian en función del diseño global de la arquitectura de la aplicación.

En este contexto, el estudio se centra en los aspectos arquitectónicos y tecnológicos de los componentes responsables de la presentación de la aplicación. Estos componentes son los que residen en el cliente junto con los que interactúan con ellos desde el lado del servidor. Las tecnologías que no se apliquen a este aspecto no son tratadas aquí.

Debido a que la arquitectura condiciona la aplicabilidad de ciertas tecnologías se considera necesario dedicar atención a este tema. Arquitectura cliente/servidor y en capas son términos no estandarizados y algo confusos, la bibliografía no es uniforme respecto a ellos e inclusive en ocasiones confunde ambos términos. Aquí se los define por separado y se determina una relación entre ambos conceptos que representa nuestra visión sobre el tema ya que no se encontró una referencia adecuada. El tratamiento de estos temas sirve además para uniformizar la terminología a lo largo del trabajo y permite abstraer las características comunes que puedan tener un grupo de tecnologías. Una vez comprendidas estas características, resulta más sencillo comprender los detalles.

Este documento está organizado de la siguiente forma: los capítulos 1 y 2 definen los conceptos de arquitectura cliente/servidor y arquitectura en capas respectivamente; el capítulo 3 describe la relación de correspondencia entre los componentes de las arquitecturas de los dos capítulos anteriores. En el capítulo 4 se distinguen los modelos tecnológicos posibles de acuerdo a las tecnologías disponibles actualmente y su correspondencia con la relación definida en el capítulo 3. El capítulo 5 estudia las tecnologías concretas correspondientes a cada modelo y el capítulo 6 resume aplicaciones reales de algunas de las tecnologías tratadas.

Capítulo 1
Arquitectura Cliente/Servidor

1.1 Introducción

En este capítulo se presentan los conceptos básicos de la arquitectura cliente/servidor que resultan elementales para el tratamiento de aplicaciones distribuidas.
Además se motiva el uso de tecnologías Internet/Intranet para el desarrollo de aplicaciones empresariales, como un medio de reducir la complejidad que representa la conectividad entre módulos de software distribuidos a través de una red.

1.2 ¿Qué es Cliente/Servidor?

1.2.1 Definición

No hay un consenso sobre el significado exacto del término. Diferentes autores proponen diferentes definiciones. La siguiente definición introduce el concepto en forma simple:

El modelo cliente/servidor es un modelo de comunicación de computadores en el cual el computador cliente solicita servicios al computador servidor por medio de mensajes. La diferencia entre el cliente y el servidor es que el cliente es el que inicia el contacto y el servidor es el que responde a dicha solicitud de conexión.

1.2.2 Cliente y Servidor

Clientes y servidores son entidades físicas diferentes que operan en conjunto a través de una red para realizar una tarea.

La arquitectura cliente/servidor está compuesta por tres elementos básicos, el cliente, el servidor y el middleware, como se puede observar en el diagrama de la figura 1.1.

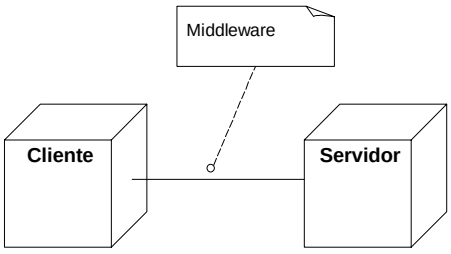


Figura 1.1

1.2.3 Middleware

El middleware abarca todo el software distribuido necesario para el soporte de interacciones entre clientes y servidores. Es el enlace que permite que un cliente obtenga un servicio del servidor. Empieza en el módulo de API de la parte del cliente que se emplea para invocar un servicio y comprende la transmisión de la solicitud por la red y la respuesta resultante. No incluye al software que presta el servicio real ni a la interfaz del usuario.

El middleware puede clasificarse como de bajo o alto nivel.

- Bajo nivel: Brinda las primitivas y servicios usuales de bajo nivel. Un ejemplo se servicio de bajo nivel son los sockets, mientras que un ejemplo de middleware de bajo nivel es TCP/IP.
- Alto nivel: Se emplea para cumplir un tipo particular de servicio, y está usualmente implementado sobre middleware de bajo nivel.

Ejemplos:

- Para base de datos: ODBC y JDBC
- Para objetos: CORBA, DCOM y RMI
- Para Internet/Intranet: HTTP

1.2.4 Características del modelo Cliente/Servidor

Servicio
Cliente/servidor es fundamentalmente una relación entre procesos ejecutados en computadores distintos. El proceso del servidor hace de éste un proveedor de servicios. El cliente es un consumidor de servicios.

Recursos compartidos
Un servidor puede atender a muchos clientes al mismo tiempo y regular su acceso a recursos compartidos.

Protocolos asimétricos
Entre cliente y servidor se establece una relación de *n* a 1. Son siempre los clientes los que inician el diálogo al solicitar un servicio. Y los servidores aguardan pasivamente las solicitudes de los clientes.

Transparencia de ubicación
El servidor es un proceso que puede residir en el mismo equipo que el proceso cliente o en un equipo distinto a lo largo de una red. Suele ocultarse a los clientes la ubicación del servidor mediante el redireccionamiento de las llamadas de servicio en caso de ser necesario. Un programa puede ser un cliente, un servidor o ambos a la vez.

Mezcla e igualdad
El software ideal de cliente/servidor es independiente del hardware y de la plataforma de software del sistema operativo. Se debe poder mezclar e igualar las plataformas del cliente y del servidor.

Intercambio basado en mensajes
Clientes y servidores tienen bajo acoplamiento e interactúan a través de un mecanismo determinado de transmisión de mensajes. El mensaje es el mecanismo de entrega para las solicitudes y de respuestas de servicios. (se considera como mensaje tanto a la solicitud como a la respuesta).

Encapsulamiento de servicios
Un mensaje le indica al servidor que servicio se solicita. Los servidores pueden ser sustituidos sin afectar a los clientes, siempre y cuando la interfaz implementada por el servidor no cambie. Ver figura 1.2.

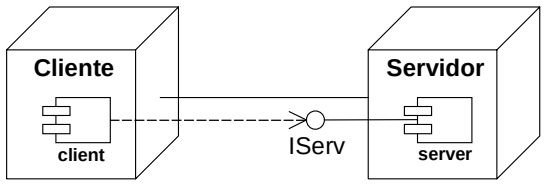


Figura 1.2

Facilidad de Escalabilidad
Los sistemas cliente/servidor pueden escalarse horizontal o verticalmente. Por escalabilidad horizontal se entiende el agregar o eliminar estaciones de trabajo de cliente con apenas un ligero impacto en la performance. Escalabilidad vertical significa migrar el servidor actual a un servidor más potente o a múltiples servidores.

Integridad
El código del servidor y los datos se conservan centralmente, lo que resulta en un mantenimiento de menor costo y en la protección de los datos de la integridad de los datos compartidos. A su vez, los clientes mantienen su individualidad e independencia.

1.2.5 Ventajas y desventajas del modelo Cliente/Servidor

En esta subsección se presentan las ventajas y desventajas de la arquitectura cliente/servidor frente al modelo centralizado donde existe un único nodo físico.

Ventajas

- Posibilidad de reducir costos de desarrollo.
- Mejores herramientas de desarrollo.
- Modificabilidad es compatible con buen diseño.
- Flexibilidad en el cliente.
- Permite un mejor control sobre permisos de acceso a información.
- Escalabilidad. Si la empresa cambia en lo que respecta al tamaño, alcance, requerimientos de información, la escala de la arquitectura cliente/servidor puede ser cambiada.
- La tecnología disponible es tan variada que puede utilizarse la que mejor se adecue a la realidad de la empresa.
- El software puede ser desarrollado desde varios lugares, lo que reduce el costo de desarrollo.
- Soporta los conceptos de comercio electrónico.

Desventajas

- Falta de profesionales calificados.
- Necesidad de reimplementación de software existente, ya que el software monolítico no suele ser compatible con la arquitectura cliente/servidor.
- Necesidad de entrenamiento de usuarios, debido al cambio de las aplicaciones existentes.

1.3 Uso de Tecnología Web

La World Wide Web (en adelante web) está redefiniendo la computación de cliente/servidor. En muchos casos esta provocando que LANs y WANs se conviertan en Internet, aunque técnicamente son intranets, dado que se ocultan tras firewalls.
Estas intranets están derribando las tradicionales barreras entre la computación cliente/servidor interdepartamental y empresarial.

1.3.1 Definiciones

Internet
Conocida como la red de redes, es una red de alcance mundial integrada por miles de redes y millones de computadores con fines comerciales, educativos, gubernamentales, personales, todas conectadas entre sí.

Intranet
Una red autocontenida que utiliza los mismos protocolos de comunicación y formato de archivos que Internet. Una intranet puede, aunque no necesariamente debe, estar conectada a Internet. Muchas empresas usan intranets para su comunicación interna.

1.3.2 Motivación

Las empresas se han estandarizado en TCP/IP para contar con soporte de HTTP (HyperText Transfer Protocol) en sus redes principales de cliente/servidor. HTTP permite el uso de software de cliente y servidor web de muy bajo costo. Los browsers resultan muy sencillos y amigables para los usuarios, y corren en cualquier plataforma. Los servidores web brindan gateways (punto de entrada) a todas las plataformas conocidas, lo que los convierte en un punto estratégico para la recopilación y distribución de información.

El hecho de llevar las tecnologías web a las LANs ha permitido aplicar las tecnologías de desarrollo web a aplicaciones empresariales, las que heredan todos los beneficios de estas tecnologías, de su modelo, y de su implementación.

Capítulo 2
Arquitecturas en Capas

2.1 Introducción

El término arquitectura cliente/servidor es utilizado comúnmente para hablar de la arquitectura del software. De igual forma, los términos arquitectura en 2 y 3 capas se aplican al mismo concepto. Debe distinguirse entre ambos.

Se entiende por arquitectura física a la *topología de la aplicación*. Independientemente de ésta, desde un punto de vista lógico, una aplicación puede ser dividida en componentes que denominamos capas. Estas son unidades altamente cohesivas, con responsabilidades de alto nivel, bien definidas y autocontenidas. A la organización del software en términos de estos componentes le llamamos *arquitectura lógica.*

Arquitectura cliente/servidor habla de la topología del software, mientras que decir que su arquitectura es en capas habla de su arquitectura lógica.

Las aplicaciones de software presentan tres aspectos fundamentales: debe hacer que los datos sean persistentes (D), debe procesarlos en forma acorde a la lógica de negocios (L), y debe presentarlos adecuadamente a los usuarios (P).

Las aplicaciones en 1 capa (P+L+D), donde no se distingue una separación lógica de estos tres aspectos, son muy grandes, difíciles de mantener, de distribuir, incompatibles con la arquitectura cliente/servidor, pesadas y con gran consumo de recursos.

Un primer acercamiento a la distribución de las responsabilidades de la aplicación en dos unidades lógicas fue la arquitectura en 2 capas, que es presentada en la siguiente subsección.

Por último, en la actualidad se tiende a desarrollar aplicaciones con arquitectura en 3 capas, donde cada uno de los aspectos se corresponde a una unidad lógica.

2.2 Arquitectura en 2 Capas

Un arquitectura en 2 capas distribuye la aplicación en dos componentes lógicos. Las responsabilidades de cada componente hacen a las variantes de esta arquitectura.

Surge la arquitectura en 2 capas como consecuencia de la arquitectura cliente/servidor. Esta topología permite distribuir la carga de la aplicación a dos computadores diferentes, lo que llevó naturalmente a distribuir las responsabilidades de la misma a dos unidades lógicas.

2.2.1 Arquitectura P+L/D

Una primer variante es retirar el manejo de datos de la aplicación. Esto permite a varios clientes utilizar el mismo juego de datos. P+L es una unidad lógica por sí, donde el manejo de interfaz de usuario y el manejo de la lógica no se los distingue como módulos independientes. Típicamente P+L se encuentra en el cliente, mientras que D se encuentra en el servidor.
Un ejemplo de aplicaciones con esta arquitectura es una aplicación que delega la persistencia a un manejador de base de datos.

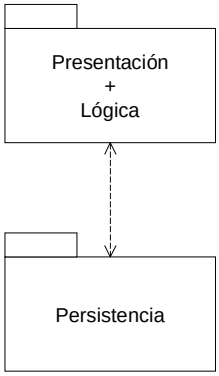


Figura 2.1

2.2.2 Arquitectura P/L+D

El hecho de tener la misma lógica en cada cliente permitió factorizarla, llevando la misma al servidor. Aquí la lógica de la aplicación se encuentra embebida al manejo de la persistencia de datos. En este tipo de aplicaciones la lógica resuelve los problemas de persistencia encargándose ella misma de dicha tarea, no necesariamente utilizando un manejador de base de datos, o embebiendo toda la lógica de negocios en el mismo.

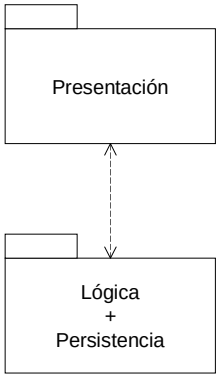


Figura 2.2

2.2.3 Arquitectura P+L/L+D

Una tercer variante es repartir la tarea de la lógica, una parte junto a la interfaz de usuario, y otro junto al manejo de persistencia de datos.
Un ejemplo de aplicaciones con esta arquitectura son aplicaciones similares a las que tienen arquitectura P+L / D, que tienen implementada parte de la lógica en procedimientos almacenados en el manejador de la base de datos.

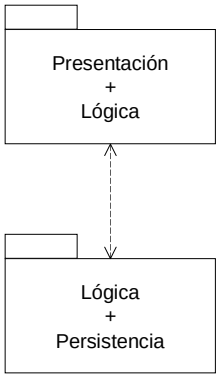


Figura 2.3

2.2.4 Desventajas de la Arquitectura en 2 capas

- La lógica de la aplicación no puede ser reusada ya que está ligada o a la interfaz de usuario o al manejo de persistencia de datos.

- Las estaciones de trabajo pueden tener serias restricciones de recursos. Los desarrolladores deben estar entrenados para optimizar la aplicación de forma que pueda ser utilizada en dichos entornos.

- Incremento de la carga de la red: dado que el procesamiento de los datos se realiza en el cliente, gran cantidad de información debe ser transmitida desde el servidor.

- El PC procesa y presenta la información. Lleva a aplicaciones monolíticas, caras y difíciles de mantener. (“fat client”).

- La “lógica de negocios” está implementada en el PC. Notar que la lógica de negocios nunca usa el sistema de ventanas.

- Implica un procedimiento de distribución complicado, ya que en caso de un cambio todos los PCs deben ser actualizados. Es difícil garantizar que un cliente está corriendo una versión anterior

2.3 Arquitectura en 3 Capas

La arquitectura en 2 capas, con su variante P/L+D, dio lugar a la arquitectura en 3 capas. El hecho de que la lógica de negocios y el manejo de persistencia sean una unidad presentaba desventajas importantes: el manejador de base de datos resultaba pequeño y quería migrarse a otro, debía actualizarse la versión, o se deseaba incorporar datos de nuevas fuentes.

En esta arquitectura la lógica de la aplicación ocupa una capa intermedia; está separada tanto de los datos como de la interfaz de usuario (P/L/D). Los procesos pueden ser administrados y desplegados en forma autónoma, sin relación con la interfaz de usuario y el manejador de base de datos. En teoría, los sistemas en 3 capas son de más fácil ampliación y más robustos y flexibles. Además, pueden integrar datos de múltiples fuentes.

Es importante notar que los límites entre las capas son lógicos, por lo que es posible ejecutar las tres capas en la misma máquina. Lo importante es que el sistema está claramente estructurado y que hay una buena planificación de los límites entre las diferentes capas.

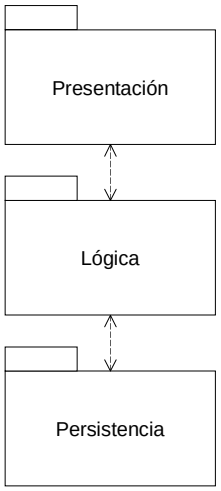


Figura 2.4

En esta sección, a diferencia de la anterior, no se trata la correspondencia de cada capa con respecto a los nodos físicos. En la arquitectura en 2 capas esta correspondencia es directa, pero en este caso no lo es. Esta correspondencia es estudiada en detalle en el capítulo siguiente.

2.3.1 Responsabilidades de las capas

Capa de presentación

- Es responsable de la presentación de los datos, recibiendo los eventos de los usuarios y controlando la interfaz de usuario.

Capa de lógica de negocios

- Esta capa es nueva, es decir, no está presente en la arquitectura en 2 capas en forma explícita
- Los objetos de negocios que implementan las reglas de negocios “viven” aquí, y están disponibles para la capa de presentación
- Esta capa es la clave para resolver los problemas de la arquitectura en 2 capas
- Protege del acceso directo a la información desde la capa de presentación

Capa de persistencia

- Es responsable del almacenamiento de los datos
- Es común reusar sistemas existentes de bases de datos en esta capa
- Actualmente se usan manejadores relacionales: son avanzados, permiten el uso de triggers y paquetes. Existen manejadores Orientados a Objetos

2.3.2 Ventajas de la arquitectura en 3 capas

- Separación clara de la interfaz de usuario de la lógica de la aplicación. Esta separación permite tener diferentes presentaciones accediendo a la misma lógica
- La redefinición del almacenamiento de información no tiene influencia sobre la presentación
- En contraste con una arquitectura en 2 capas, donde solamente datos están accesibles al público, los objetos de negocios pueden brindar servicios (lógica de la aplicación) por la red

Otras ventajas dependen de la distribución de los componentes lógicos sobre los nodos físicos. Estas serán discutidas en el siguiente capítulo.

Capítulo 3
Distribución de los Componentes Lógicos

3.1 Introducción

Los capítulos 1 y 2 introducen dos enfoques diferentes para describir la arquitectura del software: un enfoque físico y otro lógico. Los mismos son ortogonales, de modo que el diseño de una aplicación debe considerar a ambos.

El diseño lógico fue estudiado a fondo en el capítulo 2, obteniendo como conclusión que una arquitectura en 3 capas es recomendable a una arquitectura en 2 capas. El siguiente diagrama muestra la arquitectura lógica de una aplicación en 3 capas:

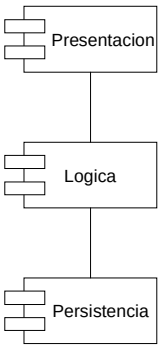


Figura 3.1

El diseño físico fue introducido en el capítulo 1. Las ventajas del modelo lo hacen muy atractivo para aplicaciones de gran porte donde numerosas estaciones de trabajo deben trabajar en el mismo sistema.

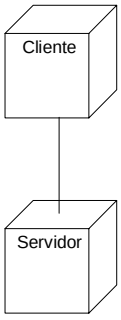


Figura 3.2

Optar por la utilización de ambos modelos en el desarrollo de una aplicación de software es razonable, aunque debe realizarse consideraciones acerca de como distribuir los componentes lógicos sobre los nodos físicos.

El presente capítulo presenta consideraciones de diseño para el desarrollo de aplicaciones en 3 capas sobre un modelo cliente / servidor.

3.2 Distribuciones

3.2.1 Monolítica

La distribución básica no considera una arquitectura cliente / servidor. Refiere a una aplicación que reside en un único nodo físico. Como se dijo anteriormente, esta distribución no tiene las ventajas que aporta un modelo cliente / servidor.
El siguiente diagrama presenta esta distribución:

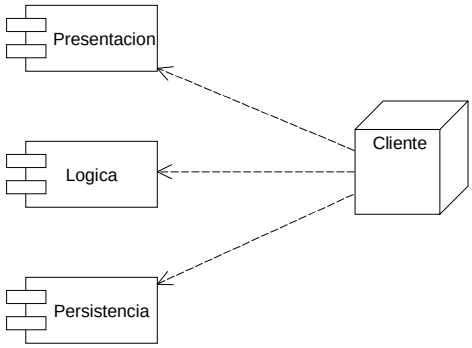


Figura 3.3

3.2.2 Lineal

La distribución lineal consiste en ubicar cada componente lógico en un nodo físico. La forma usual es ubicar la capa de presentación en el nodo cliente, y ubicar la capa lógica y de datos en el nodo servidor, ver figura 3.4. Otra variante es ubicar la capa de datos en un segundo servidor, accedido desde el nodo físico donde reside la capa lógica. Esta distribución cuenta con las ventajas del modelo cliente / servidor y la distribución se realiza en forma sencilla, ya que solo se necesita implementar un mecanismo de conexión entre las capas, ver figura 3.5.

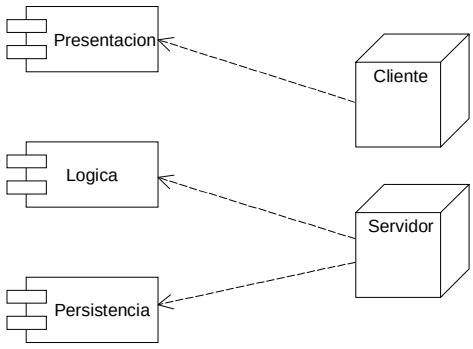


Figura 3.4

Puede considerarse al conjunto de servidores como un paquete de servidores, indicando que cada componente reside en el paquete, sin explicitar si esta en uno u en otro. Es decir, podemos abstraer el conjunto de servidores, y considerarlo como un nodo físico.
El siguiente diagrama presenta esta noción:

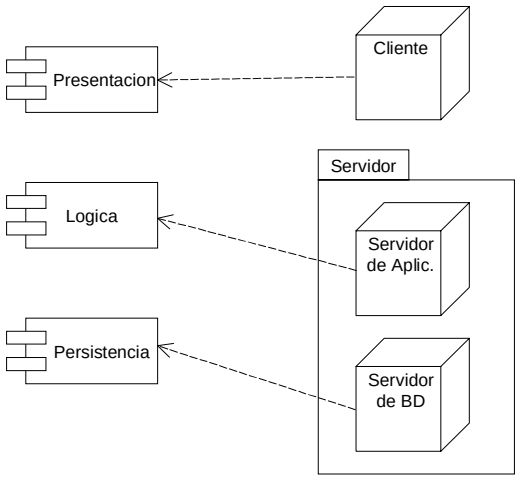


Figura 3.5

3.2.3 Componente distribuido

Un caso especial de distribución es cuando se ubica un componente en más de un nodo físico.
El siguiente diagrama presenta esta idea:

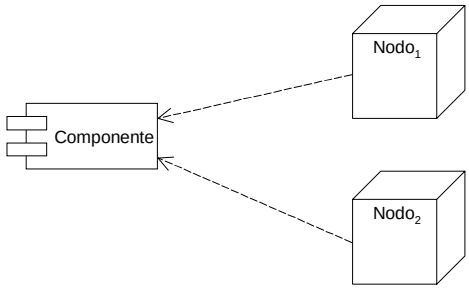


Figura 3.6

Esta distribución no aparece en forma natural. Dado que el problema se resuelve fácilmente con una distribución lineal, sin mayor esfuerzo de desarrollo, por qué considerar el caso donde es necesario distribuir uno de los componentes. Limitaciones tecnológicas, mejora en la performance y modularización pueden ser razones que motiven a distribuir un componente.

Se debe considerar que para distribuir un componente, éste debe ser rediseñado, o ser diseñado originalmente con tal filosofía. El componente quedará compuesto por dos o más componentes, cada uno con sus propias responsabilidades (asignadas en el diseño). Estos componentes se corresponderán a un nodo físico.
El diagrama de la figura 3.7 presenta el concepto anterior.

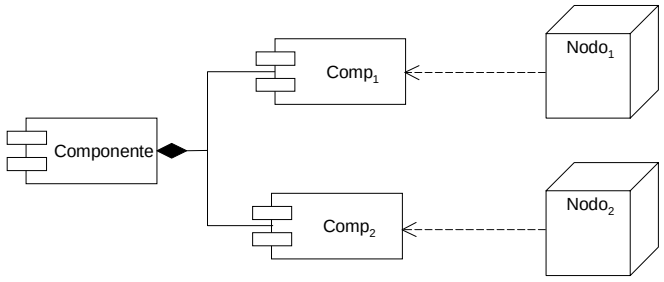


Figura 3.7

Un ejemplo típico de distribución de la capa de datos (D) es un manejador de base de datos distribuidas. Otra aplicación de este concepto es modularizar en subsistemas la capa lógica (L) de la aplicación.

Un caso de distribución de la capa de presentación (P), al cual denominamos *distribución orientada al web*, será tratado en la siguiente subsección.

3.2.4 Orientada al Web

La distribución orientada al web es un caso particular de la distribución del componente de la capa de presentación (P).

La particularidad de este caso es la utilización de tecnología web en la capa de presentación. Se distribuye el componente en dos subcomponentes, uno reside en el cliente, y el otro en el servidor.

El componente a nivel del cliente es un cliente web tradicional (web browser).
El componente a nivel del servidor consiste en un web server capaz de generar contenido dinámico, junto a la implementación de este contenido.

El diagrama de la figura 3.8 presenta esta distribución.

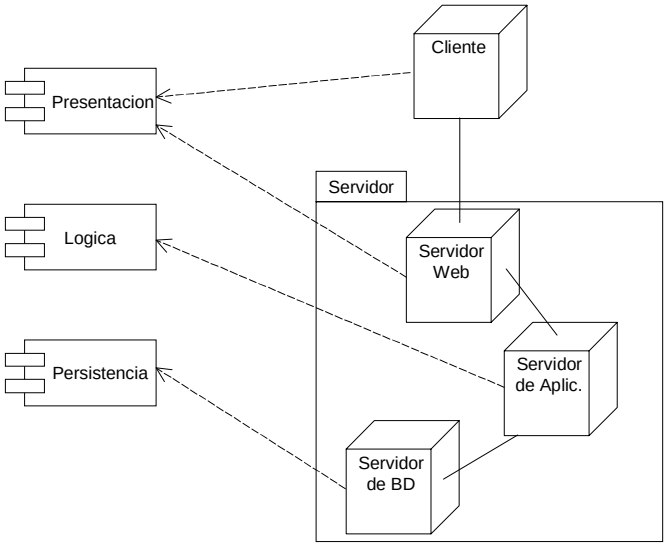


Figura 3.8

3.3 Consideraciones

La distribución monolítica carece de las ventajas que brinda el modelo cliente/servidor. Tal distribución puede tentar a llevar el motor de base de datos a un servidor, dejando las otras dos capas en el cliente. Como ya fue explicado, esto presenta serias desventajas.

La distribución lineal y de componente distribuido son las más aplicables.

La primera no implica un trabajo de desarrollo extra, salvo la comunicación de las capas utilizando el middleware disponible. Es más fácil y rápido intercambiar un componente en un servidor que en muchas estaciones de trabajo. Los servidores son sistemas confiables por lo que es más fácil de obtener protección y seguridad. Tiene sentido entonces ejecutar procesos de negocios críticos y que trabajan con datos importantes, en el servidor
Sin embargo presenta una desventaja; si la aplicación presenta un cambio (por ejemplo un cambio en los requerimientos), el mismo se verá reflejado en todas las capas, lo que implica llevar a cada estación de trabajo una nueva versión del componente de presentación.

La distribución de componente distribuido presenta muchas variantes con respecto a la ubicación de los subcomponentes en los nodos físicos.

La distribución del componente de persistencia está fuera del alcance de este documento ya que no influye sobre el nodo cliente.

La distribución del componente de lógica es un caso muy usual, en particular cuando el sistema notoriamente puede dividirse en subsistemas. Si la distribución se realiza en forma horizontal, es decir que todos los componentes residen a nivel de servidor, no presenta grandes dificultades. Si la distribución se hace vertical, es decir, se lleva parte de la lógica al nodo cliente, la aplicación presentará las desventajas de la arquitectura en 2 capas P+L/D y P+L/L+D.

La distribución del componente de presentación permite centralizar la generación de la interfaz de usuario en el nodo del servidor.

La distribución orientada al web adopta el enfoque de distribuir el componente de presentación, llevando la tarea de la generación de interfaz al servidor (o a uno de sus componentes). En este caso el desarrollo extra necesario es pequeño ya que se cuenta con una aplicación en el cliente capaz de interpretar cualquier interfaz de usuario, otra en el servidor capaz de generarla, y el middleware capaz de transportarla. Es necesario la implementación de la parte que brinda el contenido.

Capítulo 4
Modelos Tecnológicos

4.1 Introducción

Los capítulos anteriores presentaron las ventajas de desarrollar aplicaciones con arquitectura cliente/servidor, en 3 capas y sin lógica en el cliente. Presentaron además distintas distribuciones de los componentes lógicos sobre los nodos físicos.

Actualmente existe un amplio abanico de tecnologías para el desarrollo de aplicaciones con esta arquitectura. Algunas tecnologías presentan características comunes entre sí. Esto motiva una categorización de las tecnologías según su modelo de funcionamiento.

Este capítulo presenta un estudio general de las categorías mencionadas, enmarcando las mismas en las distribuciones presentadas antes.

El próximo capítulo presenta un análisis detallado de cada tecnología en particular.

4.2 Modelos

Se presenta a continuación tres modelos o categorías.

Modelo Clásico

Este modelo consiste en dos aplicaciones, donde una reside en el cliente y la otra en el servidor, como se ilustra en la figura 4.1.

Figura 4.1

Modelo de Componentes Gráficos

Este modelo consiste de una aplicación conocida como cliente universal (web browser), que reside en el cliente, un servidor web y una aplicación que residen en el servidor, ver figura 4.2.

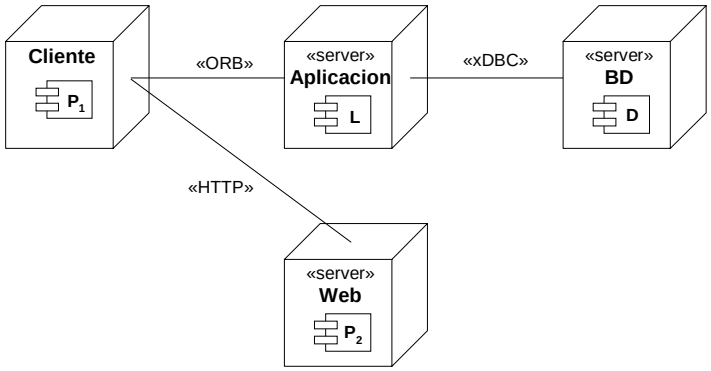


Figura 4.2

Modelo de Contenido Dinámico

Este modelo consiste en una aplicación conocida como cliente universal (web browser), que reside en el cliente, un servidor web que reside en el servidor. Ver figura 4.3.

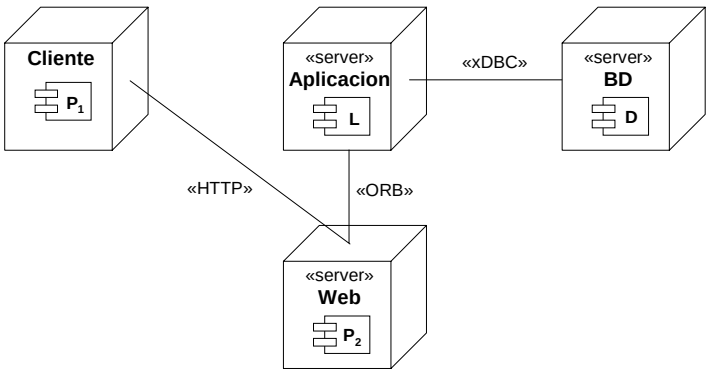


Figura 4.3

4.3 Modelo Clásico

4.3.1 Descripción del modelo

Este modelo consiste de dos aplicaciones, donde una reside en el cliente y la otra en el servidor.

Puede tomarse distintos enfoques para este modelo según la asignación de responsabilidades a cada aplicación. Se analizará a continuación los más usuales.

El primer enfoque, la aplicación en el cliente es responsable de la interfaz de usuario, mientras que la aplicación en el servidor brinda acceso a la lógica de negocios, como se ilustra en la figura 4.4.
Notar que este enfoque presenta una distribución lineal por lo que tiene las características del mismo, vistas en la sección 3.2.

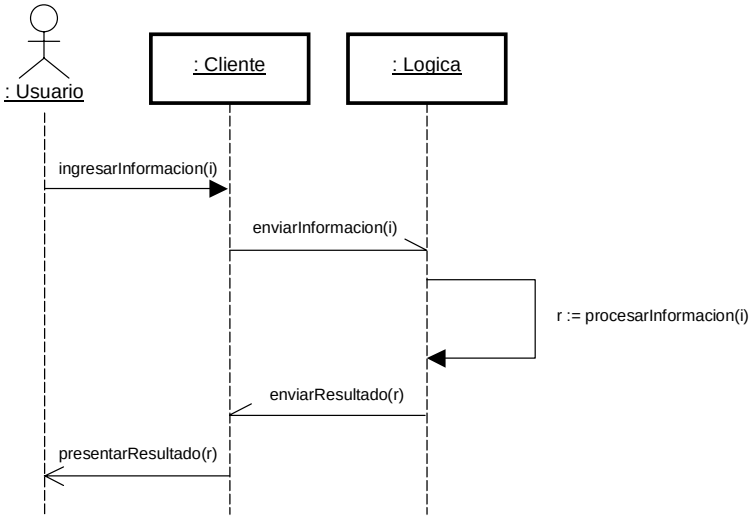


Figura 4.4

Un segundo enfoque es distribuir el componente de la lógica, poniendo parte del mismo en la aplicación en el cliente y parte en la aplicación en el servidor.
Notar que este enfoque presenta una distribución de componente distribuido, donde lo que se distribuye es el componente de la lógica, llevándolo al cliente. Este enfoque presenta las características vistas en la sección 3.2.

Un tercer enfoque es distribuir el componente de la presentación. La aplicación en el cliente se encarga del despliegue de la interfaz de usuario, mientras que el servidor se encarga de generarla.
Notar que este enfoque presenta distribución de componente distribuido, donde lo que se distribuye es el componente de la presentación. En particular, hereda varias de las ventajas de la distribución orientada al web.
Sin embargo exige un esfuerzo de desarrollo grande ya que las aplicaciones a implementar deben brindar iguales funcionalidades que la tecnología web. Existen otras tecnologías que facilitan el desarrollo mediante este enfoque. Las mismas se tratarán en los modelos siguientes.

4.3.2 Tecnologías

En este modelo las aplicaciones son aplicaciones tradicionales desarrolladas en lenguajes de programación tradicionales como C++, Java, Visual Basic, entre otros.

Los aspectos de interoperabilidad entre las aplicaciones van a depender directamente del middleware utilizado. Esto es, no todos los lenguajes soportan todos los tipos de middleware disponibles.

4.3.3 Middleware

El middleware empleado por este modelo conecta la presentación que reside en el cliente con la lógica que reside en el servidor. Esta conexión puede implementarse utilizando sockets o un recurso de más alto nivel como ORBs (Object Request Broker).

4.3.4 Cualidades

Legibilidad	Directamente proporcional al buen estilo de programación aplicado.
Mantenibilidad	La utilización de sockets hace que la implementación sea difícil de mantener. Utilizar ORB puede simplificar esta tarea. Un cambio, en particular en la presentación, implica un deployment.
Extensibilidad	Una extensión implica el deployment de la aplicación cliente completa. Si se utiliza ORBs no hay problema ya que es una invocación a un nuevo objeto; en cambio, si se utiliza sockets, implica una modificación en el protocolo entre los componentes.
Deployment	Llevar una nueva versión de un componente de la aplicación cliente a cada estación de trabajo no es una tarea automatizada. Puede tenerse problema de versiones en los componentes en la misma estación y deployment parciales puede llevar a tener estaciones de trabajo con varias versiones de atraso.
Carga de la red	En el primer enfoque viaja por la red solamente la información necesaria, ya que la interfaz reside, por completo, en el cliente. Dependiendo del protocolo, cada pieza de información puede significar uno o más solicitudes a través de ORB. Se debe tener serias consideraciones en el diseño para minimizar esta importante carga.
Tiempo de respuesta	No es simple desarrollar una aplicación servidor capaz de atender gran cantidad de clientes sin que el sistema sufra saturación en el servicio. El middleware puede influir sobre el tiempo de respuesta. Utilizar cierto ORB puede hacer que el tiempo de respuesta aumente significativamente.
Tiempo de desarrollo	El desarrollo de la interfaz gráfica puede resultar complicado en algunos lenguajes. Depende además de las virtudes del IDE utilizado.
Portabilidad	El middleware determina si se puede o no intercambiar el desarrollo de una aplicación hecho en un lenguaje por otro. El middleware y el lenguaje puede determinar la plataforma sobre la cual correrá la aplicación.
Escalabilidad a nuevos clientes	Depende de las limitaciones que presente el servidor. Implica además exigir a cada nuevo cliente que corra la aplicación que se ejecute la aplicación correspondiente, con las consecuencias de deployment que esto conlleva.
Robustez	El modelo en sí es robusto, pero depende de la robustez del ORB, de la plataforma y de la aplicación propiamente.
Requerimientos	Requiere de un equipo potente en el cliente capaz de correr una aplicación con alto contenido visual. Se debe contar con los módulos requeridos para utilizar el middleware elegido.

4.4 Modelo de Componentes Gráficos

4.4.1 Descripción del modelo

Este modelo consiste de una aplicación conocida como cliente universal (web browser), que reside en el cliente, un servidor web y una aplicación que residen en el servidor.

El funcionamiento del modelo es el siguiente:

El cliente solicita una página del servidor web. La misma contiene un componente que será ejecutado en el cliente. Este componente despliega la información al usuario y recoge información del mismo. Ésta es enviada al servidor ya sea por sockets o ORB. La aplicación en el servidor procesa la información y envía su respuesta. Esta es procesada por el componente, el cual puede realizar otra solicitud al servidor web o a un servicio de la aplicación en el servidor. Ver figura 4.5.

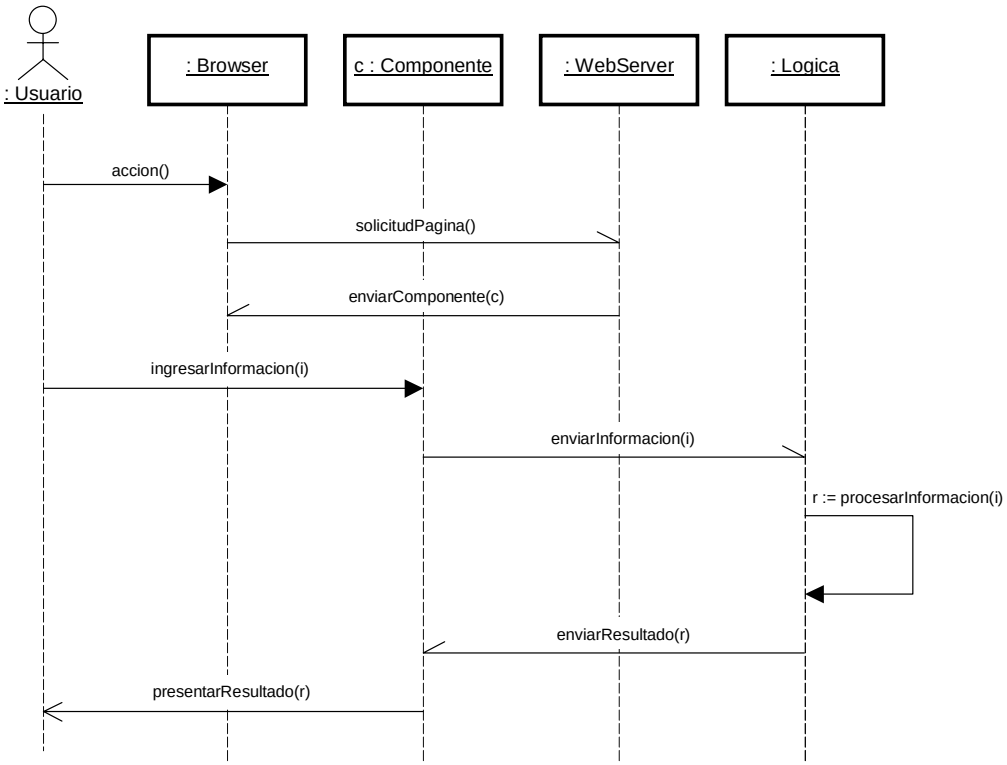


Figura 4.5

Este modelo es muy similar al modelo clásico. Se cambia la aplicación cliente por un cliente universal capaz de ejecutar componentes. En el servidor se cuenta con una aplicación brindando servicios de lógica de negocios, al igual que en el modelo clásico. Los componentes utilizan los mismos mecanismos para conectarse a la aplicación servidor.

La novedad en el modelo es la presencia de un servidor web sirviendo al cliente páginas y componentes embebidos en ellas. Además, se cuenta en el cliente de una aplicación conocida por los usuarios, estándar, y de distribución gratuita.

El servidor web y la aplicación servidor no están comunicados. El cliente utiliza el web server para acceder a los componentes que brindan la interfaz, y accede a la aplicación servidor para utilizar los servicios de lógica de negocios mediante mecanismos diferentes.

Notar que este enfoque presenta una distribución orientada a web, por lo que tiene las características vistas en la sección 3.2.

4.4.2 Tecnologías

Tanto el web server como el web browser son dependientes de la plataforma. Sin embargo, existen distribuciones gratuitas de ambos prácticamente para cualquier plataforma.

Se aplican las mismas consideraciones que en modelo clásico para la aplicación corriendo en el servidor.

Para el desarrollo del componente puede optarse por dos tecnologías similares en funcionalidad, de diferentes proveedores; estas son ActiveX (Microsoft) y Java Applet (Sun).

La elección de la tecnología en el cliente puede determinar el cliente universal (Netscape Navigator no soporta ActiveX), la plataforma del cliente (ActiveX corre sobre plataforma Windows), y el middleware a utilizar entre el componente y la aplicación en el servidor (a no ser que se utilicen *bridges* que enmascaren una tecnología sobre otra).
Dependiendo de la tecnología utilizada la aplicación servidor deberá residir en el mismo nodo que el servidor web. Esto se debe a restricciones en las tecnologías que limitan al componente alojado en el cliente a comunicarse únicamente con el servidor desde el cual fue descargado. Otras limitaciones incluyen acceso a disco del cliente.

4.4.3 Middleware

Este modelo utiliza dos tipos de middleware diferentes. Uno conecta el cliente universal con el servidor web, y el otro conecta el componente con la aplicación en el servidor.

El primero es el protocolo HTTP (HyperText Transfer Protocol), estándar del web actual, en uso en Internet e intranets.

Para el segundo tipo de middleware, puede encontrar información en la subsección 4.3.3.

4.4.4 Cualidades

Legibilidad Directamente proporcional al buen estilo de programación aplicado.

Mantenibilidad La utilización de sockets hace que la implementación sea difícil de mantener. Utilizar ORB puede simplificar esta tarea. Un cambio, en particular en la presentación, implica un deployment.

Extensibilidad Si se utiliza ORBs no hay problema ya que es una invocación a un nuevo objeto; en cambio, si se utiliza sockets, implica una modificación en el protocolo entre los componentes.
Una extensión implica un deployment.

Deployment El mismo es relativamente automático, aunque dependerá de la tecnología.
Aquí se cuenta con la ventaja de que si solamente uno de los componentes fue actualizado, solamente se requerirá el deployment del mismo, y no de toda la aplicación.
Puede tenerse problema de versiones en los componentes en la misma estación y

	deployment parciales puede llevar a tener estaciones de trabajo con varias versiones de atraso. Según este modelo el desarrollo se realiza exclusivamente del lado del servidor y es necesario visitar las estaciones de trabajo solamente cuando se desee actualizar el cliente universal.
Carga de la red	Este modelo tiene algo más de carga en la red que el clásico. Cuando el cliente se conecta al servidor, como parte de la interfaz en HTML viaja también el control, aunque únicamente la vez que es descargado ya que en sucesivos pedidos se empleará la versión que quedó almacenada en el cliente. Al igual que en el modelo clásico, deben tenerse las mismas consideraciones de diseño para economizar solicitudes ORB.
Tiempo de respuesta	En este modelo se emplean dos servidores. La mayoría de los servidores web, que sirven la presentación, son capaces de atender miles de solicitudes en paralelo. Sin embargo el otro servidor (la aplicación), que sirve la lógica, es análogo a la aplicación servidor del modelo clásico. Notar que es éste último quien recibirá la mayor parte del flujo de solicitudes. Para obtener la interfaz, no sólo es necesario descargar la página HTML correspondiente, sino que además hay que obtener el control, ya sea descargándolo desde el servidor o cargándolo desde el disco local.
Tiempo de desarrollo	Este modelo se ve favorecido por no tener que implementar una aplicación que controle a *toda* la interfaz gráfica, ya que esta está formada por un conjunto de componentes. El desarrollo de los componentes de la interfaz gráfica puede resultar complicado en algunos lenguajes. Depende además de las virtudes del IDE utilizado.
Portabilidad	El middleware determina si se puede o no intercambiar el desarrollo de la aplicación hecho en un lenguaje por otro. El middleware y el lenguaje puede determinar la plataforma sobre la cual correrá la aplicación. La tecnología utilizada en el componente puede determinar la plataforma y el cliente universal.
Escalabilidad a nuevos clientes	Depende de las limitaciones que presente el servidor de la lógica. Puede implicar que el cliente este restringido a utilizar cierto cliente universal, o que tenga un equipo de potencia considerable. Implica además realizar deployment hacia los nuevos clientes.
Robustez	Las consideraciones respecto a la robustez del modelo, en términos de la comunicación del componente con la aplicación servidora, son análogas a las del modelo clásico. El punto débil del modelo es la ejecución del componente en el entorno brindado por el cliente universal. Si la tecnología del componente y el cliente universal provienen del mismo fabricante se puede esperar el comportamiento deseado, no necesariamente así en caso contrario.
Requerimientos	Requiere de un equipo potente en el cliente capaz de correr cliente universal con un componente gráfico embebido. Requiere un servidor web y módulos para utilizar el middleware elegido.

4.5 Modelo de Contenido Dinámico

4.5.1 Descripción del modelo

Este modelo consiste de una aplicación conocida como cliente universal (web browser), que reside en el cliente, un servidor web que reside en el servidor.

El funcionamiento del modelo es el siguiente:

El cliente solicita una página del servidor web. El servidor web recibe esta solicitud, y envía la página con un formulario (HTML Form) embebido al cliente. El browser presenta la página y el formulario, recoge del usuario la información y la envía al servidor web, quien la recibe. El servidor web reenvía la información al servidor de lógica quien la procesa y devuelve los resultados. El servidor web genera una página dinámicamente con la información recibida y se la envía al cliente, quien la presenta al usuario. Ver figura 4.6.

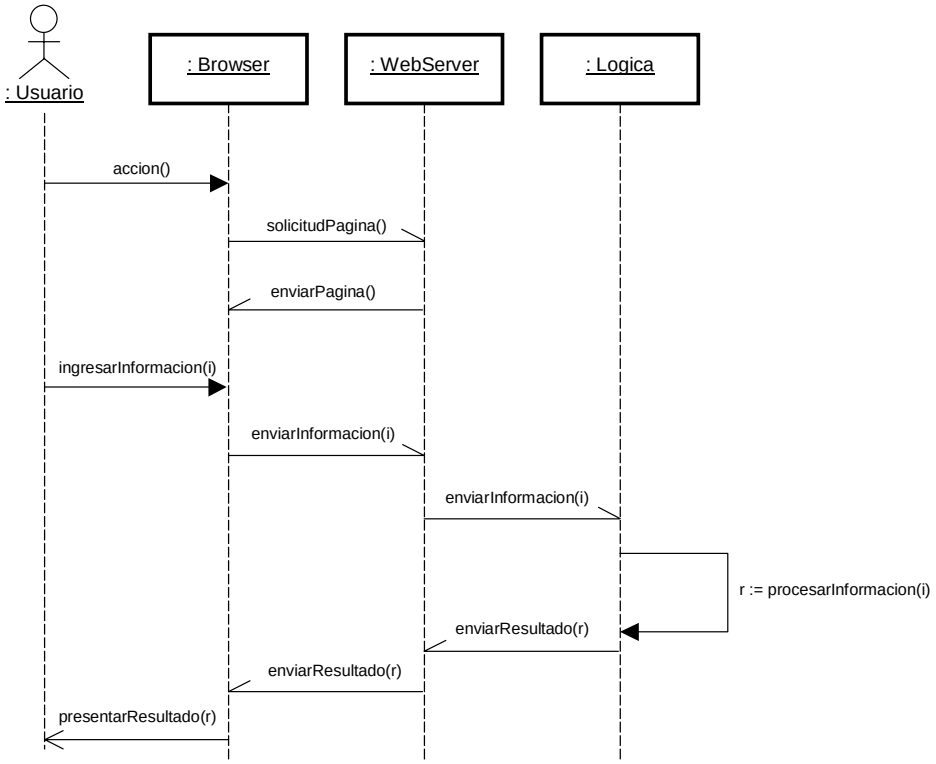


Figura 4.6

Este modelo difiere de los modelos anteriores en que el cliente no se comunica directamente con el servidor de lógica. La parte de la presentación que reside en el servidor web se comunica con la lógica a través de la ejecución de aplicaciones (CGI – Common Gateway Interface) o instanciando objetos en el entorno de la página generada, los cuales se comunicarán con la lógica utilizando ORB.
Además, a diferencia del modelo de componentes gráficos, no depende de la capacidad del browser de ejecutar componentes.

La información que llega al cliente puede ser HTML y Forms. En los casos en que la capacidad de HTML no sea suficiente, ya sea por altos requerimientos de manejo gráfico, o por la complejidad de alguna interfaz de usuario, podría utilizarse componentes. Estos, en este caso, son utilizados con fines

de resolver problemas de interfaz, y aunque no contienen lógica de negocios, si deben comunicarse con tal servicio según el modelo de componentes gráficos.

Notar que este enfoque presenta una distribución orientada a web, por lo que tiene las características vistas en la sección 3.2.

4.5.2 Tecnologías

Tanto el web server como el web browser son dependientes de la plataforma. Sin embargo, existen distribuciones gratuitas de ambos prácticamente para cualquier plataforma.

A nivel de cliente no hay tecnologías especiales a utilizar. Las virtudes de los web browser actuales son suficientes para este modelo, ya que su única función es interpretar HTML.

En cambio el servidor web debe ser más potente que el del modelo de componentes gráficos. Debe soportar tecnologías cuya función es generar contenido dinámico a ser enviado al cliente.

Las tecnologías varían en su funcionamiento. El primer enfoque utilizado fue CGI. La evolución del web llevó a servidores más potentes, capaces de ser extendidos para lograr funcionalidades específicas. Surgieron entonces nuevos enfoques como Servlets, JSP (JavaServer Pages) y ASP (Active Server Pages).

4.5.3 Middleware

Este modelo utiliza únicamente HTTP como protocolo. El mismo se utiliza para conectar el cliente universal con el servidor web.

Una consideración de interés es que la parte de presentación que reside en el servidor web debe conectarse, por algún medio, al componente lógico de la aplicación. Esta comunicación es mediante un middleware, pero interno al servidor. Esto es, no refiere al middleware que conecta al cliente con el servidor, sino el que conecta componentes diferentes a nivel de servidor.

Puede adoptarse dos enfoques diferentes para resolver esta comunicación. Un primer enfoque es que el servidor web utilice el mismo middleware que se indicó para el modelo clásico. Un segundo enfoque, más sencillo en la implementación, es utilizar el componente de la lógica en el propio servidor web; aquí el servidor web instancia objetos del componente lógico para resolver las solicitudes.

4.5.4 Cualidades

Legibilidad El buen estilo de programación no garantiza la legibilidad. Algunas tecnologías permiten definir el documento original que contiene tanto información estática como los mecanismos de generación de contenido dinámico. El mismo en interpretado o precompilado. De todas formas, la legibilidad se dificulta ya que un mismo documento tiene información de interés tanto para el diseñador (de la página) como para el programador.

Mantenibilidad La falta de legibilidad hace que el mantenimiento se dificulte.
Por otro lado, por cada interfaz desarrollada se tiene un documento diferente, lo que facilita la localización.
Un cambio no implica un deployment de componentes, ya que la información de la

	generación de la interfaz *viaja* en cada solicitud.
Extensibilidad	Agregar nuevas funcionalidades implica desarrollar un nuevo documento (o aplicación) con las indicaciones de cómo generar el contenido, y publicarlo en el servidor.
Deployment	No hay deployment como tal. En cada solicitud del cliente, el servidor devuelve la información de cómo generar la interfaz adecuada. Notar que este contenido es significativamente liviano, ya que lo que viaja es la información (en HTML) de cómo generarlo, no la interfaz generada. Según este modelo el desarrollo se realiza exclusivamente del lado del servidor y es necesario visitar las estaciones de trabajo solamente cuando se desee actualizar el cliente universal. A diferencia del modelo de componentes gráficos, asegura que las estaciones de trabajo tienen siempre la última versión de la interfaz.
Carga de la red	Presenta una carga extra ya que en cada transferencia se envía, además de los datos, la información de generación de la interfaz. Sin embargo no se utiliza ORB que es más pesado que el protocolo HTTP.
Tiempo de respuesta	El tiempo de respuesta de los servidores web es aceptable. Sin embargo, el servidor web debe conectarse con la lógica a través de los mecanismos mencionados antes. Esta comunicación puede hacerse por ORB mediante invocación remota (como sucede en los casos anteriores) o mediante invocación local, haciendo coincidir al servidor web y al componente de lógica en el mismo nodo. Una mejora en la performace puede obtenerse embebiendo la lógica en el servidor web, en cuyo caso no se utiliza ORB y la invocación es directa.
Tiempo de desarrollo	Al igual que el modelo de componentes gráficos, éste se ve favorecido por no tener que implementar una aplicación que controle toda la interfaz gráfica. El desarrollo de la interfaz gráfica se realiza en forma sencilla utilizando un editor HTML con manejo de Forms. Sin embargo, en algunos casos es dificultoso el desarrollo de la parte de la presentación que genera el contenido dinámico. Usualmente esta implementación va embebida en el archivo HTML (el cual recibe otro nombre: ASP, JSP, etc.). Para alguna de las tecnologías existen IDEs adecuados, lo que facilita el desarrollo. Actualmente están apareciendo en el mercado IDEs para desarrollar utilizando las otras tecnologías.
Portabilidad	A nivel de cliente es totalmente portable. Puede optarse por cualquier cliente universal y cualquier plataforma. A nivel de servidor web, no todos los servidores soportan todas las tecnologías. Algunos están optimizados para una tecnología en particular. Actualmente existen en el mercado productos de terceros fabricantes que permiten utilizar ciertas tecnologías en servidores que no la soportan nativamente.
Escalabilidad a nuevos clientes	Un servidor web tradicional es capaz de soportar miles de clientes sin sufrir baja de performance. No implica realizar deployment ni configuraciones especiales en los nuevos clientes, ni en el servidor. Notar, sin embargo, que la performance puede bajar por aumentar la cantidad de solicitudes que realiza el servidor web al servidor de lógica.

Robustez

Las consideraciones de robustez del modelo clásico se aplican en este caso a la comunicación entre el servidor web y el servidor de lógica.
Con respecto a la comunicación entre el cliente y el servidor web, utiliza un protocolo de uso masivo optimizado para tal fin.
El cliente universal y el servidor web son productos utilizados por miles de compañías para brindar y acceder a servicios, lo que hace que el modelo sea confiable en este aspecto.

Requerimientos

Requiere de un servidor web capaz de generar contenido dinámico.

No hay requerimientos específicos para los clientes, ya que se necesita solamente un cliente universal interpretando HTML.

Capítulo 5
Tecnologías Aplicables

5.1 Introducción

Este capítulo tiene por objetivo presentar las características de algunas de las tecnologías aplicables a cada uno de los modelos tratados en el capítulo anterior. Se discute diferentes tipos de middleware, se describen los lenguajes de programación más utilizados para el desarrollo de aplicaciones cliente/servidor y en capas. Se analiza dos tipos de componentes gráficos como son los controles ActiveX y los Applets. Por último se trata el lenguaje HTML como medio de presentación de información estática y sus formularios, y las tecnologías de generación dinámica de información como ser CGI Script, Servlets, JSP y ASP. El final del capítulo contiene comparativos entre estas tecnologías.

5.2 Middleware

Una de las dificultades que presenta el desarrollo de una aplicación distribuida es atravesar la barrera que representa la red que comunica a sus componentes.

5.2.1 Sockets

Esta barrera puede ser cruzada a través de RPCs y programación directa sobre la red. El uso de RPC presenta como desventaja que no es orientado a objetos. Por otra parte, la programación directa sobre la red se realiza usualmente en base a sockets, que es una interfaz de comunicaciones de red de bajo nivel. Las aplicaciones basadas en sockets son costosas de desarrollar y modificar.

5.2.2 Object Request Broker

Otra alternativa de comunicación es utilizar ORBs. ORB actúa como un middleware entre un objeto que necesita un servicio de otro en una maquina diferente. En lugar de escribir código de socket para el cliente y el servidor, simplemente se deja que el ORB se encargue de las tareas de comunicación por la red. Esto representa un recurso de más alto nivel que los sockets.

Actualmente existen diferentes variedades de ORBs con diferentes niveles de compatibilidad con plataformas y lenguajes. Estos son CORBA, propuesto por la OMG; RMI de Sun y DCOM de Microsoft. La tabla 5.1 resume las compatibilidades de estos ORBs.

DCOM conecta dos objetos, escritos en cualquier lenguaje, siempre y cuando ambos estén corriendo sobre plataforma Windows.

RMI conecta dos objetos en forma independiente a la plataforma, pero condiciona la implementación a utilizar el lenguaje Java.

CORBA es el más general de los tres ya que conecta objetos implementados en cualquier lenguaje corriendo sobre cualquier plataforma.

	Red	Plataforma	Lenguaje
DCOM	✓	×	✓
RMI	✓	✓	×
CORBA	✓	✓	✓

Tabla 5.1

5.2.3 HyperText Transfer Protocol

Es el protocolo subyacente al web. Define la forma de transmisión y el formato de los mensajes, y las acciones que deben realizar los browsers y servidores web ante cada comando. Por ejemplo, cuando se ingresa una URL (Uniform Resource Locator) en un cliente web, este envía un comando HTTP al servidor web indicándole que obtenga y transmita la página web solicitada.
El protocolo HTTP no presenta estado, ya que cada comando es ejecutado independientemente. Esta es la principal razón de la dificultad de implementar sitios web que reaccionen inteligentemente ante la entrada del usuario. Este inconveniente de HTTP esta siendo solucionado con un número de tecnologías incluyendo componentes gráficos y generadores de contenido dinámico (excepto CGI).
Actualmente tanto los clientes web como los servidores soportan la versión 1.1 de HTTP. Una de las principales características de esta versión es que soporta conexiones persistentes. Esto significa que una vez que el cliente web se conecta al servidor web puede recibir múltiples archivos a través de la misma conexión. Esto mejora la performance en un 20% respecto de la versión anterior.

5.3 Lenguajes de Programación

5.3.1 Lenguaje C++

C++ es un lenguaje de programación de propósito general creado por Bjarne Stroustroup en base al lenguaje de programación C.
Es un lenguaje muy potente aunque es posible introducir bugs con facilidad. Es muy difundido por estar basado en el lenguaje C. Existen compiladores de este lenguaje para casi todas las plataformas.

Sus principales características son:

- Es un lenguaje orientado a objetos, tiene herencia simple y múltiple, pero no soporta interfaces.
- Permite un manejo de muy bajo nivel de la memoria mediante aritmética de punteros.
- Permite polimorfismo con binding estático y dinámico.

5.3.2 Lenguaje Visual Basic

Visual Basic es un lenguaje que evolucionó del Basic original y ahora contiene una gran variedad de construcciones, funciones y métodos, muchos relacionados con la Windows GUI.
Provee un set completo de herramientas para simplificar y acelerar el desarrollo de aplicaciones. Por la amigabilidad de su entorno de desarrollo y por la simplicidad de su sintaxis es muy simple de aprender.

Las características principales del mismo son:

- Es orientado a objetos, soporta interfaces pero no tiene herencia.
- El polimorfismo se da a través de las interfaces y tiene binding dinámico.
- El lenguaje y su entorno de desarrollo están pensados para el desarrollo de aplicaciones basadas en GUI.

5.3.3 Lenguaje Java

Java es un lenguaje que fue desarrollado en 1991 por un grupo de ingenieros de Sun Microsystems. Está basado en C++ pero modifica ciertas características que lo tornan más manejable que su inspirador. Utiliza garbage collector y elimina por completo el manejo explícito de la memoria. No permite la herencia múltiple, pero a cambio introduce el concepto de interfaz.
Es un lenguaje híbrido que se compila a bytecode, el cual es interpretado por una máquina virtual. Actualmente existen implementaciones de esta máquina para las principales plataformas.
Se distribuye gratuitamente con un conjunto de paquetes con una gran y rica gama de utilidades.

Algunas de sus ventajas son las siguientes:

- Independencia de la plataforma. Eso es recomendable para desarrollar aplicaciones para Internet.
- Tiene sintaxis similar a la de C++. Es una sintaxis económica sin ser absurda.
- Es completamente Orientado a Objetos. Aún más que C++, porque en Java todos son objetos, salvo algunos tipos básicos.
- Es más fácil el debugging y crear código libre de errores que en C++.

Java es un lenguaje simple, distribuido, independiente de la arquitectura y con soporte para hilos múltiples, lo que hace de Java un lenguaje muy potente.

5.3.4 Lenguaje Perl

Es un lenguaje de programación de alto nivel escrito por Larry Wall. Deriva del lenguaje de programación C y en un grado inferior de SED, AWK, el shell de Unix y de otras herramientas y lenguajes.

Las principales características de este lenguaje son:

- Es un lenguaje interpretado.
- Brinda facilidades para el manejo de archivos y de texto.
- Tiene un fácil acceso a bases de datos.
- Orientado a objetos, soporta herencia simple y múltiple.
- Utiliza garbage collector

La mayoría de las aplicaciones CGI involucran la manipulación de datos accediendo a aplicaciones externas. Perl provee herramientas simples para realizar estas tareas.

5.4 Componentes Gráficos

Las tecnologías de controles ActiveX y Applets de Java son análogas en su concepción y funcionamiento.

Se estudiará a continuación los detalles de las mismas.

5.4.1 Controles ActiveX

5.4.1.1 Definiciones técnicas

Control
Es un objeto en una ventana o caja de diálogo.
Ejemplo de controles son botones, barras de desplazamiento, cajas de texto, etc.

Tecnología ActiveX
Es un conjunto de tecnologías que permite integrar componentes de software en un entorno de red, más allá del lenguaje en que fueron creados. Esta integración de componentes permite a los desarrolladores de software y de contenido crear aplicaciones interactivas y sitios web.

Control ActiveX
Un control ActiveX es un componente de software similar a un control Visual Basic, desarrollado en cualquier lenguaje que soporte tecnología ActiveX, que puede ser integrado en páginas web y ejecutado en un cliente web.
Un control ActiveX permite interacción con los lenguajes de scripting, como VBScript, JScript y JavaScript, que se utilicen en la página en la que el control fué descargado.
Un control ActiveX ejecutando en un cliente web es equivalente a una aplicación, por ejemplo Visual Basic, ejecutando en su propia ventana. La diferencia radica principalmente en que el control es de menor tamaño que la aplicación y no requiere de instalación manual en la terminal, ya que puede ser descargado desde un servidor web e instalado automáticamente.

5.4.1.2 Funcionamiento

Los controles ActiveX son comúnmente desarrollados en Visual Basic. Pueden ser utilizados como componentes de otras aplicaciones o en una página web. En este caso, tanto la página escrita en HTML puro como el control residen en el servidor web a la espera de una solicitud. Cuando un cliente solicita la página, ésta es descargada junto con el control hacia la máquina del cliente. Esto se produce porque la página contiene mínimamente el siguiente fragmento de código HTML:

```
<object
      id="nombre_control"
      classid="CLSID:nnnn"
      codebase="control.CAB#version=x,x,x,x">
</object>
```

donde el atributo `codebase` indica el paquete donde obtener el control, junto a la versión del mismo. De esta manera el cliente al interpretar el precedente código solicita el control al servidor. Una vez descargado, éste es instalado en la terminal del cliente, para ser ejecutado luego por el cliente web.

El control ActiveX una vez en ejecución, puede establecer una conexión de red (vía sockets u ORB) con otra máquina, interactuar con scripts incluidos en la página ya sea invocándolos o dejando visible sus propiedades y lanzar aplicaciones que ejecuten en la máquina del cliente. Además, un control por poder ser entendido como una aplicación, puede contener lógica de negocios e inclusive aplicar toda la potencia gráfica que el lenguaje utilizado para su implementación lo permita. En ciertos tipos de aplicaciones, este último uso es muy frecuente.

Cuando un cliente vuelve a solicitar una página que contiene un control que ya fué descargado, éste no se vuelve a descargar, sino que es utilizado el que ya se encuentra en la máquina del cliente. Este funcionamiento ocurre exceptuando el caso en que el servidor contiene una versión del control posterior al que se encuentra en el cliente (el cliente detecta este caso examinando el tag `<object>`), en cuyo caso se efectúa la descarga.

5.4.2 Java Applet

5.4.2.1 Definiciones técnicas

Un applet es una instancia de una clase llamada Applet contenida en el JDK de Java. Si bien su anatomía no es igual a la de una aplicación Java, es análoga. Esto quiere decir que un applet puede ser entendido como una aplicación Java que fue concebida para ser descargada a través del web e interpretada por la máquina virtual de un cliente web. En general los bytecodes de Java son reducidos en tamaño, por lo que no fueron necesarias consideraciones especiales por el hecho de la transferencia a través de la red.
Al igual que los controles ActiveX, un applet puede interoperar con lenguajes de scripting.

5.4.2.2 Funcionamiento

El funcionamiento de los applets es análogo al de los controles ActiveX. Es descargado a un cliente como consecuencia de la solicitud de una página HTML. Esta página contiene un código similar al siguiente:

```
<applet
      code="Applet.class"
      width=nnnn
      height=mmmm>
</applet>
```

donde el atributo `code` indica el nombre del archivo bytecode que contiene el applet que se desea descargar, y los otros dos el tamaño que ocupará la interfaz del applet en la del cliente web.

El applet una vez en ejecución puede establecer una conexión de red, pero exclusivamente con la máquina que aloja al servidor web desde donde fue descargado. Al igual que los controles ActiveX, puede interactuar con scripts, pero no puede lanzar aplicaciones que ejecuten en el cliente, ni tampoco acceder el sistema de archivos local.
Como fue dicho, un applet es un objeto pero puede requerir de otros objetos. Estos objetos pueden ser descargados también desde el servidor web o pueden ser parte del entorno de ejecución Java local. Un applet por razones de seguridad no puede cargar código que no sea Java. Por otra parte un applet también puede desplegar toda la potencia de Java para presentar interfaces gráficas muy elaboradas o inclusive para realizar lógica de negocios.

El mecanismo de descarga es también análogo al de los controles ActiveX. Una vez descargado, un applet queda almacenado en el equipo local y no es necesario volverlo a descargar.

5.4.3 Consideraciones sobre estas tecnologías

Como se dijo antes ambas tecnologías son similares.

Difieren, sin embargo, en que un control ActiveX tiene libre acceso a los recursos del sistema, mientras que el applet tiene acceso restringido. Para resolver este problema Microsoft desarrolló un sistema de registro para que los web browser puedan verificar la autenticidad de los controles antes de bajarlos y ejecutarlos. Este mecanismo no es robusto y es factible de ser víctima de hacking.

Otra diferencia es que los applets pueden ejecutar en cualquier plataforma, ya que todos los web browser cuentan con su propia máquina virtual de Java. En cambio, los controles ActiveX están limitados al entorno Windows.

La potencia de los applets es la que brinda el lenguaje Java, mientras que la potencia de un control ActiveX depende del lenguaje utilizado para la implementación.

Como última consideración, la experiencia indica que no todos los web browser despliegan los applet de la misma forma, por lo que el desarrollador no puede asegurar que la interfaz diseñada se presente de la misma forma en todas los clientes.

5.5 Contenido Estático

Originalmente el web se utilizaba para mostrar información estática. Esta información viajaba a través de la red utilizando un protocolo específico (HTTP). La información presentaba contenido formateado; tal formato era especificado en el propio documento utilizando un lenguaje llamado HTML.

La necesidad de interacción con el usuario llevo a mejorar tal lenguaje, soportando algún mecanismo de entrada de información. Los formularios HTML son la respuesta a tal problema.

5.5.1 HyperText Markup Langauge

HTML es el lenguaje universal para la publicación de hipertexto en el World Wide Web. Tiene formato no-propietario basado en SGML, aunque no es necesariamente un subconjunto estricto de éste. Puede ser creado y procesado por una amplia gama de herramientas, desde editores de texto simples a complicadas herramientas de edición WYSIWYG. HTML usa tags para estructurar el texto en encabezados, párrafos, listas, enlaces, etc.

Se entiende por *tag* un comando insertado en un documento para especificar como el documento, o una sección del mismo, debe ser formateado.

HTML 4.0, creada por la organización W3C (World Wide Web Community), se transformó en recomendación en diciembre de 1997. Una revisión (HTML 4.01) fue publicada en diciembre de 1999.

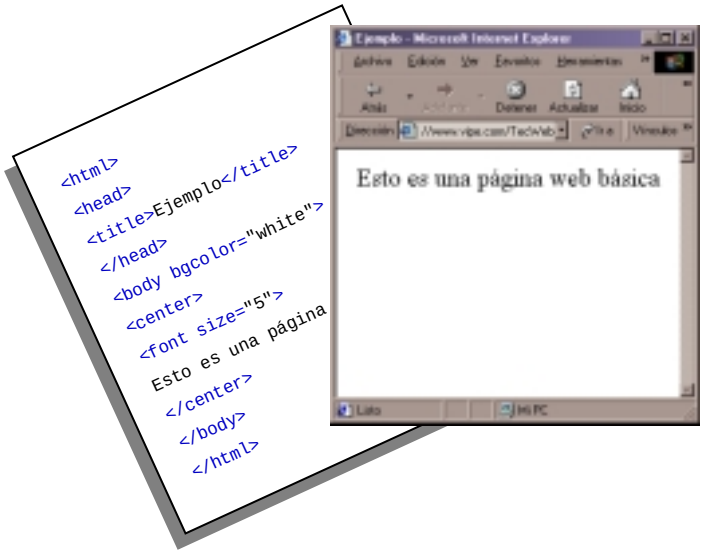


Figura 5.1

5.5.2 Formularios

Los formularios de HTML son una de las formas posibles de permitir una interacción entre un usuario y el servidor web. Estos formularios permiten visualizar los widgets más comunes en las interfaces de usuario de los lenguajes de programación.

Un usuario completa el formulario modificando los widgets, esto es, ingresando texto, seleccionando ítem de menús, etc., para luego enviar la información ingresada a un servidor web para su proceso.

El formulario esta definido en términos de tags de HTML y la interpretación de éstos es responsabilidad del cliente web, quien muestra los widgets correspondientes a cada tag.

La figura 5.2 muestra un formulario visualizado en un cliente web y la figura 5.3 muestra el código fuente (en HTML) de la página web que contiene el formulario.

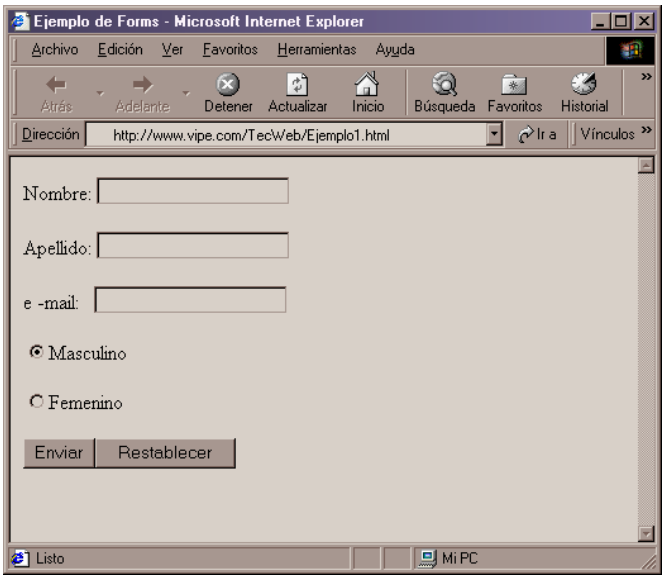


Figura 5.2

```
<html>
<head>
<meta http-equiv="Content-Language" content="es">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Pagina nueva 1</title>
</head>
<body>
<form method="POST" action="--WEBBOT-SELF--">
  <p>Nombre: <input type="text" name="T1" size="20"></p>
  <p>Apellido: <input type="text" name="T2" size="20"></p>
  <p>e -mail:   <input type="text" name="T3" size="20"></p>
  <p><input type="radio" value="V1" checked name="R1">Masculino</p>
  <p><input type="radio" name="R1" value="V2">Femenino</p>
  <p><input type="submit" value="Enviar" name="B1">
     <input type="reset" value="Restablecer" name="B2">
  </p>
</form>
</body>
</html>
```

Figura 5.3

5.6 Generación de contenido dinámico

CGI surgió como el mecanismo para presentar información generada dinámicamente en el web. Permite al servidor web generar páginas web instantáneamente al momento en que el usuario lo requiere en lugar de ser escritas de antemano por alguien.

La implementación de la interfaz CGI es la parte del servidor web que puede comunicarse con otros programas que corren en el servidor. Con CGI el servidor web puede lanzar un programa pasándole datos específicos del usuario.

CGI significa:

Common: Asegura que CGI puede ser utilizado por muchos lenguajes e interactuar con diversos tipos de sistemas. No ata al desarrollador a una forma de hacer lo que se desea.

Gateway: Sugiere que el poder de CGI está no en lo que hace por si mismo, sino en el acceso potencial a otros sistemas tales como base de datos, generadores de gráficos, etc.

Interface: Significa simplemente que CGI proporciona una manera bien definida de utilizar sus características, es decir, que se puede escribir programas que la utilizan.

Un primer enfoque es CGI Script donde el servidor web dispara una aplicación, escrita en cierto lenguaje de programación (por ejemplo Perl, Visual Basic, C, etc.) la que se encarga de la generación del contenido. Esta aplicación se ejecuta, procesando la entrada obtenida del servidor web. En su ejecución genera el contenido completo de la página web que será enviada por el servidor web al cliente.

Un segundo enfoque es servlets. Los mismos son una extensión del servidor web capaces de atender solicitudes de los clientes web. Son desarrollados en lenguaje Java pudiendo explotar todas las herramientas que este provee, acceso a base de datos a través de JDBC, uso de socket, RMI, etc.

JSP es la última tecnología Java para desarrollar aplicaciones web, basada en la tecnología servlet. Los servlets son muy buenos en muchos aspectos, pero están reservados generalmente para programadores. JSP es considerado como una forma más barata de programar servlets de Java. Diseñadores pueden trabajar directo sobre estos documentos ya que gran parte de ellos están escritos en HTML.

ASP es la propuesta de Microsoft para la generación de contenido dinámico. ASP y JSP son soluciones análogas en términos de comportamiento, aunque difieren en su funcionamiento interno.

5.6.1 CGI Script

5.6.1.1 Definiciones técnicas

Un CGI Script es una aplicación escrita generalmente en un lenguaje de scripting, aunque es posible usar lenguajes compilados también. Esta aplicación es lanzada por el servidor web recibiendo la información provista por el usuario en un formato que respeta la interfaz CGI (de ahí el nombre). Esta aplicación ejecuta en un proceso separado al del servidor web.

Se puede utilizar cualquier lenguaje para desarrollar aplicaciones CGI. Depende siempre de la plataforma donde vaya a ejecutarse el servidor web. Es recomendable que el lenguaje de programación tenga facilidad para manipulación de texto, pueda interactuar con otras bibliotecas y utilidades, y pueda acceder a variables de ambiente.

5.6.1.2 Funcionamiento

El cliente solicita la página web con un formulario e ingresa la información en el mismo. La información es enviada al servidor web quien dispara una aplicación CGI, encargada de procesar esa información y generar la página web resultado. Esta página es enviada por el servidor web al cliente web. Ver figura 5.4.

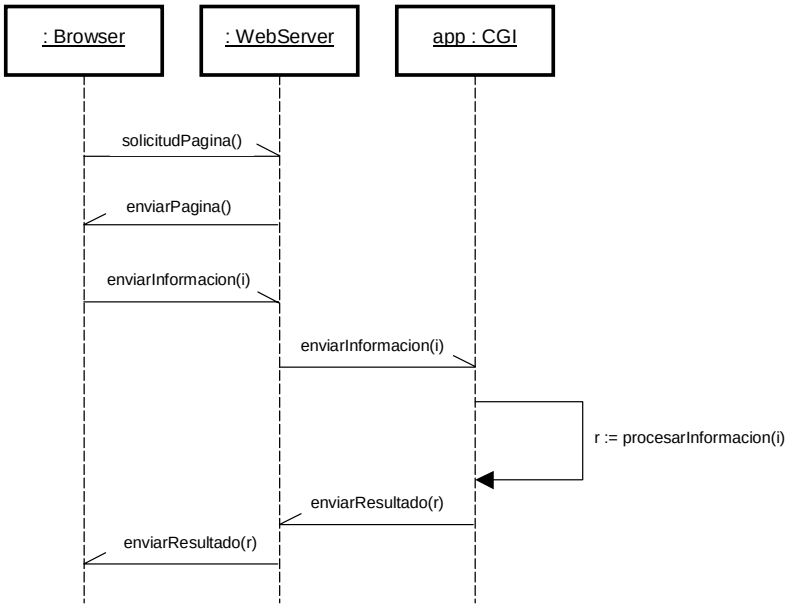


Figura 5.4

Como se indicó antes el protocolo HTTP no presenta estado. El enfoque brindado por CGI Scripts no agrega esta funcionalidad. Por lo tanto no puede guardar a nivel del servidor el estado de la interacción con el cliente. Para resolver este problema, guarda el estado a nivel de cliente. La página que causa la invocación al programa CGI contiene atributos ocultos (parte de un formulario, pero no visible para el usuario) donde guarda los datos de la sesión.

5.6.2 Servlets

5.6.2.1 Definiciones técnicas

En términos simples un servlet es un objeto que agrega nuevas funcionalidades a un servidor (típicamente a un servidor web).

Un servlet es una clase de Java que usa el API (Application Programming Interface) llamado Servlet. Este API consiste en un conjunto de clases e interfaces que definen métodos que permiten procesar solicitudes HTTP en forma independiente al servidor web.

Así como un servlet se implementa en Java, el desarrollador puede verse tentado a embeber aquí la lógica. Como se fundamentó antes, esto no es recomendable. Se utiliza tecnología JavaBean para implementar la lógica; éstos componentes JavaBean son accedidos desde el servlet para generar el contenido dinámico.

5.6.2.2 Funcionamiento

Cuando un servidor web recibe una solicitud web que deba ser manejada por un servlet primero verifica si una instancia de la clase del servlet existe. Sino, crea una. Esto es conocido como *cargar* un servlet. Luego indica al servlet que procese la solicitud. Una vez que éste ha sido cargado la misma instancia es utilizada para procesar solicitudes sucesivas. Eventualmente el servidor web necesita detener el servlet (shutdown), típicamente cuando el mismo servidor tiene que detenerse. Éste, informa primero al servlet y le da la chance de realizar operaciones de limpieza, por ejemplo cerrar una conexión a una base de datos, sockets o archivos abiertos, etc.

Una solicitud al servlet se traduce en una invocación a un método del mismo donde recibe dos objetos como parámetro: *request* que contiene la información de la solicitud, y *response* donde se carga el contenido de la respuesta, típicamente una página web generada dinámicamente. Ver figura 5.5.

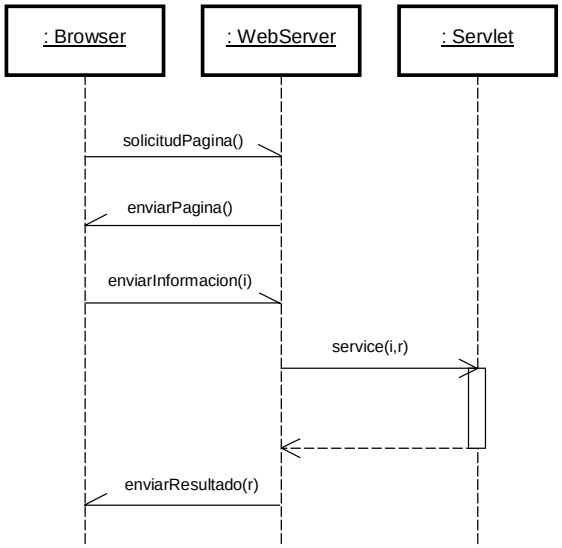


Figura 5.5

5.6.3 JavaServer Pages

5.6.3.1 Definiciones técnicas

JSP es una tecnología para desarrollar páginas web que incluyen contenido dinámico. A diferencia de una página HTML, cuyo contenido es estático, una página JSP puede cambiar su contenido en base a cualquier número de ítems variables. Como por ejemplo la identidad del usuario, el tipo de web browser, información provista por el usuario, y selecciones hechas por el usuario.

Una página JSP contiene elementos de marcado estándar (tags de HTML) al igual que una página web tradicional. Sin embargo contiene además elementos JSP especiales que permiten al servidor insertar contenido dinámico en esa página. Los elementos JSP pueden ser usados para una gran variedad de propósitos como recuperar información de una base de datos, o registrar preferencias del usuario. Estos elementos consisten en fragmentos de código Java puro.

La tecnología JSP funciona como una capa por sobre la tecnología servlet para facilitar el desarrollo de aplicaciones con esta tecnología. En particular, una página JSP es convertida internamente (y automáticamente) a un servlet.

El hecho de tener que utilizar invocaciones a `out.println()` para la generación de cada línea de código HTML se convirtió en un problema para el uso real de servlets. JSP permite extraer del código fuente del servlet el código HTML. El mecanismo mostrado en la figura 5.6 sintetiza esta mejora la cual inspiró la tecnología JSP.

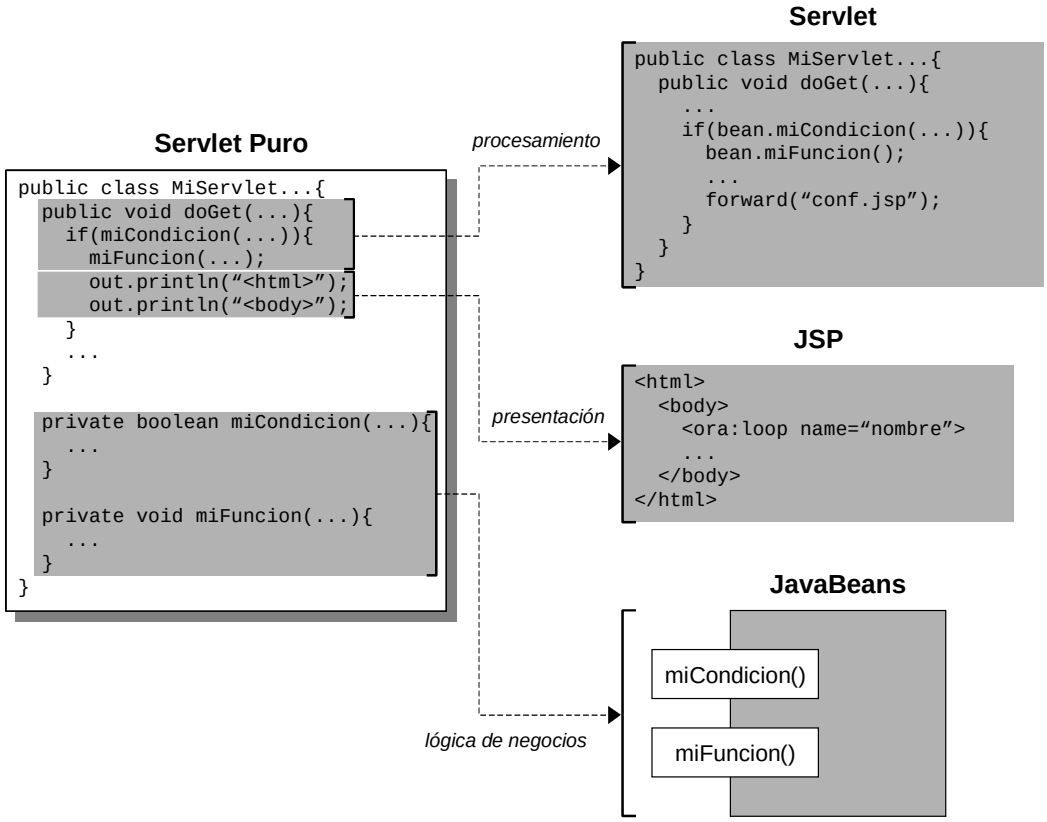


Figura 5.6

Así como en el servlet podía embeberse la lógica en el código del servlet, puede embeberse tal funcionalidad en la página JSP. Como se dijo, esto hace que se pierdan todas las ventajas de una aplicación en 3 capas. Puede utilizarse en JSP, al igual que con servlets, tecnología JavaBean para la lógica. JSP brinda funcionalidades extra para entregar al bean la información ingresada por el usuario.

En la siguiente subsección se trata el procesamiento de una página JSP.

5.6.3.2 Funcionamiento

Una página JSP no puede ser mandada directamente a un web browser. Todos los elementos JSP deben ser procesados primero por el servidor. Esto se hace convirtiendo una página JSP en un servlet, y luego ejecutando el servlet.

Primero llega la solicitud de la página JSP al servidor web. Este accede al archivo JSP local y extrae el código Java generando el fuente del servlet. El servlet es compilado y ejecutado, para luego enviar el resultado al servidor. En el caso de que el servlet de la página JSP ya haya sido construido (generado y compilado), el servidor web solamente ejecuta el servlet. Notar que si se hubiera optado por interpretar el fuente de la página JSP esto degradaría la performance. Ver figura 5.7.

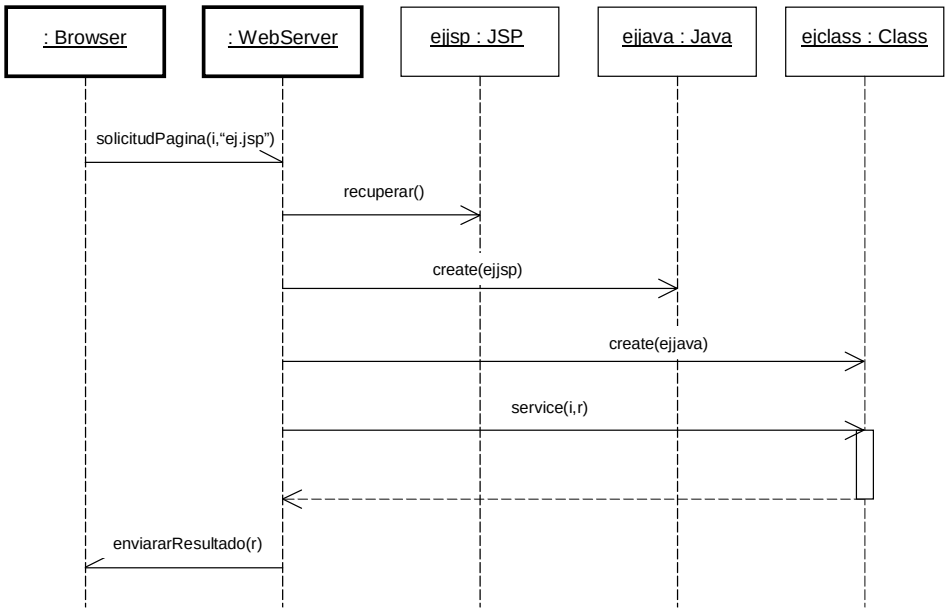


Figura 5.7

5.6.4 Active Server Pages

5.6.4.1 Definiciones técnicas

Es una tecnología de scripting que puede ser utilizada para crear contenido dinámico e interactivo para el web. Una página ASP es una página HTML que contiene scripts que son procesados por el servidor web antes de ser enviada al cliente web.

Es posible extender los scripts de ASP usando componentes COM. El mismo provee un mecanismo para acceder a la lógica. Puede utilizarse estos componentes desde cualquier lenguaje de script que soporten Automation. Los componentes COM brindan funcionalidad equivalente a la que brindan los beans a la tecnología JSP. Los lenguajes utilizados en ASP son lenguajes de scripting como VBScript, derivados de lenguajes de programación, los cuales brindan menor expresividad y funcionalidad.

En resumen, el enfoque de ASP es análogo al de JSP, donde la diferencia radica en el funcionamiento interno. El servidor web debe ser capaz de interpretar el scripting de las páginas ASP en lugar ejecutar una aplicación Java (como es el caso de JSP al generar un servlet).

5.6.4.2 Funcionamiento

El funcionamiento es más sencillo que el de JSP, pagando el precio de interpretar el scripting de las páginas ASP en cada solicitud.

Los scripts en la página ASP son interpretados cuando el web browser realiza la solicitud al web server. El servidor web procesa la página ASP de principio a fin, interpretando los comandos en los scripts. Tal proceso resulta en fragmentos de código HTML el cual es sustituido por el script en la página, resultando código HTML puro. Ver figura 5.8.

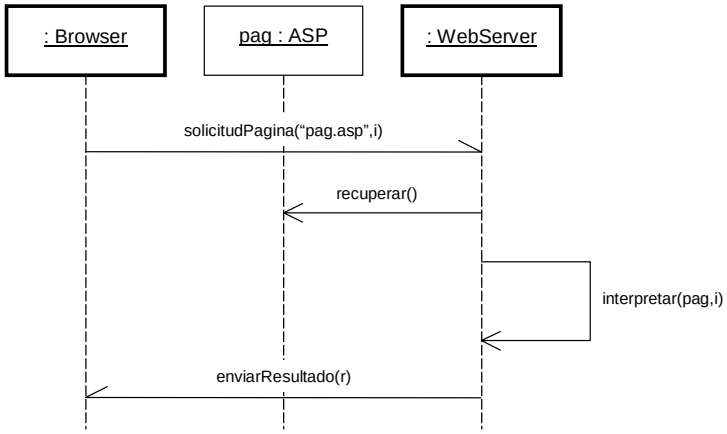


Figura 5.8

5.6.5 Consideraciones sobre estas tecnologías

La presente sección muestra comparativos entre algunas de éstas tecnologías. Todas ellas tienen en común la generación de contenido dinámico, aunque existen diferencias de enfoque. Esta diferencia hace que ciertas comparaciones pierdan interés, ya que o es análoga a la comparación de una tecnología subyacente (ej. CGI vs. JSP), o presenta una filosofía diferente (ej. Servlet vs. ASP).

Estos comparativos fueron tomados directamente de las referencias, y no representan la opinión personal de los autores sobre las tecnologías.

5.6.5.1 CGI Scripts vs. Servlets

En los comienzos de la web, CGI Script era la única herramienta para lograr contenido dinámico. Sin embargo, por cada solicitud el servidor web tiene que crear un nuevo proceso en el sistema operativo, cargar el intérprete y el script, ejecutar el script, y luego liberar los recursos. Esto es demasiado costoso para el servidor y no es escalable a mayor tráfico.

Numerosas alternativas aparecieron en el mercado, tales como mod_perl de Apache, NSAPI de Netscape, ISAPI de Microsoft, Java Servlets de Sun Microsystems.

Dado que Java Servlets fue diseñado para satisfacer "Write Once, Run Anywhere" heredado del lenguaje Java, los servlets pueden ser ejecutados sin modificación en cualquier plataforma.

Servlets presenta varias ventajas sobre CGI Scripts. Como se indicó, CGI Script requiere que un nuevo proceso sea creado por cada solicitud, consumiendo recursos del servidor y aumentando el tiempo de respuesta. Además los scripts CGI son dependientes de la plataforma y no pueden aprovechar las funcionalidades del servidor web, ya que corren en un proceso separado (por ejemplo, no pueden escribir en el log del servidor web). Los servlets son eficientes y escalables porque no requieren de un nuevo proceso para ejecutar, sino que corren en un nuevo hilo en el mismo proceso del servidor.

5.6.5.2 Servlets vs. JSP

Un servlet contiene procesos de solicitudes y acceso a la lógica de negocios (usualmente a través de beans), y también el código HTML de respuesta embebido directamente en el código del servlet a través de llamadas a `out.println()`, como muestra la figura 5.6.

Una aplicación basada en servlets más estructurada aísla las diferentes piezas de procesamiento en varias clases utilitarias reutilizables y puede también utilizar una biblioteca de clases separada para generar los elementos HTML de la respuesta.

Aun así, el enfoque basado en servlets puros tiene los siguientes inconvenientes:

- Se necesita un detallado conocimiento del lenguaje Java para mantener todos los aspectos de la aplicación ya que el código de procesamiento y los elementos de HTML están juntos.
- Cambiar la apariencia de una aplicación o agregar soporte para nuevos tipos de cliente, por ejemplo WML, requiere la actualización y recompilación del código del servlet.
- Es difícil aprovechar herramientas de desarrollo de páginas web al diseñar la interfaz. Si esas herramientas se utilizan para desarrollar la estructura del documento el código HTML generado debe ser luego embebido en el código del servlet. Este proceso consume mucho tiempo, es propenso a errores, y extremadamente tedioso.

JSP no brinda funcionalidades extras sobre servlets pero es más conveniente escribir HTML que cientos de sentencias `println`. Además, separar la apariencia del contenido permite asignar personal específico a tareas específicas. Una página web puede ser diseñada por expertos en diseño, quien dejan espacios que los programadores completarán con contenido dinámico.

5.6.5.3 JSP vs. ASP

Debido a sus similitudes, ninguna de las dos tecnologías es superior a la otra. Ambas tienen sus puntos fuertes y débiles. Ambas son una excelente elección para desarrollar contenido dinámico. Las principales consideraciones para optar entre ASP y JSP depende de los siguientes factores:

- El tamaño del proyecto
- La experticia del equipo de desarrollo
- Dónde y cómo se va a publicar el sitio web
- La cantidad de usuarios concurrentes esperados

Tamaño del proyecto

Para proyectos grandes JSP es claramente superior. Si se usa en forma adecuada, puede asignarse los recursos humanos en forma más sencilla en un proyecto JSP. Parte del equipo se concentra en la construcción de los JavaBeans mientras los otros miembros se concentran en el desarrollo de la presentación. Además, como Java es orientado a objetos, es más fácil separar lógicamente el proyecto y asignarlo a distintos miembros del staff. En proyectos grandes el código de ASP se torna confuso con facilidad, y cuestiones de mantenimiento pueden hacer fracasar el proyecto. JSP es más legible y fácil de mantener en proyectos grandes.

Para proyectos pequeños ASP es preferible. Esto se debe a que el IDE de ASP es simple. Además presenta menos inconvenientes en poner en funcionamiento y mantener el sitio web. La integración de ASP con Microsoft IIS es armoniosa. Por último, el acceso a fuentes de información utilizando ADO (ActiveX Data Object) es simple y eficiente. Esta característica le da potencia para completar rápidamente proyectos de menor porte.

Experticia del equipo de desarrollo

No hay razón particular para descartar ASP por JSP si el equipo de desarrollo presenta experticia en ASP. Si no se cuenta con experticia en ninguna tecnología, es razonable optar por ASP, ya que la curva de aprendizaje lo favorece. Si el equipo presenta experticia en Java, entonces JSP es preferible.

Dónde y cómo se va a publicar el sitio web

Si la publicación del sitio se hará fuera de la empresa ASP es preferible, ya que la oferta de hosting para ASP es ampliamente superior a la de JSP. Dos años atrás la oferta para ASP también era pobre, pero eso cambió rápidamente. Se espera que ocurra lo mismo para JSP.

En el caso de que la publicación se haga en la misma empresa, que es el caso de las intranets por ejemplo, esta limitación no existe. Incluso si el servidor web a utilizar no es IIS, se recomienda JSP. Esto se debe a que JSP es independiente del servidor y ASP está pensado básicamente para funcionar con IIS, a pesar de que existan productos de terceros que resuelvan esta limitación.

Cantidad de usuarios concurrentes esperados

En sitios con baja carga de solicitudes concurrentes, tanto ASP como JSP funcionan adecuadamente. Cuando se trata de configuraciones mayores con alta carga de accesos concurrentes la situación es diferente. JSP tiene mayor capacidad para manejar sitios de gran porte.

Tendencias actuales de uso

Puede medirse la popularidad de las tecnologías según su presencia en los buscadores de Internet. La ventaja de este método es que aproxima de buena manera la distribución en Internet. La desventaja es que las páginas que no son accesibles desde los buscadores no son consideradas. Tampoco se considera el uso corporativo en intranets.

La tabla 5.2 muestra los resultados del test realizado el 23 de febrero de 2001 por los autores del documento.

	JSP	ASP
Altavista	226.436 páginas	15.224.295 páginas
Google	2.020.000 páginas	33.900.000 páginas
Infoseek	57.301 páginas	154.669 páginas
Northern Light	52.893 páginas	5.688.560 páginas
Yahoo	142 páginas	4564 páginas

Tabla 5.2

Según cifras obtenidas del buscador Altavista, relación de aumento en el número de páginas publicadas desde el 12 de octubre de 2000 hasta la fecha de escritura de este informe es de 1:926 para JSP y 1:480 para ASP.

Conclusión

Es de esperarse que el mercado se divida en predilección por ambas tecnologías. Salvando variaciones regionales, a nivel global es previsible un balance entre ambas. JSP se perfila como la alternativa para sitios corporativos, mientras que ASP mantendrá su liderazgo entre sitios de pequeño porte.

Capitulo VI
Aplicaciones Reales

6.1 Introducción

Este capítulo presenta aplicaciones reales de las tecnologías presentadas en el capítulo anterior.

Los ejemplos de aplicaciones reales están categorizados utilizando la misma estructura que el capítulo anterior.

Se presentan aplicaciones reales solamente de las tecnologías de interés.

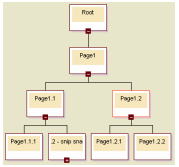
6.2 Componentes Gráficos

El uso principal que se le ha dado a esta tecnología es para explotar la potencialidad gráfica y el poder de conectarse al servidor sin necesidad de solicitar la página nuevamente.

Los siguientes ejemplos presentan aplicaciones actuales utilizando estas tecnologías.

6.2.1 Controles ActiveX

Site Map ActiveX control – http://www.viksoe.dk/sitemap/



Este control muestra un árbol, es capaz de ocultar ramas del mismo. Soporta árboles n-arios y varios niveles de profundidad.

Desarrollado por la empresa

[Información tomada de http://download.cnet.com/downloads/ enero 2001]

Real Player – www.real.com

Control capaz de ejecutar archivos de sonido de RealAudio.

[Información tomada de http://download.cnet.com/downloads/]

6.2.2 Applets

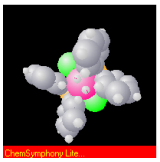
LandRover – www.landrover.com

"Sin dejar sus teclados, los clientes de LandRover pueden ver como lucirá su nuevo vehículo modificándole diferentes propiedades del mismo."

Utilizan un applet para mostrar al cliente vehículos personalizados, según las opciones que elige el cliente.

[Información tomada de http://java.sun.com/features/1997/oct/applets.html
octubre de 1997
Ver 'Aplicaciones reales de JSP' por información actualizada.]

Cruise Molecular Structures – www.cherwell.com



Cherwell Scientific ha desarrollado ChemSymphony, un conjunto de applets para la visualización de moléculas y el procesamiento de datos químicos.

El tiempo de desarrollo fue de aproximadamente 18 meses por O. Krassavine.

[Información tomada de http://java.sun.com/features/1997/oct/applets.html
octubre de 1997
Cherwell Scientific tiene su homepage en http://www.cherwell.com/]

Puede encontrarse otros ejemplos en
http://java.sun.com/features/1997/oct/applets.html
http://java.sun.com/features/1997/dec/applets2.html
http://java.sun.com/features/1998/07/applet.power.iii.html

6.3 Generación de contenido dinámico

6.3.1 CGI Script

Lycos – www.lycos.com

El servidor de búsqueda Lycos permite a los usuarios localizar documentos web. El usuario ingresa la información que busca y el servidor le devuelve una lista de los sitios que probablemente sean de su interés, dado la información que buscaba.

[Información tomada de "CGI Programming on the World Wide Web"]

Coloring Book – www.coloring.com

Permite al usuario elegir un conjunto de imágenes, y colorearlas.

Este es un buen ejemplo de la interacción con el usuario utilizando scripting en el cliente y CGI en el servidor

[Información tomada de "CGI Programming on the World Wide Web"]

6.3.2 Servlets

Marquette Banks – http://www.marquette.com/

Marquette Banks está implementando su producto "Total Web Banking" que permite a los clientes a utilizar todos los servicios a través de Internet

Utilizan en este proyecto tecnología Java Servlet.

[Información tomada de http://java.sun.com/products/servlet/success.html]

Sapient Health Network – http://www.beasys.com/customers/weblogic/server/shn.shtml

Es un servicio de información de salud, gratuito, basado en el web, para personas con enfermedades crónicas y de tratamiento permanente.

Utilizan en este proyecto tecnología Java Servlet.

[Información tomada de http://java.sun.com/products/servlet/success.html]

Puede encontrarse otros ejemplos en:
http://java.sun.com/products/servlet/success.html

6.3.3 JSP

LandRover – www.landrover.com

Utiliza un conjunto de páginas JSP para que el usuario pueda personalizar su nuevo vehículo. Antes utilizaban applets para este servicio.

Utilizan en este proyecto tecnología JSP.

[Información tomada de http://www.landrover.com/]

The Works – http://www.theworksusa.com/

Herramienta de búsqueda escalable de alta performance que provee acceso a más de 400.000 listas de empleo y curriculums de potenciales empleados.
Han licenciado su herramienta de búsqueda, Excalibur RetrievalWare WebExpress, para proveer la información en tiempo real.

Utilizan en este proyecto tecnología JSP.

[Información tomada de
http://industry.java.sun.com/javanews/stories/story2/0,1072,19240,00.html]

Puede encontrarse otros ejemplos en:
http://java.sun.com/products/jsp/success.html

6.3.4 ASP

Unisys – www.unisys.com

Cool ICE es un servidor de aplicaciones de Internet de Unisys. Permite a desarrolladores crear, organizar y administrar aplicaciones y servicios de negocios basadas en generación dinámica de contenido web.

Utilizan tecnología ASP para su proyecto.

[Información tomada de http://www.chilisoft.com/allaboutasp/compapps.htm]

Concur Technologies – http://www.concur.com/solutions/default.stm

Su producto EmployeeDesktop™ automatiza procesos de negocios ineficientes, en función de los empleados, socios de negocios, proveedores y proveedores de servicios. Este producto se integra al suite de aplicaciones para el web y provee un portal de negocios.

Utilizan en este proyecto tecnología ASP.

[Información tomada de http://www.chilisoft.com/allaboutasp/compapps.htm]

Puede encontrarse otros ejemplos en:
http://www.chilisoft.com/allaboutasp/compapps.htm

Referencias Bibliográficas

Arquitectura Cliente/Servidor

- Cliente / Servidor – Guía de supervivencia
  ISBN: 0-471-15325-7
  Robert Orfali – Dan Harkey – Jeri Edwards
  McGraw-Hill
  Segunda edición: 1999

- http: / / www.landfield.com / faqs / client-server-faq /
  "Client / Server Frequently Asked Questions"
  Artículo que da respuesta a preguntas usuales acerca de la arquitectura cliente / servidor

Arquitectura en Capas y Distribución

- http: / / www.fi.infn.it / DFS / docs-osf.org / dce-mag / 9_OH.html
  "2-tier vs. 3-tier, stop the debate"
  Terry M. Olkin
  Artículo que compara las arquitecturas en 2 y 3 capas

- http: / / www.personal.kent.edu / ~jnattey / spag11.htm
  "Types of Client / Server Architecture"
  Joseph O. Nattey
  Artículo que trata posibles arquitecturas en capas para cliente / servidor

Tecnologías

- Programming Web Components
  ISBN: 0-07-912316-3
  Reaz Hoque – Tarun Sharma
  McGraw-Hill
  Primera edición: 1998

- CGI Programming on the World Wide Web
  ISBN: 1-56592-168-2
  Shishir Gundavaram
  O'Reilly & Associates, Inc.
  Primera edición: marzo 1996

- JavaServer Pages
  ISBN: 1-56592-746-X
  Hans Bergsten
  O'Reilly & Associates, Inc.
  Primera edición: enero 2001

- JavaServer Pages Specification
  Version 1.1 – diciembre 1999
  Sun Microsystems.
  Especificación formal de JaveServer Pages

- Tecnologías Java
  Martín Varela – Daniel Perovich
  Introduce los conceptos de orientación a objetos. Presenta el lenguaje de programación Java. Presenta un comparativo con respecto a los lenguajes C++ y Visual Basic.

- http://java.sun.com
  Principal fuente de información en Internet sobre tecnología Java

- http://java.sun.com/servlet
  Home de la tecnología Java Servlet

- http://java.sun.com/applet
  Home de la tecnología Java Applet

- http://java.sun.com/products/jsp
  Home de la tecnología JSP

- http://java.sun.com/products/jdk/1.1/docs/guide/misc/applet.html
  "The APPLET Tag"
  Sun Microsystems – 1996

- http://serverwatch.internet.com/articles/servlets/overview_d.html
  Artículo básico que presenta algunas ventajas de la tecnología JSP con respecto a otras tecnologías de generación de contenido dinámico

- http://www.servlets.com/soapbox/problems-jsp.html
  Artículo que presenta un estudio detallado de los problemas que presenta JSP

- http://www.servlets.com/soapbox/problems-jsp-reaction.html
  Artículo del mismo autor que el anterior, que presenta la reacción de los lectores a lo presentado en el artículo original

- http://www.inforingpress.com/computer_information/activex.htm
  Artículo sobre tecnología ActiveX

- http://members.tripod.com/cuinl/tutorials/ocx-11.htm
  Artículo sobre tecnología ActiveX

- http://msdn.microsoft.com/workshop/server/asp/asptutorial.asp
  "Active Server Pages Tutorial"
  Microsoft Corp. – diciembre 2000

- http://msdn.microsoft.com/workshop/server/asp/ASPover.asp
  "An ASP You Can Grasp: The ABCs of Active Server Pages"
  Microsoft Corp. – mayo 1997

- http://msdn.microsoft.com/workshop/server/asp/comtutorial.asp
  "COM Objects in ASP"
  Wayne Berry – 15 Seconds – enero 1998

- http://java.sun.com/products/jsp/jsp-asp.html
  "Comparing JavaServer PagesTM and Microsoft Active Server PagesTM Technologies"
  Artículo publicado por Sun Microsystems presentando una comparación de las tecnologías JSP y ASP

- http://www.geocities.com/gslender/
  "ASP vs. JSP Test Page"
  Presenta dos casos de estudio implementados con ambas tecnologías. Muestra gráficas de tiempo de respuesta en función de la cantidad de solicitudes concurrentes al servidor web

- http://www.orionserver.com/benchmarks/benchmark.html
  Benchmarks de su producto Orion Application Server contra IIS de Microsoft. Presenta gráficos comparativos según el nivel de stress del servidor

- http://php.weblogs.com/popularity
  Fuente de información de popularidad de las tecnologías. Se utilizó los buscadores de sitios para información actualizada

- http://www.chilisoft.com/
  Home site de la empresa que desarrolla módulos para poder utilizar tecnología ASP en múltiples plataformas

- http://www.mhsoftware.com/resources/iisjserv.html
  "Comparing IIS/ASP and Java™ Servlets: A Developer's Perspective"
  George Sexton – MH Software, Inc.

- http://www.execpc.com/~gopalan/misc/compare.html
  "A Detailed Comparison of CORBA, DCOM and Java/RMI"
  Gopalan Suresh Raj – setiembre 1998

- http://webopedia.internet.com/
  Definición de términos técnicos de informática

- http://www.w3c.org/
  World Wide Web Consortium