



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 6

Introducción.

1. Arquitecturas cliente-servidor. Conceptos generales.

1.1. Principales características.

1.1.1. Compartir recursos.

1.1.2. Concurrencia.

1.1.3. Arquitectura abierta.

1.1.4. Escalabilidad.

1.1.5. Tolerancia a fallos.

1.1.6. Transparencia.

2. Tipología.

2.1. Evolución de los sistemas cliente-servidor.

2.1.1. Aplicaciones cliente/servidor.

2.1.2. Integración de aplicaciones.

2.1.3. Modelos de distribución cliente-servidor.

2.1.4. Arquitecturas de aplicaciones.

3. Componentes.

3.1. Elementos básicos.

3.2. El middleware.

4. Interacción entre los componentes.

4.1. Desarrollo de n-capas.

4.1.1. Ventajas del modelo.

4.1.2. La evolución.

4.1.3. Desarrollo de aplicaciones basado en componentes.

5. Ventajas e inconvenientes.

5.1. Seguridad.

5.2. Control del usuario.



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 6

Arquitecturas cliente-servidor. Tipología. Componentes. Interacción entre los componentes. Ventajas e inconvenientes.

INTRODUCCIÓN.

A la hora del diseño de la arquitectura de sistemas de información, una de las posibilidades más utilizadas es el paradigma denominado «arquitectura cliente-servidor». Con el término cliente-servidor se quiere hacer referencia a un tipo de sistema distribuido en el que los recursos están divididos en subsistemas de alguna forma independientes que interactúan al menos de dos en dos y dónde uno toma el rol de cliente y el otro el de servidor. Así, suele hablarse de proceso cliente y proceso servidor, bien entendido que en este caso el término proceso hace referencia a todo un (sub)sistema, y toda la infraestructura hardware que lo soporta. Aparece de esta forma el concepto de servicio que por definición es el que proporciona el proceso o (sub)sistema servidor. Si este servicio se da de una forma preestablecida y bajo condiciones estándar, podremos considerarlo separado del cliente como un sistema en sí cuya misión es la de proporcionar ese servicio, pudiendo prestarlo así a más de un cliente incluso simultáneamente.

Si el grado de estandarización en el acceso al servicio y su prestación es elevado, puede publicarse esta prestación del servicio para otros clientes, es decir sistemas en principio desconocidos con el único requerimiento de que el acceso y petición de los recursos se haga de forma estándar.

Es importante resaltar que en el modelo cliente-servidor no es relevante la ubicación de un proceso respecto del otro, es decir, el servicio que presta un servidor a un cliente puede darse de forma local (digamos en una misma máquina) o remota. En este último caso cobra una especial importancia el sistema de comunicaciones que permite la interacción, que ha de ser conocido por ambos sistemas y fiable. Si además el acceso se estandariza hasta tal punto que se pueda acceder a aplicaciones y datos situados en cualquier lugar de forma completamente transparente, estaremos ante un sistema abierto.

Así las cosas, y dada la explosión en el progreso de los sistemas de comunicaciones y las posibilidades de interconexión que ofrece Internet no es extraño que en la red de redes prácticamente todas las aplicaciones funcionen en «modo» cliente-servidor, los sistemas web (http), de correo (pop y smtp), de terminal virtual (telnet), de resolución de nombres (dns), etc.

Si bien no está exenta de inconvenientes la implementación de sistemas cliente-servidor, como su construcción, la complejidad de gestión del software, su actualización, la dependencia del hardware y de las comunicaciones, saturación, coste, seguridad, etc. este tipo de arquitectura goza de ventajas que los superan con creces, ya que compartiendo los recursos gestionados por un servidor sobre la plataforma adecuada, entre muchos clientes heterogéneos, se consigue su optimización, economía, escalabilidad, potencia de proceso, tolerancia a fallos, etc. Tanto es así que muy pequeño y monolítico tiene que ser un sistema que se diseñe actualmente para que no funcione en modo cliente-servidor aunque sea internamente.

1. ARQUITECTURAS CLIENTE-SERVIDOR. CONCEPTOS GENERALES.

Las principales características de los sistemas cliente/servidor coinciden básicamente con las de los sistemas distribuidos

De entre las múltiples definiciones existentes de sistema distribuido, se podría escoger por su simplicidad aquella que los define como un conjunto de ordenadores autónomos enlazados mediante una red de comunicaciones y equipados con un software distribuido (sistema operativo distribuido o de red) que coordina sus distintos procesos y les permite compartir los recursos del sistema (datos, hardware y software), de tal manera que los usuarios del mismo no perciben la existencia de múltiples ordenadores y dispositivos, sino la de un único sistema integrado.

El desarrollo de los sistemas distribuidos tuvo sus inicios a principios de la década de los setenta con la aparición de las primeras redes de área local y en la actualidad, con la disponibilidad general de ordenadores personales, redes, sistemas servidores y desarrollos de software en entornos distribuidos, se está produciendo una tendencia generalizada hacia la utilización de sistemas distribuidos frente a los tradicionales sistemas centralizados.

1.1. PRINCIPALES CARACTERÍSTICAS.

Las principales características de un sistema distribuido son: compartir recursos, concurrencia, arquitectura abierta, escalabilidad, tolerancia a fallos y transparencia. A continuación se describen brevemente.

1.1.1. Compartir recursos.

En primer lugar, cabe hacer la precisión de que el término «recurso» debe entenderse en un sentido amplio, ya que a los efectos que nos ocupan abarca el equipo físico (ordenadores, discos, impresoras, etc.), los equipos y sistemas de comunicaciones, el equipo lógico tanto de base como los aplicativos específicos, y los datos (ficheros, bases de datos, etc.).

Compartir recursos comunes en un sistema centralizado es una situación que el sistema operativo del ordenador central se encarga de resolver, pero cuando se aplica a equipos individuales enlazados mediante una red de comunicaciones (como ordenadores personales accediendo a un servidor de red de área local por ejemplo), es preciso que cada recurso a compartir sea gestionado por un programa que proporcione el interface de comunicaciones necesario para acceder, manipular o actualizar el recurso de forma fiable y consistente. Esta función la lleva a cabo un módulo de software conocido como gestor de recursos cuyas funciones son, entre otras, la provisión de un sistema de nombres de recursos, el mapeo de dichos nombres a direcciones de comunicaciones, la coordinación de accesos concurrentes para asegurar la consistencia de recursos cuyo estado cambia, etc.

Por ejemplo en el modelo de sistema distribuido cliente-servidor, todos los recursos compartidos son gestionados por procesos servidores mientras que los procesos clientes envían peticiones a dichos servidores cada vez que precisan acceder a uno de sus recursos. Y en el modelo de sistema distribuido basado en objetos cada recurso compartido se considera como un objeto, de tal forma que los programas que precisan utilizar un recurso envían un mensaje conteniendo la petición al objeto correspondiente.

1.1.2. Concurrency.

En un sistema distribuido basado en compartir recursos como se ha descrito, se produce una ejecución paralela de procesos debido a que muchos usuarios pueden interactuar con aplicaciones y solicitar servicios de forma simultánea, para ello, muchos procesos servidores se ejecutan concurrentemente, respondiendo cada uno de ellos a cada petición procedente de un proceso cliente.

La concurrencia y la ejecución de procesos en paralelo aparecen ligadas a los sistemas distribuidos como consecuencia de las actividades diferenciadas de los usuarios (concurrencia de usuarios), de la independencia de los recursos y de la ubicación de los procesos servidores en equipos distintos, siendo imprescindible la sincronización de los accesos y actualizaciones concurrentes.

1.1.3. Arquitectura abierta.

Esta característica determina hasta que punto puede extenderse un sistema, tanto por lo que respecta al hardware como al software o a las comunicaciones. En el caso de los sistemas distribuidos, el aspecto clave consiste en la medida en que pueden añadirse servicios que permitan compartir recursos sin interferir o alterar los servicios existentes.

La condición necesaria para alcanzar una arquitectura abierta consiste en especificar y documentar los principales interfaces de software del sistema, poniendo la documentación a disposición de los desarrolladores de aplicaciones (interfaces públicos).

Un ejemplo de utilización de arquitecturas abiertas lo representa DCE (Distributed Computing Environment del Open Group) que constituye un estándar de facto acordado por los fabricantes del sector sobre sistemas distribuidos. Permite que los ordenadores de distintos proveedores se comuniquen y compartan recursos de forma transparente mediante la utilización de un conjunto de tecnologías de software que se integran en la infraestructura informática de los sistemas distribuidos y que incluyen servicios tales como: RPC, seguridad, directorio, transacciones y archivo distribuido. El Open Group (también conocido como Open Software Foundation) es un consorcio de fabricantes y usuarios de informática que colaboran en la tecnología de sistemas abiertos y que proporcionan el código fuente que los fabricantes incorporan en sus equipos, las especificaciones válidas para los desarrolladores y un conjunto de pruebas utilizadas para certificar la conformidad de los productos con el estándar DCE. Esta tecnología incluye servicios de software que se instalan encima del sistema operativo, middleware que utiliza el sistema operativo de bajo nivel, y recursos de red.

Esta interoperabilidad permite que los usuarios puedan trabajar con entornos heterogéneos y así poner en plataformas con software y hardware muy potentes sistemas servidores que presten servicios a sistemas pequeños con software y hardware de distinta tipología, fabricante, etc. sin problemas de funcionamiento.

1.1.4. Escalabilidad.

Esta característica implica que no deba ser preciso cambiar significativamente ni el sistema operativo ni las aplicaciones cuando se modifica el tamaño del sistema distribuido. Esto se cumple en la mayoría de los sistemas distribuidos, pero cuando son muy grandes o cuando intervienen elementos de interconexión de redes (internetworking) la cuestión deja de ser trivial, especialmente en lo referente a los equipos y sistemas de comunicaciones (un ejemplo de falta de escalabilidad lo podría constituir el agotamiento de direcciones de red).

Escalabilidad vertical / horizontal: facilidad para añadir o suprimir estaciones o elementos activos en la red (escalabilidad horizontal) o migrar a servidores mayores / menores (escalabilidad vertical) sin introducir cuellos de botella. Se trata de garantizar que el futuro del sistema no se verá comprometido, incluso aunque cambien las necesidades iniciales de diseño.

1.1.5. Tolerancia a fallos.

El diseño de sistemas distribuidos tolerantes a fallos se basa en dos factores complementarios: la redundancia de equipos hardware (incluyendo aquí comunicaciones) y la utilización de software específico para la recuperación de fallos.

Otro término relacionado con este aspecto es la disponibilidad del sistema, entendiéndola como una medida de la fracción de tiempo que el sistema está disponible para su utilización. Cada día es más frecuente que dispongamos de servicios de los denominados de misión crítica, 7 x 24, etc. y cada vez aparecen más servicios con este requisito.

La tolerancia a fallos está muy relacionada con la transparencia, entendida no como que el cliente pueda ver o conocer el interior del sistema, sino al contrario, que no tenga que importar cómo se resuelve la petición de un servicio que solicita.

1.1.6. Transparencia.

Se refiere a la percepción que tiene el usuario del sistema distribuido como un todo integrado y no como un conjunto de equipos independientes. Se basa en la separación de componentes y en la existencia de sistemas de comunicaciones y de gestión integrada que permitan la ejecución de programas en paralelo, la recuperación de fallos aislados sin alterar el resto del sistema, la seguridad global del sistema, y el crecimiento o disminución gradual del mismo mediante la adición o eliminación de componentes.

El modelo de interconexión de sistemas abiertos de la International Standards Organization (modelo OSI de la ISO) identifica diferentes formas de transparencia:

- Transparencia de acceso: permite el uso de operaciones análogas para el acceso a objetos de información tanto locales como remotos.
- Transparencia de ubicación: permite el acceso a los objetos de información sin necesidad de conocer su ubicación en el sistema distribuido.
- Transparencia de concurrencia: permite que varios procesos operen concurrentemente utilizando objetos de información compartidos sin interferencia entre ellos.

- **Transparencia de réplicas:** permite la utilización de múltiples instancias (réplicas) de los objetos de información a fin de aumentar la fiabilidad y las prestaciones del sistema, sin que los usuarios de los programas tengan que conocer la existencia de tales réplicas.
- **Transparencia ante fallos:** permite que los usuarios y las aplicaciones puedan completar sus tareas a pesar de fallos en el hardware o software del sistema (o retrotraerlas al instante anterior a la ocurrencia del fallo).
- **Transparencia de migración:** permite el cambio de ubicación de los objetos de información dentro del sistema sin que ello afecte a la operatoria de los usuarios o de las aplicaciones.
- **Transparencia de prestaciones:** permite la reconfiguración del sistema para mejorar sus prestaciones a medida que cambia la carga del mismo.
- **Transparencia de escalabilidad:** permite que el sistema pueda crecer sin necesidad de cambios en su estructura o en los algoritmos de las aplicaciones.

Las más importantes son la de acceso y la de ubicación (conocidas conjuntamente en ocasiones como transparencia de red) puesto que son las que afectan en mayor medida a la utilización de recursos distribuidos.

Un ejemplo de cumplimiento de la transparencia de red podría ser la utilización de direcciones de correo electrónico en Internet: cuando se escribe la dirección con los nombres de usuario y de dominio, no es necesario que el usuario conozca la ubicación física o la dirección de la red (transparencia de ubicación); por otra parte, la forma de enviar un mensaje no depende de la ubicación del destinatario (transparencia de acceso).

2. TIPOLOGÍA.

2.1. EVOLUCIÓN DE LOS SISTEMAS CLIENTE-SERVIDOR.

Desde el punto de vista del hardware, podemos partir de un escenario que se caracteriza por la tendencia creciente a la instalación de ordenadores personales y estaciones de trabajo, inicialmente trabajando de forma aislada y posteriormente, de forma progresiva, interconectados mediante redes de área local a través de las cuales los usuarios comienzan a compartir recursos: datos, aplicaciones, dispositivos, etc.

Al mismo tiempo, las empresas que disponían de una típica estructura informática organizada en base a mainframes y terminales no inteligentes, sustituyen éstos por ordenadores personales trabajando inicialmente en modo emulación, pero también los emplean en tareas de proceso cooperativo. Estas opciones se basan en entornos de comunicaciones consistentes y homogéneos (como LU-0 o APPC, Advanced Program to Program Communication), y suelen tener una distribución de funciones fija entre el ordenador central y las estaciones de trabajo, de tal manera que el control de la aplicación y la mayor parte del software de aplicación residen en dicho equipo central.

Posteriormente la utilización de herramientas de desarrollo para aplicaciones cliente-servidor que inicialmente se empleaban casi exclusivamente en interfaces GUI (Graphical User Interface), se extiende a otro tipo de aplicaciones más generales: los proveedores de sistemas de gestión de bases de datos ofrecen herramientas que abarcan todos los aspectos del desarrollo de aplicaciones, los sis-

temas de grupos de trabajo (groupware) evolucionan desde la ejecución de aplicaciones de productividad a otro tipo de sistemas, etc. Empieza a existir un claro partido por las soluciones basadas en una red de área local, se hacen frecuentes términos como downsizing o rightsizing, y aparecen en el mercado sistemas operativos de red (Novell Netware, Banyan Vines, etc.) que facilitan la compartición de recursos mediante una completa gama de servicios (archivo distribuido, seguridad, directorio, ...).

Actualmente la situación más frecuente es encontrarse con ordenadores personales que trabajan en un entorno de aplicaciones múltiples, poniendo a disposición de los usuarios un interface GUI (a menudo orientado a objetos), y ofreciendo la integración de las aplicaciones de la empresa con otras disponibles comercialmente (hojas de cálculo, correo electrónico, etc.) mediante la utilización de técnicas que permiten compartir e intercambiar datos, en la mayoría de los casos con acceso a una intranet donde se publican las aplicaciones corporativas, y que además permite el acceso a Internet, habitualmente de forma selectiva y a través de elementos de seguridad (proxy, cortafuegos, etc.) pero que no rompen la conectividad.

Se trata de la era del network computing en la que todos los ordenadores de una empresa están interconectados entre sí, con independencia de que estén ubicados dentro de un mismo edificio o de que la interconexión se realice mediante una WAN.

Esto pone de manifiesto que desde el punto de vista hardware, que el requisito imprescindible para los sistemas cliente-servidor es la conectividad, con independencia de que ésta se consiga con redes de área local (LAN), de área metropolitana (MAN), de área extensa (WAN) y, por supuesto, de la tecnología del medio físico que se emplee (cables coaxiales, de pares, fibra óptica, wireless, etc.).

2.1.1. Aplicaciones cliente/servidor.

Una aplicación cliente-servidor se compone de varios procesos clientes y servidores que se pueden distribuir en una red. Para ayudar a la optimización de esta arquitectura, se desarrolla un conjunto de servicios que permiten que las aplicaciones distribuidas interoperen sobre la red de comunicaciones, evitando que el personal de desarrollo o el usuario final tengan que conocer la complejidad del sistema y posibilitando que las peticiones de servicios y las respuestas lleguen a su destino de forma transparente a través de dicha red. En otras palabras, se separan las aplicaciones de los aspectos de distribución liberando al programador de las tareas de comunicaciones, directorio, gestión de transacciones y seguridad...

Para que sea posible obtener todas las ventajas que ofrece el modelo cliente-servidor, las aplicaciones deben tener los siguientes atributos:

- Separación de funciones.

Para poder distribuir partes de una aplicación entre los distintos procesadores debe ser posible dividirla en los siguientes módulos funcionales: software de presentación, software de negocio, software de datos. Con un diseño e implantación adecuados estos módulos funcionales pueden distribuirse y redistribuirse en distintos procesadores y se pueden cambiar los flujos de trabajo en función de las necesidades. Además, se pueden elegir herramientas específicas y/o tecnologías para implantar los distintos tipos de funciones. La elección del middleware determina, a su vez, el nivel de portabilidad e interoperabilidad deseado.

- Encapsulación de servicios.

A fin de posibilitar el acceso de clientes y servidores a los distintos módulos funcionales, las funciones deben encapsularse, es decir, rodearse de un interface bien definido, accesible mediante mensajes o llamadas definidas (encapsulación: los detalles de gestión de memoria y de otros dispositivos utilizados para implementar los recursos deben permanecer ocultos a los procesos cliente, incluso cuando se trate de clientes locales).

- Portabilidad.

Para que sea posible la ubicación de las distintas partes de las aplicaciones en diversos procesadores, es necesario que las mismas hayan sido desarrolladas con atributos de portabilidad que faciliten su implantación en sistemas de diferentes arquitecturas y fabricantes con mínimos o nulos cambios.

- Operación síncrona/asíncrona.

Las peticiones que dirigen los clientes a los servidores pueden ser iniciadas por aquéllos en modo síncrono o asíncrono. En el proceso síncrono el cliente debe esperar hasta que el servidor haya cumplimentado su petición (o hasta que haya expirado un plazo determinado). En cambio, en el proceso asíncrono, el cliente puede continuar con su proceso una vez que haya recibido del servidor el acuse de recibo de la petición que le ha formulado y, por su parte, el servidor una vez haya concluido el proceso lanzado como consecuencia de la petición del cliente le notificará el resultado a éste.

2.1.2. Integración de aplicaciones.

En el pasado las aplicaciones se construían principalmente de forma vertical y sin ningún tipo de integración. Una aplicación vertical estaba formada por código y sus propios datos asociados, lo que, en la mayoría de los casos, producía duplicidad de datos y falta de sincronización entre las bases de datos.

Una de las principales ventajas del modelo cliente-servidor es que, contando con un adecuado diseño, permite la integración de las aplicaciones no sólo a nivel de datos sino también de funciones. En lugar del concepto de aplicaciones monolíticas en las que datos y código residen en el mismo equipo, la distribución de funciones de la aplicación y la adopción de la orientación a objetos contribuyen fundamentalmente a ese nuevo diseño de las aplicaciones. Una aplicación se puede descomponer en muchos procesos cliente y servidor auto-contenidos, de los cuales algunos se ocupan de la interacción con el usuario a través de objetos o gráficos, y otros realizan funciones encapsuladas que pueden invocarse desde diversos procesos clientes pertenecientes a aplicaciones diferentes.

Bajo esta perspectiva, las funciones base definidas durante la fase de diseño de la aplicación se traducirán en procesos cliente y servidor, dando como resultado global un entorno de aplicación que contendrá componentes distribuibles. Tales componentes pueden ser clientes, que hacen de interface con el usuario, o servidores que proporcionan servicios a los clientes y que pueden solicitarlos de otros servidores, todos distribuidos a través de diversos procesadores en la red. Los servidores pueden ser funciones de negocio encapsuladas desarrolladas a medida o también aplicaciones estándar disponibles en el mercado.

En este entorno añadir una nueva aplicación significa: añadir un nuevo proceso cliente, habilitar dicho cliente para que pueda acceder a los servidores ya existentes, añadir un nuevo servidor para alo-

jar la nueva aplicación (también podría ser utilizado para otras aplicaciones) o modificar alguno de los servidores existentes para alojarla, modificaciones en los procesos cliente existentes para que puedan beneficiarse de la nueva aplicación. Debido a estas características, el entorno también recibe el nombre de entorno de aplicaciones incremental.

A pesar de la distribución de los componentes de las aplicaciones y del hecho de que éstas se compongan de partes de desarrollo a medida y de paquetes comerciales, al usuario se le ofrece una imagen de sistema único mediante la integración de los diversos elementos a través de la interfaz gráfica de usuario, compartiendo datos comunes o mediante comunicación directa e intercambio entre aplicaciones.

2.1.3. Modelos de distribución cliente-servidor.

A pesar de que no existen reglas sobre por donde dividir una aplicación para distribuir sus componentes, la regla general consiste en intentar situar las funciones y los datos lo más cerca posible del lugar en que sean más necesarios para la operativa de la organización, permitiendo el uso óptimo de los recursos. Dependiendo de las funciones que se distribuyan se pueden obtener tres escenarios o modelos distintos:

- Modelo de presentación distribuida.

En este modelo la parte de la aplicación que se ocupa de la presentación de los datos está situada remotamente respecto de la lógica de negocio y de los datos. El software de presentación se desarrolla habitualmente con alguna de las herramientas cliente/servidor y la presentación que se consigue consiste, en su forma más simple, en una interfaz gráfica de usuario para acceder a los programas de proceso de transacciones ya existentes, aunque también se puede incluir la funcionalidad correspondiente al diálogo con el usuario, la validación de datos, la invocación del software de aplicación y el acceso a datos.

- Modelo de datos distribuidos.

Se trata de separar el acceso y manejo de datos del resto de la aplicación. Existen dos funciones principales: ficheros distribuidos (servidor de ficheros) y bases de datos distribuidas (servidor de base de datos), que están fuera de la lógica de aplicación, distribuidas de forma transparente y realizadas por el software del sistema, de manera que el programador no tenga que saber si dichas funciones se ejecutan en la misma estación de trabajo.

En el caso de ficheros distribuidos o servidor de ficheros se trata principalmente de una forma de compartir los recursos de la red que consiste en una redirección física de las solicitudes de lectura y escritura. Como característica hay que señalar que el software de aplicación siempre se ejecuta en el proceso cliente, puesto que las exigencias sobre el sistema por parte del procesador que gestiona el recurso común son notablemente mayores que las del relativamente pequeño código cliente que se ejecuta en el procesador solicitante.

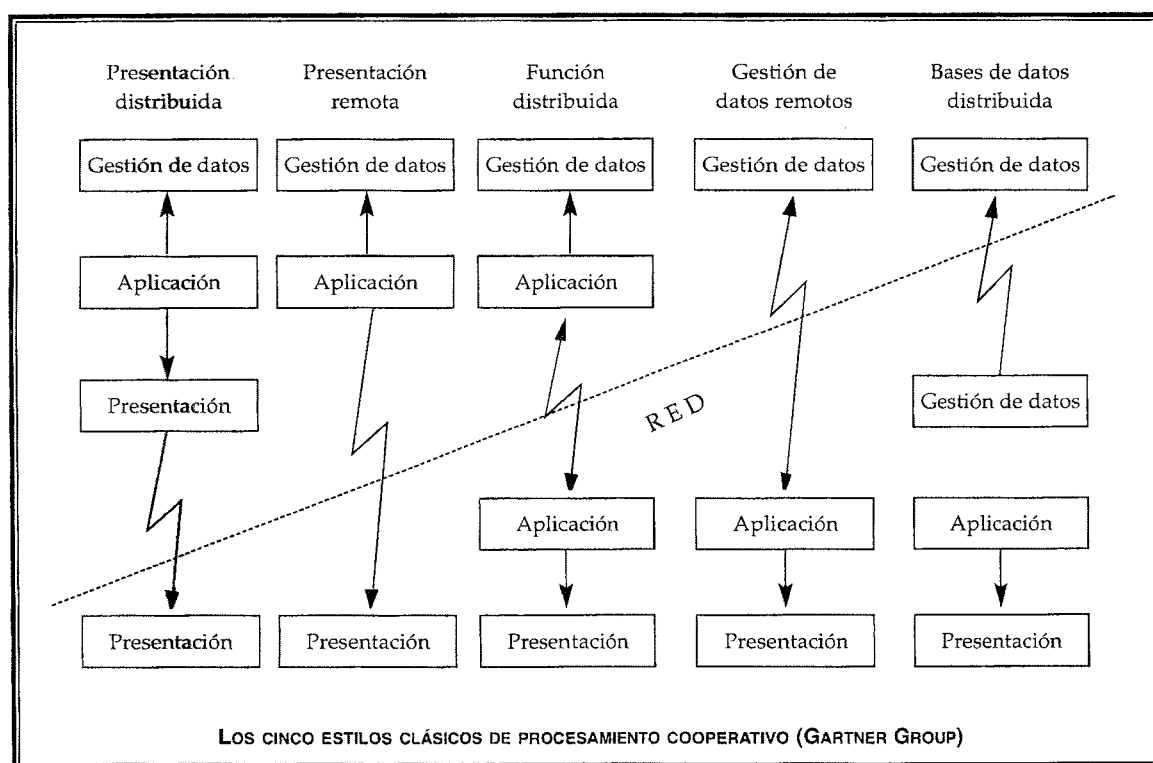
Por lo que respecta a las bases de datos distribuidas, en este caso se produce una utilización más eficiente del procesador que controla la B.D. ya que el proceso cliente pasa las peticiones SQL en forma de mensaje a dicho servidor, siendo éste el que se encarga de ejecutar las búsquedas que resulten precisas y de devolver el resultado a la aplicación invocante. El inconveniente de este sistema es que si el resultado de la consulta consiste en un volumen elevado de datos, su transmisión en un entorno WAN puede afectar negativamente al rendimiento global. La lógica de la aplicación se suele ejecutar en el procesador cliente y también en este caso las

exigencias sobre el sistema del procesador que gestiona el recurso son muy superiores a las del cliente, sucediendo además que el servidor ejecuta considerablemente más funciones que en el caso de ficheros distribuidos.

- Modelo de funciones distribuidas.

Se trata del modelo que proporciona la máxima flexibilidad y permite a los desarrolladores un control total sobre la decisión de ubicar las distintas funciones de red. A este respecto es primordial evitar que la distribución de funciones de la aplicación quede fija por diseño, ya que en ese caso una redistribución de funciones y datos implicaría importantes modificaciones o incluso un rediseño de la aplicación.

A continuación se presenta el conocido esquema del Gartner Group que refleja de forma gráfica los distintos modelos de arquitectura cliente/servidor:



2.1.4. Arquitecturas de aplicaciones.

Las aplicaciones cliente-servidor pueden clasificarse en aplicaciones de dos y de tres niveles, basándonos en la forma en que se distribuye la aplicación entre el cliente y el servidor. Normalmente el middleware y/o las herramientas de desarrollo utilizadas dictan el tipo de distribución, así como el nivel de flexibilidad y los procesadores soportados.

2.1.4.1. Aplicaciones de dos niveles.

La aplicación se divide en dos partes:

- Los servicios de presentación, la lógica de presentación y la lógica de aplicación y acceso a datos se ejecutan en la estación cliente.
- La base de datos está situada físicamente en el servidor.

Estos desarrollos se construyen sobre sistemas de gestión de bases de datos (servidores de bases de datos) y el formato del mensaje entre cliente y servidor es SQL. Algunos sistemas aportan las funciones de procedimientos almacenados y disparadores (stored procedures y triggers) que algunos proveedores consideran funcionalmente equivalentes a la arquitectura de tres niveles pero, puesto que no proporcionan una verdadera separación de la lógica de aplicación ya que ésta está ligada al gestor de base de datos, no se puede hablar propiamente de ello.

La utilización típica se presenta en los sistemas de soporte de decisiones y en aplicaciones de carácter departamental, debido, entre otras razones, a la ventaja de un rápido desarrollo de nuevas aplicaciones en un entorno de alta productividad. Por lo que se refiere a sus desventajas potenciales, se pueden señalar: solución prioritaria basada en una arquitectura cerrada, falta de flexibilidad para distribuir funciones y datos, dependencia del suministrador, falta de escalabilidad en las aplicaciones, ausencia de servicios estándar de seguridad y posibles disminuciones en el rendimiento a causa de que la lógica de aplicación está físicamente ligada al gestor de bases de datos que, normalmente, sólo soporta SQL dinámico.

2.1.4.2. Aplicaciones de tres niveles.

Las funciones de la aplicación están divididas en lógica de presentación, lógica de aplicación o de negocio y lógica de datos, pudiéndose distribuir cada una de estas partes entre los múltiples procesadores de la red. Probablemente los servicios y la lógica de presentación se ejecutarán en la estación cliente, mientras que la lógica de aplicación y la de datos pueden distribuirse entre diferentes procesadores. Al ser posible la separación de la lógica de aplicación se consigue un grado de flexibilidad que no es posible con las aplicaciones a dos niveles.

La utilización típica es en aplicaciones corporativas que soportan procesos sujetos a modificaciones; especialmente adecuadas para aplicaciones grandes, dinámicas o complejas. Las ventajas potenciales de este modelo son: máximo nivel de flexibilidad en la distribución de funciones y datos; aplicaciones escalables, flexibles y abiertas; servidores de aplicación reutilizables; selección de las herramientas más adecuadas para cada módulo. Por lo que respecta a los inconvenientes, el principal consiste en la mayor complejidad de desarrollo y en la consiguiente inversión inicial para el diseño del sistema.

3. COMPONENTES.

3.1. ELEMENTOS BÁSICOS.

Los elementos básicos en el uso de arquitecturas cliente/servidor pueden enumerarse como: Cliente o front-end con capacidad de procesamiento, Servidor o back-end (sistema posterior) y red. Un típico entorno cliente/servidor es el tradicional host o «mainframe», servidores (miniordenadores), clientes o workstations, interconectados mediante sistemas de comunicaciones.

Pero como casi siempre, los elementos considerados básicos cada vez son más, de manera que podemos esperar como componentes y arquitectura exigibles a sistemas distribuidos:

- Servicios de directorio con:
 - Un espacio o dominio bien definido y estructurado.
 - Nombres de recursos y su traducción a direcciones
 - Los esquemas de nombres.
- Cada vez mayor velocidad de transmisión en las comunicaciones, software de comunicaciones sencillo y rápido, buena política de distribución de recursos.
- Sistemas de comunicación de procesos:
 - Transmisión de datos + sincronización (síncrono o asíncrono).
 - Distribución de la carga de trabajo.
 - Pool de procesadores: servidores con varios procesadores que son asignados dinámicamente a tareas de usuarios.
 - Multiprocesador con memoria compartida: servidores con varios procesadores y suficiente memoria caché, con acceso al área de memoria compartida.
 - Mantenimiento de la consistencia.

Sin embargo, en la situación actual con lo que se ha dado en llamar «Paradigma del Web» aparecen arquitecturas para desarrollar aplicaciones distribuidas mediante otro tipo de componentes, componentes software, que sean más autónomos y reutilizables, pero que necesitan una serie de convenciones para poder interactuar, así, surgen especificaciones como: DDE, CORBA, COM (DCOM), ActiveX, Java...

3.2. EL MIDDLEWARE.

El middleware (conjunto de servicios que permiten la interoperabilidad de las aplicaciones distribuidas en una LAN o una WAN, aislando del personal de desarrollo o de los usuarios la complejidad del sistema de comunicaciones y permitiendo un acceso transparente a todos los recursos de la red) lidera las estrategias de sistemas abiertos ya que cada vez surgen más productos y herramientas de desarrollo que deben proporcionar interfaces abiertas que permitan la coexistencia de diferentes desarrollos.

Una tendencia de mercado impulsa el concepto de network computer como futuro desarrollo del modelo cliente-servidor. Dicho concepto se basa en la utilización como estaciones de trabajo de equipos sin disco duro, sin disquetera, con un controlador de red de área local incorporado de base, con una gran cantidad de memoria, y dotados de la facilidad de autoarranque directo del sistema operativo y de las aplicaciones desde el servidor.

Las principales ventajas de este enfoque son:

- Facilita la gestión de revisiones de las aplicaciones instaladas en los clientes puesto que ya no residen en sus equipos sino que se cargan desde el servidor cada vez que inician una sesión.

- Disminuye considerablemente el coste de las estaciones de trabajo.
- Aumenta la rapidez de instalación de nuevos clientes.
- Garantiza en mayor medida la seguridad de acceso.
- Asegura que las estaciones de trabajo van a ser utilizadas únicamente para su función.

En resumen se podría decir que mezcla las cualidades de los terminales X-Windows con las de los ordenadores personales, consiguiendo las prestaciones de un entorno de proceso distribuido pero sin tener que hacer frente a la problemática derivada de la gestión de dicho entorno con un número elevado de clientes. Como inconveniente para su implantación podría señalarse la necesidad de un sistema de comunicaciones de las prestaciones suficientes para soportar el elevado tráfico de red que se genera y el coste asociado a dicho sistema cuando debe implementarse sobre redes públicas de datos.

El middleware es una categoría de software que oculta la red subyacente y sus protocolos de comunicación a las aplicaciones; provee los servicios necesarios para unir las partes distribuidas de las aplicaciones a través de la red, ayudando a los usuarios a superar las dificultades del «interfacing» de los componentes de las aplicaciones dentro de entornos heterogéneos de red. Los productos middleware aparecen cuando los usuarios necesitan el acceso a los servicios de muchos servidores diferentes, dichos servidores utilizan sistemas operativos y protocolos de comunicación distintos, los servidores de bases de datos disponen de conjuntos de órdenes SQL incompatibles que dificultan los cambios entre sistemas, los usuarios precisan el acceso a este tipo de servicios desde varias aplicaciones. Podríamos resumir la evolución del middleware mencionando las siguientes arquitecturas:

- RPC o llamadas a procedimiento remoto.

Es un procedimiento síncrono que hace posible la transferencia de llamadas entre rutinas de dos aplicaciones, a través de la red y de forma transparente. Para ello existe una herramienta denominada compilador RPC que detalla el modo en que la rutina llamante invoca a la llamada (datos, formato,...) y genera un módulo de código (stub) instalado tanto en el cliente como en el servidor cuya finalidad es, en el lado del cliente, convertir las llamadas a procedimientos locales en llamadas a procedimientos remotos y, en el lado del servidor, recuperar los argumentos de la llamada, invocar al procedimiento local pertinente y, cuando éste devuelva los resultados, responder al stub cliente. Al tratarse de un procedimiento síncrono, el programa solicitante no continúa con la ejecución de su tarea hasta que no se ha completado su petición.

El Open Group (Open Software Foundation) dispone de un estándar de facto DCE (Distributed Computing Environment) para RPC.

- Conversaciones.

Una conversación es un diálogo continuo entre dos o más sistemas sobre una conexión lógica. A diferencia de las RPCs, una conversación puede constar de ejecuciones solapadas sobre un entorno distribuido, por lo que son especialmente útiles para la actualización de bases de datos distribuidas en las cuales los cambios efectuados en lugares múltiples se deben sincronizar completamente. Está basado en la comunicación igual a igual (peer to peer).

Como ejemplo de este tipo de procedimiento se puede citar el Advanced Program to Program Communication (APPC de SNA-IBM).

- Mensajería y ORB (Object Request Brokers).

En general, los sistemas de mensajería (o cola de mensajes) comunican los componentes cliente y servidor mediante mensajes que contienen las consultas y las respuestas en uno y otro sentido, y que son manejados por un sistema de transporte de mensajes. Pueden ser transferidos en modo síncrono o asíncrono, siendo este último el más común ya que implica que los mensajes pueden ser enviados sin necesidad de establecer una sesión o incluso sin que el otro componente se halle disponible.

La interfaz cola de mensajes (MQI) está desarrollada y disponible en múltiples sistemas (IBM, Unix, HP-UX, SunOS, etc.)

En el campo de la orientación a objetos se han desarrollado también métodos específicos para su aplicación a sistemas distribuidos. En concreto, por lo que se refiere a la comunicación entre procesos y el consiguiente intercambio de mensajes, destaca la arquitectura CORBA (Common Object Request Broker Architecture o arquitectura común de gestores de solicitudes de objetos), consistente en una especificación de mensajes basada en objetos desarrollada por el Object Management Group y publicada por X/Open que establece un estándar mediante el cual pueden emitirse mensajes en forma de peticiones a objetos en un formato predecible.

- **El framework microsoft.net.**

Es el modelo de programación de Microsoft.Net, para desarrollar aplicaciones Web, aplicaciones cliente y servicios Web XML. Incluye:

- Common Language Runtime: es responsable de los servicios en tiempo de ejecución:
 - Integración multilenguaje, gestión de excepciones multilenguaje
 - Gestión de memoria, hilos y procesos.
 - Gestión de la seguridad.
 - Gestión fuerte de tipos.
 - Enlace dinámico.
- Librerías de clases: proporcionan funcionalidad estándar, como entradas/salidas, manipulación de strings, gestión de la seguridad, comunicaciones en redes, hilos, etc. Esto incluye también el acceso a bases de datos con ADO.Net y páginas Web dinámicas y Servicios Web con ASP.Net.
- C# es el nuevo lenguaje de Microsoft, introducido en la plataforma.Net junto con las adaptaciones de lenguajes anteriores, como Visual Basic.Net. Tiene mucho en común con el lenguaje Java.
 - La sintaxis es muy parecida a la de C++ (o Java).
 - Proporciona gestión de memoria automática (recolector de basura), aunque permite usar punteros y gestión manual de la memoria dentro de un bloque de código convenientemente marcado.

- Se pueden escribir comentarios en XML para generar luego documentación automática.
- Permite herencia simple y definición de interfaces.
- Dispone de soporte nativo para COM y otras APIs de Microsoft.

Como ventajas, podemos enumerar: es una apuesta muy fuerte de un fabricante de software líder; es más eficiente que las anteriores plataformas de Microsoft, y las aplicaciones creadas sobre .Net son más fáciles de instalar; C# es un buen lenguaje, y la migración desde Visual Basic a Visual Basic .Net, aunque no es inmediata, es factible; la curva de aprendizaje es relativamente suave.

• J2EE.

La plataforma J2EE es una especificación de Sun Microsystems para proveer un estándar simple y unificado para aplicaciones distribuidas a través de un modelo de aplicaciones basado en componentes.

Proporciona un modelo de programación que consiste en un conjunto de APIs y dirige una manera de construir aplicaciones, basada en una infraestructura de aplicación proporcionada por los contenedores de las implementaciones de J2EE. Este modelo tiene ciertas ventajas sobre otras arquitecturas: Es independiente de la plataforma, tanto hardware como software, (se proporciona una implementación de referencia en la que debe funcionar cualquier aplicación); los objetos son gestionados por los contenedores, de forma que los desarrolladores sólo tienen que centrarse en la lógica de negocio; las aplicaciones son modulares y reusables, de forma que puede optimizarse el desarrollo. En sus inconvenientes debe anotarse al menos que: necesariamente hay que usar java como lenguaje de programación; y que incluso para pequeños desarrollos hay que desplegar casi toda la complejidad de la arquitectura.

Una de las características más importantes es el uso del concepto de componentes, que en este caso se materializan en los «Enterprise Java Beans» o EJB's. Un EJB es un componente software del lado del servidor, que debe ser desplegado sobre un contenedor de EJBs. Un EJB está formado por varios ficheros, que consisten en interfaces y clases Java y un descriptor de despliegue en XML. El contenedor consiste en un entorno software sobre el que pueden ejecutarse los EJBs, les proporciona servicios y controla su ciclo de vida. Independientemente del número de objetos que formen un EJB, éste siempre es accesible para el cliente a través de una única interfaz, que como el resto del EJB, debe cumplir la especificación.

Cumplir la especificación implica implementar y extender ciertas interfaces, exponiendo unos determinados métodos. La especificación EJB 2.0 define tres tipos de EJBs: Session Beans, que se suelen usar para modelar procesos de negocio; Entity Beans, que suelen modelar datos del negocio; y Message-driven Beans, similares a los Session Beans pero con la diferencia de que sólo pueden invocarse mediante mensajes.

Una de las principales funciones del contenedor es la de interponerse entre el cliente y los EJBs. El contenedor pone a disposición de los componentes una serie de servicios conocidos como middleware. Estos servicios middleware consisten en aspectos tales como la seguridad, persistencia, o transacciones.

Tradicionalmente, como por ejemplo en CORBA, el middleware ha sido explícito. Esto significa que las llamadas a métodos de APIs que ofrecieran estos servicios tenían que escribirse mezcladas con

el resto del código de la lógica de negocio. En las nuevas tecnologías basadas en componentes, el middleware es implícito. Esto significa que el desarrollador de los EJBs sólo tiene que preocuparse de escribir código que implemente la lógica de negocio, y puede olvidarse de escribir código para controlar el acceso a datos, la seguridad, o la persistencia. Para obtener estos servicios, es suficiente con declararlo en el descriptor de despliegue, un fichero que se escribe en XML.

Cuando despleguemos los EJBs el contenedor leerá el descriptor y sabrá qué servicios debe poner a disposición de los EJBs. De esta manera, cuando el cliente invoque métodos del EJB, el contenedor se interpone asegurándose de que se cumpla todo lo declarado en el descriptor, y es entonces cuando se delega la invocación al EJB.

El cliente no invoca de manera directa a los métodos del enterprise bean. El objeto que se interpone entre el cliente y la clase bean se llama objeto EJB. Es un objeto inteligente, que entiende de redes, transacciones, seguridad, etc., y que sabe cómo llevar a cabo lógica intermedia que pueda requerirse antes de delegar la invocación a la instancia de la clase bean.

El objeto EJB expone al cliente los mismos métodos de negocio que tiene el bean. El objeto EJB es generado por el contenedor, y para que éste sea capaz de hacerlo el desarrollador debe escribir una interfaz especial llamada interfaz remota, que contenga todos los métodos de negocio de la clase bean y que extienda la interfaz `javax.ejb.EJBObject`.

Esta interfaz extiende la interfaz `javax.ejb.Remote`, que es parte de RMI-IIOP (Remote Method Invocation over the Internet Inter-ORB Protocol). Por lo tanto, como el objeto EJB implementa la interfaz remota, esto hace que los objetos EJB sean objetos remotos, es decir, que puedan ser invocados desde otras JVMs, desde otras máquinas virtuales Java.

4. INTERACCIÓN ENTRE LOS COMPONENTES.

Una de las características más sobresalientes de la interacción entre los componentes software es el salto que se ha producido en el sentido de considerar que cualquier componente pueda comunicarse con cualquier otro con independencia de la localización física de ambos, es decir, hemos de suponer el escenario más desfavorable: que se encuentren en máquinas distintas. Para conseguirlo son necesarios, y así se establecen, servicios de nombrado, de localización de objetos o componentes, de publicación de interfaces o funcionalidades que permitan a un componente descubrir, identificar, localizar y utilizar a otro u otros.

Como es habitual en la resolución de nuevos retos o necesidades, para la consecución de estos objetivos, se utilizan conocimientos, mecanismos o estrategias ya conocidas a la vez que se intentan nuevas investigaciones. En este sentido podemos considerar como un punto de partida los mecanismos utilizados en los comienzos de los sistemas operativos distribuidos, entre los que podemos mencionar Mach de Carnegie-Mellon University & Nucleus de Chorus Systems, o los sistemas operativos actuales como Windows o Unix que pretenden asegurar la transparencia al usuario y gestionan los recursos con filosofías de encapsulación o procesamiento recurrente, sin olvidar las consideraciones de seguridad, con: Llamadas procedimientos remotos, Invocación de métodos de objetos, etc.

Dada la especial importancia que conlleva, mencionaremos explícitamente la trascendencia del procesamiento de transacciones, que aunque ahora tengan que hacerse entre objetos o componentes de una, o varias aplicaciones, no debe prescindirse de las mismas características que se consideran cuando se hacen entre sistemas funcionales diferentes (por ejemplo: cómo se hacen transacciones tradicio-

nalmente contra una base de datos). Siempre hay que asegurar las propiedades de atomicidad, consistencia, independencia, persistencia... (Técnica «Two phase commit»: compromiso en la ejecución y orden de ejecutar el compromiso)

4.1. DESARROLLO DE N-CAPAS.

El modelo n-tier (n-capas) de informática distribuida ha emergido como la arquitectura predominante para la construcción de aplicaciones multiplataforma en la mayor parte de las organizaciones. Este cambio radical en los modelos de computación, desde los sistemas monolíticos basados en main-frame y los tradicionales sistemas cliente-servidor, hacia sistemas distribuidos multiplataforma altamente modulables, representa simplemente la punta del iceberg de lo que está por llegar en el mundo del desarrollo de aplicaciones, tal y como se pone de manifiesto en las últimas tendencias de las grandes empresas de tecnología

4.1.1. Ventajas del modelo.

- Desarrollos paralelos (en cada capa).
- Aplicaciones más robustas debido al encapsulamiento.
- Mantenimiento y soporte más sencillo (es más sencillo cambiar un componente que modificar una aplicación monolítica).
- Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
- Alta escalabilidad. La principal ventaja de una aplicación distribuida bien diseñada es su buen escalado, es decir, que puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más hardware. El crecimiento es casi lineal y no es necesario añadir más código para conseguir esta escalabilidad.

Como tecnología, las arquitecturas de n-capas proporcionan una gran cantidad de beneficios para las empresas que necesitan soluciones flexibles y fiables para resolver complejos problemas inmersos en cambios constantes.

CORBA, DNA (DCOM/COM+), EJB, XML, Java, Servidores de Aplicaciones, etc. Estas y otras tecnologías relacionadas con las arquitecturas en n-capas y las comunicaciones están teniendo gran acogida en las organizaciones de tamaño medio y grande.

Todas las aplicaciones basadas en n-capas suelen permitir trabajar con clientes ligeros, tal como navegadores de Internet, WebTV, Teléfonos Inteligentes, PDAs (Personal Digital Assistants o Asistentes Personales Digitales) y muchos otros dispositivos preparados para conectarse a Internet.

De este modo, las arquitecturas de n-capas se posicionan rápidamente como la piedra angular de los desarrollos de aplicaciones empresariales y las compañías están adoptando esta estrategia a una velocidad de vértigo como mecanismo de posicionamiento en la economía emergente que tiene su base en la red (lo que se ha venido a denominar «nueva economía»).

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

4.1.2. La evolución.

Las arquitecturas basadas en n-capas son el siguiente paso lógico en un proceso de evolución, el cual, está basado en las arquitecturas convencionales cliente-servidor (2 y 3 capas) más la convergencia de dos tecnologías tremendamente potentes:

1. Desarrollo de aplicaciones basadas en componentes - relacionado directamente con la Programación Orientada a Objetos (Lenguajes y Técnicas).
2. Internet - primer ejemplo de un sistema complejo de n-capas cliente-servidor.

Los sistemas de n-capas utilizan técnicas de desarrollo basadas en componentes combinados con los estándares abiertos de Internet, para crear aplicaciones multiplataforma muy potentes con bajos costes, fáciles de mantener y con gran efectividad. Lo que realmente es nuevo en el modelo de n-capas es la posibilidad de distribuir objetos independientes sobre el número de capas que sean necesarias y enlazarlas dinámicamente, cuando sea necesario, para proporcionar una flexibilidad ilimitada a la aplicación.

4.1.3. Desarrollo de aplicaciones basado en componentes.

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

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

Estas y otras tecnologías en rápida evolución proporcionan la infraestructura necesaria y la fontanería relacionada que permite a las compañías operar en un entorno complejo, multiplataforma y con capacidades de computación distribuida, tanto interna como externamente según se requiera en cada caso.

5. VENTAJAS E INCONVENIENTES.

Se han esbozado las ventajas e inconvenientes en cada tipo de implementación de una arquitectura distribuida, no obstante, cada implementación de un sistema de información tiene que estar con-

dicionada por un estudio individualizado de todos sus ámbitos, en el caso de su arquitectura podrá ser distribuida o no, y en su caso, podrá tener un mayor o menor grado de descentralización. No obstante los sistemas que utilicen el paradigma cliente-servidor suelen tener en común ciertas ventajas e inconvenientes que podemos resumir a continuación:

- Ventajas:
 - Redimensionamiento más fácil y elegante (downsizing y rightsizing).
 - Distribución de la carga de trabajo (cliente/servidor).
 - Almacenamiento más eficiente.
 - Compartición de recursos, que en general se traduce en optimización.
- Inconvenientes:
 - Desarrollo más complejo, considerando el problema de la concurrencia.
 - Gestión y mantenimiento de aplicaciones/sistema distribuido.
 - Saturación de las redes de comunicaciones.
 - Coste de las comunicaciones.

Se tratarán a continuación, de forma más pormenorizada los posibles problemas de seguridad, así como una característica intrínseca de los sistemas distribuidos que puede considerarse tanto desde la perspectiva de las ventajas como de los inconvenientes, que es el hecho de que el usuario toma un mayor control en el flujo de la ejecución de las aplicaciones.

5.1. SEGURIDAD.

Los sistemas cliente-servidor, al igual que los sistemas centralizados convencionales, deben utilizar procedimientos de seguridad adecuados tales como autenticación o autentificación, autorización o control de acceso, confidencialidad, integridad y no repudio.

- Disponibilidad.

Una característica de los servicios que se prestan a gran cantidad de usuarios es que suelen tener un acceso público para llegar a invocarlos. Esto puede convertirse en una vulnerabilidad si no se tiene controlado el acceso concurrente, por ejemplo con un sistema de balanceo de carga o distribución de peticiones, ya que, un ataque típico consistente en realizar peticiones indiscriminadas simultánea e iterativamente, incluso desde diferentes máquinas (que pueden llegar a ser un número realmente elevado), puede provocar que el sistema se colapse y no pueda atender ninguna, incluso puede llegar a provocar la caída del sistema.

- Autenticación.

Se trata de identificar a los usuarios que inician sesión en la red. Dicha identificación puede servir como prueba de su autenticidad para el resto de los dispositivos de la red. Los sistemas de

autenticación suelen ir asociados a los procedimientos de inicio de sesión: una palabra clave (password) que tan sólo conoce un usuario y que está asociada con su cuenta de red garantiza la autenticidad de dicho usuario. En algunos casos puede ser necesario reforzar estos mecanismos mediante la verificación de determinadas características físicas y biológicas del usuario.

- Autorización o control de acceso.

Los usuarios previamente autenticados deben obtener acceso a los recursos de la red de acuerdo con los derechos de acceso que les correspondan. Se puede implementar mediante diversos mecanismos tales como listas de acceso y derechos, contraseñas, limitación del número de intentos, duración del acceso, etc.)

- Confidencialidad.

Se debe garantizar que la información transmitida entre receptor y transmisor no ha sido interceptada y que los datos quedan ocultos frente a accesos no autorizados.

- Integridad.

Se debe garantizar que los mensajes son auténticos y que no han sufrido alteración. Para ello es preciso garantizar que no se han producido manipulaciones sobre los datos tales como permutaciones, repeticiones, inserciones, pérdidas o alteraciones, mediante la utilización de secuencias numéricas, cadenas criptográficas o referencias horarias.

- No repudio.

Asegura que el originador de una información no puede rechazar la autoría sobre su transmisión o su contenido, y/o que el receptor de la misma no puede negar su recepción o su contenido. Se puede implementar mediante técnicas de certificación y firma digital.

5.2. CONTROL DEL USUARIO.

En un entorno de proceso distribuido como cliente-servidor, el control de la aplicación pasa al usuario y a su estación de trabajo u ordenador personal, a través del cual accede a los servicios locales de la red departamental o a los servicios remotos de una WAN manejando los iconos de su interfaz gráfica. Las consecuencias de esta situación son las siguientes:

- Seguridad a nivel del ordenador personal. A fin de garantizar la seguridad e integridad de los datos, el ordenador personal o estación de trabajo debe disponer de los dispositivos de seguridad necesarios para proceso local, acceso a datos locales y para aplicaciones y datos de la red (identificación y autenticación del usuario, control de acceso, protección de los datos transmitidos por la red), adicionales a los dispositivos de seguridad de que ya se disponga centralizadamente.
- Control del flujo de trabajo (servicios de aplicación). Es necesario ligar las funciones que el sistema pone a disposición del usuario para organizarlas en la secuencia requerida por el flujo de trabajo y el código que controle esta ejecución secuencial del proceso de negocio debe quedar fuera de la lógica de las funciones, para evitar que cambios en el flujo de trabajo obliguen a la recodificación de las mismas.

• **Referencias.**

Temario de las pruebas selectivas para ingreso en el Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado.

ASTIC.

Temario de las pruebas selectivas para el acceso, al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado.

MINISTERIO PARA LAS ADMINISTRACIONES PÚBLICAS.

<http://www.glosarium.com>.

<http://www.sun.com>.

