

Aplicaciones, servicios y procesos distribuidos

Una visión para la construcción de software

Patricia Bazán (coordinadora)

FACULTAD DE
INFORMÁTICA

e
exactas



UNIVERSIDAD
NACIONAL
DE LA PLATA

APLICACIONES, SERVICIOS Y PROCESOS DISTRIBUIDOS

UNA VISIÓN PARA LA CONSTRUCCIÓN DE SOFTWARE

Patricia Bazán
Alejandro Fernández
Nicolás del Rio
Lia Molinari
Juan Pablo Pérez
Matías Banchoff

Patricia Bazán
(coordinadora)

Facultad de Informática



Índice

Prólogo	4
Capítulo 1	
Introducción a los Sistemas Distribuidos	6
<i>Patricia Bazán</i>	
Capítulo 2	
Evolución de los Sistemas Distribuidos	24
<i>Patricia Bazán</i>	
Capítulo 3	
La Distribución y los Sistemas Operativos	36
<i>Matias Banchoff, Nicolás del Rio, Lía Molinari, Juan P. Perez</i>	
Capítulo 4	
Clientes y Servidores en Sistemas Distribuidos	57
<i>Patricia Bazán</i>	
Capítulo 5	
Arquitecturas Distribuidas	69
<i>Patricia Bazán</i>	
Capítulo 6	
Servidores de Base de Datos y la Distribución	82
<i>Patricia Bazán</i>	
Capítulo 7	
GroupWare	97
<i>Alejandro Fernández</i>	
Capítulo 8	
Computación en la Nube	108
<i>Patricia Bazán</i>	
Bibliografía	121
Los autores	123

Prólogo

En los orígenes de la ciencia de la computación todo el énfasis estuvo puesto en desarrollar sistemas que automatizaran tareas que se hacían manualmente. Este era suficiente desafío.

En la actualidad, se han automatizado muchas tareas y el gran desafío es cómo mejorar la capacidad de los sistemas para alcanzar nuevos requerimientos: agregar nuevas interfaces, combinar múltiples fuentes de datos en una sola, interactuar con dispositivos móviles y reemplazar viejas aplicaciones con nuevas.

En este sentido, el desarrollo de software se ha evolucionado hacia modelos distribuidos donde los componentes cooperan y colaboran para lograr el objetivo y ocultar la distribución al usuario.

Sin lugar a dudas, la vertiginosa evolución de la tecnología Web y el uso de Internet como mecanismo de comunicación, ha impactado fuertemente en los paradigmas de desarrollo de software. Pero ésta no es la única causa de la evolución de la construcción de aplicaciones con una perspectiva modular, desacoplada y que facilite los nuevos requerimientos que se mencionan más arriba.

Este libro tiene por objetivo desarrollar todos los conceptos que deben aprenderse e incorporarse para concebir software distribuido y se encuentra orientado a alumnos avanzados de carreras de informática que cuenten con conocimientos de redes, sistemas operativos, técnicas de ingeniería de software y algunas nociones de programación distribuida y concurrente.

La obra se encuentra organizada en ocho capítulos. En el Capítulo 1 se presentan las definiciones de los sistemas distribuidos y se dan algunos ejemplos. En el Capítulo 2 se analiza la evolución cronológica de los sistemas distribuidos para comprender el giro de los acontecimientos sucedidos ante los avances tecnológicos.

El Capítulo 3 aborda conceptos básicos de sistemas operativos y su funcionalidad en entornos distribuidos.

El Capítulo 4 está destinado a comprender los dos componentes fundamentales de un sistema distribuido como lo son Cliente y Servidor, como modelo clásico de distribución tanto funcional como física.

El Capítulo 5 presenta distintos modelos de arquitecturas distribuidas, cómo se organizan sus componentes y cómo se comunican. Por su parte el Capítulo 6 revisa el componente de administración del recurso dato, presente en todo sistema de información, y cómo se ve afectado por el modelo distribuido.

El Capítulo 7 aborda el concepto de GroupWare como elemento intrínsecamente colaborativo y cooperativo, analizando sus prestaciones a la luz de tecnologías clásicas y también actuales. Finalmente, el Capítulo 8 presenta la noción de computación en la nube como un nuevo modelo de servicio que impacta fuertemente en la manera en que se conciben las aplicaciones distribuidas.

o

CAPÍTULO 1

Introducción a los Sistemas Distribuidos

Patricia Bazán

Los sistemas distribuidos pueden concebirse como aquellos cuya funcionalidad se encuentra fraccionada en componentes que al trabajar sincronizada y coordinadamente otorgan la visión de un sistema único, siendo la distribución, transparente para quien hace uso del sistema.

En términos computacionales, se dice que un sistema distribuido es aquel cuyos componentes, de hardware o de software, se alojan en nodos de una red comunicando y coordinando sus acciones a través del envío de mensajes.

El concepto de componente como pieza funcional autónoma y con interfaces bien definidas alcanza al área del desarrollo de software pero también puede concebirse dentro de otras disciplinas como la idea de una pieza que, cuando se combina con los demás componentes, forma parte de un todo.

En términos estrictamente informáticos, un componente es cada parte modular de un sistema de software.

Entre las motivaciones para construir sistemas distribuidos, sin lugar a dudas está el hecho de compartir recursos, pero también favorece el diseño de software modular donde cada componente se ejecuta en la plataforma más adecuada y brinda un servicio más eficiente debido a la especificidad de su construcción. Por ejemplo, concebir un sistema de software donde uno de los componentes es la gestión y manipulación de los objetos de información (datos), le da un papel importante a los Sistemas de Gestión de Bases de Datos - en inglés DataBase Management Systems o DBMS - que logran dar un servicio de acceso a los datos eficiente, correcto y consistente.

En este capítulo presentamos un conjunto de definiciones básicas que ilustran la idea de sistemas distribuidos, sus complejidades y desafíos, así como la manera que se administran y comunican.

El capítulo se organiza de la siguiente manera: en la Sección 1 se presenta la motivación y las principales definiciones de sistemas distribuidos. La Sección 2 muestra ejemplos de sistemas distribuidos de la vida real y desde un enfoque funcional. La Sección 3 analiza los desafíos a los que se enfrentan los sistemas distribuidos y los objetivos que persiguen. Finalmente, en las Secciones 4 y 5 se detallan los modelos arquitectónicos y los mecanismos de comunicación existentes en los sistemas distribuidos con un enfoque conceptual. En la Sección 6 se arriban a algunas conclusiones.

1.1. Motivaciones y Definiciones

Como hemos mencionado, la principal motivación para construir y utilizar sistemas distribuidos se sustenta en la idea de compartir recursos. La idea de recursos abarca desde un elemento de hardware - impresoras, unidades de disco, memoria -, hasta entidades de software - archivos, bases de datos, servicios, objetos, elementos multimedia -.

En este sentido una definición de sistema distribuido aportada por Colouris en [Colouris, 2000] es:

Un sistema en el cual las componentes de hardware y software se ubican en una red de computadoras y comunican y coordinan sus acciones solo por envío mensajes.

Otra definición que se puede formular es:

Un sistema que consiste de una colección de dos o más computadoras independientes que coordinan su procesamiento a través del intercambio sincrónico o asincrónico de mensajes.

Por su parte, Tanenbaum en (Tanenbaum, 2007) afirma:

Un sistema distribuido es una colección de computadoras independientes que se muestran al usuario del sistema como un sistema único.

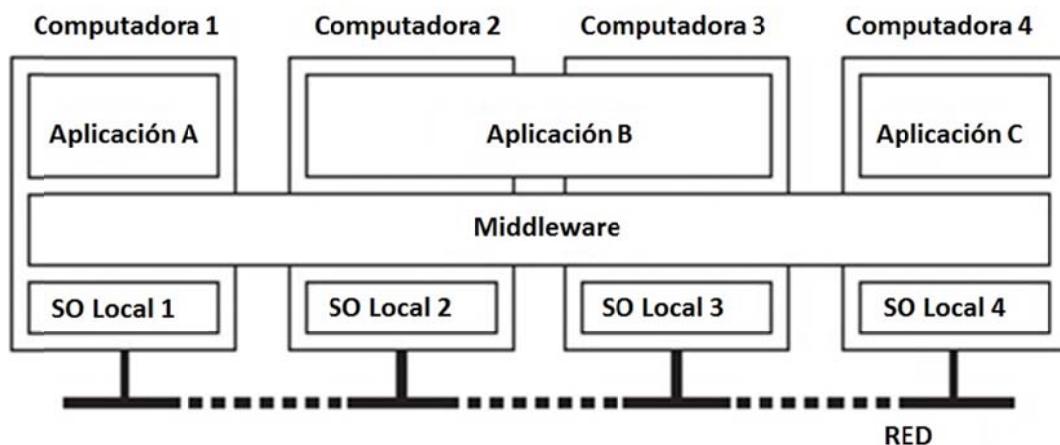


Fig. 1. Componentes de un Sistema Distribuido

En la Figura 1 muestra que no necesariamente debe existir una correspondencia única entre computadoras y aplicaciones. También se observa que la visión del sistema único es lograda por la capa middleware que oculta los detalles del entorno de ejecución local de cada computadora.

Finalmente, también se puede asegurar que:

Un sistema distribuido es una colección de computadoras autónomas enlazadas a través de una red con software diseñado para producir facilidades de cómputo integradas.

A la luz de estas definiciones se advierte la coexistencia de dos conceptos complementarios: sistema distribuido y red de computadoras. Se puede decir que mientras una red de computadoras son un conjunto de computadoras físicamente visibles y que debe ser explícitamente direccionada, un sistema distribuido es aquel donde las múltiples computadoras autónomas son transparentes pero se basa en ellas.

Sin embargo existen problemáticas comunes a ambos conceptos: las redes son a su vez sistemas distribuidos si pensamos en su servicio de nombres y además todo sistema distribuido se basa sobre los servicios provistos por las redes de computadoras.

Otro elemento conceptual es la noción de middleware software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. Se profundizará sobre este concepto en próximos capítulos.

Se revisan a continuación las principales razones para distribuir sistemas:

Distribución funcional

Este tipo de distribución radica en que cada computadora posee capacidades funcionales diferentes pero coordinan sus acciones con un objetivo común. Por ejemplo, cliente/servidor, existe un componente que brinda servicios y otro que necesita consumirlos. Otro caso es la relación host/terminal, un equipo posee toda la funcionalidad (host) y existe otro que se limita a exponer dicha funcionalidad (terminal). Asimismo la concurrencia de datos/procesamiento de datos, es otro caso de sistema distribuido donde el recurso a compartir es el dato y las funcionalidades están vinculadas a su manipulación y procesamiento como es el caso de los sistemas de gestión de bases de datos - en inglés DataBase Management Systems o DBMS.

Distribución inherente al dominio de aplicación

Este tipo de distribución se refiere a la que se realiza dentro de un mismo dominio de aplicación y cuyas componentes se identifican y modelan dentro de este dominio. Ejemplos de tal tipo de distribución lo constituyen un sistema de cajas de un supermercado y el sistema de inventario respectivo o el trabajo colaborativo soportado por computadoras.

Balanceo de carga y distribución

Este tipo de distribución se realiza para poder asignar tareas a distintos procesadores a fin de mejorar el rendimiento general del sistema.

Replicación del poder de cómputo. Varios computadores sumados pueden alcanzar una velocidad de cómputo que nunca se lograría con un super computador.

Razones económicas. Un conjunto de microprocesadores ofrecen una mejor relación precio/rendimiento que un gran mainframe¹ que es 10 veces más rápido pero 1000 veces más caro.

Entre los interrogantes que se plantean podemos encontrar:

¿Por qué usar sistemas distribuidos y no hardware o software aislado? para compartir recursos, para incrementar la comunicación entre personas, para potenciar la capacidad de cada componente (de hardware y de software) y otorgar flexibilidad al sistema

¿Qué problemas aparecen en los sistemas distribuidos y conectados? el diseño de software se complejiza porque intervienen más componentes, pueden surgir dependencias de la infraestructura de ejecución subyacente, la facilidad de acceso a datos compartidos puede afectar la seguridad.

Consecuencias

Los sistemas distribuidos son concurrentes. Esto implica que cada componente es autónomo (lo que implica diferenciar proceso de programa) y ejecuta tareas concurrentes. De este modo debe haber una sincronización y coordinación de dichas tareas y a su vez, aparecen los problemas de la concurrencia como deadlocks² y comunicación no confiable.

Ausencia de reloj local. Debida a la comunicación por medio de mensajes, se dificulta la precisión con la que cada componente debe sincronizar su reloj, apareciendo el concepto de latencia³.

Ausencia de contexto global: Debido a la concurrencia y al mecanismo de comunicación por mensajes, no existe un proceso único que conozca o albergue el estado global del sistema. Por este motivo se dice que los sistemas distribuidos son sistemas “sin estado”.

Nuevos modos de falla. Los procesos son autónomos en su ejecución, de modo que las fallas individuales pueden no ser detectadas y a su vez desconocen el impacto en el sistema completo.

1 Una computadora central (en inglés mainframe) es una computadora grande, potente y costosa, usada principalmente por una gran compañía para el procesamiento de una gran cantidad de dato

2 El bloqueo mutuo (también conocido como interbloqueo, traba mortal, deadlock, abrazo mortal) es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema

3 Latencia a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

1.2. Ejemplos de Sistemas Distribuidos

En esta sección se describen ejemplos típicos de sistemas distribuidos. En cada uno de ellos se evidencian los distintos componentes y se pone de manifiesto lo enunciado anteriormente respecto a que el sistema funciona como un todo donde cada componente se comunica con otro para poder dar respuesta al sistema de carácter global.

Ejemplo 1. La Internet

La Internet es una red que aglutina computadoras y aplicaciones heterogéneas implementadas bajo el protocolo de pilas análogo al modelo OSI para redes de computadoras.

Este protocolo se basa en una comunicación horizontal capa a capa donde las capas de nivel superior invocan servicios de niveles inferiores mediante interfaces de servicio. Esta visión por capas como proveedoras y consumidoras de servicios es un método de abstracción para aislar a los niveles superiores de detalles de la transmisión física. En la Figura 2 se puede observar el protocolo de pilas usado por Internet donde se visualiza cómo los niveles superiores tienen una visión más abstracta respecto de lo que sucede a nivel de comunicación en la capa enlace.

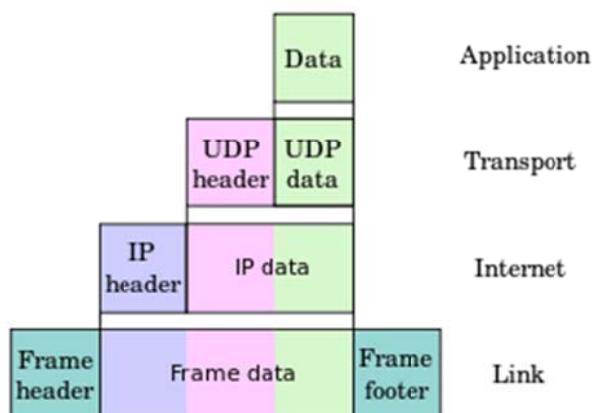


Fig. 2. Encapsulamiento de datos en la capa de aplicación descendiendo a través de los niveles

En la Figura 3 se observa cómo se lleva a cabo la comunicación entre capas (protocolos) y cuál es el flujo de datos a través de cada capa.

La capa de aplicación

Es el ámbito donde las aplicaciones crean datos de usuario y se comunican con otras aplicaciones del mismo equipo o de otro. Las aplicaciones, o procesos, hacen uso de los servicios provistos por los niveles inferiores, especialmente con la capa de transporte que provee un medio confiable o no, con otros procesos.

La capa de transporte

Es la capa que realiza la comunicación computadora-computadora sobre el mismo o diferente equipo de la red local o de redes remotas separadas por routers. Asegura la comunicación punta a punta mediante un servicio de datagramas

La capa de red

Es la capa que intercambia datagramas a través de la red y provee una interfaz uniforme que oculta la topología subyacente de conexión. Es la capa que realiza lo que se denomina “internetworking” (interconexión) que define Internet. Este nivel define las direcciones y las estructuras de ruteo usadas por el protocolo TCP/IP.

La capa de enlace

Es la capa que define los métodos de conexión con alcance en la red local en la que las computadoras se comunican sin ruteo. Esta capa incluye los protocolos usados para describir una topología de red local y las interfaces necesarias para efectivizar la transmisión de los datagramas de la capa de red, hacia sus vecinos.

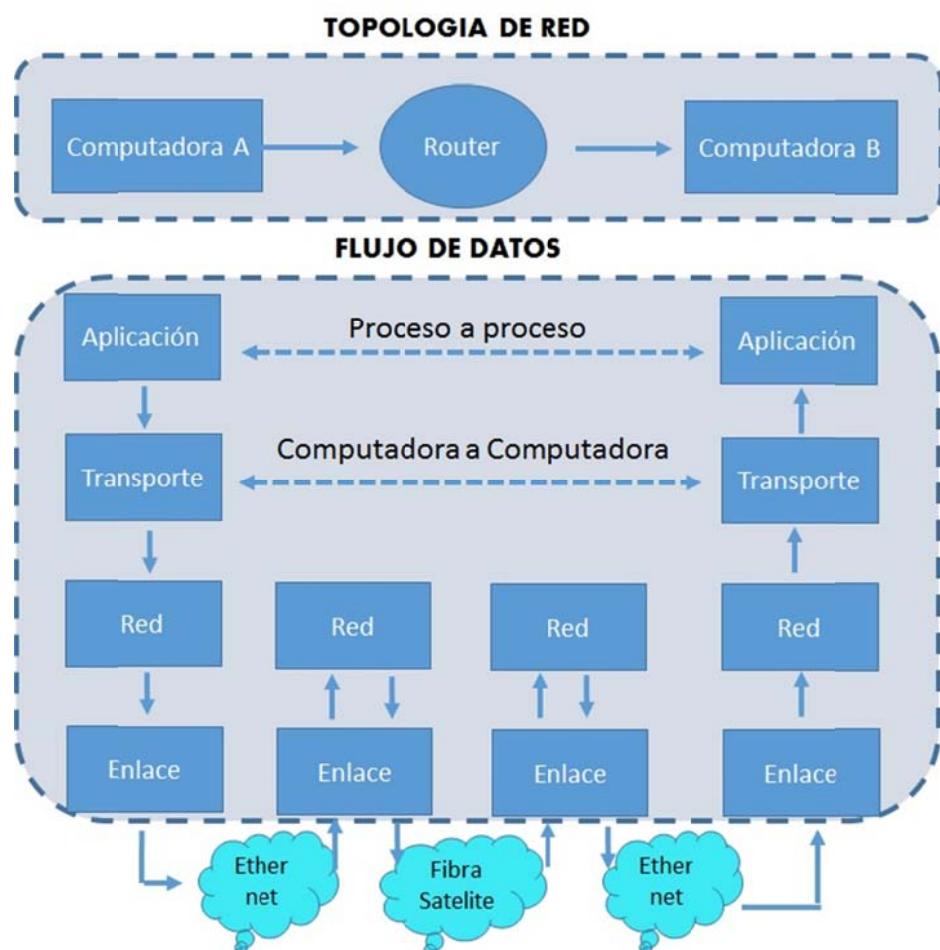


Fig. 3. Topología de red y flujo de datos entre capas

La Figura 4 muestra una configuración típica de Internet como ejemplo de sistema distribuido.

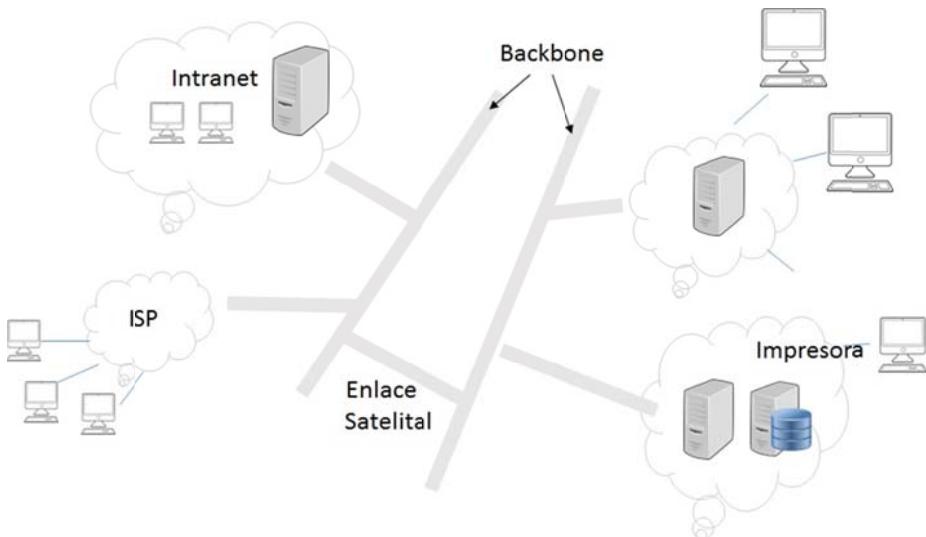


Fig. 4. Configuración global típica de Internet

Ejemplo 2. Intranets

Las intranets son redes administradas localmente, usualmente propietarias (la red del campus de una universidad, la red de una organización o empresa) y se comunican a Internet ubicando firewalls⁴.

La Figura 5, muestra una configuración posible de una intranet.

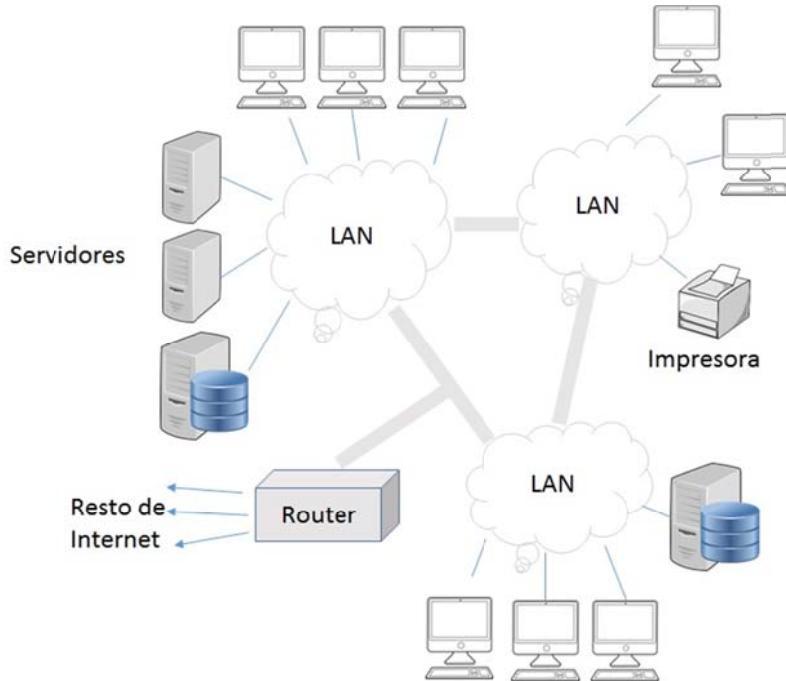


Fig. 5. Configuración de una intranet

⁴ Un cortafuegos (firewall) es una parte de un sistema o una red que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas

Ejemplo 3. Sistemas de cómputo móviles y ubicuos

Este tipo de sistemas distribuidos se caracterizan porque sus componentes son móviles - dispositivos pequeños que pueden portarse y ser utilizados durante su transporte -, y ubicuos - dispositivos que admiten estar conectados todo el tiempo sin importar el lugar.

En la Figura 6 se observa una configuración posible para un sistema distribuido móvil y ubicuo donde los dispositivos pueden ser cámaras fotográficas, impresoras, teléfonos celulares, laptop e incluso PDAs y tabletas.

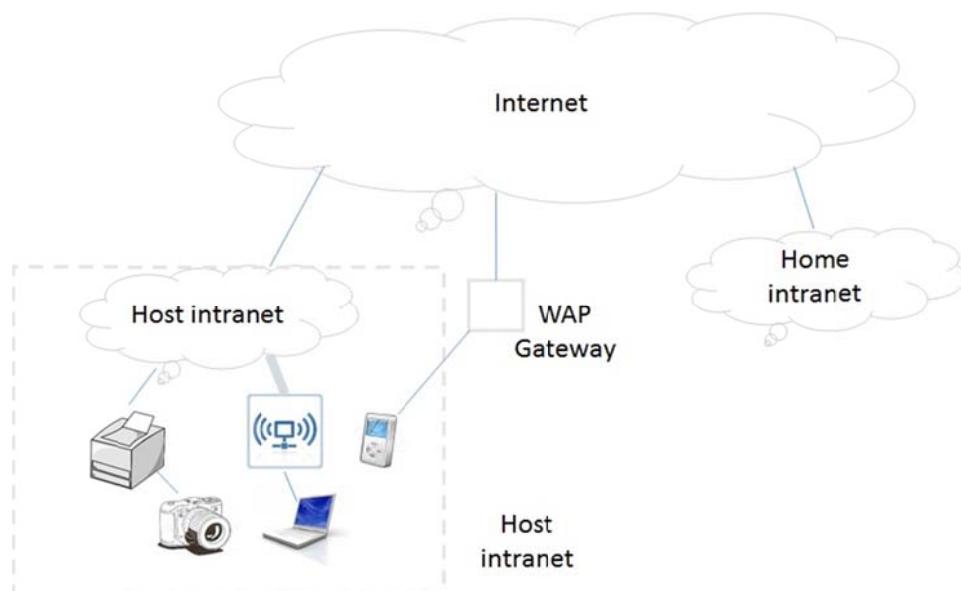


Fig. 6. Configuración de un sistema distribuido móvil y ubicuo

1.3. Desafíos y Objetivos

En esta sección se plantean los principales desafíos que enfrenta un sistema distribuido, así como los objetivos que persigue. Estos últimos son la apertura, escalabilidad, accesibilidad y transparencia, mientras que los desafíos se puede enumerar como la concurrencia, la seguridad y la heterogeneidad, algunos de los cuales son consecuencia de los objetivos.

Apertura

Un sistema distribuido es abierto cuando ofrece extensibilidad y mantenibilidad mediante la adhesión a estándares que describen la sintaxis y semántica de los servicios que ofrece.

Escalabilidad

Un sistema distribuido es escalable cuando ofrece capacidad de crecimiento en términos de cantidad y tipo de recursos que administra y cantidad usuarios que lo acceden.

Accesibilidad

Un sistema distribuido es accesible cuando garantiza el acceso a los recursos distribuidos que lo componen a todos los usuarios y en todo momento.

Transparencia

Un sistema distribuido es transparente en la medida que oculta su heterogeneidad y su distribución a los usuarios, quienes conservan la visión de un sistema único.

Como consecuencia de perseguir estos objetivos, los sistemas distribuidos se enfrentan a los siguientes desafíos:

Concurrencia

Los recursos de un sistema distribuido se caracterizan por estar expuestos al acceso concurrente de uno o más usuarios, debiendo ser capaz de planificar los accesos evitando deadlocks.

Seguridad

Además de garantizar la seguridad a nivel de transporte de datos, debido al alto grado de promiscuidad de la red como medio de comunicación, un sistema distribuido debe abordar la seguridad en términos de autenticación de usuarios, permisos de acceso sobre los recursos y auditoría del uso de los mismos.

Heterogeneidad

Un sistema distribuido es heterogéneo en cuanto es capaz de permitir que los usuarios accedan a servicios y ejecuten aplicaciones bajo cualquier plataforma (esto es diferentes redes, hardware, sistemas operativos y lenguajes de programación)

1.4. Modelos de Distribución

Los modelos de distribución pueden responder a aspectos arquitectónicos como a aspectos fundamentales o semánticos [Colouris, 2000].

El modelo arquitectónico de un sistema distribuido determina su estructura en función de cada una de sus componentes por separado.

El modelo fundamental o semántico revela los problemas claves a los que se enfrenta un desarrollador en un sistema distribuido y que condiciona la metodología de trabajo para alcanzar sistemas confiables en términos de seguridad, correctitud y fiabilidad.

Arquitecturas y Modelo Arquitectónico

La arquitectura de un sistema distribuido define la estructura completa de un sistema según sus componentes específicos, determinado el rol y funciones que cumple cada componente, la ubicación dentro de la red y la interrelación entre las componentes lo que determina su rol o interfaz de comunicación.

En cuanto a las funciones de las componentes, se definen como procesos servidores (atienden requerimientos solicitados por otros procesos), procesos clientes (inician la comunicación, solicitan el requerimiento y esperan la respuesta) y procesos pares (procesos que cooperan y comunican simétricamente para realizar una tarea).

Esta visión de componentes amerita considerar una estructura de software expresada por niveles de servicio y que facilitan la interacción en una misma computadora o en varias de ellas.

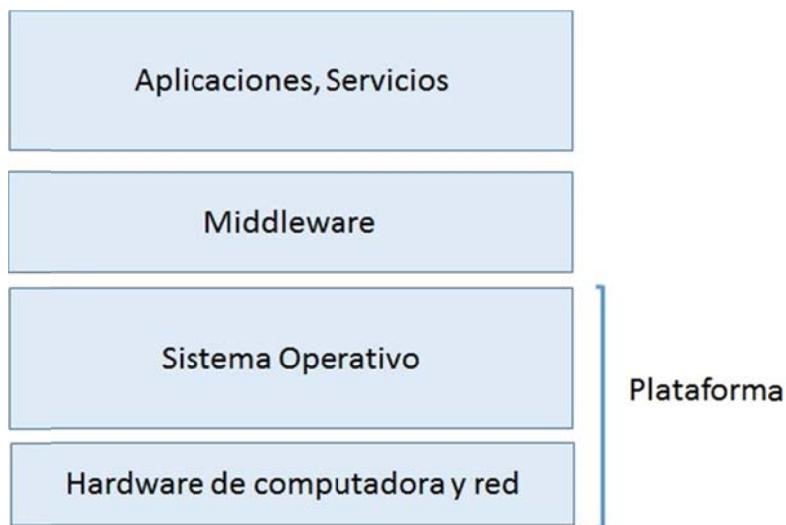


Fig. 7. Niveles de Servicio en una Arquitectura Distribuida

La plataforma está conformada por el software y hardware de bajo nivel que establece el entorno de ejecución de cada componente, sin dicho entorno, no es viable ejecutar aplicaciones siendo esto válido tanto para componentes distribuidas como para componentes que se ejecutan en un sistema único.

El middleware es el nivel de software que otorga abstracción al programador y a las aplicaciones así como también enmascara la heterogeneidad de la plataforma subyacente. El lenguaje RPC, las invocaciones basadas en SQL y los protocolos para invocación de Web Services, constituyen ejemplos de middleware que retomaremos en próximos capítulos.

El modelo arquitectónico en un sistema distribuido, simplifica y abstrae las funciones de cada componente dándoles una ubicación y una interrelación entre ellas.

Modelo Cliente/Servidor

El modelo cliente/servidor cuenta con procesos que ofrecen servicios (servidores) y procesos que utilizan o consumen dichos servicios (clientes). Se valen de un mecanismo de comunicación basado en pregunta-respuesta, siendo siempre el cliente quien inicia dicha comunicación.

Los servicios pueden ser implementados por varios procesos en diferentes computadoras que los alojan (host). Esto significa que tanto servidores como clientes pueden residir en diferentes máquinas, siendo el mecanismo de comunicación lo que pone en evidencia la distribución.

Este mecanismo de comunicación conforma una tercer componente o middleware, que establece la manera en que los procesos (tanto clientes como servidores) van a interactuar, determinado si la comunicación será sincrónica o asincrónica. Se profundizará sobre este concepto en el capítulo 4.

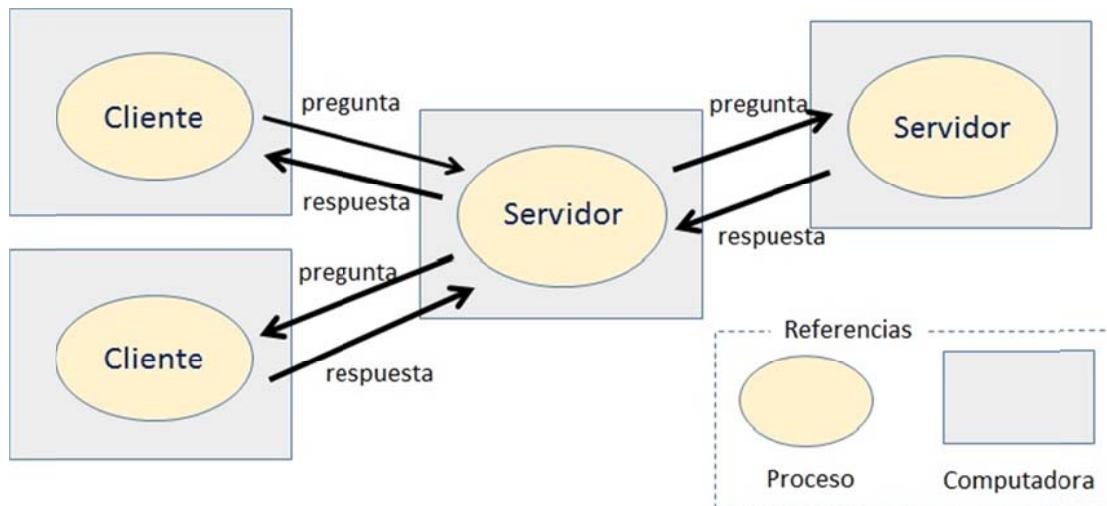


Fig. 8. Modelo Cliente/Servidor

Modelo peer-to-peer

En este modelo todos los procesos juegan roles similares considerándose pares (o peers) sin distinción entre clientes y servidores. El modelo adhiere claramente a un esquema descentralizado pero el mecanismo de comunicación no establece jerarquía, por lo que el código del proceso en cada nodo debe mantener la consistencia de los recursos y sincronizar las acciones de la aplicación.

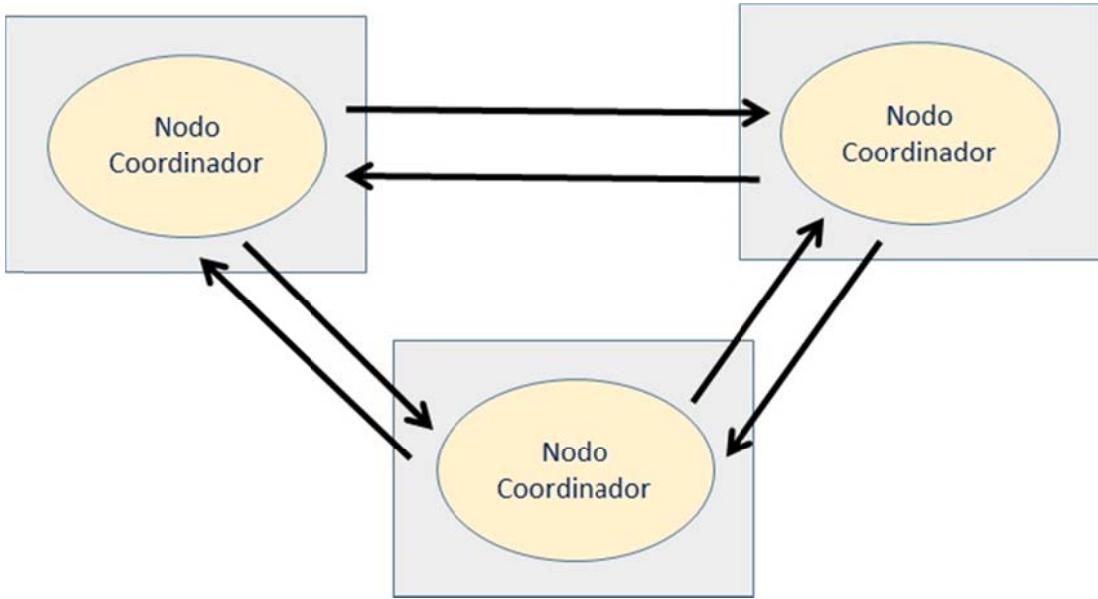


Fig. 9. Modelo Peer To Peer

Arquitecturas de software

Los diferentes modelos arquitectónicos se fundamentan en la arquitectura de software subyacente que determina la organización lógica (no física) de los componentes entre varias máquinas.

En la Figura 10 se muestran una organización en componentes lógicas diferentes, y distribución de las componentes entre varias máquinas. Estas arquitecturas pensadas como:

- a) Modelo en capas usado por sistemas cliente-servidor
- b) Modelo basado en objetos usado en sistemas de objetos distribuidos

Establecen un mecanismo de comunicación sincrónico siendo el sistema distribuido altamente acoplado en el tiempo.

Esto significa que las componentes del sistema distribuido deben estar presentes y activas para que el sistema pueda funcionar.

En el caso del modelo de capas determina un flujo de comunicación dentro de cada componente que deberá recorrer el camino inverso cuando llegue a destino y además el destino debe estar disponible.

El modelo de objetos distribuidos, por su parte, sigue el paradigma de la orientación a objetos donde cada objeto posee un estado interno y un comportamiento que será puesto en ejecución ante la llegada de un mensaje que lo requiera. En este caso no hay niveles o capas pero sí se requiere la disponibilidad del objeto que recibe el mensaje y de la computadora que lo contiene.

Por otra parte, la Figura 11 presenta dos modelos desacoplados en espacio y tiempo como lo son:

- a) Publicar/Suscribir
- b) Espacio de datos compartido

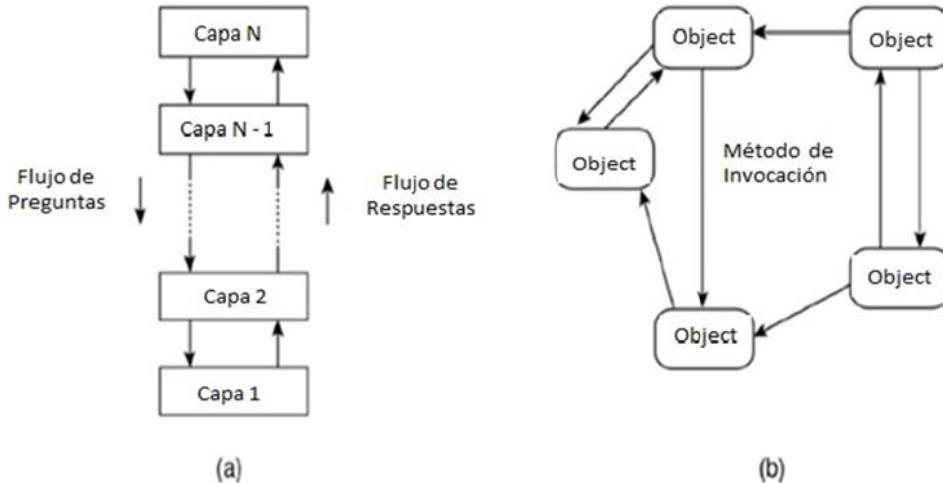


Fig. 10. Modelos en Capas usados por sistemas Cliente/Servidor

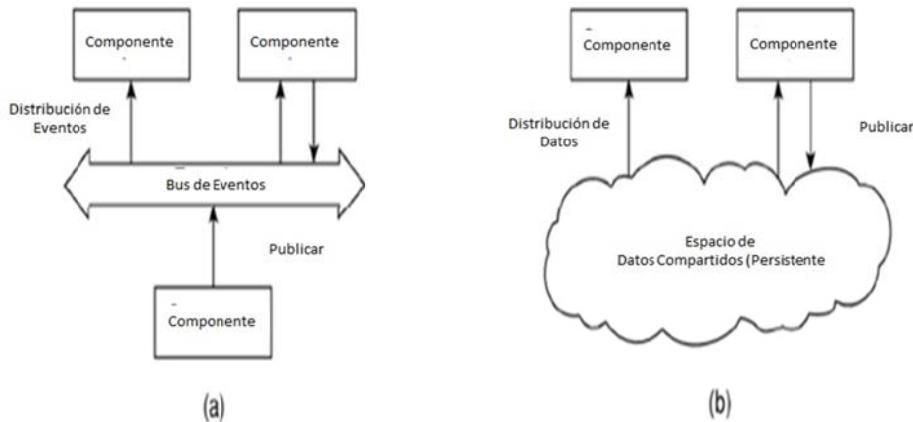


Fig. 11. Modelos de Objetos usados en Sistemas de Objetos Distribuidos

El modelo de publicación/suscripción no requiere sincronismo (coincidencia en el tiempo) debido a que las componentes publican sus funcionalidades en un *bus* de eventos que se ocupa de dar respuesta a las componentes que se suscriben a cada una de las funcionalidades publicadas. Tampoco se requiere acoplamiento en espacio, dado que las componentes como el *bus* de eventos mismo pueden estar distribuidos en la red.

Por su parte, el modelo de espacio de datos compartido permite que la comunicación entre las componentes se encuentre absolutamente desacoplada en el tiempo y favorece el acceso a los datos usando una descripción más que una referencia explícita.

1.5. Mecanismos de Comunicación

Los mecanismos de comunicación en los sistemas distribuidos son aquellos que definen la manera en que intercambian información los componentes del sistema de manera tal que el funcionamiento global se perciba como un sistema único. Estos mecanismos de comunicación son un componente más del sistema y se lo denomina middleware.

Tal como sucede con los otros componentes del sistema distribuido, el middleware cubre las necesidades de comunicación tanto a bajo nivel (comunicación a nivel física y de red) como a alto nivel (comunicación entre aplicaciones o procesos distribuidos).

El middleware a nivel del sistema operativo de red define el mecanismo de comunicación entre componentes y será desarrollado en el Capítulo 3.

Los mecanismos de comunicación entre procesos distribuidos son aquellos que se definen para dar respuestas a las siguientes preguntas:

¿Cómo conversan los componentes?

¿Cómo se sincronizan los requerimientos con las respuestas?

¿Cómo se obtiene independencia de la representación física de datos?

¿Qué sucede cuando algunos de los extremos no está disponible?

Estos mecanismos de comunicación o middleware se representan por debajo de la capa de aplicaciones y por sobre el sistema operativo tal como indica la Figura 12.



Fig. 12. Nivel de Comunicación o Middleware

Los mecanismos para comunicar procesos entre diferentes computadoras pueden clasificarse como sincrónicos o asincrónicos. Esta clasificación determina si las preguntas y respuestas entre componentes se sincronizan o no en el tiempo y a su vez esto determina si se requiere o no disponibilidad de los componentes para poder establecer la comunicación:

Entre las variantes sincrónicas encontramos sockets y RPC/RMI, mientras que MOM (Message Oriented Middleware) es un caso de mecanismo de comunicación asincrónico.

Mecanismo de Comunicación por Sockets

El mecanismo de comunicación por *sockets* establece un canal de comunicación entre dos computadoras con una conexión ya existente a nivel de transporte. Generalmente se resuelve sobre el protocolo TPC pero existen mecanismos similares sobre otros protocolos.

Una vez establecido el canal de comunicación, el socket se define como un par que contiene la dirección IP de cada computador y un número de puerto. A partir de allí, el programador debe resolver, mediante la escritura de líneas de código, todas las funcionalidades para que la comunicación a nivel de programa de aplicación pueda realizarse.

Para usar sockets es necesario:

- Establecer la dirección del socket (IP, Nro. de puerto)
- Conocer la dirección de destino
- Usar las operaciones del socket

La Figura 13 muestra un ejemplo:

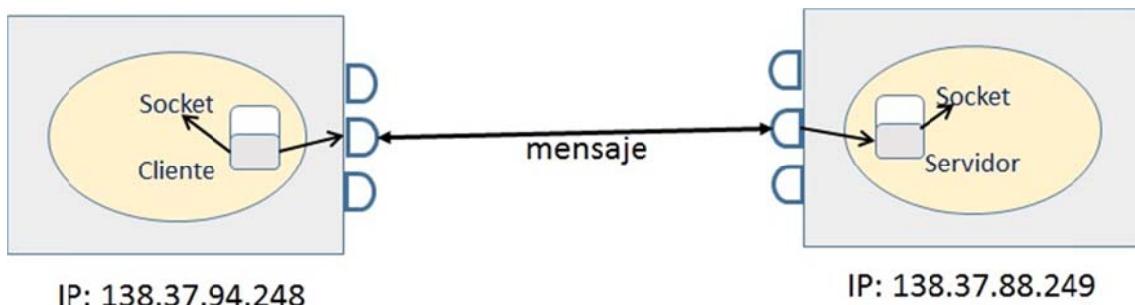


Fig. 13. Mecanismo de Comunicación por Socket

Las operaciones de los socket son:

- 1- Se crea un socket con la función `socket`
- 2- Los clientes deben conectarse con `connect`
- 3- Los servidores aceptan conexiones de clientes usando la función `accept`
- 4- Clientes y servidores intercambian datos con `read` y `write`
- 5- Las conexiones se cierran con `close`

El mecanismo de comunicación por socket carece de un entorno que facilite las tareas del programador, sólo dispone de un conjunto de operaciones básicas, siendo responsabilidad del programador resolver:

Diferencias en la representación de datos: cada computadora posee su propio hardware y sistema operativo, con lo cual es posible que se trate de arquitecturas diferentes y que los datos no sean codificados de la misma manera. En ese caso, el programador del socket debe tener en cuenta estas diferencias.

Manejo de errores: si se producen errores en la ejecución de los programas remotos, el programador deberá tener en cuenta los mismos y escribir el código necesario para manejar los mismos.

Asociación de cada componente distribuida: como se ha observado, se debe conocer de antemano las direcciones de las computadoras que participarán en la comunicación.

Mecanismo de comunicación por RPC/RMI

RPC (Remote Procedure Call) es un mecanismo de comunicación entre computadoras definido en la década del 60 y que se transformó más tarde en uno de los primeros lenguajes para programación distribuida.

Se trata de un mecanismo de comunicación sincrónico que se caracteriza por incluir las siguientes características:

- Lanzamiento y localización de funciones del servidor. El código compilado en RPC cuenta con un componente servidor y un componente cliente y esto determina que el componente servidor pueda lanzarse y permanezca escuchando en una determinada localización.
- Pasaje y definición de parámetros entre cliente y servidor. Si bien RPC intenta simular el modelo call-return de la programación clásica (no distribuida), este modelo ejecutado en componentes distribuidas se tropieza con el hecho de que no existe el concepto de memoria compartida. RPC garantiza que la pila de ejecución del proceso remoto que se invoca reciba los parámetros y retorna los resultados. Para ello utiliza el modelo de pasaje de parámetros por entrada/salida en lugar del modelo por referencia, que sería inviable dada la falta de memoria compartida.
- Manejo de fallas y caídas. Las fallas de ejecución de programas en entornos distribuidos, llevan a revisar diversas fuentes de error, en contraposición con los errores en entornos únicos, donde la cancelación por error de un programa es absoluta responsabilidad del único programa que se está ejecutando.

En entornos distribuidos, la falla puede producirse por caída del medio de comunicación (red) o por caída de los extremos (computadores), debiendo considerarse una semántica para manejar dichos errores.

1. Semántica idempotente: las operaciones se pueden realizar varias veces y siempre se obtiene el mismo resultado que si se realiza una vez.
2. Semántica *exactamente una vez*: la operación se lleva a cabo una sola vez sin reintentos. En caso de falla, la operación queda sin ejecutarse.
3. Mecanismo *a lo sumo una vez*: la operación se lleva a cabo una vez y se aplican tantos intentos como sean necesarios hasta garantizar que se ejecutó, pero solo una vez.

La Figura 14 resume los tres modelos semánticos detallando su comportamiento en cada caso: retransmisión del requerimiento, filtro de duplicados y retransmisión del mensaje.

			Semántica de Invocación
Retrasmite Mensaje	Filtrar Duplicados	Re-ejecuta Proceso	
NO	No Aplica	No Aplica	Idempotente
SI	NO	Re-Ejecuta	A lo sumo una vez
SI	SI	Re-Trasmite	Al menos una vez

Fig. 14. Comportamiento en los Distintos Modelos Semánticos

El lenguaje RPC posee un compilador que genera por cada código fuente, cuatro archivos: componente servidor, componente cliente, componente para resolver la representación de datos (mediante mecanismo de serialización o marshaling) y el código ejecutable propiamente dicho.

Una evolución del RPC se hizo presente ante la aparición del paradigma de programación orientado a objetos. Surge así RMI (Remote Method Invocation) que es una herramienta similar al RPC pero para Java y que es análoga al RPC pero aplicado a objetos distribuidos.

Mecanismo de Comunicación Orientado a Mensajes (MOM)

El mecanismo de comunicación orientado a mensajes se basa en el concepto que los componentes distribuidos se comunican enviando mensajes de cualquier tipo y que quedan encolados hasta que son leídos por el destinatario.

Este mecanismo es asincrónico y oculta totalmente los detalles del medio de comunicación (red) y de la arquitectura de los componentes, eliminando algunos de los inconvenientes de los métodos sincrónicos.

Sin embargo, aparece el inconveniente de decir si las colas de mensaje serán persistentes o no y, en caso que lo sean, como asegurar esa persistencia. Es decir, que pasa con los mensa-

jes no persistidos si el componente que los ha recibido y no los ha procesado, deja de funcionar o cae en una condición de error.

Existen numerosos productos de software que otorgan distintas variantes de solución e implementación para los mecanismos orientados a mensajes.

La Figura 15 resume algunas de las características de los mecanismos de comunicación y cómo se comportan en el caso de sincronismo o en el caso de asincronismo.

Característica	MOM	RPC
Metáfora	Como oficina postal	Como teléfono
Tiempo de relación C/S	Asincrónico. Clientes y servidores operan en distinto tiempo.	Sincrónico. Clientes y servidores corren a la vez y se esperan
Secuenciamiento C/S	Sin secuencia fija	Los servidores se lanzan antes que los clientes.
Estilo	Cola	Call/Return
Datos persistentes	SI	NO
Compañero disponible	NO	SI
Balanceo de carga	Una cola simple FIFO o con prioridades	Requiere un monitor de TP adicional
Soporte transaccional	SI (algunos productos)	NO (requiere RPC transaccional)
Filtrado de mensajes	SI	NO
Performance	Baja	Alta
Procesamiento asincrónico	SI (se requiere colas y triggers)	Limitado (requiere hilos y trucos de programación)

Fig. 15. Comparación de Mecanismos de Comunicación Sincrónicos y Asincrónicos

1.6. Conclusiones

El capítulo presenta algunas definiciones clásicas de los sistemas distribuidos y también marcó los desafíos que dichos sistemas deben afrontar así como los objetivos que persiguen. Los ejemplos que se plantean permiten conocer diversos tipos de sistemas distribuidos clasificados por las funciones y prestaciones que brindan.

El análisis arquitectónico permite comprender gráficamente la manera que se ubican los componentes y también cómo interactúan.

Los mecanismos de comunicación se constituyen en la componente fundamental para agrupar cada parte de un sistema distribuido dando la visión de un sistema único, definiendo así el concepto de middleware.

El próximo capítulo presenta la evolución de los sistemas distribuidos a través del tiempo y a través de las diferentes tecnologías que fueron surgiendo como soporte de dichos sistemas.

CAPÍTULO 2

Evolución de los Sistemas Distribuidos

Patricia Bazán

Los sistemas distribuidos concebidos como aquellos que funcionan como si se tratara de un sistema único, no hacen ninguna presuposición acerca de la variedad funcional de sus componentes ni de cómo estos componentes se comunican. De esta manera las variantes que existen han dado origen a sistemas distribuidos de distinta índole y que van de la no-distribución (sistemas monolíticos) hacia la distribución completa (servicios orquestados por procesos de negocio).

En este capítulo se analiza esta evolución tanto desde el punto de vista tecnológico que sin dudas se ve afectado por la evolución de las comunicaciones, plataformas y hardware.

El capítulo se organiza de la siguiente manera: la Sección 1 presenta el concepto de sistema monolítico y sus mejoras ante la aparición de distintos componentes funcionales. En la Sección 2 se describen los sistemas Cliente/Servidor y su evolución hacia sistemas de n-capas. En la sección 3 se analizan los sistemas distribuidos con objetos como antecesores a las arquitecturas orientadas a servicios presentadas en la Sección 5. Mientras tanto, en la Sección 4 se describen las distintas alternativas de integración de aplicaciones. En la Sección 6 se analizan a los procesos de negocio como consumidores de Servicios. Finalmente en la Sección 7 se arriban a algunas conclusiones.

2.1. Sistemas Monolíticos

Un sistema de información monolítico es aquel que se concibe como un único elemento funcional donde sus prestaciones se ofrecen a través de una sola pieza de código que resuelve tanto la presentación al usuario (interface), como el acceso a los datos (trata con operaciones de entrada/salida) y a su vez resuelve la lógica algorítmica del problema que aborda.

Estos sistemas de información se construían en un único lenguaje de programación, sobre un único computador y con un único sistema operativo como plataforma. La comunicación con el usuario y también con el programador, era a través de terminales de caracteres que responden únicamente a comandos lanzados por consola. Su ubicación en el tiempo se remonta a los años '70.

Una versión mejorada de los sistemas de información así concebidos, pudo aportarse con las técnicas de programación estructurada, los lenguajes de programación y las metodologías

de programación modular introducidas por Edsger Dijkstra. En este sentido, los sistemas de información comenzaron a ser concebidos como un conjunto de componentes o niveles de complejidad, aunque permanecían ejecutándose en una misma computadora y por lo tanto podemos considerarlos no-distribuidos.

La Figura 16 la evolución marca que el primer paso fue reconocer un componente funcional que resuelve el acceso a los datos (con la aparición de los DBMS-DataBase Management Systems), para luego identificar un componente de manejo de interface de usuario [Weske, 2008].

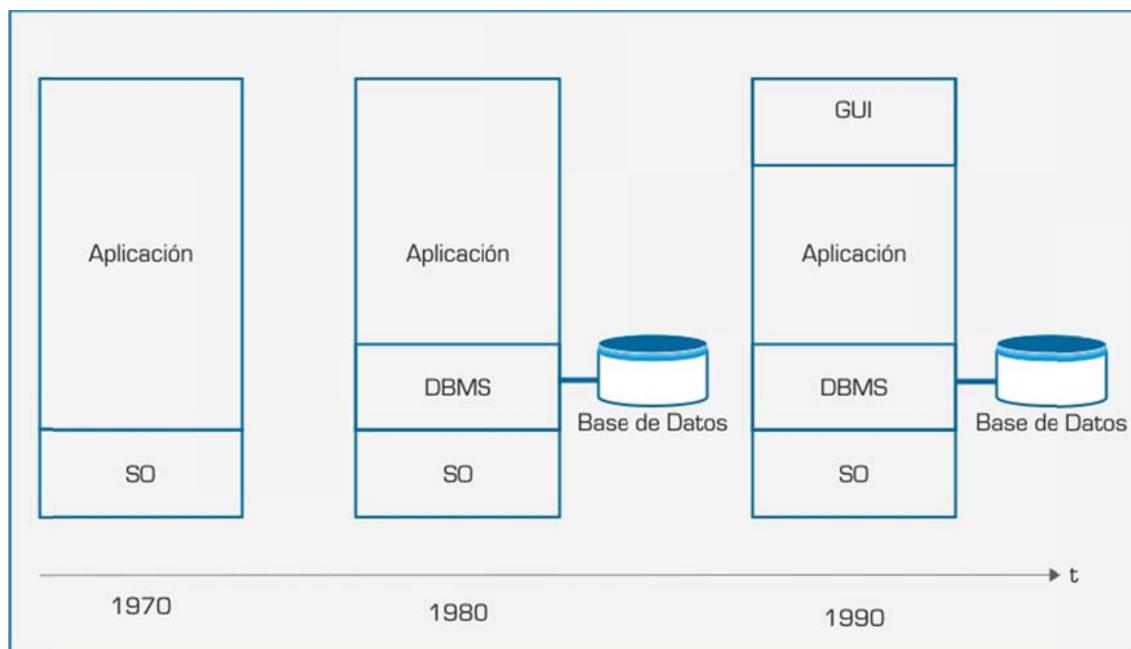


Fig. 16. Evolución tecnológica de los sistemas de información

2.2. Cliente/Servidor y su evolución a n-capas

La distribución funcional entre lógica de aplicación propiamente dicha y lógica de acceso a datos vislumbrada en la década del 80, condujo a concebir la idea que estos dos componentes podría residir en computadoras diferentes aprovechando las consultas SQL que podían interpretar los DBMS, junto con la gran proliferación de PCs sobre LAN (Local Area Network), que se estaba dando en esos tiempos.

Este escenario dio origen a la primera versión de sistema distribuido, desde el aspecto funcional, que fue el denominado Cliente/Servidor o arquitectura de 2 capas (En el Capítulo 4 se analizará en detalle esta arquitectura).

La arquitectura Cliente/Servidor o arquitectura de 2 capas clásica, se sustenta en la idea que existe una componente Cliente que resuelve la lógica de la interface con el usuario. El código de este componente se escribe generalmente en un lenguaje visual de cuarta generación (Delphi, VisualBasic, PowerBuilder). La capa 2 sería un componente Servidor que resuelve la lógica de acceso a los datos, al menos con la gestión que hace los mismos un DBMS.

En esta arquitectura, la lógica de la aplicación no cuenta con un componente físico específico para su ejecución y despliegue, sino que es absorbida por algunas de las dos componentes antes mencionadas: Cliente o Servidor, tal como se muestra en la Figura 17.



Fig. 17. Sistema distribuido en 2 capas o Cliente/Servidor clásico

Este modelo de sistema distribuido posee ventajas y desventajas.

Ventajas

- › **Aprovechamiento del poder de las PC reduciendo costos**
- › **Permite que el procesamiento resida cerca de la fuente de datos reduciendo el tráfico de red**
- › **Facilita el uso de GUI**
- › **Acepta el desafío de los sistemas abiertos**
- › **Favorece el diseño modular**

Desventajas

- › **Un desbalanceo de tareas entre cliente y servidor puede provocar cuellos de botella**
- › **El desarrollo de aplicaciones es más complejo creciendo también la complejidad de las herramientas de desarrollo y programación**

La desventaja fundamental que llevó a buscar evolucionar hacia un modelo superador lo constituyó sin dudas el desbalanceo de carga entre cliente y servidor, lo cual se vio acompañado por la instauración cada vez más firme de la Internet como la mejor solución a la no disponibilidad de recursos de cómputo y la globalización de los sistemas de información.

Por su parte, el modelo Cliente/Servidor clásico se transformó en poco escalable (capacidad de crecimiento) tanto a nivel vertical (aumento de cantidad de clientes y por ende de requiri-

mientos al servidor), como horizontal (un solo servidor accesible a través de una LAN, ya no era suficiente, se requería acceso a través de WAN).

Surge así el modelo de 3 capas donde, básicamente, se le otorga a la capa media (lógica de la aplicación) una plataforma de ejecución propia. El más claro ejemplo de este tipo de modelos lo constituye la tecnología Web. De este modo, y dado que el componente medio posee su propia plataforma de ejecución y que se cuenta con un modelo Cliente/Servidor, es que la capa media puede ser una sola o varias - siendo todas estas alternadamente cliente y servidor de la siguiente - y obteniendo la arquitectura n-capas, como se muestra en la Figura 18.

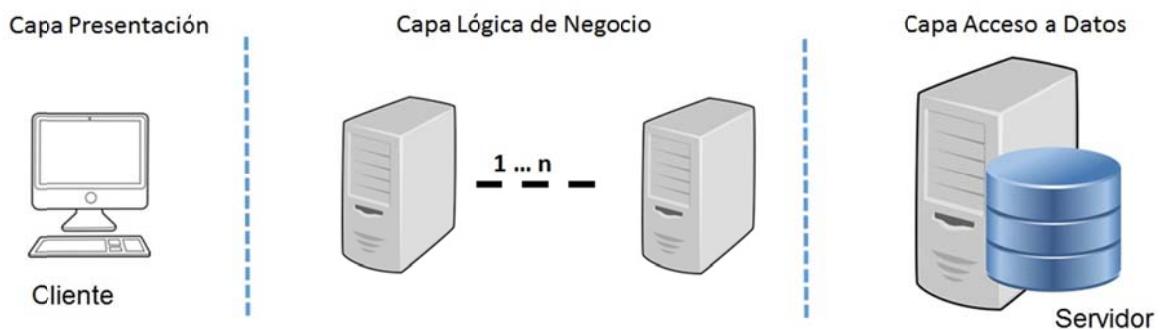


Fig. 18. Sistema de Información en N capas

Tecnología Web: un caso particular de Cliente/Servidor

La tecnología Web es aquella hace uso de Internet para construir sistemas de información. En este sentido, dichos sistemas de información se construyen haciendo uso de los protocolos y mecanismos de comunicación provistos por Internet, esto es: acceder a recursos a través de una identificación única (URL), obtener dichos recursos con un protocolo de transferencia de archivos (FTP/HTTP) y mostrar los resultados al usuario (Navegador o Browser).

Web Browser: componente cliente para visualizar un documento HTML

Web Server: componente que contiene páginas HTML que envía a los clientes que los visualizan en el Web Browser

Web Site: estructura de directorio para almacenar las páginas HTML y otros archivos.

La Figura 19 muestra una solución en tecnología Web simple que se compone por un Web Browser que interpreta páginas HTML (Browser), obtenidas por HTTP de un Web Server.

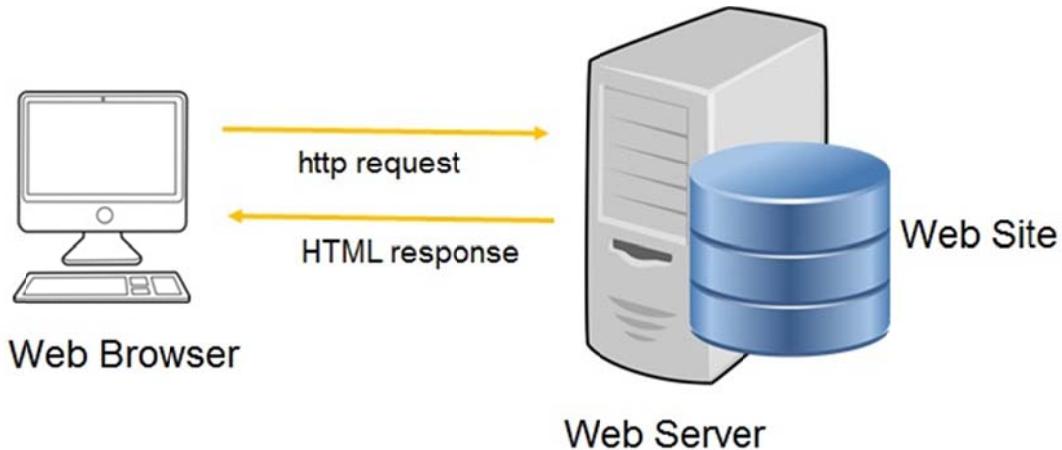


Fig. 19. Tecnología Web Simple

Tecnología Web en 3 capas: Web Server Dinámico

El componente Web Server en sistemas distribuidos de tecnología Web, provee un tipo de servicio que se limita a recibir peticiones por HTTP y responder con un conjunto de registros (archivo) estructurado como HTML.

Este escenario resulta insuficiente para construir sistemas de información debido a: 1- ausencia de un componente que permita acceder a los datos y mantenerlos persistentes y 2- ausencia de un componente que ejecute código para producir transformaciones en dichos datos.

Surge así lo que damos en llamar *web server dinámico*, es decir, aquel que es capaz de desencadenar cómputo que se encuentra codificado dentro de las páginas HTML que administra, tal como muestra la Figura 20.

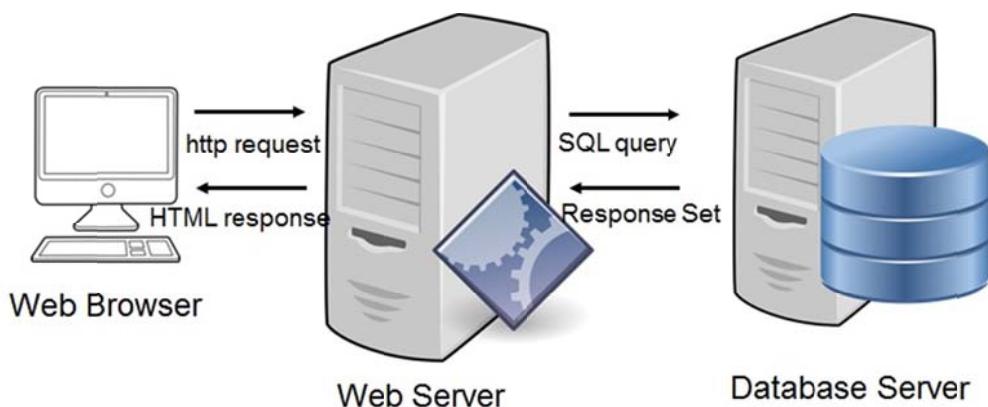


Fig. 20. Tecnología basada en Web en 3 capas

En el Capítulo 5 retomaremos este tipo de tecnología y profundizaremos en ella.

2.3. Sistemas Distribuidos con Objetos

A medida que fue aumentando la complejidad de las soluciones distribuidas fue necesario incorporar conceptos, metodologías y buenas prácticas en la construcción de software y llevarlas al entorno distribuido.

El concepto de *objeto* en el sentido clásico de la Programación Orientada a Objetos y que tiene sus raíces principales en el lenguaje Smalltalk, se mantiene a nivel de diseño de aplicaciones pero cambia algunas características cuando se lo lleva a un entorno distribuido.

Un *objeto distribuido* es un *objeto* (componente funcional con estado interno, comportamiento, heredable y encapsulado), que por estar en un entorno distribuido también debe ser:

- Transaccional: los cambios de estado que produce se deben ejecutar completamente o no ejecutarse.
- Seguro: debe evitar vulnerabilidades que corrompa el estado del sistema.
- *Lockable*: debe fijar un cerramiento que garantice su ejecución correcta ante accesos concurrentes.
- Persistente: su estado interno debe conservarse más allá de su ciclo de vida.

Esta definición de objeto distribuido nos conduce a la necesidad de contar con una infraestructura que facilite la comunicación entre este tipo de componentes, logrando a su vez que logren esta comunicación a través de distintas plataformas de ejecución, sistemas operativos y lenguajes de programación.

Así surge el estándar CORBA (Common Object Request Broker Architecture)⁵ [2.3], definido por la OMG (Object Management Group) y que surge de la iniciativa de un consorcio de empresas interesadas en establecer una arquitectura interoperable sobre la base del paradigma orientado a objetos. CORBA se constituye así en un ejemplo de sistema distribuido basado en objetos.

Entre los elementos distintivos del estándar - y que se puede decir que sentaron las bases para lo que se conoce actualmente como *web services* - encontramos:

- Un lenguaje de especificación de interface, IDL Interface Definition Language, que define los límites de las componentes o sea las interfaces contractuales con los potenciales clientes.
- Un *bus* de objetos u ORB (Object Request Broker) que comunicar objetos independientes de su locación. El cliente se despreocupa de los mecanismos de comunicación con los que se activan o almacenan los objetos servidores.

Provee un vasto conjunto de servicios de middleware distribuido y tiene claramente un mecanismo de comunicación mucho más complicado que los clásicos RPC y MOM.

⁵ <http://www.corba.org/>

2.4. Integración de Aplicaciones

Más allá de los esfuerzos por construir estándares para mejorar la interoperabilidad y de la existencia de plataformas Web mucho más accesibles al programador y al usuario, el desarrollo de aplicaciones distribuidas a gran escala, seguía adoleciendo de problemas de integración.

La idea de construcción de aplicaciones integradas permitió que las organizaciones desarrollen software que resuelva cada parte de su negocio y se integre con aplicaciones que gestionen la parte administrativa de dicho negocio. En este sentido la idea de integración de aplicaciones comienza a cobrar un sentido muy relevante.

En este contexto surgen los sistemas ERP (Enterprise Resource Planning) que proveen variada funcionalidad integrada por un único repositorio de datos.

No se tardó demasiado en intentar agregar valor a los desarrollos de las organizaciones aportando sistemas de CRM⁶ (Customer Relationship Management) o sistemas de DataWarehouse⁷ que requieren algún tipo de integración con los sistemas de índole operativa o propia del negocio.

Esta integración se enfoca principalmente en la integración vía el modelo de datos. Este tipo de integración ha sufrido una evolución e incluye diversas variantes que se describen a continuación.

Integración Punto a Punto

Se basa en integración uno a uno sustentada generalmente por un middleware asincrónico basado en colas de mensajes. Si bien es un esquema de alta disponibilidad, dada por el mecanismo de comunicación, es rígido y difícil de adaptar a los cambios, además de resultar muy costoso de gestionar, monitorear y extender. Es una arquitectura accidental, completamente sincrónica, de grano grueso y poco escalable.

La Figura 21, presentada por M.Weske en (Bazán,2010), muestra el escenario de una integración con el mecanismo punto a punto.

6 Sistemas informáticos de apoyo a la gestión de las relaciones con los clientes, a la venta y al marketing (Wikipedia)

7 Es una colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza (Wikipedia)

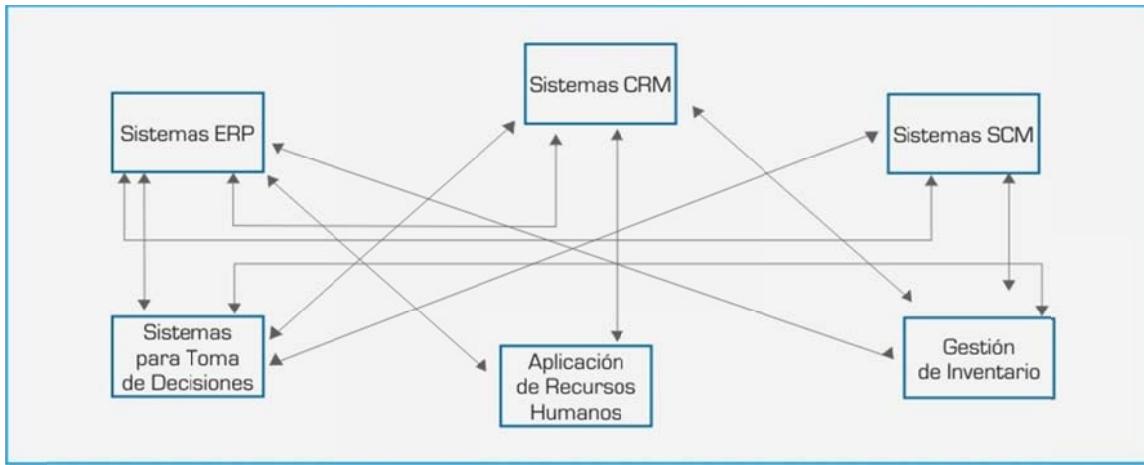


Fig. 21. Mecanismo de integración punto a punto

Integración por adaptadores

La interacción se realiza por un *hub* donde se conecta cada elemento a integrar previa construcción del adaptador correspondiente para poder *enchufarse*. Cada uno de los elementos está desconectado y no requiere utilizar el mismo lenguaje ya que existen adaptadores para establecer la comunicación.

La complejidad está en la construcción del adaptador.

La Figura 22 presenta una evolución de la Figura 21 hacia un mecanismo de integración por adaptadores.

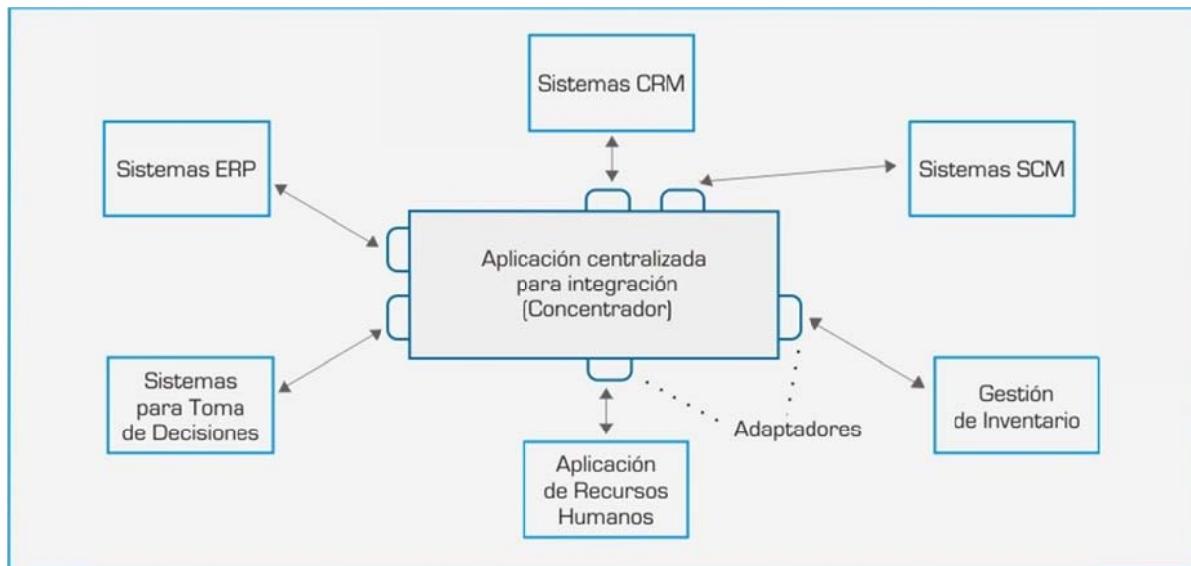


Fig. 22. Mecanismos de Integración por Adaptadores

Mediador de mensajes

Este mecanismo de integración representa una evolución del modelo anterior hacia una generalización del hub permitiendo extraer la lógica de la integración fuera de las aplicaciones. Se define declarativamente la forma de comunicación de las aplicaciones y se traslada al mediador o *broker* la lógica necesaria para producir las transformaciones que generen salidas válidas para el otro extremo.

Utiliza colas para garantizar la distribución de los mensajes, que se manejan con un esquema de publicador/suscriptor. Esto asegura que el tráfico que se genera sea solamente el requerido.

La Figura 23 muestra el próximo paso evolutivo desde la Figura 22 donde se muestra la integración a través de un broker de mensajes.

2.5. Arquitectura Orientada a Servicios

La integración de aplicaciones como se ha planteado en 2.4, si bien ha sido mejorada con diversos mecanismos como el uso de adaptadores y brokers, siguen siendo una arquitectura accidental (se construye para cada caso), de mantenimiento costoso y lento y difícil de gestionar, monitorizar y extender.

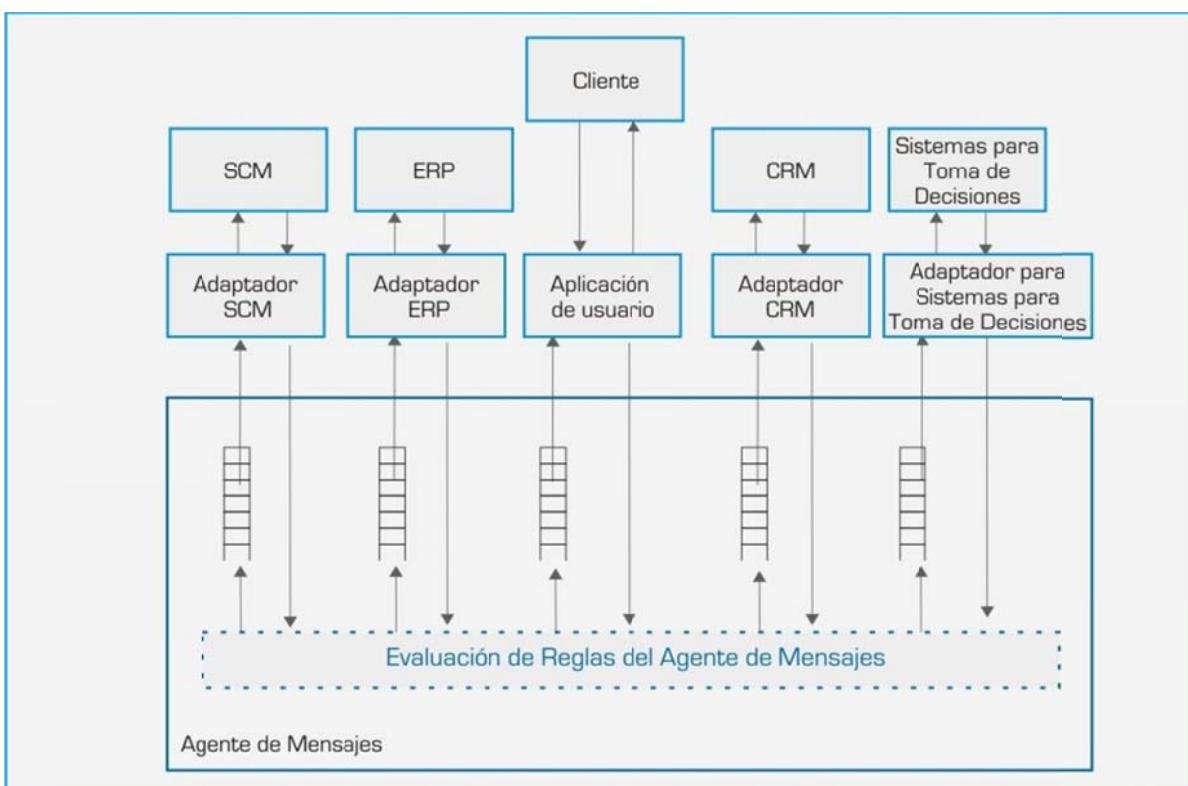


Fig. 23. Mecanismo de Integración por Mediador de Mensajes

La Arquitectura Orientada a Servicios (SOA) representa un cambio radical en la relación entre el mundo del negocio y el área de tecnología de la información. SOA constituye mucho más que un conjunto de productos aglutinados por una tecnología. Es un nuevo enfoque en la construcción de sistemas de IT que permite a las empresas aprovechar los activos existentes y abordar fácilmente los inevitables cambios en el negocio.

Si bien la industria del software ha venido enfocándose en una arquitectura orientada a servicios desde hace más de 20 años con la noción de reusabilidad y su aplicación a la construcción de software, lo cierto es que en los últimos años esto se ha fortalecido con la definición de estándares y la conformación de consorcios que participan en su definición.

Según IEEE una arquitectura de software es la organización fundamental de un sistema, reflejado por sus componentes, relaciones entre ellos y entorno, así como los principios que regirán su diseño y evolución (1471-2000) (Bazán,2010).

Según OASIS, se define como la estructura o estructuras de un sistema de información formado por entidades y sus propiedades externamente visibles, así como las relaciones entre ellas (Modelo de Referencia para SOA 1.0 – Agosto de 2006) (Bazán,2010).

Adaptándose a la definición de OASIS, se define SOA como un paradigma capaz de organizar y utilizar las capacidades distribuidas, que pueden estar bajo el control de distintas organizaciones, y de proveer un medio uniforme para publicar, descubrir, interactuar y usar los mecanismos oportunos para lograr los efectos deseados.

Podemos resumir los conceptos subyacentes fundamentales en este paradigma en los siguientes:

- Proveedor: entidad (organización o persona) que ofrece el uso de capacidades mediante servicios.
- Necesidad: carencia de una empresa para resolver la actividad de su negocio.
- Consumidor: entidad (organización o persona) que busca satisfacer una necesidad particular a través de las capacidades ofrecidas por servicios.
- Capacidad: tarea que el proveedor de un servicio puede proporcionar al consumidor.
- Servicio: mecanismo que permite el acceso a una o más capacidades alcanzables por medio de una interfaz preestablecida, y que se llevará a cabo de forma consistente con las normas establecidas para él.
- Descripción del servicio: información necesaria para hacer uso del servicio.
- Interacción: actividad necesaria para hacer uso de una capacidad con el objeto de obtener efectos deseados.

En el Capítulo 5 se retoman y amplían los conceptos de Arquitecturas Orientadas a Servicios.

2.6. Procesos de Negocio como Consumidores de Servicios

En términos generales, cuando se habla de integración de aplicaciones nos referimos tanto a datos como a procesos.

Los datos pueden integrarse o puede resolverse su integración contando con cualquiera de los esquemas anteriormente planteados. Cualquiera de ellos implica el desencadenamiento de un flujo o cadena de mensajes que quedan incrustados dentro del esquema de integración mismo.

Así, los sistemas de gestión de workflow surgen como ciudadanos de primera clase a la hora de realizar una integración de aplicaciones eficiente y fácil de mantener y actualizar.

Gran parte del esfuerzo de implementar una arquitectura orientada a servicios orquestada por procesos de negocio, se ha centrado sobre la integración de aplicaciones y el workflow. Sin embargo existe una preocupación de los arquitectos de software acerca de la incapacidad para acceder a datos de negocios y administrarlos de una manera ágil tal como sucede con la lógica de negocio en las aplicaciones.

La integración de datos dirigida por procesos puede ayudar a enriquecer los servicios de negocios SOA y los procesos de negocio a través de una secuencia de servicios de datos combinados de manera reusable que incorpora la intervención de tareas humanas transformando la información en exacta, consistente y oportuna.

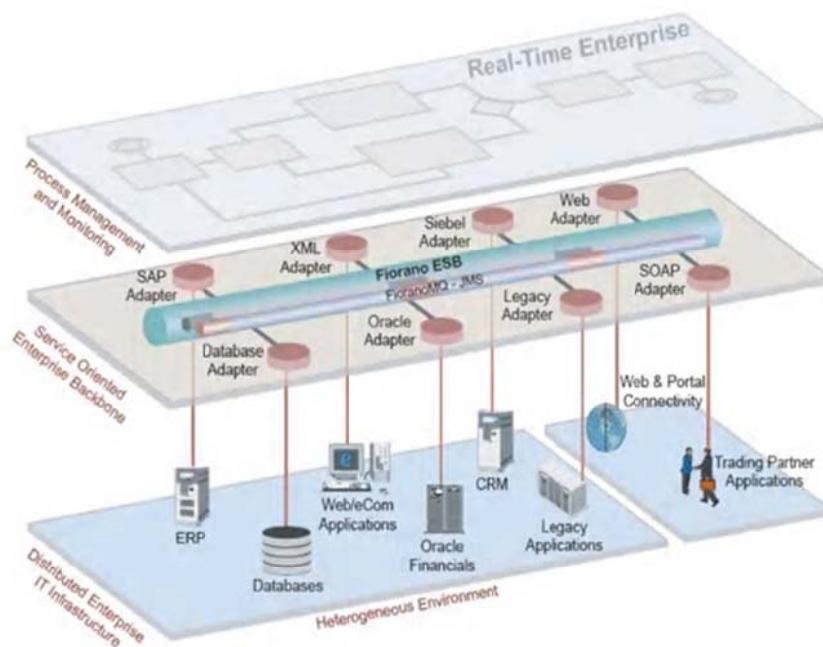


Fig. 24. Procesos de Negocio como Consumidores de Servicios

2.7 Conclusiones del Capítulo

Este capítulo presenta la evolución tanto cronológica como en complejidad de los sistemas distribuidos. El esquema de distribución planteado se corresponde tanto con lo funcional como con lo estructural, siendo la combinación de ambos tipos de distribución lo que hace más complejas las soluciones.

Los avances tecnológicos en términos de comunicaciones y el crecimiento en prestaciones de los sistemas operativos modernos, han sentado buenas bases para que las arquitecturas de software puedan extenderse y consolidarse.

En el próximo capítulo se presentan los pilares conceptuales más importantes de los sistemas operativos en entornos distribuidos.

CAPÍTULO 3

La Distribución y los Sistemas Operativos

*Matias Banchoff, Nicolás del Rio, Lía Molinari
y Juan P. Pérez*

Quizá el lector sea un avezado conocedor de sistemas operativos. Quizás no. Lo que podemos afirmar es que seguramente está familiarizado con ellos. La masividad en el uso de computadoras y celulares hizo que aún los más neófitos, los recién llegados, hayan oído hablar de Windows, de Linux, de Android, etc.

El sistema operativo de una computadora surge injustamente como el primer culpable de cualquier mal que la aqueje. Que si es lento, que no es suficientemente amigable, que no soporta esta aplicación, que hay que actualizarlo. Pero... es imprescindible para poder interactuar con el hardware y ejecutar los programas.

Por eso, en un acto de reivindicación, vamos a dedicarle este capítulo en el contexto de los sistemas distribuidos y analizaremos su rol trascendental.

3.1. ¿Qué es un sistema operativo?

Hace muchos años para definir a los Sistemas Operativos era suficiente hacerlo como un administrador de recursos de hardware y software. La interacción con el usuario no era una variable a considerar. Quienes utilizaban las computadoras eran especialistas. La élite de los albores: matemáticos, físicos, ingenieros.

Hoy cuando definimos los sistemas operativos lo hacemos desde dos miradas: como administrador de recursos y la facilidad de uso. La masividad y la ubiquidad de las TICs exigieron que los desarrolladores de los sistemas operativos dieran a la amabilidad un rol preponderante en las nuevas versiones.

En este libro, donde nos referimos a los sistemas distribuidos, cabe analizar cuál es el rol de los sistemas operativos en ellos, cuáles son los recursos a administrar, qué funciones debe llevar a cabo y las características que debe presentar ante la homogeneidad o heterogeneidad de los equipos conectados.

Un sistema distribuido es un conjunto de computadoras independientes que aparecen ante los usuarios como una única computadora. Las ventajas esperadas con respecto a velocidad,

confiabilidad, distribución, escalabilidad y extensibilidad no podrían lograrse si no existiera una entidad que administre los recursos y garantice su disponibilidad y sincronización.

La compartición de recursos es uno de los temas clave cuando se analiza el contexto de los sistemas distribuidos.

Ante estos escenarios donde interactúan varias CPU, una de las principales cuestiones es de qué forma lo hacen. Puede ser que tengan independencia e interactúen cuando lo precisen; que lo hagan mediante una red, o compartiendo memoria; que cada CPU tenga un rol definido o que entre varias lleven a cabo una tarea.

Este acoplamiento define el grado de integración.

El sistema operativo, como administrador de recursos, dará el soporte necesario de acuerdo a cómo se realiza esta integración.

Por lo tanto, una y otra vez en este libro aparecerá el término abstracción. Para soportar la heterogeneidad de recursos se desarrollan capas de software para integrarlos, que *abstraen* las características propias de estos recursos: redes, hardware, lenguajes de programación, e incluso, sistemas operativos. Una de esas capas de software es el middleware, sobre el cual nos referiremos en párrafos posteriores.

Podríamos incluso pensar en un sistema operativo como un software de abstracción. Por ejemplo, facilita la integración de diferentes dispositivos de almacenamiento o de comunicación. Gracias a él no necesitamos hablar de sectores de disco, o sockets. La masividad en el uso de la tecnología no hubiera sido posible sin la evolución de los sistemas operativos, ¿o es al revés?

Volvamos al concepto de pensar en un conjunto de computadoras interconectadas.

Puede ser que cada una de ellas tenga su propio sistema operativo y que puedan acceder a recursos remotos, por ejemplo, a un servidor de bases de datos.

En este esquema, cada nodo tiene autonomía en la gestión de sus recursos. Un usuario puede acceder a otro nodo y ejecutar allí un programa. Pero no hay planificación de procesos entre los distintos nodos.

Analicemos otra alternativa donde el usuario que quiere ejecutar un programa en otro nodo no tiene que conectarse a ese nodo, o no debe indicar dónde se ubica un recurso remoto, porque existe un nivel de transparencia que permite acceder a lo remoto casi como si fuera local.

Estas dos posibilidades son lo que se llama Sistemas operativos en red y sistemas operativos distribuidos.

El sistema operativo distribuido controla todos los nodos y puede enviar a ejecutar un programa en cualquier nodo de acuerdo a una política de planificación.

3.2. Hitos en la historia de los sistemas operativos

Cuando se describe la evolución de los sistemas operativos, es ineludible hablar de los mainframes. En la saga de libros de Sistemas Operativos de Avi Silberschatz, Peter Baer Gal-

vin y Greg Gagne, se utilizan los dinosaurios para representar la etapa de la informática donde los mainframes eran los reyes de la comunidad. Enormes, pesados, lentos pero sumamente seguros en su esquema fuertemente centralizado, aunaban software de base, ambientes de desarrollo y bases de datos del mismo proveedor, garantizando un cliente cautivo, obligado a adaptarse a cambios de versiones y consecuentes migraciones. La empresa proveedora tenía una fuerte injerencia en las decisiones tecnológicas de la organización para garantizar el servicio y la integración. El proveedor marcaba las tendencias.

Los modelos como los 360, 370, 390 de IBM, con sus sistemas operativos VM, VSE, MVS, Z/OS, o los provistos por Burroughs, Unisys u otros, o renovados soportando Linux, como Linux para Zseries en nuestros días. Mantienen sus adeptos entusiasmados por la posibilidad de procesamiento de transacciones a gran escala, el soporte de miles de usuarios y aplicaciones, el acceso simultáneo a los recursos, gran capacidad de almacenamiento, comunicaciones mediante gran ancho de banda. Y en forma muy destacada: seguridad, robustez.

A mediados de los 90, en el mercado argentino irrumpen los sistemas abiertos, como amenazante alternativa al monopolio de los mainframes.

El atributo de abierto surge de la posibilidad de combinar interoperabilidad, portabilidad y uso de computadoras de diferentes proveedores. Para ello se especifican interfaces, servicios y formatos.

Este nuevo contexto influyó en un cambio de capacidades y habilidades en el personal de administración de sistemas o soporte técnico. La integración de sistemas operativos, bases de datos, administración de redes y lenguajes revolucionó las hasta el momento bastante pasivas áreas de soporte, que sólo conocían los productos de un sólo proveedor, que además, definía el presente y el futuro de la infraestructura de la organización.

Esta etapa fue la que, sin intención, comenzó a preparar a los profesionales de TI en la revolución del software libre que la sucedió. Integración. Interoperabilidad. Colaboración.

El sistema operativo Unix deja de ser un producto para universidades y las grandes marcas revolucionan el mercado con AIX-IBM, HP-UX. La terminología que se utilizaba evidenciaba a qué bando se pertenecía: los de IBM *ipeleaban* (argentinismo que proviene de ejecutar el IPL, Initial Program Loader) y la gente de Unix, *booteaba* (argentinismo por ejecutar el proceso boot, de arranque).

3.3. Cliente-servidor: un importante paso hacia los sistemas distribuidos

Las arquitecturas cliente-servidor impusieron un modelo determinado por los servicios.

Este esquema es un modelo de aplicación distribuida, en el que las tareas o actividades se reparten entre un conjunto de procesos proveedores de recursos o servicios llamados servidores, y un conjunto de procesos demandantes de recursos o servicios llamados clientes. Si bien este esquema puede ser aplicado a procesos que se ejecuten en una misma computadora, el

mayor potencial se presenta al pensar a los mismos interactuando de manera remota sobre una red de computadoras.

Hoy en día podemos encontrar el modelo cliente/servidor en numerosas situaciones que utilizamos a diario: servicio de Correo Electrónico, acceso a la *World Wide Web*, aplicaciones en nuestros teléfonos móviles para compra de productos o servicios, servidores de bases de datos, entre muchas más.

En esta arquitectura la capacidad de proceso está distribuida entre los procesos clientes y los procesos servidores, lo que brinda una importante ventaja centralizando la gestión de la información y separando las responsabilidades en la organización de las aplicaciones.

La separación entre cliente y servidor es lógica. El servidor no necesariamente se ejecuta sobre una sola computadora ni exige la existencia de un único proceso. Esta característica permite mejorar los tiempos de respuesta.

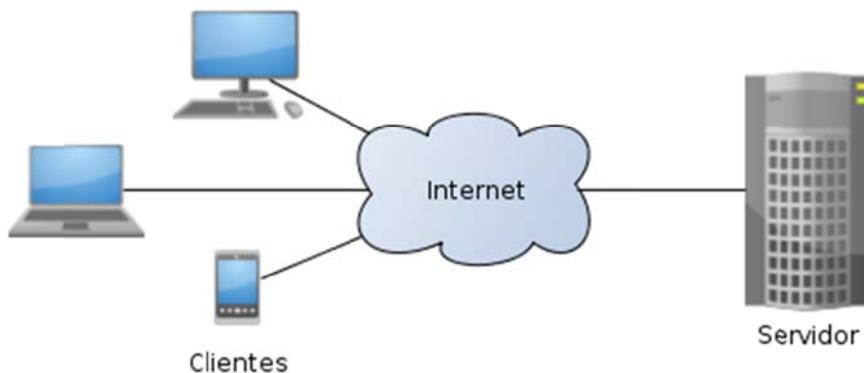


Fig. 25. Diferentes Clientes usando un Servidor

La arquitectura cliente servidor se basa en el modelo de comunicación requerimiento-respuesta (*request-reply protocol*). Esto es, la comunicación se basa en un mensaje del cliente al servidor (el requerimiento) y un mensaje del servidor al cliente (la respuesta). En este modelo, el cliente esperará desde que envía el mensaje hasta que recibe la respuesta al mismo.

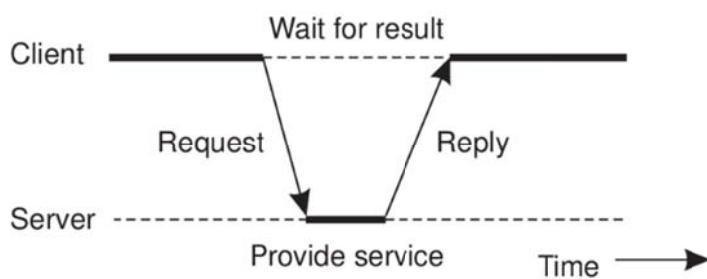


Fig. 26. Esquema del modelo de comunicación Requerimiento - Respuesta

Veamos algunas características de ambos tipos de procesos. Por un lado, los clientes:

- son los responsables de iniciar las solicitudes por lo que tienen un papel activo en la comunicación;

- esperan y reciben las respuestas del servidor;
- pueden conectarse a varios servidores a la vez para cumplir una tarea específica;
- normalmente interactúan con los usuarios finales mediante alguna interfaz gráfica de usuario o GUI (Graphical User Interface).
- En cuanto a los servidores:
- al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñando entonces un papel pasivo en la comunicación;
- ante la recepción de una solicitud, la procesan y luego envían la respuesta al cliente;
- pueden aceptar las conexiones de un gran número, aunque limitado, de clientes.

Comúnmente a la arquitectura tradicional cliente/servidor se la conoce como arquitectura de dos capas: una capa cliente y un servidor. En el siguiente gráfico podemos observar diferentes esquemas que se pueden presentar. Una alternativa plantea toda la lógica a cargo del servidor dejando únicamente la presentación de la información en el cliente. En otro caso el servidor sólo resulta responsable de los datos y el cliente lo es del resto de las actividades - presentación, gestión, etc., existiendo además esquemas alternativos entre los dos extremos mencionados.

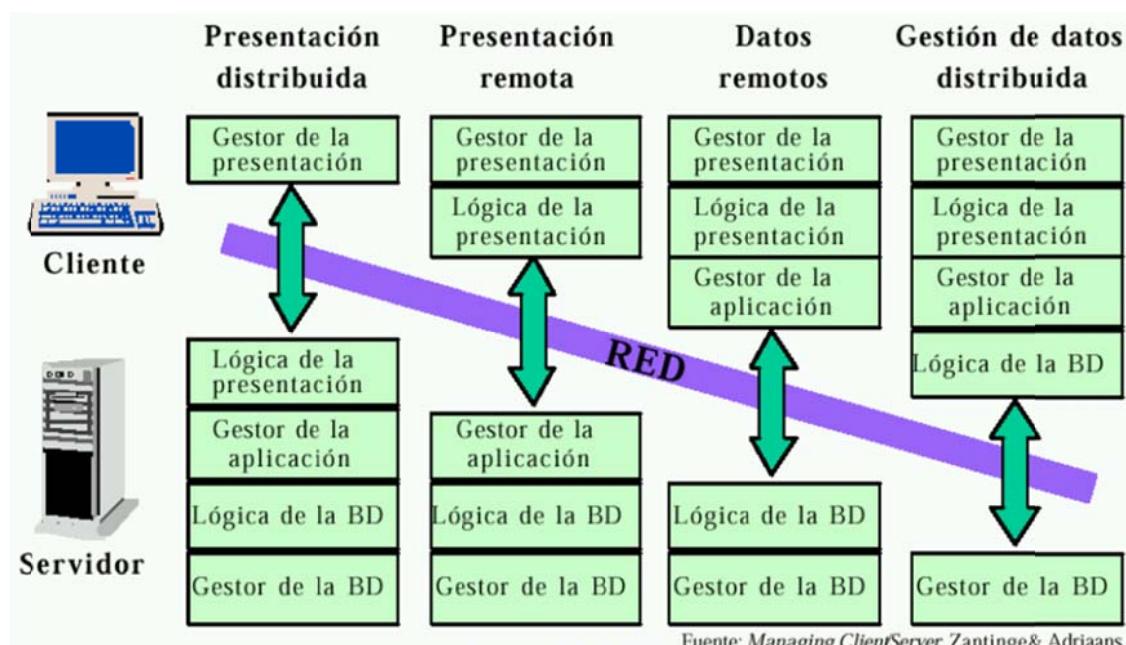


Fig. 27. Posibles organizaciones Cliente/Servidor en un modelo de Dos Capa

Otra posible organización es la conocida como de tres capas:

- Una capa cliente que interactúa con los usuarios finales.
- Una capa servidor de aplicación, la cual se encarga de procesar los datos para los clientes.
- Una capa servidor de base de datos, responsable de almacenar los datos y brindar los mismos a la capa servidor de aplicación.

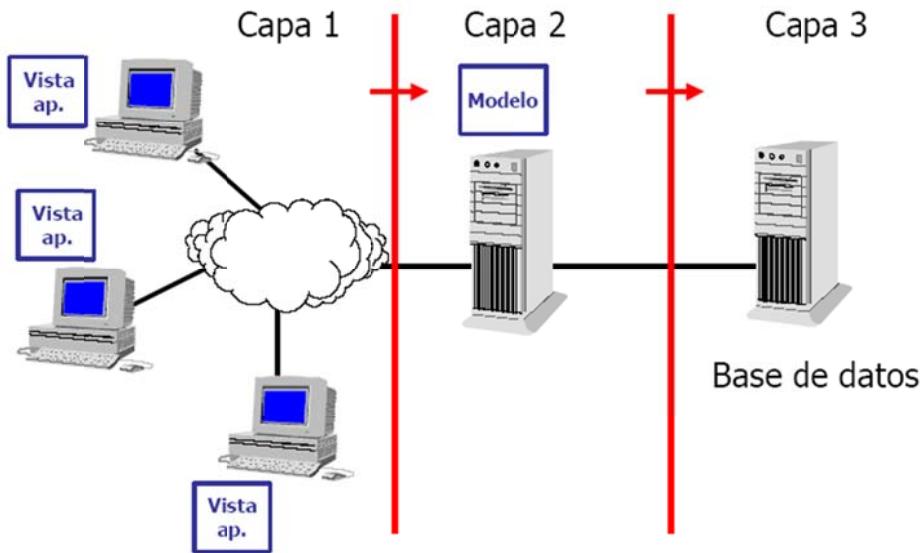


Fig. 28. Organización Cliente/Servidor de Tres Capas

La utilización de múltiples capas nos presenta la ventaja de obtener un mayor desacople en los pasos requeridos para dar una respuesta a un usuario, mejorando el balance de la carga de cada nodo participante y haciendo la solución más escalable. Las desventajas que se presentan las encontramos en la mayor carga producida en la red - por el aumento del tráfico de información - así como en la dificultad de desarrollar, probar y desplegar aplicaciones respecto a una solución de 2 o 3 capas.

De acuerdo con (Bochenksi 1994), son diez las características que definen un ambiente cliente/servidor. La existencia de las cinco primeras es de carácter obligatorio, mientras que las otras cinco es de cumplimiento opcional:

1. Una arquitectura cliente/servidor consiste de un proceso cliente y un proceso servidor que pueden ser distinguidos uno del otro y que pueden interactuar de manera independiente.
2. Las partes cliente y servidor pueden operar, aunque no necesariamente, en plataformas computacionales diferentes.
3. Tanto la parte cliente como la del servidor pueden ser actualizadas individualmente sin que la otra deba serlo también.
4. El servidor es capaz de dar servicio a múltiples clientes en forma concurrente. En algunos sistemas pueden acceder múltiples servidores.
5. Un sistema cliente/servidor incluye algún tipo de capacidad de red.
6. Una porción significativa (a veces la totalidad) de la lógica de la aplicación reside en el cliente.
7. El procesamiento es iniciado usualmente en el lado del cliente, no del servidor. Sin embargo, los servidores de bases de datos pueden iniciar acciones basadas en "disparos automáticos", "reglas del negocio" o procedimientos almacenados.
8. Una interfaz gráfica de usuario amigable generalmente reside en el lado del cliente.

9. La capacidad de un lenguaje estructurado de consultas es una característica de la mayoría de los sistemas cliente/servidor.
10. El servidor de base de datos debería proporcionar seguridad y protección a los datos.

Por su parte (Orfali 1994) resume así las características de los sistemas cliente/servidor:

- **Servicio:** El ambiente cliente/servidor conforma una relación entre procesos. El proceso servidor ofrece servicios, mientras que los clientes los solicitan.
- **Recursos compartidos:** Los servidores regulan el acceso a los recursos comunes por parte de los clientes.
- **Protocolos asimétricos:** Los clientes pueden pertenecer a una amplia variedad de tecnologías, y son los responsables por iniciar las solicitudes de servicios. Un servidor es un ente pasivo que se encuentra en espera permanente por dichas solicitudes.
- **Transparencia de localización:** Aun cuando un proceso servidor puede residir en el mismo equipo que los procesos clientes, es indispensable que los sistemas cliente/servidor oculten a éstos la localización física de los servidores, redireccionando apropiadamente las llamadas a los servicios requeridos.
- **Apertura:** El software cliente/servidor debe ser lo más independiente posible de las plataformas de hardware y sistemas operativos involucrados.
- **Intercambios basados en mensajes:** Los servidores y los clientes participan de sistemas *débilmente acoplados*, cuya relación se implementa con mecanismos de paso de mensajes (Message-Passing).
- **Encapsulación de servicios:** Los procesos servidores deben poder ser actualizados sin que esto provoque cambios en los clientes. Para lograr lo anterior, únicamente es necesario mantener sin alterar la interfaz de comunicación entre ambos componentes.
- **Escalabilidad:** El escalamiento de los sistemas cliente/servidor puede ser horizontal (adición o eliminación de clientes sin afectar significativamente el rendimiento global de un sistema) o vertical (crecimiento hacia configuraciones más grandes y eficientes).
- **Integridad:** Los servicios y datos de un servidor se ubican en un lugar centralizado, lo que simplifica su mantenimiento y protección.

Las principales ventajas que brinda la arquitectura cliente/servidor son:

- **Mantenibilidad:** la descomposición de sistemas rígidos y monolíticos hacia partes discretas intercomunicadas facilita el mantenimiento y reduce los costos. Como sucede con la mayoría de productos de la ingeniería, es más fácil dar servicio, reemplazar y arreglar componentes con interfaces bien definidas, que hacer el equivalente en unidades monolíticas.

- **Modularidad:** la arquitectura cliente/servidor está construida sobre la base de módulos conectables. Tanto el cliente como el servidor son módulos del sistema independientes uno del otro y pueden ser reemplazados sin afectarse mutuamente. Se agregan nuevas funciones al sistema ya sea creando nuevos módulos o mejorando los existentes.
- **Adaptabilidad:** el desacoplamiento del cliente y del servidor permite una rápida solución ante cambios del entorno del cliente, con modificaciones mínimas o nulas.
- **Escalabilidad:** las soluciones cliente/servidor pueden ser orientadas a satisfacer las necesidades cambiantes de la empresa.
- **Portabilidad:** actualmente el poder de procesamiento se puede encontrar en varios tamaños: super servidores, servidores, desktop, notebooks, máquinas portátiles. Las soluciones cliente/servidor basadas en estándares permiten a las aplicaciones estar localizadas donde resulte más ventajoso u oportuno.
- **Sistemas abiertos:** los sistemas cliente/servidor pueden desarrollarse bajo la premisa de sistemas basados en estándares de la industria.
- **Autonomía:** las máquinas cliente pueden presentar diversas configuraciones, tamaños, marcas y arquitecturas. Con una configuración adecuada, cada cliente puede trabajar en forma independiente o como parte de la red distribuida de la empresa.

Si bien vimos que esta arquitectura proporciona numerosas ventajas, no conforman la solución perfecta para las necesidades de administración de la información en una empresa, debido a que imponen también ciertas restricciones, entre las que encontramos (Price 1995):

- Si una parte importante de la lógica de las aplicaciones es trasladada al servidor, éste puede convertirse en un *cuello de botella* del sistema global. En este caso, los recursos limitados del servidor tienen una alta demanda por un número cada vez más creciente de usuarios.
- Las aplicaciones distribuidas, en especial aquellas basadas en un modelo cooperativo, son más complejas que las no distribuidas, e imponen cargas adicionales de comunicación y por ende de transferencia de información (datos del usuario y *overhead* del sistema).
- Se requiere de un alto grado de compatibilidad y de sujeción a estándares de parte de los dispositivos (hardware y software) que conforman un sistema cliente/servidor, para que éste pueda funcionar de una manera efectiva y transparente para el usuario.
- La mayoría de las herramientas cliente/servidor obligan a los usuarios a aprender esquemas de desarrollo de aplicaciones totalmente nuevos para muchos.
- La complejidad y el esfuerzo requerido para administrar y soportar un ambiente cliente/servidor grande basado en sistemas abiertos es un punto a considerar. Hay pocas herramientas bien reconocidas que soportan manejo de configuraciones, monitoreo del rendimiento y distribución de versiones de software.

Pero ¿qué pasa con los sistemas operativos en el caso que el cliente y el servidor funcionen en computadoras diferentes?

El sistema operativo de la máquina donde se ejecuta el cliente, posiblemente esté orientado a un uso pensado en el ámbito del consumidor de servicios, por ejemplo, un puesto de trabajo donde el usuario accede a servidores de correo, o a servidores de páginas web, o que usa aplicaciones cliente específicas de una aplicación determinada.

No obstante, hay sistemas operativos que permiten alojar un pequeño servidor web con algunas limitaciones.

Los sistemas operativos de las máquinas que ejecutan procesos servidores están orientadas a optimizar el servicio que se ofrece a los clientes. Estas computadoras habitualmente están siempre encendidas y tendrán características físicas orientadas a su servicio. Si es un servidor de archivos, por ejemplo, tendrá que tener capacidad de almacenamiento adecuada. Si es un servidor de una aplicación que se espera sea muy accedida, debe estar configurado y preparado para atender múltiples demandas concurrentes.

Si fuera accedido por el resto de las computadoras en una red, en este servidor se alojarían los servicios de dominio, de DNS, de DHCP, entre otros.

3.4. La nube y la grid

En párrafos anteriores se enunciaron los conceptos tradicionales en el marco de los sistemas operativos. Pero la evolución de la tecnología en las últimas décadas favoreció el surgimiento de nuevos modelos de computación distribuida. Es el caso de la computación en la nube (*cloud computing*) y computación en la *grid* (*grid computing*). La administración de recursos se hizo más compleja pues se agrega una arquitectura de software para lograr la abstracción que se pretende en la interacción con estos sistemas y el usuario.

El uso de *grid computing*, se inicia a principios de los 90s como respuesta a una necesidad planteada desde el mundo científico, facilitando la integración de recursos computacionales. Es una forma de computación distribuida que fue ganando adeptos en universidades, grandes centros de investigación y laboratorios.

Cloud computing, también llamada *computación en la nube*, es un paradigma que surge con el objetivo de ofrecer servicios de computación a través de Internet. Hasta aquí no queda muy claro cuál es la diferencia entre ambos modelos: al fin y al cabo, ambos ofrecen servicios por Internet.

Más allá de las diferencias planteadas por Foster et al, ambos son modelos de computación distribuida.

Un sistema operativo para la nube (OS *cloud computing*), también llamado sistema operativo virtual, está diseñado para administrar los recursos de este entorno, de acuerdo al modelo de prestación. La virtualización, de uso habitual en el contexto de la nube, también define

nuevas pautas para la creación, operación y mantenimiento de las máquinas virtuales, servidores e infraestructura.

Una forma de definir la *grid* es mediante un conjunto de *clusters* geográficamente dispersos comunicados por redes de alta velocidad y con una infraestructura de software para permitir la interacción e interoperación. Este software es llamado middleware.

¿Es el middleware un sistema operativo? Administra recursos y permite trabajar en forma remota con diferentes grados de abstracción. De acuerdo a la definición tradicional de sistemas operativos, podría entrar en esa categoría. Pero este middleware es complejo de configurar, exigiendo un importante grado de especialización, difícil de conseguir en ambientes no científicos. Además está orientado a administrar un subconjunto de recursos, no todos los recursos.

Sea el contexto que planteemos, la definición más amplia acerca de qué es un sistema operativo, lo generalizamos aceptando que es el software que administra los recursos de una manera eficiente y facilita su acceso con un alto grado de transparencia.

3.5. Una introducción a la virtualización

Virtualizar es un término que se ha incorporado en nuestro cotidiano. Es el mecanismo por el cual se logra aquello que *existe sólo aparentemente y no es real*.

La virtualización es un término directamente relacionado con la abstracción. Y no es un término ajeno al mundo de los sistemas operativos. En este ámbito, no es un algo nuevo.

La primera versión de un sistema operativo que soportaba máquinas virtuales es de 1972: VM/370 (*Virtual Machine Facility/370*). Pero el concepto ya se venía investigando en la década del 60.

La virtualización es una capa de abstracción sobre el hardware para obtener una mejor utilización de los recursos y flexibilidad. Permite que haya múltiples máquinas virtuales (MV) o entornos virtuales (EV), con distintos (o iguales) sistemas operativos corriendo en forma aislada. Cada MV tiene su propio conjunto de hardware virtual (RAM, CPU, NIC, etc.) sobre el cual se carga el SO huésped (*guest*). Virtualización se transformó en la solución para este mundo de integración que se describió en párrafos anteriores.

Ante situaciones tales como las que se enuncian a continuación, la virtualización se constituye en una solución:

- Existencia de muchas máquinas servidores, poco usados
- Convivencia con aplicaciones heredadas (*legacy*) que no pueden ejecutarse en nuevo hardware o sistema operativo
- Testeo de aplicaciones no seguras
- Ejecución con recursos limitados o específicos
- Necesidad de usar un hardware con el que no se cuenta
- Simulación de redes de computadoras independientes.
- Ejecución de varios y distintos sistemas operativos simultáneamente.

- Testing y monitoreo de performance
- Facilidad de migración.
- Ahorrar energía (tendencias de green IT, o tecnología verde)

Entre otras ventajas, la facilidad de uso se evidencia en el hecho de que las máquinas virtuales están encapsuladas en archivos, lo que las hace fácil de almacenar y de copiar. Sistemas completos (aplicaciones ya configuradas, sistemas operativos, hardware virtual) pueden moverse rápidamente de un servidor a otro.

La dinámica en el caso de los sistemas operativos consiste en un software anfitrión (host) y un software huésped (guest).

Algunas máquinas virtuales en realidad son emuladas. La emulación provee toda la funcionalidad del procesador deseado a través de software. Se puede emular un procesador sobre otro tipo de procesador. Tiende a ser lenta.

La virtualización simula un procesador físico en distintos contextos, donde cada uno de ellos corre sobre el mismo procesador.

En este contexto se hace imprescindible un “manejador/monitor de máquinas virtuales” (VMM, virtual machines monitor) lo que se llama un hipervisor (hypervisor).

En el ámbito de los sistemas operativos, un hipervisor es una plataforma de virtualización que permite múltiples sistemas operativos corriendo en un host al mismo tiempo. El hipervisor interactúa con el HW, realiza la multiprogramación y presenta una o varias máquinas virtuales al sistema operativo.

Hay distintos tipos de hipervisor:

- Hipervisor tipo 1. Se ejecuta en modo kernel. Cada VM se ejecuta como un proceso de usuario en modo usuario. Es decir que hay modo kernel virtual y modo usuario virtual.
- Hipervisor tipo 2. Se ejecuta como un programa de usuario sobre un sistema operativo host. Este host es quien se ejecuta sobre el HW.

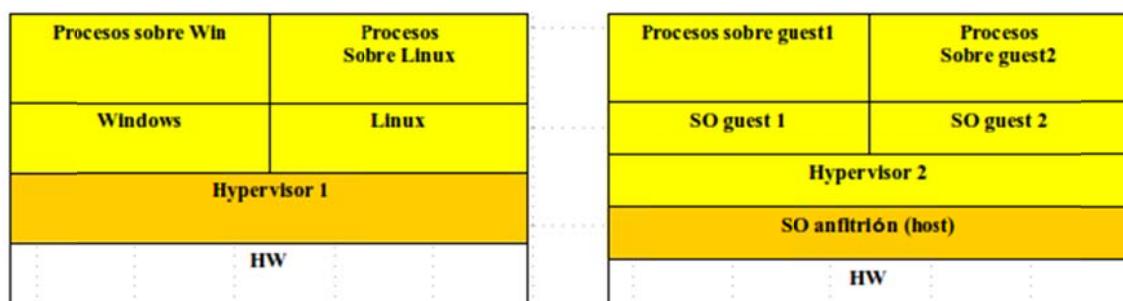


Fig. 29. Organización lógica de un Hipervisor Tipo 1 y uno tipo 2.

A continuación se listan las características de una máquina virtual, enunciadas en 1974 por Popek y Goldberg:

- Equivalencia / Fidelidad: un programa ejecutándose sobre un VMM debería tener un comportamiento idéntico al que tendría ejecutándose directamente sobre el hardware subyacente.
- Control de recursos / Seguridad: El VMM tiene que controlar completamente y en todo momento el conjunto de recursos virtualizados que proporciona a cada guest.
- Eficiencia / Performance: Una fracción estadísticamente dominante de instrucciones tienen que ser ejecutadas sin la intervención del VMM, o en otras palabras, directamente por el hardware.

Pero analicemos que ocurre a nivel de las instrucciones. Para ello merece hacerse un pequeño repaso.

Una interrupción es un evento que altera la secuencia de instrucciones ejecutada por un procesador. Diferenciamos interrupciones enmascarables de no enmascarables.

La interrupción enmascarable es ignorada por la unidad de control por el tiempo que permanezca en ese estado.

La interrupción no enmascarable será siempre recibida y atendida por la CPU. Sólo unos pocos eventos críticos (como fallos del hardware) pueden generar interrupciones no enmascarables.

Las excepciones son causadas por errores de programación o condiciones anómalas que deben ser manejadas por el *kernel*. Algunas son detectadas por el procesador y otras son programadas. Entre los tipos de excepción están los *traps*.

Recordemos además que generalizando en cuanto a modos de ejecución, en modo supervisor se puede ejecutar cualquiera de las instrucciones que acepta el procesador. En modo usuario, no.

Veamos un ejemplo. En VM/370, hay una máquina CMS (*Conversational Monitor System*) para cada usuario, con “la ilusión” del hardware completo. Una aplicación sobre CMS hace una *system call*, y CMS la “atrapa” (es un *trap*). Y se lo comunica a la MVV.

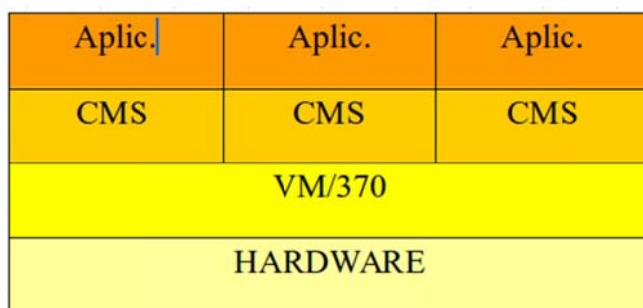


Fig. 30. Organización lógica del IBM VM/370

En la virtualización, el sistema operativo *guest* (ejecutándose en modo usuario) emitirá una instrucción privilegiada que no debe ser ignorada.

INTEL 386 ignoraba las instrucciones privilegiadas que se invocaban en modo usuario. Las máquinas virtuales se ejecutan en modo usuario. Cuando se ejecuta una instrucción privilegiada no debe ignorarse, y debe ser captada por la VMM. Las instrucciones no privilegiadas deben

ejecutarse nativamente (es decir, eficientemente). Popek y Goldberg hicieron un importante aporte al mundo de la virtualización con la definición de instrucciones sensibles e instrucciones privilegiadas. Las instrucciones sensibles siempre generan una excepción y pasan el control al VMM. Las instrucciones privilegiadas generan una interrupción.

Una arquitectura es virtualizable si todas las instrucciones sensibles son privilegiadas.

Cuando se inicia un guest, se ejecuta allí hasta que haga una excepción, que debe ser *atra-
pada* por el hipervisor.

En el hipervisor tipo 1 cuando la VM ejecuta una instrucción sensible, se produce un trap que procesa el hipervisor.

En los modelos distribuidos que se utilizan hoy en día, la virtualización se ha transformado en un elemento clave, por sus características de aislamiento, fácil administración y ahorro de energía, entre otras ventajas.

3.6. Servicios básicos y extendidos de un sistema operativo distribuido

Cuando pensamos acerca de los servicios básicos y extendidos que provee un sistema operativo, surgen los conceptos de administración de memoria, CPU y hardware como servicios básicos y por ejemplo autenticación y autorización de usuarios como servicios extendidos (entre otros). En entornos distribuidos muchos de los servicios que se consideran básicos para un sistema operativo ya se asumen satisfechos por cada nodo, mientras que surgen otros que resultan de vital importancia con el fin de interoperar con el resto de los nodos que componen el sistema.

En un sistema distribuido, cada nodo posee su propio espacio de memoria, sus dispositivos de entrada salida y sus planificadores. Inclusive, cada nodo representa y administra los datos y recursos, de acuerdo a la arquitectura de hardware y sistema operativo que corra en cada uno. Es de este último concepto, conocido como heterogeneidad que surge una de las premisas más importantes que deben respetar los servicios en entornos distribuidos. Los sistemas deben proveer las correspondientes especificaciones con el fin de poder definir interfaces comunes que permitan construir sobre ellas los servicios que el sistema distribuido brindará. Esto último permitirá lograr la visión de sistema único que interopera con un fin específico.

Dentro de los servicios básicos de un sistema operativo distribuido se pueden considerar:

- **Servicio de comunicación:** “La diferencia más importante entre un sistema distribuido y un sistema de un único procesador es la comunicación entre procesos” (Tannenbaum, 2007). En los sistemas operativos tradicionales, la comunicación entre procesos se realiza a través del pasaje de mensajes asumiendo áreas de memoria compartida. En los sistemas distribuidos, no existe la memoria compartida, ya que cada nodo posee su propia área de memoria que gestiona de modo independiente. La comunicación entre procesos, resulta ahora más compleja y requiere de meca-

nismos que prevean la distribución e inclusive la heterogeneidad. Dichos mecanismos se encuentran plasmados en reglas conocidas como protocolos de comunicación. En líneas generales, cuando un proceso A desea comunicarse con un proceso B residente en otro nodo, el primer proceso construye el mensaje en su propio espacio de direcciones, ejecuta una llamada al sistema para indicarle a su sistema operativo que envíe el mensaje a través de la red y se espera que el proceso B en el nodo remoto reciba los datos. En cada uno de los pasos mencionados, el mensaje atraviesa distintas capas de software que se encargan de *normalizar* el mensaje, de modo tal que pueda ser transmitido y recibido por el destinatario de forma íntegra. Más adelante, en este mismo capítulo, se analizará en detalle el sistema de comunicación y su analogía con las distintas capas de software.

- **Servicio de sincronización:** “Además de la comunicación, es fundamental la forma en que los procesos cooperan y se sincronizan entre sí” (Tanenbaum, 2007). En los sistemas operativos tradicionales, la sincronización es llevada a cabo por el sistema base en un punto centralizado que puede ser consumido por todos los procesos. En los sistemas distribuidos, cada nodo posee su propio reloj, por lo cual surge la necesidad de contar con un mecanismo de sincronización global. Una primera aproximación a la sincronización de relojes en entornos distribuidos, es la que se conoce como *sincronización de relojes físicos*. En esta técnica, el reloj de cada nodo se sincroniza contra un punto central conocido como servidor de hora. De este modo, todos los sistemas operan bajo el mismo esquema horario. Si bien esta aproximación provee buenos resultados, cuando es necesario sincronizar una gran cantidad de nodos, es muy común la producción de desfasajes.

Otra técnica es la utilización de *relojes lógicos*. En esta última, no es necesario contar con una sincronización exacta de los relojes, sino que es más relevante contar con un orden de ocurrencia de los eventos. De este modo, se define la relación *sucedió antes*, que permite determinar que si dos procesos se encuentran relacionados, un evento de uno de ellos suceda antes que otro.

- **Servicio de gestión distribuida de procesos:** La gestión de procesos es una función básica de cualquier sistema operativo. El mismo realiza la gestión en forma centralizada. Sin embargo, cuando se trata de realizar la gestión en un entorno distribuido, existen distintas consideraciones que deben tenerse en cuenta. En un sistema distribuido, los procesadores virtuales se encuentran divididos en los nodos que componen al mismo. Para la ejecución de los procesos, es necesario organizar los procesadores de acuerdo a la arquitectura elegida. Existen tres arquitecturas posibles para realizar la gestión:

- **Modelo de estaciones de trabajo:** En este esquema, cada proceso se ejecuta en un nodo, mientras que se utilizan sistemas de archivos compartidos para compartir los datos. Este modelo puede provocar que ciertos nodos tengan mucha carga, mientras que otros puedan estar ociosos.

- **Pila de procesadores:** Cada vez que se debe ejecutar un proceso, el mismo es agregado a una pila de donde se seleccionan para su ejecución. De este modo cada nodo que posea recursos para realizar alguna tarea, seleccionará los procesos de la pila global para ser ejecutados.
- **Modelo híbrido:** Los trabajos interactivos son ejecutados bajo el modelo de estaciones de trabajo, mientras que los no interactivos se ejecutan bajo el modelo de pila de procesadores.

En todos los casos es responsabilidad del sistema distribuido realizar una correcta gestión de la carga de trabajo con el fin de optimizar el uso de los recursos.

- **Servicio de memoria compartida distribuida:** En un sistema distribuido, la memoria es gestionada por cada nodo independientemente, pero se comparte lógicamente en el sistema. Cada nodo ejecuta los procesos en su memoria física privada, pero puede consumir recursos de otros nodos que comparten sus espacios de memoria. La forma de acceder a áreas de memoria compartida de otros nodos es a través del pasaje de mensajes. Cada nodo de la red aporta una porción de su memoria local para construir lo que se conoce como el espacio global de direcciones virtuales, el cual será utilizado por los procesos que se ejecuten en el sistema distribuido. El servicio de memoria compartida distribuida se encargará de interceptar las referencias a memoria que realice cada proceso y satisfacerlas ya sea local o remotamente. Los accesos serán enmascarados por el servicio de modo tal de lograr la transparencia y garantizar la heterogeneidad.

Con el fin de disminuir la sobrecarga en la red por la transferencia de locaciones de memoria, es posible implementar técnicas de caching de las páginas más accedidas. Si bien esta técnica provee mejoras, deben implementarse mecanismos adecuados para evitar incoherencias de los datos. Al momento de realizar actualizaciones de una página, deberá implementarse un protocolo que permita replicar dichos cambios en todas las copias cacheadas localmente por los nodos.

Dentro de los servicios extendidos de un sistema distribuido pueden considerar:

- **Servicio de nombres y localización:** En los sistemas distribuidos, los nombres son utilizados para referirse a recursos como por ejemplo nodos, servicios, usuarios, etc. Los nombres facilitan la comunicación y la compartición de recursos. En líneas generales para que un proceso pueda compartir información con otros, el mismo debe estar nombrado de modo tal que otros puedan localizarlo y accederlo. El ejemplo más común de servicio de nombres es el servicio DNS (Domain Name Service). Se trata de un sistema de nomenclatura jerárquica que permite dar nombre a entidades y realizar búsquedas siguiendo un esquema estructurado.
- **Servicios de Seguridad:** El concepto de seguridad es amplio y abarca a los sistemas distribuidos principalmente en dos aspectos. El primero, referente a la comunicación, tiende a asegurar la comunicación entre procesos residentes en distintos

nodos. La seguridad en la comunicación de procesos que residen localmente en un nodo es llevada a cabo por el sistema operativo. En entornos distribuidos es necesario garantizar la confidencialidad e integridad de los datos transmitidos a través de la red. El segundo aspecto prevé la autorización, la cual debe garantizar que un proceso tenga acceso únicamente a los recursos que le fueron concedidos.

3.7. Conceptos de sistema operativo de red

Cuando nos referimos al término de sistema operativo de red, estamos hablando de una clase especial de software que permite que una red de nodos de computación pueda interoperar entre sí. Utilizando dicho software, los nodos pueden compartir información y recursos.

Dentro de sus funciones principales se encuentran la conexión de los equipos, periféricos y demás dispositivos de red; coordinar las funciones de los nodos y controlar el acceso a los datos y elementos. El sistema operativo de red determina la forma de compartir y acceder a los recursos que gestiona. Se encuentra estrechamente ligado a la arquitectura de red (cliente servidor o trabajo en grupo).

Hoy en día es muy común hablar de sistemas operativos que trabajan en red, y a ellos se los conoce como sistemas operativos de red. Es el caso común de los sistemas operativos Microsoft Windows o GNU/Linux. Netware de Novell, es un sistema operativo de red donde el software de red del equipo cliente se integra al sistema operativo del equipo.

Los sistemas operativos de red se desarrollaron en su gran mayoría siguiendo la arquitectura cliente - servidor. A nivel general, existen dos modelos de arquitectura para su construcción:

- **Modelo de acceso remoto:** Este modelo ofrece a los clientes una forma de acceder transparentemente a los recursos gestionados por algún servidor remoto. Debido a que los clientes desconocen la ubicación de los recursos, el sistema operativo provee mecanismos de localización e interacción con los servicios. La característica más importante de este modelo, es que los datos residen en el servidor remoto y son invocados por los clientes. Un ejemplo de este modelo es el protocolo SSH para acceder remotamente a una línea de comando, o TELNET.
- **Modelo de carga y descarga:** En este caso, el cliente realiza la conexión con el servidor con el fin de descargar los datos del mismo y procesarlos localmente. En la máquina cliente se genera una copia de los datos obtenidos desde el servidor. Una vez que el cliente terminó de utilizar los datos necesarios, vuelve a realizar la carga en el servidor para que las modificaciones se vean reflejadas. Un ejemplo típico de este modelo es el protocolo FTP.

3.8. Relación entre los sistemas distribuidos y las pilas de OSI y TCP/IP

Existen varias definiciones para *Sistema distribuido*, pero en todas se hace referencia a una colección de computadoras conectadas entre sí y apareciendo como un único sistema de cara al usuario. Esta caracterización implica que los sistemas distribuidos presentan ciertas propiedades, como transparencia por ejemplo; más aún, dado que un sistema distribuido se compone de varias partes, cada una realizando una función particular, es posible vislumbrar un paralelismo entre los sistemas distribuidos como un todo y las pilas de red OSI y TCP/IP. Tanto es así que el propósito de esta sección es establecer una analogía entre los sistemas distribuidos en general y los modelos de referencia OSI y TCP/IP.

A modo de repaso, recordemos que el modelo OSI sirve de referencia a la hora de hablar acerca de protocolos de red. Por ejemplo, cuando se dice que IP es *un protocolo de capa 3*, estamos haciendo referencia al modelo OSI y situando a IP en la Capa de Red. Sin embargo, aunque el modelo OSI sea muy conocido y utilizado como referencia, la pila de red más usada actualmente es la denominada TCP/IP (en referencia a los dos protocolos más importantes que la componen).

Tanto TCP/IP como OSI están pensados en términos de *capas* o *layers*, de modo que una capa brinda servicios a la siguiente. Por ejemplo, en TCP/IP, la capa de aplicación usa los servicios brindados por la capa de transporte. Análogamente, los sistemas distribuidos están pensados para que una componente pueda consumir servicios de otras. Esta es otra similitud que sirve para elaborar una analogía entre las pilas de red y los sistemas distribuidos.

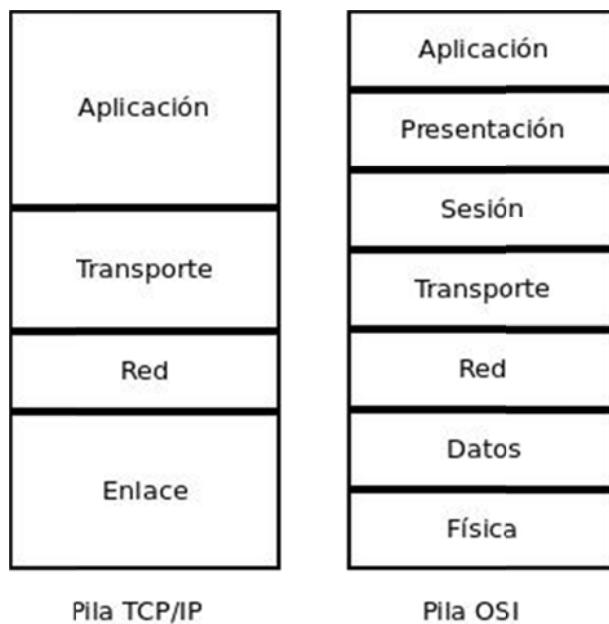


Fig. 31. Capas OSI y TCP

3.9. Sobre las propiedades de un sistema distribuido

Los sistemas distribuidos presentan varias propiedades o características, como son transparencia, apertura (*openness*) y escalabilidad.

Transparencia

Se busca que un sistema distribuido sea transparente para ocultar el hecho de que los recursos y procesos que lo componen se encuentran repartidos entre varias componentes, de modo que los usuarios ven al sistema como un todo. Por ejemplo, cuando un usuario realiza una consulta de DNS, son varias las componentes involucradas en devolver una respuesta (aunque el usuario no percibe esa complejidad).

De forma similar operan las pilas OSI y TCP/IP. En este caso la transparencia se da mediante el encapsulamiento de los datos. De esta manera los datos enviados por una aplicación se rutean hacia su destino, sin que la aplicación conozca las rutas e intermediarios transitados.

Existen varios sentidos en los cuales un sistema puede ser transparente:

- **Acceso:** La transparencia en el acceso implica ocultar la representación de los datos y la manera de accederlos. Por ejemplo, cuando un usuario hace una consulta de DNS, no sabe el formato en el que se almacenan los datos solicitados ni cuáles son los servidores encargados de generar la respuesta, pero de todas maneras recibe una.

En el caso de los modelos OSI y TCP/IP, las capas inferiores abstraen a las superiores de los pormenores a la hora de acceder a los diferentes medios de comunicación en uso. Por ejemplo, un servidor web que opera en la capa de aplicaciones no sabe si está brindando servicio sobre una red WiFi o cableada, porque las capas inferiores de la pila realizan la abstracción necesaria.

- **Locación:** Se dice que existe transparencia de locación o localidad cuando los recursos pueden residir en cualquier lugar sin que esto afecte a la calidad y/o funcionamiento del servicio. En otras palabras, el usuario no sabe dónde está físicamente el recurso en el sistema.

Algo similar ocurre con los dispositivos conectados a una red. Y esta transparencia de locación la brinda la capa de Red (El protocolo IP en el caso de la pila TCP/IP). Obviamente aquí se asume que la latencia del medio de comunicación es aceptable para el tipo de servicio que se quiere brindar.

- **Concurrencia:** En el contexto de concurrencia transparente se espera que el usuario no perciba el hecho de que un mismo recurso puede estar siendo accedido y usado por otros clientes al mismo tiempo.

La analogía con las distintas capas de los modelos OSI y TCP/IP es bastante obvia: las distintas capas arbitran el acceso a los recursos y su utilización. Por

ejemplo, la capa de acceso al medio se encarga de multiplexar el tráfico y arbitrar el acceso al medio.

- **Fallas:** Se busca que la aparición y recuperación de fallas sea transparente al usuario. Por ejemplo, si un servidor de DNS falla, automáticamente se pregunta a algún otro servidor que tiene información para la zona en cuestión.

Análogamente, IP fue pensado para lidiar con fallas en los enlaces, de modo que un paquete puede transitar por rutas alternativas. De todas maneras, IP no asegura la entrega del paquete, es un *protocolo de mejor esfuerzo*. Por otro lado, TCP sí asegura la entrega de todos los datos, o la certeza de que no se pudieron entregar.

Apertura

Por otro lado, cuando se habla de la apertura (*openness* en inglés) de un sistema, se está haciendo referencia al hecho de que el sistema brinda servicios según reglas cuya sintaxis y semántica están bien definidas. A modo de ejemplo se puede tomar la Web, donde existen clientes y servidores web que *hablan* un *idioma* cuya sintaxis y semántica está bien definida: el protocolo HTTP.

Una consecuencia importante de la existencia de estos estándares es la posibilidad de implementarlos sobre diversas plataformas, dando lugar a sistemas distribuidos heterogéneos: No es necesario que todas las componentes corran sobre el mismo hardware, sistema operativo y usen las mismas librerías y lenguajes de programación.

Esta misma noción de servicios bien definidos es lo que permite crear fácilmente nuevos protocolos en las distintas capas de TCP/IP. Por ejemplo, dado que los protocolos UDP y TCP están bien definidos, es relativamente fácil crear protocolos de capa de aplicación que los utilicen.

Escalabilidad

La última propiedad nombrada implica que los sistemas distribuidos deben ser capaces de crecer y contraerse sin perder calidad en el servicio ofrecido: Escalabilidad.

Esta propiedad se puede medir de diferentes maneras. Por un lado, se puede hablar de escalabilidad refiriéndose al tamaño del sistema (cantidad de nodos, datos, usuarios, enlaces). También se habla de escalabilidad geográfica cuando el sistema continúa siendo viable sin importar la lejanía geográfica entre sus usuarios y componentes. Y por último, existe el concepto de escalabilidad administrativa, que implica que las organizaciones se pueden adherir al sistema sin que la calidad del servicio brindado disminuya.

De forma implícita, se espera que la usabilidad y administrabilidad del servicio no disminuya a medida que este crece.

Un ejemplo clásico es el servicio de DNS. La escalabilidad en DNS se da agregando más zonas y servidores al sistema; estos servidores están repartidos geográficamente alrededor del mundo y su administración está delegada a las organizaciones de las zonas correspondientes.

Aquí la analogía puede hacerse con la inclusión de nuevos protocolos a la pila. Por ejemplo, cuando se creó la pila TCP/IP, el protocolo HTTP no existía; sin embargo, fue fácil incluirlo en la capa de aplicaciones.

Otra manera de pensar la escalabilidad en TCP/IP es en la cantidad de nodos que componen el sistema. Las estadísticas⁸ muestran el crecimiento de los usuarios de Internet (Y por ende de los usuarios del protocolo IP) a lo largo de los años y está claro que el protocolo fue capaz de soportar ese crecimiento.

3.10. Analogía basada en los servicios

Existen varios modelos que describen la interacción entre las distintas partes de un sistema distribuido. Uno de estos modelos se conoce como Cliente-Servidor, ya descripto en este capítulo. Muchos de los sistemas usados diariamente están implementados según el modelo de Cliente-Servidor. Algunos ejemplos de esta clase de protocolos son HTTP, DNS, SMTP y SSH.

Como se dijo en el párrafo anterior, la modalidad de la interacción consiste en que los clientes envían solicitudes a los servidores, los servidores realizan alguna operatoria para satisfacer la solicitud, y luego envían la respuesta al cliente. De forma similar, cada capa de la pila de red da un servicio a la capa que está inmediatamente encima.

Por definición un sistema distribuido se compone de varias partes y servicios. Es necesario algún mecanismo para referenciar cada una de esas partes. Para esto se utilizan nombres e identificadores.

De forma similar existen nombres e identificadores en las distintas capas de los modelos OSI y TCP/IP. Por ejemplo, la IP de un dispositivo puede verse como un nombre (pero no como un identificador). La MAC sirve al mismo propósito pero en una capa inferior. Así, también, los puertos sirven para referenciar servicios y aplicaciones en un mismo host.

3.11. Conclusiones

En los modelos distribuidos se hace evidente desde lo expuesto anteriormente la idea recurrente de *Divide y vencerás*. Las pilas de red están divididas en capas y los sistemas distribuidos exitosos están divididos en partes.

⁸ https://en.wikipedia.org/wiki/List_of_countries_by_number_of_Internet_users

Nótese que incluso dentro de una misma capa, como es el caso de la capa de transporte en TCP/IP, existe una división más, principalmente entre los protocolos TCP y UDP (pensado cada uno para tareas distintas). En definitiva, se tiende a evitar los llamados *sistemas monolíticos*.

Se destaca a la aceptación masiva de TCP/IP y de los sistemas distribuidos tomados como ejemplo (DNS, WEB, Mail). Las causas de esta aceptación son:

- Interfaces bien definidas. En otras palabras, la *apertura* de los sistemas y protocolos. Esto permitió implementar tanto la pila TCP/IP como los diferentes servicios en varias plataformas.
- La posibilidad de ejecutarse sobre varias plataformas. Aquí debe notarse que el hecho de que TCP/IP esté implementado en capas vuelve más fácil su aceptación en nuevos medios de comunicación.

Por último, no debe descuidarse la importancia de la *escalabilidad administrativa*, es decir, la posibilidad de que varias organizaciones administren partes del mismo sistema. Esta característica hizo posible el rápido crecimiento de servicios como DNS y web, y la aparición de nuevos protocolos, principalmente de capa de aplicación.

¿Qué nos espera en el futuro acerca de los sistemas operativos? Nuevas tecnologías, como Internet de las Cosas (Internet of things) exigen nuevos paradigmas, contextos ágiles tanto en desarrollo como en comunicación, sistemas embebidos. Y por lo tanto esa administración de recursos, función fundamental de los sistemas operativos, tendrá que acompañar el mundo dinámico, ágil, sin descuidar seguridad en sus tres criterios: confidencialidad, integridad y disponibilidad. Un verdadero desafío.

CAPÍTULO 4

Cientes y Servidores en Sistemas Distribuidos

Patricia Bazán

Como hemos mencionado en el Capítulo 2, la arquitectura Cliente/Servidor o arquitectura de 2 niveles, identifica dos componentes que trabajan coordinadamente para dar la visión de un sistema único. Se diferencia de otras variantes de sistemas distribuidos en los siguientes conceptos:

Servicio - La relación cliente/servidor es la relación primaria entre dos procesos que se ejecutan en computadoras separadas. El servidor provee servicios y está a la escucha de requerimientos, mientras que el cliente solicita y consume servicios.

Recursos Compartidos - Los recursos compartidos en un sistema distribuido son administrados por los servidores que regulan el acceso a los mismos por parte de los clientes.

Protocolos Asimétricos - Hay una relación muchos a uno entre clientes y servidores. Por su parte la relación es asimétrica dado que los clientes siempre inician el diálogo y los servidores esperan los requerimientos.

Transparencia de la Ubicación - El servidor es un proceso que puede residir en la misma computadora del cliente o en otra computadora de la red, siendo transparente para el cliente dicha ubicación.

Comunicación basada en pregunta-respuesta - Clientes y servidores son procesos poco acoplados que interactúan a través de mensajes de tipo pregunta-respuesta, pudiendo ser estos sincrónicos o asincrónicos.

Escalabilidad - Los sistemas Cliente/Servidor pueden crecer moderadamente de manera horizontal (agregando clientes) o de manera vertical (potenciando al servidor).

Estas características aportan una distribución inteligente de funcionalidades a través de la red y otorgan un marco para el desarrollo de aplicaciones poco acopladas.

Por otra parte, para poder crear la visión de sistema único que requieren los sistemas distribuidos, es preciso contar con una infraestructura tecnológica que soporte cada una de las funciones.

La infraestructura red y comunicaciones ha crecido vertiginosamente en los últimos 20 años permitiendo un grado de interconexión global con alto ancho de banda y medios de transporte seguros y confiables.

El desafío de la industria informática es como agregar por encima de esta infraestructura física la arquitectura Cliente/Servidor que permitan unir las piezas del rompecabezas. Este desafío obliga a proveer:

- Protocolos de transporte que soporten el intercambio entre redes de computadores en forma confiable.
- Sistemas operativos de red que garanticen seguridad y privacidad y faciliten a los programas la tarea de encontrar los servicios que necesitan consumir
- Bases de datos para almacenar, recuperar y organizar grandes volúmenes de información incluso multimedial.
- Sistemas GroupWare que permitan el intercambio persona a persona y conferencias grupales.
- Agentes inteligentes que ayuden a los humanos a organizar sus actividades en el *ciberespacio*
- Plataformas de gerenciamiento de sistemas distribuidos que permitan mantener todo unido día a día.

El capítulo se organiza de la siguiente manera: en la Sección 1 se describen los roles y funciones de un componente servidor y sus variantes. En la Sección 2 se caracteriza el componente cliente y se describen algunas tecnologías asociadas. En la Sección 3 se describe la tecnología Cliente/Servidor con Java, detallando en la Sección 4 la tecnología JEE en entornos distribuidos. En la Sección 5 se presenta el patrón de diseño MVC a la luz de la tecnología Web. En la Sección 6 se arriba a algunas conclusiones.

4.1. Roles y Funciones del Servidor

El rol de un programa servidor es servir a múltiples clientes que requieren acceder a recursos compartidos que son propiedad del servidor. Veamos en detalle las funciones principales de un programa servidor:

- **Espera por requerimientos iniciados por el cliente** - Un programa servidor gasta gran parte de su tiempo esperando pasivamente requerimientos de los clientes que llegan generalmente en forma de mensajes arribando sobre una sesión de comunicación. Algunos servidores asignan sesiones dedicadas a cada cliente y otros crean un pool dinámico de sesiones reusables. Otros utilizan una combinación de ambos métodos. Lo cierto es que el servidor debe garantizar pronta respuesta a los requerimientos del cliente independientemente del mecanismo.
- Ejecuta muchos requerimientos a la vez - Un programa servidor que no soporta multitarea corre el riesgo de tener “en espera” a un cliente monopolizando recursos. El servidor debe ser capaz de servir concurrentemente a múltiples clientes protegiendo la integridad de los recursos compartidos.
- Atiende primero a los cliente VIP - El programa servidor debe ser capaz de proveer distintos niveles de prioridad en los servicios a sus clientes. Claramente un servicio batch tendrá prioridad más baja que un servicio de tipo interactivo.

- Inicia y corre tareas en segundo plano (background) - El programa servidor debe ser capaz de lanzar tareas que corren en background y que no están relacionadas con la tarea principal que se está ejecutando.
- Mantiene la corrida - Un programa servidor es una aplicación de misión crítica por lo cual debe garantizar un mantenimiento de su corrida y es tolerante a fallas.
- Crece a lo ancho y a lo largo - Un programa servidor tiene un apetito insaciable de memoria y capacidad de procesamiento por lo tanto debe ser escalable y modular.

Variantes de Servidores

Los sistemas Cliente/Servidor se clasifican en función del tipo de servicio que proveen los servidores, es decir, cuál es el recurso que administran. Estos recursos pueden ser archivos, datos, objetos, elementos de comunicación entre personas o transacciones.

- Servidores de Archivos - El recurso administrado es un archivo, concebido como conjunto de registro. El cliente solicita un archivo y el servidor responde enviando, a través de la red, bloques de registros. Es el mecanismo más primitivo para acceder

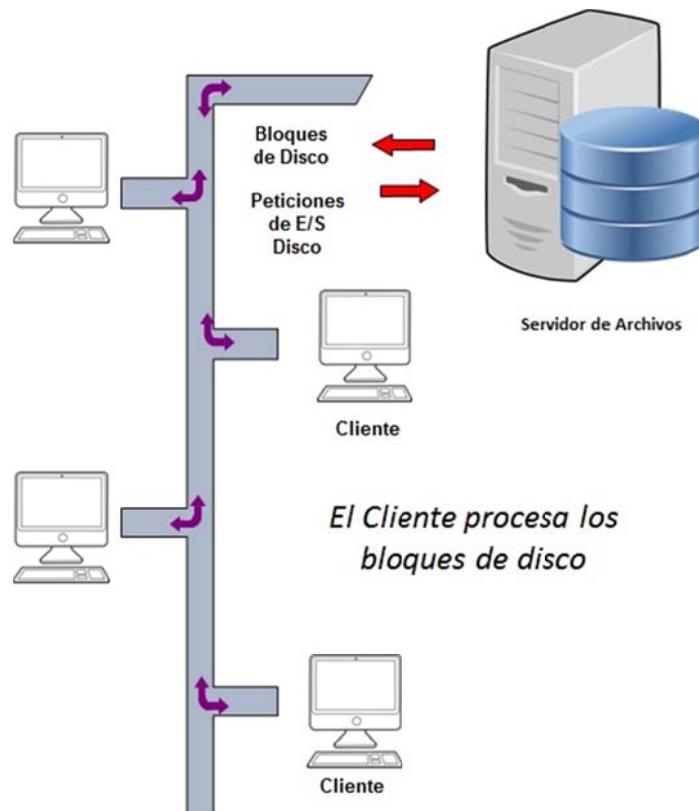


Fig. 32. Funcionamiento de un Servidor de Archivos

los datos dentro de un sistema de información. La Figura 32 muestra la comunicación de este tipo de servidores con sus clientes.

- Servidores de Base de Datos - El cliente envía requerimientos SQL como mensajes a un servidor de base de datos. El resultado de cada comando SQL es retornado a través de la red. La diferencia con el esquema anterior es que es el servidor el que procesa el mensaje y retorna los datos requeridos en lugar de retornar registro por registro como en un esquema servidor de archivos. La Figura 33 muestra la comunicación de este tipo de servidores con sus clientes.

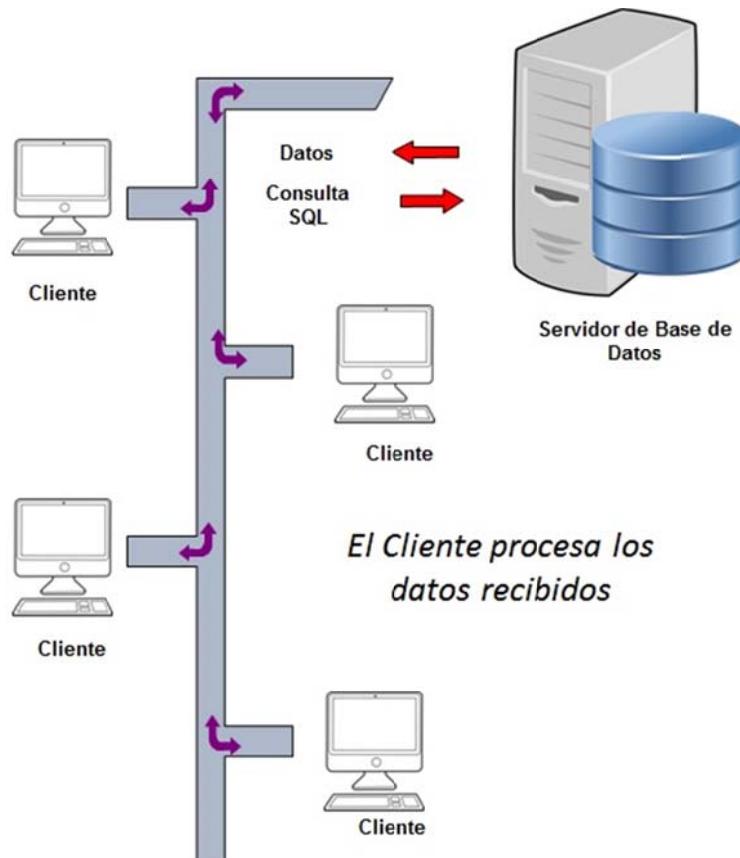


Fig. 33. Funcionamiento de un Servidor de Base de Datos

- Servidores de Transacciones - El cliente invoca un procedimiento remoto que reside en el servidor en un motor de base de datos. Este proceso remoto ejecuta un conjunto de sentencias SQL. El intercambio de red consiste en un único request/reply a diferencia del esquema anterior donde hay un mensaje request/reply por cada comando SQL. Este conjunto de sentencias SQL agrupadas se conocen con el nombre de transacción y constituyen una unidad atómica que tiene éxito o falla en su conjunto. Con este esquema se desarrollan aplicaciones donde el código se distribuye entre servidor (procesos de datos y reglas de negocios) y cliente (interface gráfica). El Capítulo 6 brinda detalles de este tipo de servidores. La Figura 34 muestra la comunicación de este tipo de servidores con sus clientes

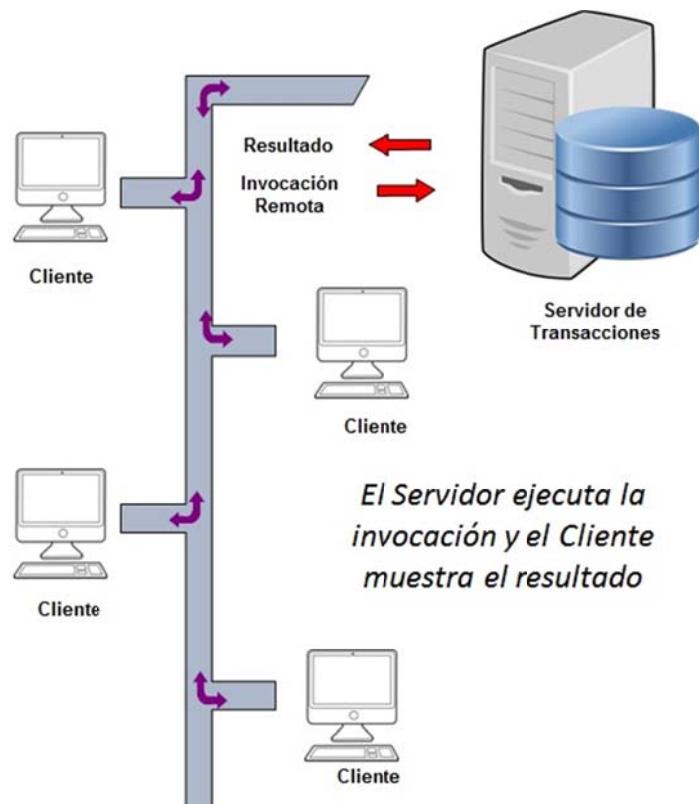


Fig. 34. Funcionamiento de un Servidor de Transacciones

- Servidores para Grupos de Trabajo - Son sistemas que permiten manejar información semi-estructurada como texto, imagen, mail, boletines y flujo de tareas. La Figura 35 muestra la comunicación de este tipo de servidores con sus clientes.



Fig. 35. Funcionamiento de un Servidor de Grupo de Trabajo

- Servidores de Objetos - Con este esquema las aplicaciones Cliente/Servidor se escribe como un conjunto de objetos que se comunican. Los objetos clientes se comunican con los servidores usando Object Request Broker (ORB). ORB localiza una instancia de la clase del objeto servidor, invoca el método requerido y retorna los resultados a la instancia del objeto cliente. La Figura 36 muestra la comunicación de este tipo de servidores con sus clientes.

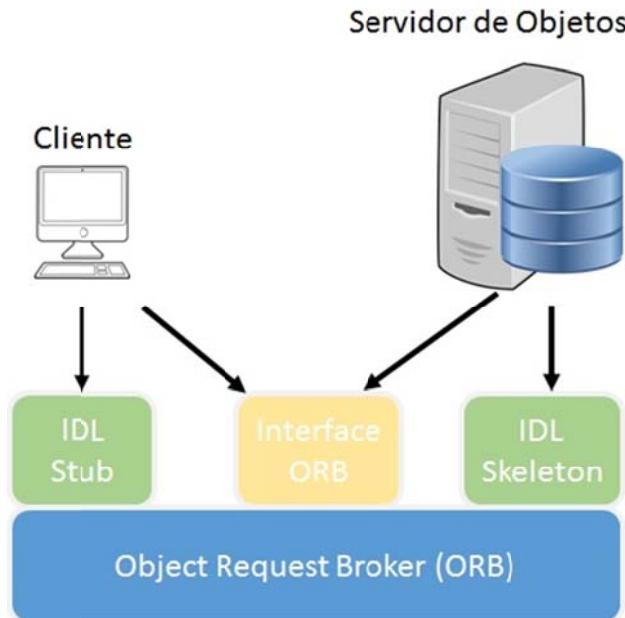


Fig. 36. Funcionamiento de un Servidor de Objetos

4.2. Roles y Funciones del Cliente

El rol del Cliente es el de iniciar el diálogo con el servidor solicitando acceso a los recursos compartidos que éste posee. Una vez obtenidos dichos recursos, es responsabilidad del cliente mostrar los resultados al usuario, para lo cual es muy probables que deba ejecutar código.

En este sentido, en una arquitectura Cliente/Servidor clásica en 2 niveles, el cliente resuelve la lógica de presentación al usuario (GUI - Graphic User Interface) y también lógica de negocio. La variedad de clientes estará dada generalmente por el tipo de lenguaje de programación o plataforma de ejecución que posee.

Los denominados lenguajes de programación de cuarta generación o lenguajes visuales, fueron los facilitadores para la construcción de clientes con la funcionalidad mencionada anteriormente.

La Figura 37 muestra algunas variantes tecnológicas para construir componentes clientes que cumplan con el rol definido en un entorno Cliente/Servidor clásico.



Fig. 37. Lenguajes de programación de cuarta generación.

4.3. Cliente/Servidor en la Web con Tecnología JAVA

En el Capítulo 2 en el que se analizó la evolución de los sistemas distribuidos, se mencionó la tecnología Web en 3 capas y se introdujo el concepto de Web Server *dinámico*.

El dinamismo que se busca en un Web Server consiste en lograr que dicho servidor que podría verse como un simple servidor de archivos, sea capaz de desencadenar algún tipo de procesamiento computacional, es decir, ejecutar código de programas.

En ese sentido, esta extensión de funcionalidad que se busca obtener, puede lograrse de modo nativo o a través de la programación de interfaces (tal como sucede siempre que se requiere extender la funcionalidad de un software).

- El modo nativo consiste en modificar el núcleo del software en cuestión dotándolo de las prestaciones que se buscan dentro mismo de su entorno de ejecución.
- El modo por programación de interfaces, consiste en contar con un componente de software que brinde las prestaciones que se buscan lograr (en este caso que el servidor no solo devuelva archivos sino que invoque rutinas para ejecutar código) y escribir las interfaces necesarias para conectar ambos componentes.

La tecnología Java para Web o Tecnología JEE [4.1][4.2] aporta una solución en modo nativo, mientras que la tecnología CGI (que será abordada en el Capítulo 5), aporta una solución por programación de interfaces.

La Figura 38, muestra las distintas variantes para obtener un Web Server “dinámico”.

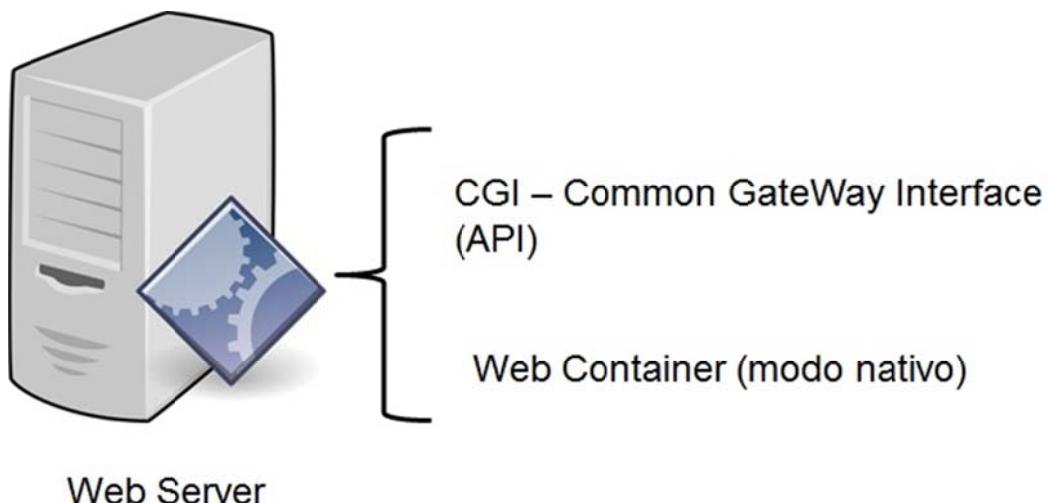


Fig. 38. Variantes de Web Server dinámico

La tecnología basada en Web modo nativo hace uso del concepto de páginas HTML dinámicas.

Una página dinámica se ejecuta dentro del Web Server pero dentro de un componente de la misma que interpreta el código de programación de manera separada del código HTML.

Se considera de modo nativo porque el código no requiere invocaciones a programas externos y no usa una API.

Ejemplos de páginas dinámicas lo constituyen el lenguaje PHP, JSP (basado en JAVA) o ASP (páginas dinámicas en tecnología Microsoft).

En el caso particular de páginas dinámicas en tecnología Java, la arquitectura del sistema se presenta en la Figura 39.

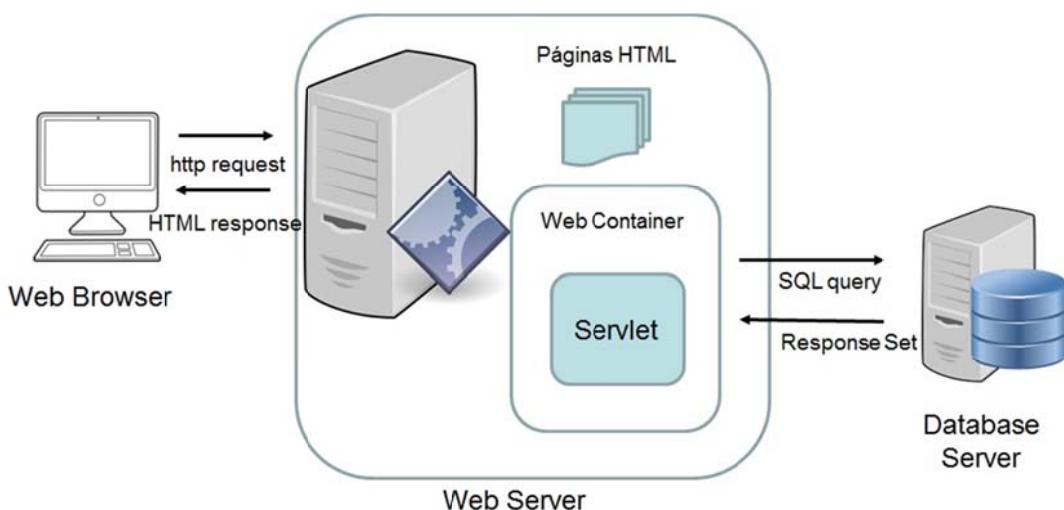


Fig. 38. Tecnología Web modo nativo en JAVA

4.4. Tecnología JEE en el Contexto de Ambientes Distribuidos

Un servlet Java es un programa Java que corre del lado del servidor y se ejecuta en el contexto de un componente del Web Server denominado Web Container.

El Web Container es una máquina virtual Java (JVM) que brinda una implementación de la interface de aplicación del servlet (API). Las instancias de los servlets son componentes que utiliza el Web Container para responder a los requerimientos HTTP.

El componente Web Container puede atender requerimientos que se ejecutan en hilos separados, logrando un mayor rendimiento y escalabilidad. Además proporciona servicios adicionales para seguridad y manejo de errores.

Los Servlets, por su parte, aprovechan las prestaciones de su entorno de ejecución y se comportan como programas servidores robustos y orientados a objetos, que por estar escritos en Java son independientes de la plataforma.

En definitiva, un Servlet es un programa Java que debe generar código HTML dentro de su respuesta, debido a que se ejecuta en el contexto de un Web Server.

Esta característica, obliga a que se mezcle la lógica de la presentación y la lógica de negocio dentro del mismo programa, por esa razón surgen las páginas dinámicas Java o JSP (Java Server Page) que son páginas HTML con etiquetas particulares para incrustar código Java.

El Web Container convierte las páginas JSP en instancias de Servlets.

El siguiente es un ejemplo de código Java que implementa un Servlet para "Hola Mundo"

```
public void generarRespuesta (HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    // Determina el nombre especificado
    String name = request.getParameter ("name");
    if ((name == null) || (name.length() == 0)) {
        name = DEFAULT_NAME;
    }
    // Especifica que el tipo de contenido es HTML
    response.setContentType ("text/html");
    PrintWriter out = response.getWriter ();
    // Generar respuesta HTML
    out.println ("<HTML>");
    out.println ("<HEAD>");
    out.println ("<TITLE>Hola Mundo</TITLE>");
    out.println ("</HEAD>");
    out.println ("<BODY BGCOLOR = 'white'>");
    out.println ("<B>Hello, " + name + "</B>");
    out.println ("</BODY>");
    out.println ("</HTML>");
    out.close();
}
```

Este mismo ejemplo pero construido con páginas JSP sería:

```

<%! private static final String DEFAULT_NAME = "Mundo" ;%>
<HTML>
<HEAD>
<TITLE> Hola Mundo en JSP </TITLE>
</HEAD>
<%-- Determinar el nombre especificado --%>
<%
    String name = request.getParameter ("name");
    if ( ( name == null ) || ( name.length () == 0 ) ) {
        name = DEFAULT_NAME;
    } %>
<BODY BGCOLOR = "white">
<B> Hello, <% =name %></B>
</BODY>
</HTML>

```

La primera vez que una página JSP es requerida, el Web Container convierte el archivo JSP en un servlet que puede responder a los requerimientos HTTP.

El procesamiento de una página JSP por parte del Web Container realiza tres pasos:

- 1- En el primer paso el Web Container traduce el archivo JSP en código fuente Java que contiene la definición de una clase Servlet.
- 2- En el segundo paso, el Web Container compila el código fuente Java en un archivo de clase Java.
- 3- En el tercer paso, el Web Container crea una instancia de la clase servlet y realiza los pasos del ciclo de vida del mismo invocando un método especial que es jsplInit.

La Figura 40 muestra el procesamiento descripto.

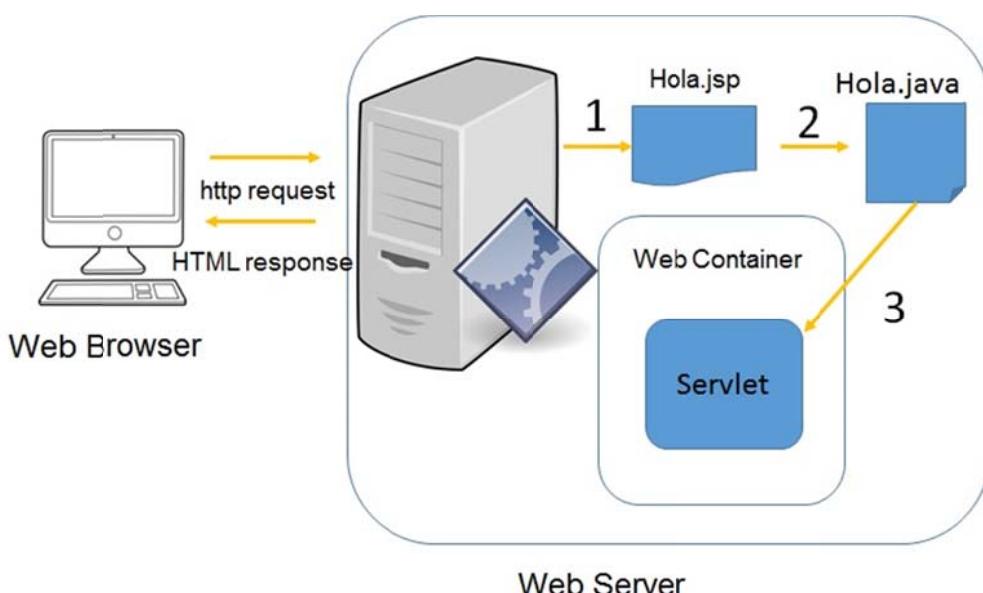


Fig. 40. Procesamiento de una página JSP

4.5. Patrón de Diseño MVC en un Entorno Distribuido

El patrón de diseño MVC - Model, View, Controller, tiene sus orígenes en la programación orientada a objetos y se basa en el concepto que un programa de software debe considerar en su diseño tres componentes:

- Model: los servicios de negocios y objetos del dominio de aplicación
- View: la *ventana* dentro de la aplicación que se presenta al usuario
- Controller: la lógica que acepta las acciones del usuario, realiza la operación y selecciona la próxima View para el usuario.

Este patrón de diseño fue adoptado por la tecnología Web y encontró la mejor forma de ser aplicado, en la tecnología Java.

- El Model es construido con clases Java o con componentes JavaBean.
- El View es construido con páginas JSP
- El Controller se implementa con servlets.

La Figura 41 muestra los tres componentes de MVC aplicado a la tecnología Web.

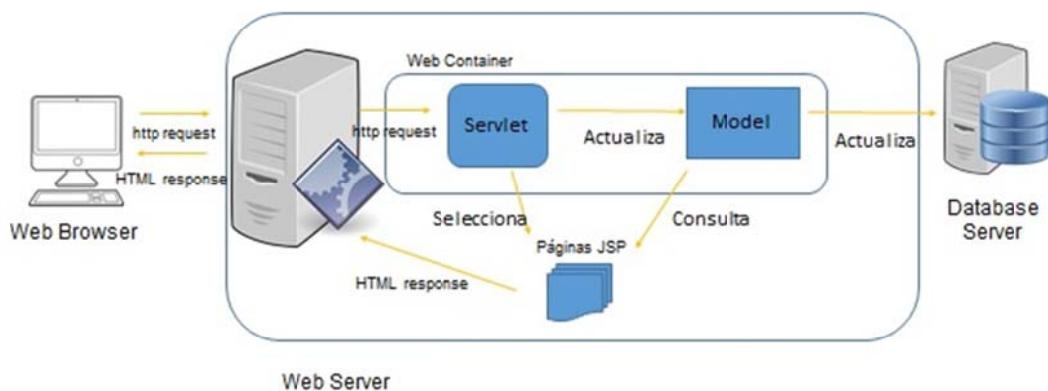


Fig. 41. Patrón MVC Web

Esta arquitectura de tecnología Web implica que la lógica de presentación, la lógica de negocios y la de acceso a los datos, si bien están conceptualmente divididas, son resueltas por el Web Server.

Esto incrementa la carga de trabajo en un único servidor y además hay que tener en cuenta que un Web Server es eficiente en la atención de requerimientos concurrentes, mientras que la lógica de negocios se caracteriza por consumir mucha CPU.

Por otra parte, si el sistema ya posee desarrollos con clientes GUI standalone (que se ejecutan fuera del Web Browser), deberán duplicarse la lógica de negocios y acceso a datos, con el correspondiente manejo de transacciones.

Por tal motivo, la tecnología JEE considera un componente de ejecución más, EJB Server, que podemos clasificar como un Servidor de Objetos tal como lo describimos en el Capítulo 2.

EJB es el acrónimo de Enterprise Java Beans, siendo un bean un elemento de diseño que establece ciertas pautas para definir una clase Java, como por ejemplo:

- Las propiedades se definen con métodos para accederlas y modificarlas (llamadas comúnmente *seters* y *geters*)
- Posee un método constructor sin argumentos que permite instanciar objetos de esa clase.
- Las variables de instancia no son públicas, es decir, sólo se acceden y modifican por los *seters* y *geters*.

La figura 42 muestra la arquitectura de un sistema distribuido con tecnología JEE.

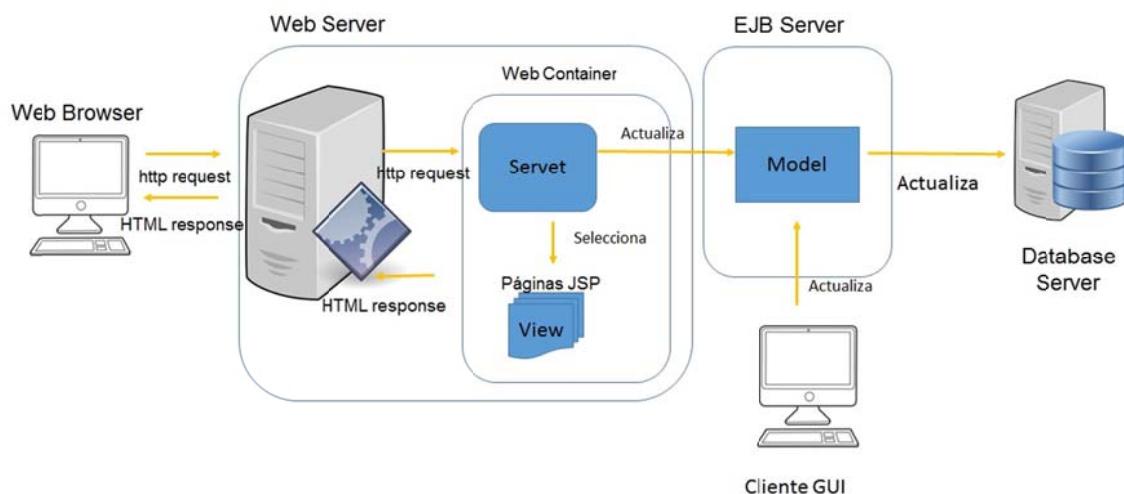


Fig. 42. Arquitectura Distribuida en 3 niveles con JEE

4.6. Conclusiones del Capítulo

En este capítulo se hace hincapié en los componentes servidor y cliente en los ambientes distribuidos, detallando sus principales características y funcionalidades.

En este sentido se analiza también la tecnología Web nativa como un caso de distribución Cliente/Servidor donde la capa media adquiere mayor relevancia y comportamiento.

Se detalla la tecnología Web en modo nativo, como un ejemplo de aplicación distribuida que también convive con Cliente/Servidor clásico y que expone los detalles de varios tipos de servidores (de archivos, de objetos y de base de datos).

CAPÍTULO 5

Arquitecturas Distribuidas

Patricia Bazán

En capítulos anteriores se presentaron distintos estilos arquitectónicos distribuidos y como se detalló en el Capítulo 1, las arquitecturas distribuidas buscan simplificar y abstraer las funciones de cada componente definiendo su intercomunicación y estableciendo su ubicación y plataforma de ejecución.

También se ha detallado la evolución de los sistemas distribuidos que fue guiada en gran medida por la misma evolución de construcción de aplicaciones y el crecimiento de Internet como mecanismo de comunicación (middleware) a nivel de red.

En la evolución presentada en el Capítulo 2 se describió someramente el concepto de arquitecturas orientadas a servicios (SOA), como evolución de los objetos distribuidos y como un paradigma adecuado para la integración de aplicaciones.

En este capítulo se profundizan estos conceptos y se analizan los Web Services como una implementación posible de una arquitectura orientada a servicios.

El capítulo se organiza de la siguiente manera: en la Sección 1 se describen las arquitecturas de n niveles, ya analizadas en el Capítulo 4, pero con tecnología CGI. En la sección 2 se presenta la Arquitectura Orientada a Servicios SOA, detallando componentes, características y motivaciones de uso. En la Sección 3 se analizan los Web Services como un medio para implementar un SOA, junto con sus variantes tecnológicas que se describen en la Sección 4. Finalmente, se arriban a algunas conclusiones.

5.1. Arquitectura n niveles. Tecnología CGI

Las llamadas aplicaciones Cliente/Servidor de misión crítica dejaron de ser un elemento teórico. Cada día, millones de transacciones de comercio electrónico se inician desde las PC's y se ejecutan en servidores distribuidos. Con Cliente/Servidor se mezcla el procesamiento local y remoto en una aplicación única. Este es de alguna manera el fundamento de Internet, intranets y extranets. Sin embargo el Cliente/Servidor clásico en dos niveles comenzó a resultar insuficiente.

Si bien el concepto de tres niveles fue inicialmente dirigido por la necesidad de escalar las aplicaciones en dos capas, el mercado actual de servicios y componentes lo están tomando

como su paradigma. Como se mencionó en el Capítulo 2, la tecnología Web es un ejemplo de evolución del modelo Cliente/Servidor clásico a un modelo en tres niveles. Pero este no es el único caso de arquitectura en más de dos niveles.

Características del modelo de 3 niveles

El cliente provee la interface de usuario gráfica (GUI - *Graphic User Interface*) e interactúa con el servidor a través de servicios remotos y métodos de invocación.

La lógica de la aplicación vive en el nivel medio y se distribuye y administra en forma separada de la interface de usuario y de la base de datos.

Las aplicaciones en 3 niveles minimizan los intercambios en la red creando niveles de servicios abstractos.

Las aplicaciones en 3 niveles sustituyen unas pocas invocaciones al servidor por varias consultas y actualizaciones SQL.

Proveen mayor seguridad al no exponer la estructura física de los datos.

La siguiente tabla muestra una comparación de características entre 2 niveles y 3 niveles

Característica	2 capas	3 capas
Administración de sistema	Compleja (más lógica en el cliente)	Menos compleja (se centralizan las aplicaciones)
Seguridad	Baja (a nivel de datos)	Alta (a nivel de servicios y métodos)
Encapsulamiento de datos	Baja (tablas expuestas)	Alta (el cliente invoca servicios y métodos)
Performance	Pobre (muchas sentencias SQL viajan por la red)	Buena (se intercambian sólo preguntas y respuestas)
Escalabilidad	Pobre	Excelente
Reuso de aplicaciones	Pobre (aplicaciones monolíticas en el cliente)	Excelente (se reusan servicios y objetos)
Soporte de Internet	Pobre (limitaciones por ancho de banda)	Excelente (los “thin” clientes son fáciles de descargar)
Soporte de BD heterogéneas	NO	SI
Elección de mecanismo de comunicación	NO (sólo sincrónico y orientado a la conexión)	SI (mensajes, colas, broadcasting)

Tecnología CGI

La tecnología CGI (Common Gateway Interface), define un escenario posible de implementación de arquitectura de 3 niveles basada en la tecnología Web detallada el Capítulo 2 donde se planteaba la necesidad de contar con alguna variante de ejecución computacional dentro del servidor Web y transformarlo en un servidor Web *dinámico*.

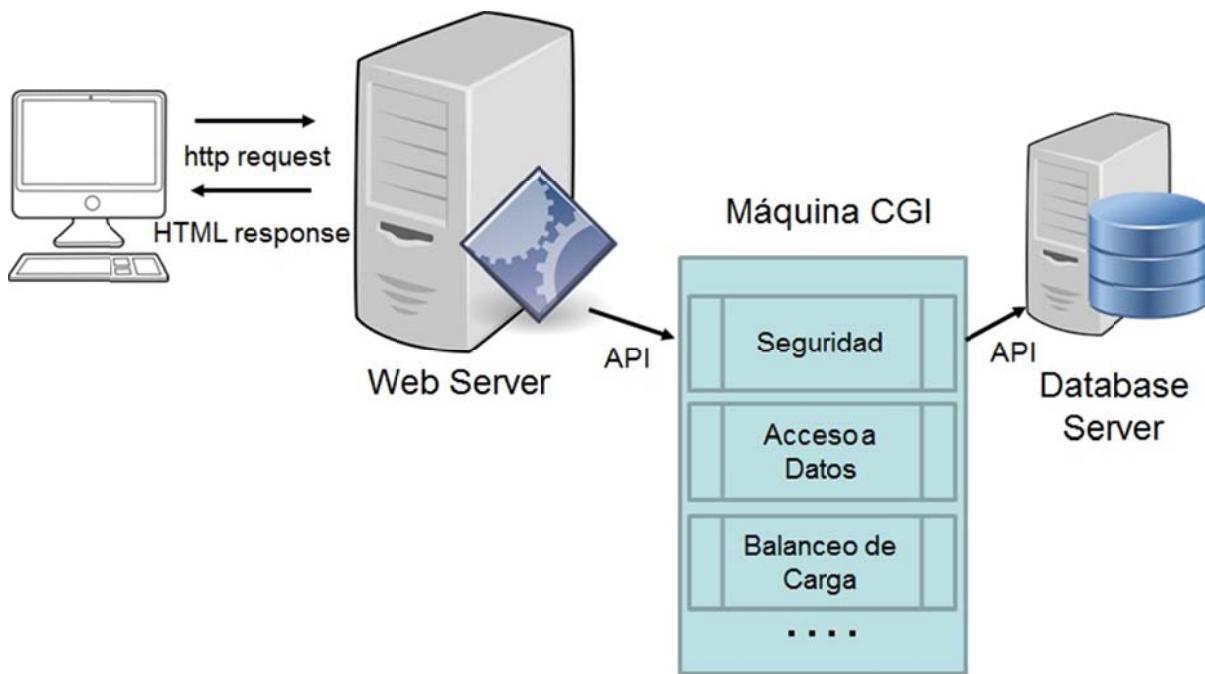


Fig. 43. Funcionalidades de una Máquina CGI

Como se detalló en el Capítulo 4, hay dos variantes para incorporar dinamismo a un servidor Web. Una de ellas, en modo nativo, a través de páginas dinámicas (la Tecnología Java es un ejemplo) o mediante una interface (API: Application Program Interface), tal es el caso de la Tecnología CGI.

La tecnología CGI es un mecanismo muy simple que permite que un servidor web ejecute un programa escrito en cualquier lenguaje de programación a través de una invocación definida por su API (implica salida al Sistema Operativo).

Para enviar datos al programa CGI se cuenta con la especificación CGI que define cómo se empaquetan dichos datos. El URL determina cuál programa CGI se ejecutará.

La Figura 43 detalla algunas de las funciones que pueden llevar a cabo un motor o máquina CGI.

Debido a que el Web Server debe encaminar la invocación a la máquina CGI, sólo puede hacerlo a través de un tag HTML y en particular a través de los métodos GET o PUT.

Veamos con un ejemplo, cómo sería esta invocación utilizando una máquina CGI específica, PBCGI050.EXE que provee el lenguaje PowerBuilder.

```
<FORM ACTION = "/scripts/pbcgi050.exe/master/uo_inet_functions/f_handlelogin"
METHOD="GET">
```

- 1- FORM le indica al WebServer que el texto que continúa es un FORM
- 2- ACTION le indica al WebServer que cuando este form sea enviado (submit) se realizará alguna acción. Esta acción puede ser bien un pedido de servicio HTTP o un llamado a una interface CGI para la ejecución de una función.
- 3- uo_inet_functions identifica el objeto no visual de master que contiene la definición de la función f_handlelogin.
- 4- /scripts/ determina el directorio donde encontrar la interface CGI y /pbcgi050.exe/ es el programa específico que implementa el CGI

Tecnología CGI: ventajas y desventajas

Las ventajas de usar tecnología CGI para lograr un Web Server dinámico son:

- Pueden escribirse en cualquier lenguaje de programación (Perl, C, PowerBuilder, VisualBasic).
- Un error en el programa CGI no afecta el servicio Web debido a que se ejecuta fuera de su contexto.
- Son programas fáciles de referenciar para un diseñador Web. Como se ve en el ejemplo, simplemente se incluye la invocación en una página HTML ya diseñada gráficamente.
- Dado que los programas CGI ejecutan su propia llamada al sistema operativo, no tienen conflicto de concurrencia con requerimientos que usen el mismo CGI.
- Todos los proveedores de servicios lo implementan.

Por su parte las desventajas son:

- Alto tiempo de respuesta, justamente por su ejecución fuera del contexto del Web Server.
- No es escalable. Si crece repentinamente la cantidad de requerimientos, el programa CGI debería hacer balanceo de carga.
- No permite un diseño modular ni separa la lógica de la presentación de la de negocio.

El nivel medio en muchas aplicaciones en 3 niveles no es implementado como un programa monolítico sino que es una colección de componentes utilizadas por varias transacciones.

Cada componente automatiza una función de negocio relativamente pequeña. Los clientes suelen combinar varias componentes del nivel medio dentro de una transacción única.

5.2. Conceptos de Arquitecturas Orientadas a Servicios

Según la definición de OASIS 9, SOA como un paradigma capaz de organizar y utilizar las capacidades distribuidas, que pueden estar bajo el control de distintas organizaciones, y de proveer un medio uniforme para publicar, descubrir, interactuar y usar los mecanismos oportunos para lograr los efectos deseados.

SOA es un modelo de referencia para entender las relaciones más significativas dentro del dominio de un problema concreto y facilitar el desarrollo de estándares o especificaciones. Se fundamenta en un pequeño número de conceptos para explicar el modelo a profanos y busca producir una semántica sin ambigüedades.

SOA es un modelo de referencia para:

- La creación y utilización de servicios a lo largo de su vida útil.
- La definición de la infraestructura que permita intercambiar datos entre diferentes aplicaciones.
- La participación de los servicios en los procesos de negocios independientemente del sistema operativo, los lenguajes de programación y si los procesos son internos o externos a la organización.

Los conceptos y sus relaciones, definidas en el modelo de referencia SOA, deben ser la base para describir la arquitectura.

Una arquitectura SOA concreta será el producto de aplicar la arquitectura de referencia desarrollada según el modelo de referencia y los patrones¹⁰ de esa arquitectura, así como los requerimientos necesarios, incluyendo los impuestos por los entornos tecnológicos.

La aplicación de un modelo de referencia para lograr una arquitectura completa, equivale a pasar de una etapa de análisis a una de diseño en analogía con las etapas del ciclo de vida del software. Implica dar un paso más en el nivel de detalle y comenzar a buscar metodologías para aplicar sobre los conceptos analizados.

Una arquitectura concreta se desarrolla en un contexto predefinido donde se fijan protocolos, perfiles, especificaciones y estándares. La plataforma SOA combina estos elementos a los efectos de generar un producto operativo. La Figura 44 (Bazán,2010) presenta un marco general entre la noción de marco de referencia, arquitectura de referencia y arquitectura específica, que muestra qué aporta y en qué consiste cada una de ellas, así como el paso de lo más abstracto hacia lo más concreto.

9 <https://www.oasis-open.org/>

10 Un patrón es una forma de realizar una tarea concreta basada en una generalización.



Fig. 44. Marco general entre modelo de referencia, arquitectura de referencia y arquitectura específica

Características de SOA

En conclusión con lo anterior, para lograr un SOA una empresa necesita entender cómo modelar procesos, servicios y componentes y cómo unir estos modelos de manera consistente

Las características o aportes fundamentales de SOA son:

- Reuso de componentes de software existentes. Este concepto se encuentra vinculado fundamentalmente con la idea de federación de aplicaciones. Un entorno informático federado es aquel en el cual los recursos y aplicaciones se encuentran unidos manteniendo autonomía y autogobierno. La exposición de servicios mediante SOA facilita la federación y por ende el reuso.
- Inteoperabilidad entre aplicaciones y tecnologías heterogéneas. Este concepto con la capacidad de compartir información entre aplicaciones y también con la independencia de las plataformas. La exposición de servicios favorece la interoperabilidad y permite el uso de la funcionalidad que otorga un proveedor con independencia de la plataforma de implementación.
- Flexibilidad para componer, integrar y escalar soluciones. La flexibilidad de una solución se vincula con un mayor alineamiento entre el dominio o negocio y la tecnología. Este alineamiento busca que el sistema reaccione rápidamente a los cambios

del entorno. Los servicios de grano fino facilitan la evolución del sistema y otorgan agilidad a la organización.

Conceptos SOA

- Servicios: Son piezas funcionales que resuelven un aspecto del negocio. Pueden ser simples (almacenar los datos de un cliente) o compuestos (el proceso de compra de un cliente)
- ESB (Enterprise Service Bus). Es la infraestructura que habilita una alta interoperabilidad entre Servicios en un contexto distribuido.
- Bajo acoplamiento. Concepto de reducir las dependencias entre sistemas. Es una idea muy difícil de desplegar, mantener y corregir.

SOA y el Modelo de Interacción

Según los conceptos subyacentes a SOA descriptos con anterioridad, el modelo de interacción es uno de los más relevantes dado que aporta la manera en que los servicios se comunican entre sí.

Este modelo de interacción sigue el paradigma triangular de Publicar-Ligar-Ejecutar como se muestra en la Figura 45 (Bazán,2010).

En este paradigma, los proveedores registran sus servicios en un registro público. Los consumidores utilizan este registro para buscar servicios que cumplan con cierto criterio. Si el registro posee tal servicio, el mismo provee al consumidor con un contrato y un punto de acceso para el servicio.

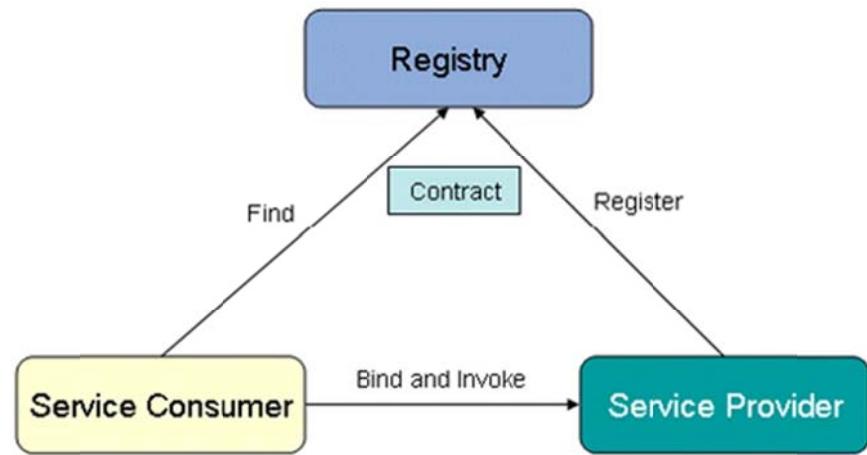


Fig. 45. Modelo de Interacción Triangular SOA

- Publicar (*Register*): para que un servicio sea usado, su descripción de servicio debe ser publicada
- Descubrir / Encontrar (*Bind and Invoke*): el consumidor busca una descripción de servicio directamente, o hace una búsqueda de servicios. La operación se puede hacer en dos puntos del ciclo de vida del cliente: durante el diseño o durante la ejecución.

- Enlazar (*Find*): el cliente del servicio invoca o inicia la interacción con el mismo durante la ejecución, contando con los detalles en la descripción del servicio para localizarlo, contactarlo e invocarlo.

Acerca de los servicios

Un servicio es un elemento que se comprende en términos de la utilidad que brinda, por lo tanto, no puede apartarse del negocio o problema para el cual debe ser útil.

Los servicios son tareas computacionales débilmente acopladas que se comunican vía una red (en el caso de los web services por Internet), y que juegan una relación cada vez más creciente en las interacciones B2B (Business To Business).

Un servicio captura funcionalidad con un valor de negocio, y está listo para ser usado. Es provisto por servidores, para lo cual requiere de una descripción que pueda ser accedida y entendida por potenciales clientes. Los servicios de software son servicios provistos por sistemas de software (Erl, 2007).

Con estas definiciones podemos decir que el conjunto de conceptos que describen los servicios son: descripción del servicio, contratos y normas y contexto de ejecución. A través de la descripción del servicio se obtiene la información que un consumidor necesita para considerar si usa o no el servicio.

Con esta información se puede informar al consumidor: si el servicio existe (alcanzabilidad), qué funciones realiza (funcionalidad), qué restricciones se aplican a su uso y cómo se debe interactuar con el servicio (interfaz) tanto en cuanto a formato como a secuencias.

Los contratos y normas representan el acuerdo entre las partes, las condiciones de uso, las restricciones y el despliegue de los servicios.

El contexto de ejecución es el conjunto de elementos técnicos y de negocios que conforman la vía para que proveedores y consumidores puedan interactuar.

La posibilidad de extender sistemas anexándole nuevos servicios y haciendo reuso de los ya existentes, con el objetivo de lograr interoperabilidad entre tecnologías y aplicaciones heterogéneas, nos permite prolongar la vida de los sistemas. Los beneficios de SOA son a largo plazo. La existencia de un único servicio, no tendrá valor si no tiene forma de complementarse con otros servicios.

Este estilo de arquitectura nos permite el desarrollo de aplicaciones débilmente acopladas, las cuales pueden ser accedidas a través de la red.

Enterprise Service Bus (ESB)

El enfoque SOA busca construir aplicaciones mediante la combinación poco acoplada de servicios interoperables. En tal sentido, el hecho de que un servicio pueda utilizarse amplia-

mente en toda la empresa por muchas aplicaciones, implicaría dar lugar a los siguientes riesgos para la infraestructura de IT de una organización:

- Tiempos de respuesta poco aceptables tanto para los usuarios como para procesos de negocio.
- Infracciones de seguridad.
- Pérdida de niveles de servicio para funciones críticas del negocio.
- Incumplimiento de normas de la industria y regulaciones gubernamentales.
- Gestión de servicios insuficiente.

El área de tecnología de una organización se ha centrado desde hace mucho tiempo en la gestión de la infraestructura como un activo para apoyar las aplicaciones y las unidades de negocio. Con SOA la atención se enfoca hacia la gestión de los servicios que prestan apoyo a los procesos de negocio.

Así, resulta evidente la necesidad de contar con una infraestructura de IT que apoye la gestión de los servicios.

Sobre una arquitectura SOA se puede definir un bus de servicios empresariales (Enterprise Service Bus o ESB) como una plataforma de software que da soporte a muchas funcionalidades resueltas a nivel de la capa de aplicación en los enfoques tradicionales de construcción de aplicaciones.

Tales funcionalidades son:

- La comunicación: el ESB se ocupa del ruteo de los mensajes entre los servicios.
- La conectividad: el ESB resuelve la conectividad entre extremos mediante la conversión de protocolos entre solicitante y servicio.
- La transformación: es responsabilidad del ESB resolver la transformación de formatos de mensajes entre solicitante y servicio.
- La portabilidad: los servicios serán distribuidos independientemente del lenguaje de programación en el que estén escritos y del sistema operativo subyacente.
- La seguridad: el ESB posee la capacidad de incorporar los niveles de seguridad necesarios para garantizar servicios que puedan autenticarse, autorizarse y auditarse.

Actualmente existen dos tendencias mayoritarias para la implementación de un ESB: los que requieren un servidor de aplicaciones y los que son totalmente distribuidos y, por lo tanto, no lo requieren (Bazán,2010).

Funcionalmente, cualquiera de las dos tendencias conserva sus propias características según se detalla en el cuadro *Servidores de aplicaciones Vs ESB*, pero se puede afirmar que un ESB totalmente distribuido que no requiere de un servidor de aplicaciones tendrá mayor independencia de la plataforma y será capaz de ofrecer una mayor ubicuidad de los servicios que gestiona.

Las siguientes características permiten determinar qué es y que no es un ESB:

- Está desarrollado sobre una arquitectura orientada a servicios, por lo tanto, es una implementación de SOA.
- Se basa en la utilización de estándares en un 100%, pero admite también elementos propietarios.
- Normalmente son multi-plataforma y multi-lenguaje.

- Son netamente distribuidos en el sentido de que no es necesario conocer la ubicación física de los servicios y que un mismo servicio puede existir en más de una localización en forma simultánea.
- Presenta un mecanismo de comunicación basado en mensajes con enrutamiento inteligente (basado en reglas o basado en contenido).
- Implementa un modelo de seguridad estandarizado con lo que se denomina nivel C2 de seguridad garantizando autenticación, autorización y auditoría.

5.3. Web Services como la Evolución Natural de la Computación Distribuida

Como hemos visto, CORBA ha constituido una base conceptual muy importante en la aparición de los conceptos de SOA.

El concepto de objeto y el de servicio guardan muchas similitudes en lo que se refiere a su modularidad y capacidad de reuso, pero también se diferencian, fundamentalmente en el modelo de interacción que presentan.

Servidores de aplicaciones Vs ESB

Los servidores de aplicaciones manejan gran parte de las interacciones entre la capa cliente y la capa de persistencia a datos en un modelo de 3-capas. Proveen una colección de servicios de middleware junto con un ambiente de ejecución para desplegar las componentes de lógica de negocios (el container). La mayoría de los servidores de aplicaciones soportan Web Services, ORB's, sistemas de mensajes, manejo transaccional, seguridad, balanceo de carga y gestión de los recursos. Proveen una solución integral a las necesidades de los sistemas de información empresariales (a gran escala). Constituyen una excelente plataforma de integración. Hoy la mayoría de los productos comerciales posicionan sus servidores de aplicaciones como máquinas de integración o los especializan agregando conexiones a back-end y sistemas legados posicionando sus productos como servidores de integración.

Si bien este tipo de servidores puede facilitar considerablemente la configuración de los diferentes productos de middleware, todavía vale la pena pensar en lo que hay debajo. Sean utilizado para desarrollo de aplicaciones o para integración, los servidores de aplicaciones son **plataformas de software**: la combinación de tecnologías de software necesarias para ejecutar aplicaciones. En este sentido, ellos definen la infraestructura de todas las aplicaciones desarrolladas y ejecutadas en los mismos.

Un ESB, por su parte, es una **infraestructura de software** en sí misma, que actúa como middleware intermediario y que dirige los requerimientos extendidos que generalmente no pueden ser cubiertos por los Web Services, como la integración entre los mismos y la inclusión de otras tecnologías de middleware y servicios de valor agregado como robustez, seguridad, gestión y control de servicios.

Un ESB dirige estos requerimientos y agrega flexibilidad para la comunicación entre servicios, haciendo posible conectar los servicios implementados en diferentes tecnologías (tales como EJB, sistemas de mensajes, componentes CORBA y sistemas legados).

Los objetos se definen con gran cohesión, en el sentido que existe gran asociación entre los métodos que atienden el objeto y una fuerte ligadura funcional, pero con bajo acoplamiento para minimizar el impacto de los cambios de una clase.

Los servicios, por su parte, necesitan un menor acople y no requieren que se conozca su nombre para utilizarlos porque poseen abundante meta-information.

Dentro de un servicio existen operaciones, no habiendo, a priori, asociaciones entre ellos, por lo tanto existe menos cohesión que en los objetos.

A cambio, se hace imprescindible contar con un marco de referencia o arquitectura, que facilite su registro y publicación para conocer su existencia (Arsanjani, 2004).

Un servicio difiere de un objeto o un procedimiento porque se define en función de los mensajes que intercambia con otros servicios.

Por último, los Web Services surgieron en forma paralela a la idea de SOA. La plataforma SOA ordenó su uso sistemático combinando protocolos, perfiles, especificaciones y estándares (Pulier, 2006).

Los Web Service constituyen una manera apropiada (no la única) de implementar interfaces de aplicaciones basadas en servicios y permiten que diferentes aplicaciones, realizadas con diferentes tecnologías, y ejecutándose en una variedad de entornos, puedan comunicarse e integrarse.

Los estándares vinculados con los Web Services son:

- WSDL (Web Service Description Language). Describe en formato XML la funcionalidad brindada por el Web Service.
- SOAP (Simple Object Access Protocol). Facilita el intercambio de información estructurada como XML. Constituye el protocolo de comunicación estándar.
- UDDI (Universal Description, Discovery and Integration). Es un estándar para facilitar el registro y descubrimiento de Web Services.

Estos estándares instancian un caso particular de modelo de interacción de servicios para Web Services, tal como se muestra en la Figura 46.

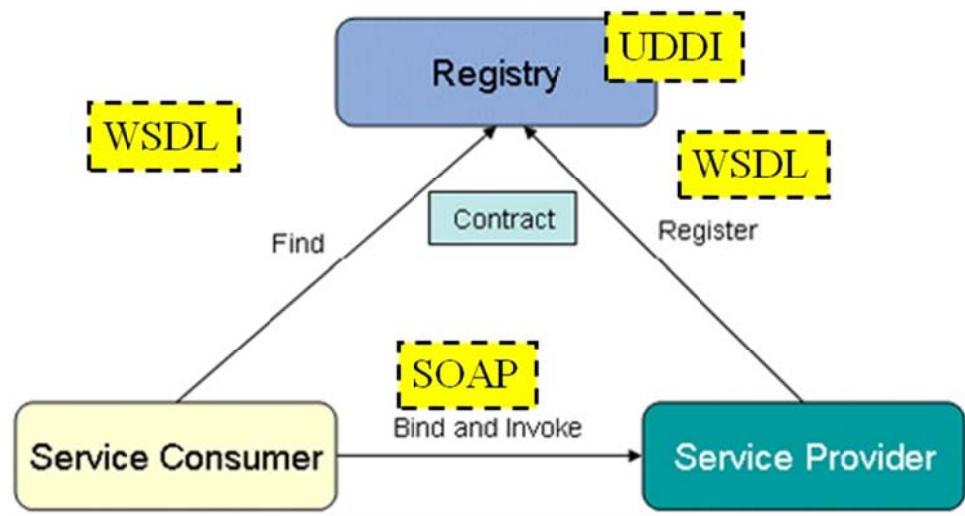


Fig. 46. Modelo de Interacción Triangular Web Services

5.4. Web Services en distintas tecnologías: REST y SOAP

Existen dos maneras de implementar Web Services y se diferencian respecto del uso o no del estándar para invocar y transportar el servicio. Estos son SOAP o REST.

SOAP (Simple Object Access Protocol)

Presentan una interface de invocación a una función (o método) distribuido similar a RPC. Comúnmente, la unidad básica de este estilo de Web Service es la operación WSDL, es decir, la especificación, en XML de lo que hace un servicio web, donde se encuentra y la forma de ser invocado. Este provee información muy importante para los desarrolladores, este lenguaje describe el formato de los mensajes que utiliza y a cuales puede responder.

REST (Representational state transfer)

Los Servicios Web RESTful emulan HTTP y protocolos similares limitando la interface a un conjunto de operaciones bien conocidas y estándares (p.e: GET, PUT, DELETE). Acá el foco esta puesto sobre la interacción con recursos, más que mensajes u operaciones. RESTful Web services pueden usar WSDL para describir mensajes SOAP sobre HTTP, lo cual define las operaciones, o pueden ser implementados puramente como una abstracción por encima de SOAP (ej: WS-Transfer).

	REST	SOAP
Formato del mensaje	XML	XML dentro de SOAP
Definición del interfaz	No es necesario	WSDL
Transporte	HTTP	HTTP, JMS, FTP, etc.

En resumen:

5.5. Conclusiones del Capítulo

En este Capítulo completamos la descripción de arquitecturas distribuidas desde su evolución en la Web a través de la tecnología CGI, hasta la aparición de las arquitecturas orientadas a servicios SOA.

Un SOA presenta un verdadero cambio en la concepción de aplicaciones pero que se ve favorecido en su instrumentación si proviene de un desarrollo basado en tecnología Web. La

instrumentación de un bus de servicios permite ordenar los servicios y además unificar los métodos de acceso evitando integraciones punto a punto.

Los Web Services provenientes de la tecnología Web son buenos instrumentadores de SOA sin provocar un impacto demasiado grande en las arquitecturas de los sistemas.

CAPÍTULO 6

Servidores de Base de Datos y la Distribución

Patricia Bazán

Podemos deducir de los temas desarrollados hasta ahora en las clases que sin NOS y un mecanismo de comunicación que cuente con al menos el nivel de transporte, es imposible crear la ilusión de tener un sistema totalmente transparente de la distribución.

Desde el punto de vista de las aplicaciones no se puede hablar de cliente/servidor hoy sin tener en cuenta a los servidores de base de datos SQL o basados en SQL.

Las bases de datos SQL o servidores SQL son el modelo dominante para crear aplicaciones cliente/servidor. Varios productos comerciales han evolucionado y perfeccionado su funcionamiento en ese sentido.

La Figura 47 muestra una arquitectura cliente/servidor clásica en dos niveles. El servidor SQL acepta peticiones SQL como mecanismo de comunicación y retorna como respuesta un conjunto de resultado que satisface la condición fijada en la consulta.



Fig. 47. Arquitectura Cliente/Servidor Clásico

El capítulo se organiza de la siguiente manera: en la Sección 1 se repasan algunos fundamentos del lenguaje SQL y bases de datos relacionales. En la Sección 2 se clasifican los motores de bases de datos SQL según su modo de almacenar datos o de procesar requerimientos. Luego, en la Sección 3 se analiza el impacto de procedimientos almacenados y disparadores en el modelo de distribución en dos niveles. La Sección 4 describe el manejo de transacciones en una base de datos como medio para asegurar la concurrencia de un sistema distribuido. La Sección 5 define el uso de transacciones distribuidas y en la Sección 6 se presentan variantes a dicha distribución mediante monitores de transacciones. Finalmente la Sección 7 arriba a algunas conclusiones.

6.1. Fundamentos de SQL y las bases de datos relacionales

El SQL como lenguaje de manipulación, definición y control de datos ha favorecido en gran medida la construcción de servidores de bases de datos de cualquier tamaño. SQL es un estándar ISO está orientado a los conjuntos, posee muy pocos comandos y fue creado para bases de datos que adhieren al modelo relacional.

Orígenes del SQL

El manejo de bases de datos que adhieren al modelo relacional fue creado por Codd en 1970 y se creó SQL como lenguaje de consulta front-end para la base de datos relacional prototípico System R. Es lenguaje orientado, al inglés, con firmes y sólidas raíces matemáticas se fundamenta en la teoría de conjuntos y el cálculo de predicados.

Con SQL se puede manipular un conjunto de datos que cumplen una determinada condición de recuperación.

Por otro lado, el modelo relacional hace una clara abstracción del almacenamiento físico de datos a través de las tablas, compuestas por filas y columnas. El modelo nos libera de concentrarnos en detalles relacionados con la forma en que está almacenada la información permitiendo un acceso puramente lógico.

Con SQL solo se especifican las tablas, las columnas y las filas involucradas en la operación. Además el SQL se ha transformado en el lenguaje predominante de mainframes, mini-computadoras y LAN servers. Las herramientas clientes SQL hacen una industria horizontal donde se mezclan herramientas front-end con servidores back-end.

¿Qué hace el SQL?

El lenguaje SQL se usa para realizar operaciones de datos complejas que insumiría cientos de líneas de código de un lenguaje tradicional.

- Los roles que cumple el SQL son:
- SQL es un lenguaje de consulta interactivo. Fue originalmente diseñado para usuarios finales y hoy se ve mejorado por las herramientas visuales para construcción de sentencias SQL sin escribir ningún comando.
- SQL es un lenguaje de programación de bases de datos. Puede estar embebido en otros lenguajes de programación para acceder a los datos o usar una conjunto de API como interface (X/Open). Por otra parte cada vendedor (Sybase, Oracle) proveen su propio dialecto de SQL para la programación.
- SQL es un lenguaje de definición y administración (manipulación) de datos. El lenguaje de definición de datos se utiliza para crear tablas simples, objetos complejos,

índices, vistas, restricciones de integridad referencial, seguridad y control de acceso. Todos los objetos definidos con SQL son almacenados en un diccionario conformado a su vez por tablas denominadas el catálogo del sistema.

- SQL ayuda a proteger los datos en un ambiente multiusuario. Provee excelentes características para validación de datos, control de integridad referencial, manejo de transacciones como unidades únicas de trabajo, cerramientos (*locks*) automáticos y detección de *deadlocks*.

La Figura 48 representa los distintos roles que puede cumplir el SQL como mecanismo de comunicación.

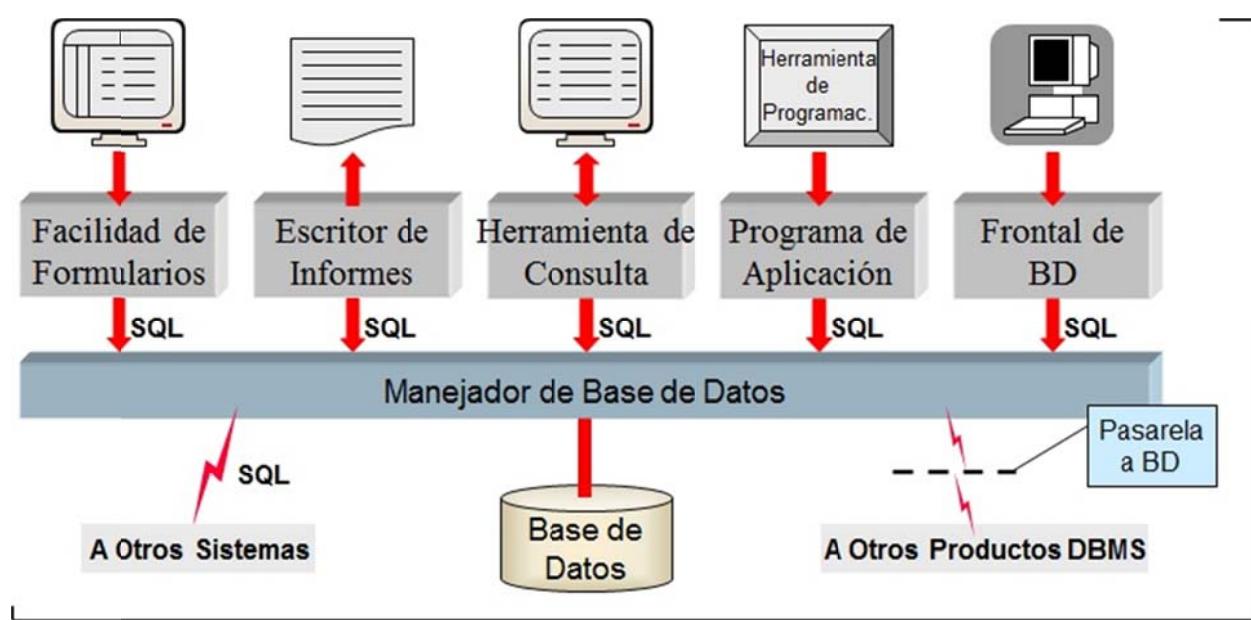


Fig. 48. Distintos Roles del SQL

¿Qué hace un servidor SQL?

En la arquitectura de aplicaciones cliente/servidor centradas en base de datos, una aplicación cliente requiere datos y servicios relacionados con ellos a un servidor de datos. El servidor de base de datos, normalmente llamado motor SQL, responde a los requerimientos del cliente y provee acceso seguro a datos compartidos. Una aplicación cliente puede recuperar datos y modificar un conjunto de datos con una petición simple.

- Un servidor SQL tiene las siguientes funciones:
- Controla y ejecuta comandos SQL.
- Provee una visión lógica y física de los datos.
- Genera planes optimizados de ejecución de los comandos SQL.
- Provee herramientas de administración y mantienen tablas de catálogos dinámicas de los objetos almacenados en dichos catálogos.
- Maneja la recuperación, concurrencia, seguridad y consistencia de la base de datos.
- Controla la ejecución de transacciones.

- Obtiene y libera *locks* durante la ejecución de la transacción en curso.
- Protege la base de datos de accesos no autorizados.

¿Qué es un servidor SQL?

Generalmente un servidor SQL es un híbrido del SQL estándar y el dialecto propio de vendedor.

Los vendedores de bases de datos ponen un gran esfuerzo en extender el SQL para otorgar nuevas funcionalidades más allá del modelo relacional fundamentalmente en lo relacionado con extensiones procedurales para los distintos sistemas operativos distribuidos, monitores de transacciones, bases de datos de objetos y administradores de objetos distribuidos.

6.2. Arquitecturas de Servidores SQL

La arquitectura de los servidores SQL pueden clasificarse conceptualmente desde dos puntos de vista: desde el punto de vista del almacenamiento y desde el punto de vista del procesamiento.

Arquitectura de Base de Datos Desde el Punto de Vista del Almacenamiento

Esta clasificación tipifica los servidores SQL según cómo organizan el repositorio físico de la información que gestionan y almacenan y se pueden encontrar do clases de variantes: 1- base de datos única - todos los objetos se almacenan en un mismo dispositivo físico o disco - y 2- base de datos múltiple - cada conjunto de objetos conforman una base de datos que ocupa cada una un dispositivo físico diferente -.

Las Figuras 49 y 50 muestran ejemplos de ambas arquitecturas.

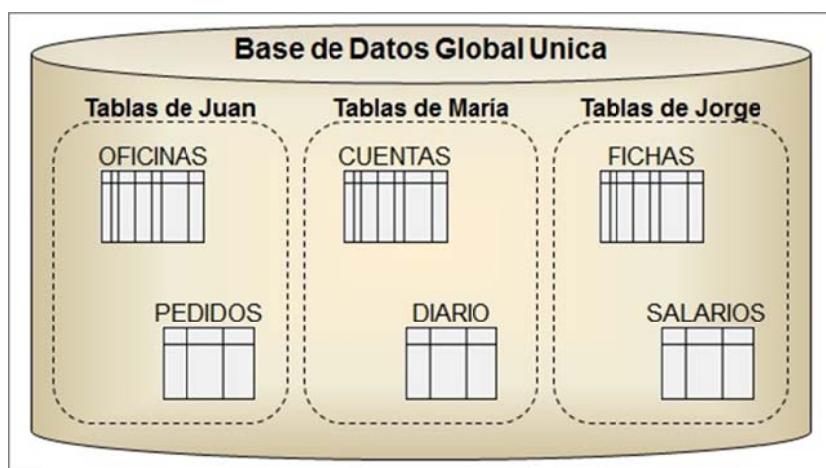


Fig. 49. Arquitectura de Base de Datos Única

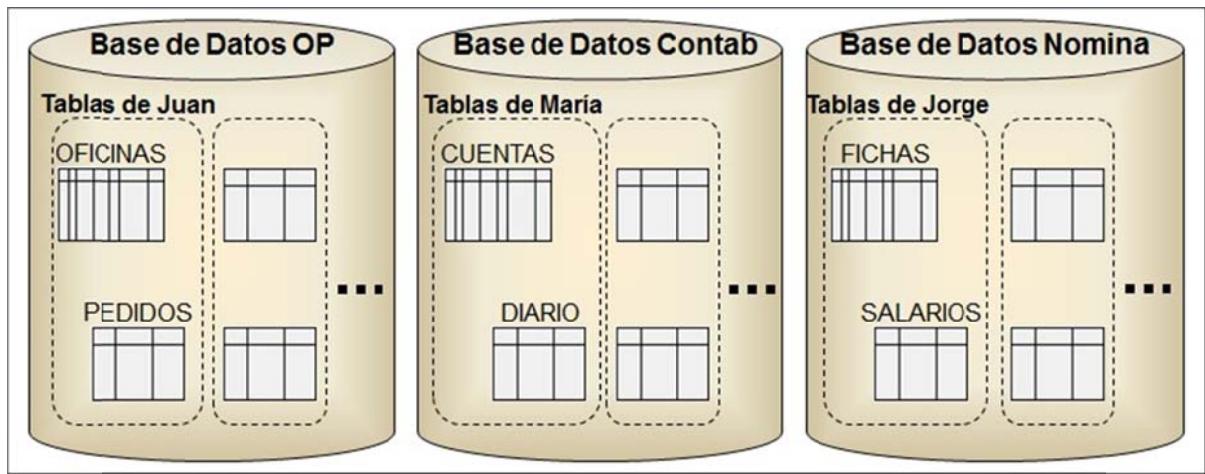


Fig. 50. Arquitectura de Base de Datos Múltiple.

Cada tipo de arquitectura posee sus ventajas y desventajas. En el caso 1, las tablas entre distintos sistemas de información pueden referenciarse fácilmente y también de manera más eficiente, debido a que el dispositivo de almacenamiento es único. El caso 2, por su parte, torna más complejo la referencia de objetos que no son propios de su base de datos y además podría provocar el aislamiento de la información si no existe un buen diseño de datos. Por ejemplo, tablas con el mismo contenido (provincias de Argentina), pueden existir en dos bases de datos distintas con el mismo nombre, duplicando innecesariamente la información.

Por otra parte, el caso 1 resulta desventajoso y hasta ineficiente a la hora de realizar copias de resguardo y/o restaurar debido al crecimiento desmedido del dispositivo que almacena todos los objetos de la base de datos. Además, al ser un dispositivo único se debe realizar la copia del dispositivo completo cada vez, aunque haya información que no se modificó o su modificación es esporádica. El caso 2, por otra parte, facilita la administración de copias y su restauración pudiendo seleccionar diferentes frecuencias para cada dispositivo y además, al ser de menor tamaño, el copiado y restauración es más rápido.

Arquitectura de Base de Datos Desde el Punto de Vista del Procesamiento

Esta clasificación tipifica a los servidores de base de datos según la manera en que atienden las peticiones que provienen de sus clientes. En definitiva, según la manera en que resuelven el acceso concurrente y se pueden encontrar tres variantes: 1- proceso por cliente - a cada cliente que hace una petición se le asigna un proceso o espacio de memoria específico, 2- multihilada - se asigna un único proceso para atención de requerimientos y cada cliente ejecuta un hilo (thread) diferente y 3- híbrida - el servidor escucha múltiples hilos pero no pasa las peticiones directamente sino que cuenta con un componente intermedio (dispatcher) que encola las peticiones y administra su distribución balanceando la carga de trabajo.

Cada tipo de servidor, en este caso posee ventajas y desventajas:

Arquitectura Proceso por Cliente (Ej: DB2, Informix)	
Ventajas	Desventajas
Protege a los usuarios entre sí y al mismo motor de ejecución de fallas ajenas.	Consumo más memoria y CPU
Cada proceso puede asignarse a un procesador si el hardware admite multiprocesamiento.	Degrada la performance debido al <i>overhead</i> que produce los cambios de contexto.

Arquitectura Multihilada (Ej: SQL Server)	
Ventajas	Desventajas
No requiere cambios de contexto y es por lo tanto más eficiente.	Un programa de usuario con errores puede provocar la caída del servidor completo
Las implementaciones de estos servidores son más portables porque no dependen del sistema operativo	La preemción de tareas necesarias para administrar los hijos nunca será realizada mejor que si se hiciera a nivel de sistema operativo.

Arquitectura Hibrida (Ej: Oracle a partir de versión 7)	
Ventajas	Desventajas
Restringe el acceso del espacio de cada usuario evitando la asignación permanente de recursos.	El balanceo de carga realizado por el dispatcher nunca será tan eficiente como un monitor de transacciones.

6.3. Procedimientos Almacenados, Disparadores y Reglas: su impacto en el Cliente/Servidor.

Las bases de datos relacionales cuentan hoy con extensiones procedurales como lo son los procedimientos almacenados, disparadores y reglas - En inglés, stored procedures, triggers y rules, respectivamente - que son extremadamente útiles pero no estándar. Los puristas del modelo relacional no ven con buenos ojos la existencia de estos mecanismos porque aseguran que van contra la esencia del modelo pues hacen que se pierda el sentido declarativo del lenguaje de los datos relacionales. Además tienen la contra de ser, en general, poco portables dado que se escriben en un dialecto de SQL propio del vendedor.

Pero más allá de todo, estas extensiones constituyen la primera aproximación al procesamiento distribuido dado que permite codificar procesos dentro del mismo motor de base de datos.

¿Qué es un stored procedure?

Muchos vendedores de base de datos ofrecen hoy un mecanismo de programación tipo RPC sobre la base de datos. Este mecanismo se conoce en general como *TP lite* o *stored procedures* (procedimientos almacenados).

Un procedimiento almacenado es una colección de sentencias SQL y lógica procedural (sentencias de control) que son compiladas, verificadas y almacenadas en el servidor de base de datos.

Un procedimiento almacenado es un objeto más de la base de datos y es almacenado en el catálogo junto a los demás (tablas, índices, vistas, etc.).

Un procedimiento almacenado acepta parámetros de entrada y puede ser usado a través de la red por múltiples clientes. El resultado de la ejecución es retornado y se gana enormemente en eficiencia y reducción de tráfico en la red.

El concepto de procedimiento almacenado fue introducido por Sybase en 1986 para mejorar la performance del SQL sobre redes. Son utilizados para asegurar reglas de negocios e integridad de datos, para realizar mantenimiento del sistema y también para administración pero la verdadera razón de ser es proveer inteligencia al servidor de base de datos y agregar funcionalidad de la lógica de las aplicaciones. Son óptimos para procesamiento de transacciones en línea (OLTP) donde básicamente se hace:

1. Recepción de un conjunto fijo de entradas desde clientes remotos.
2. Realización de múltiples comandos SQL previamente compilados contra la base de datos local.
3. Cumplimiento del trabajo (Commit).
4. Retorno de un conjunto fijo de resultados

Los procedimientos almacenados también proveen abstracción respecto de su sitio pues las modificaciones a las estructuras de datos son transparentes a las aplicaciones remotas que sigue solicitando el servicio sin importar cómo está implementado.

Para comprender las ventajas del uso de procedimientos almacenados, es importante tener presente las siguientes definiciones.

SQL dinámico y estático

El SQL estático son sentencias definidas en el código y compiladas en el momento de su definición. Si los objetos que referencia no existen, no compila.

El SQL dinámico es creado en tiempo de ejecución ofrecen más flexibilidad a expensas de velocidad de ejecución

SQL embebido y programado

SQL embebido son sentencias SQL que están incorporadas al código fuente del programa, entremezcladas con las otras sentencias del lenguaje de programación

SQL programado son las sentencias SQL se almacenan como objetos de la base de datos (triggers, Stored procedure, reglas)

Los procedimientos almacenados son un ejemplo de código SQL estático y programado, pero el SQL embebido también puede ser estático si las sentencias se compilan junto con el programa fuente en el que se encuentran o dinámico, si se incorporan a programas como cadenas de caracteres que son evaluadas en tiempo de ejecución.

A su vez, un procedimiento almacenado también puede ejecutar sentencias SQL de manera dinámica (es decir, evaluadas en tiempo de ejecución) utilizando la cláusula *dynamic*.

La Figura 51 muestra distintos aspectos evaluados en tres variantes, con procedimientos almacenados, con SQL embebido estático y con SQL embebido dinámico.

	Procedimientos almacenados	SQL embebido estático	SQL embebido dinámico
Nombres para las funciones	SI	NO	NO
Funciones compartidas	SI	NO	NO
Parametros de E/S	SI	NO	NO
Catalogado	SI	SI	NO
Logica procedural	En el objeto	Externa	Externa
Flexibilidad	Baja	Baja	Alta
Nivel de abstraccion	Alto	Bajo	Bajo
Estandarizacion	NO	SI	SI
Performance	Rapido	Mediano	Lento
Trafico	1 Request/Reply por varios comandos SQL	1 Request/Reply por cada comando SQL	1 Request/Reply por cada comando SQL

Fig. 51. Características de Funcionamiento de los Procedimientos Almacenados Respecto del SQL embebido.

Triggers y reglas

Un trigger o disparador, es una acción especial definida por el usuario (usualmente en la forma de procedimiento almacenado) que son automáticamente invocadas cuando ocurre un evento relacionado con los datos de la base de datos.

Una regla es un caso especial de trigger que se usa para chequeos simples sobre los datos.

Ambos están ligados a operaciones específicas sobre tablas específicas y realizan generalmente tareas relacionadas con auditoría de los datos, seteo de columnas por defecto y también control de integridad (de entidad o referencial).

Los triggers se habilitan por las operaciones de DELETE, INSERT y UPDATE sobre las tablas para las cuales están definidos, pueden llamar a otros triggers, e incluso puede generarse una recursión. Se diferencian de los procedimientos almacenados en que no tienen una invocación explícita sino que se disparan ante la ejecución de la operación para la que se hubieren definido.

Los procedimientos almacenados, los triggers o disparadores y las reglas constituyen código ejecutable que se cataloga y usa dentro del mismo servidor de base de datos, permitiendo que éste pueda incorporar capacidad de procesamiento y mejorar el balanceo de carga entre cliente y servidor para alojar parte de la lógica de la aplicación que de otro modo sólo podría ejecutarse en el cliente, debido a que en este modelo de distribución de dos niveles o capas no existe un componente que la aloje.

6.4. Manejo de transacciones en una base de datos

Los servidores de base de datos SQL heredan de dicho lenguaje la posibilidad de asegurar el acceso concurrente a los recursos mediante la definición de transacciones.

Una transacción en un servidor de base de datos SQL es una secuencia de una o más sentencias SQL que juntas forman una unidad de trabajo para el servidor, quien asume que todas las sentencias deben completarse para asegurar la consistencia e integridad de los datos. Esto es posible debido a que SQL cuenta con las primitivas de lenguaje COMMIT y ROLLBACK.

Una transacción se inicia con cualquier sentencia SQL y finaliza exitosamente con COMMIT (que a su vez inicia un nueva transacción) o falla con ROLLBACK.

La Figura 52 muestra los tres escenarios posibles de finalización de una transacción estándar.

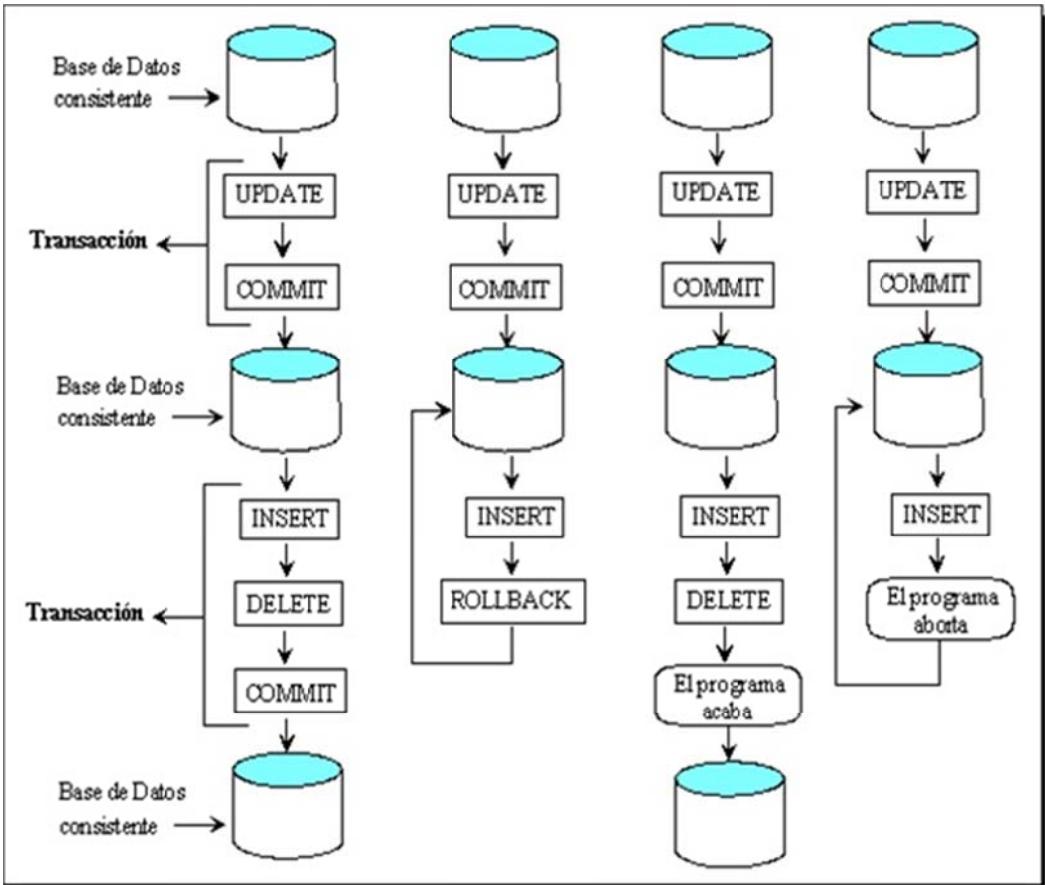


Fig. 52. Ejecución de Transacciones Estándar

Transacciones en el procesamiento multiusuario

Según Edgar Frank Codd, el creador de las bases de datos relacionales:

"Durante un transacción el usuario verá una vista completamente consistente de la base de datos. El usuario nunca verá modificaciones no cumplimentadas de otros usuarios, e incluso los cambios cumplimentados efectuados por otros, no afectarán a los datos examinados por el usuario en mitad de una transacción"

Este principio se basa en el concepto de aislamiento de las transacciones pero también requiere un mecanismo adicional que permita asegurarla. Este mecanismo se logra estableciendo niveles de cerramiento. Los niveles de cerramiento pueden fijarse sobre toda la base de datos (lo que implicaría que la misma se transformaría en un recurso monousuario perdiendo la capacidad de acceso concurrente) o a nivel de sus distintos objetos.

Las alternativas, conceptualmente serían:

- Cerramiento a nivel de tabla: ningún usuario puede acceder a una tabla utilizada por otro usuario hasta que éste último libere el recurso. Si la tabla es muy grande esto podría provocar una larga cola de usuarios en espera, siendo que quizás no quieran acceder todos a la misma porción de la tabla.
- Cerramiento a nivel de fila de una tabla: ningún usuario puede acceder a la fila o conjunto de filas utilizadas por otro usuario hasta que éste último libere el recurso.

Este esquema de cerramiento es incierto dado que depende del conjunto de filas cerrado por el usuario.

La mayoría de las implementaciones de servidores de base de datos definen su propia unidad de cerramiento (generalmente denominadas *páginas*) con un tamaño fijo medido en KBytes y *cierran* el acceso a usuarios concurrentes esa porción de datos.

A su vez, para aumentar el acceso concurrente, se utilizan esquemas con más de un tipo de cierre, siendo habitual en tal sentido implementar al menos dos tipos de cierre: Cierre Compartido y Cierre Exclusivo.

Para comprender el funcionamiento, supongamos dos transacciones A y B y veamos si puede o no acceder una a los recursos cerrados por la otra, según cada una de ellas haya fijado un cierre compartido o exclusivo. La Figura 53 muestra lo que sucede en cada caso.

Transacción A	Transacción B			
	Acción	No Cerrado	Cierre Compartido	Cierre Exclusivo
	No Cerrado	SI	SI	SI
Cierre Compartido	SI	SI	NO	
Cierre Exclusivo	SI	NO	NO	

Fig. 53. Acceso permitido o restringido según tipo de cierre de las transacciones A y B

6.5. Modelos de Procesamiento de Transacciones. Transacciones Distribuidas

El concepto de transacción tal como se ha enunciado funciona adecuadamente cuando el servidor de base de datos SQL se encuentra centralizado y es el único que ejecuta las transacciones. Pero este modelo resulta insuficiente cuando la transacción debe ejecutarse en más de un servidor (transacción distribuida) o bien cuando, por su naturaleza, la transacción debe ser cumplimentada parcialmente debido a, por ejemplo, su longitud.

En cualquiera de estos casos, las transacciones planas resultan poco adecuadas o insuficientes para lograr el objetivo.

Transacciones Encadenadas

Una transacción encadenada introduce un concepto de control lineal para secuenciar transacciones. La forma más fácil de implementarla es con puntos de guarda dentro de una transacción plana que permite salvar cambios hasta el *commit* definitivo. Si se produce un *rollback* se hace hasta el último punto de guarda, pero si el sistema se cae (*crash*) se pierden todos los puntos de guarda, es decir no cumplen la propiedad de Durabilidad.

La Figura 54 muestra el funcionamiento de transacciones encadenadas con puntos de guarda. Estos funcionan como COMMIT parciales pero no dividen la transacción en sub-transacciones.

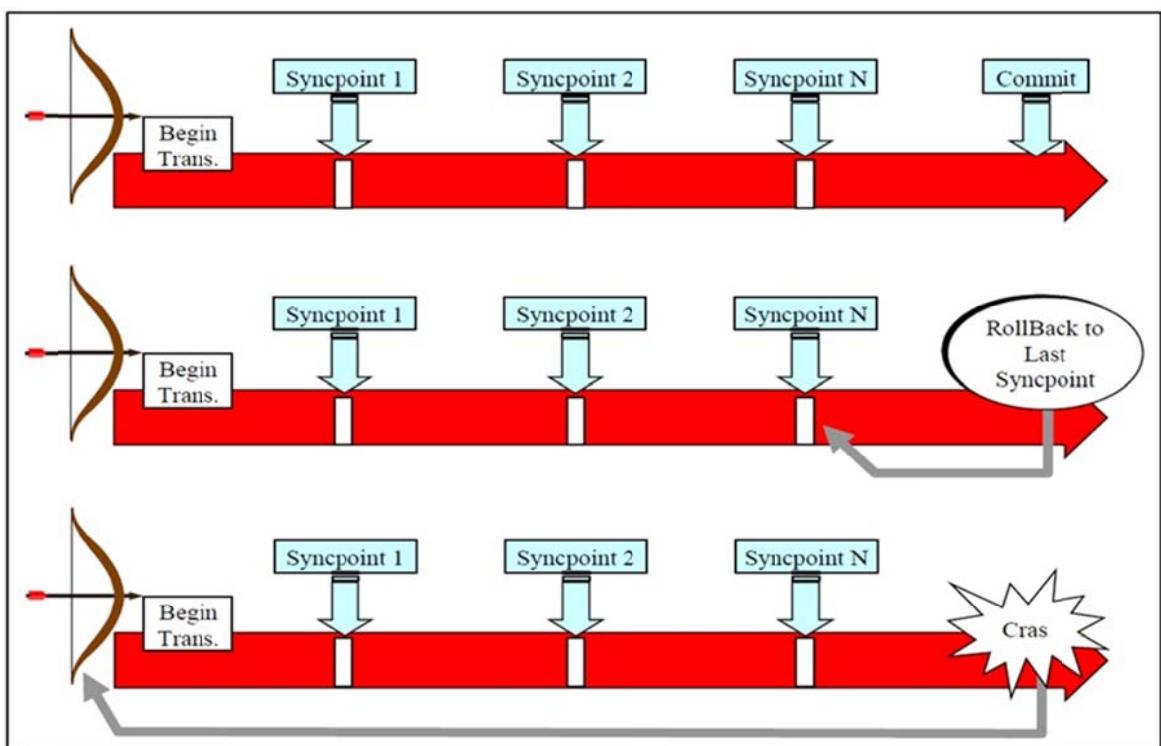


Fig. 54. Transacciones Encadenadas con Puntos de Guarda

Transacciones Anidadas

Las transacciones anidadas definen transacciones dentro de transacciones. Conservan la misma semántica que los procedimientos de un lenguaje de programación, estableciendo una relación jerárquica de sub-transacciones.

La transacción principal comienza las sub-transacciones y cada una de ellas puede hacer commit y rollback de sus piezas de trabajo locales. Cuando una sub-transacción termina, los resultados están sólo disponibles para el parent.

El commit de una sub-transacción se hace permanente después del commit local y el de todos sus ancestros.

La Figura 55 muestra el funcionamiento de las transacciones anidadadas.

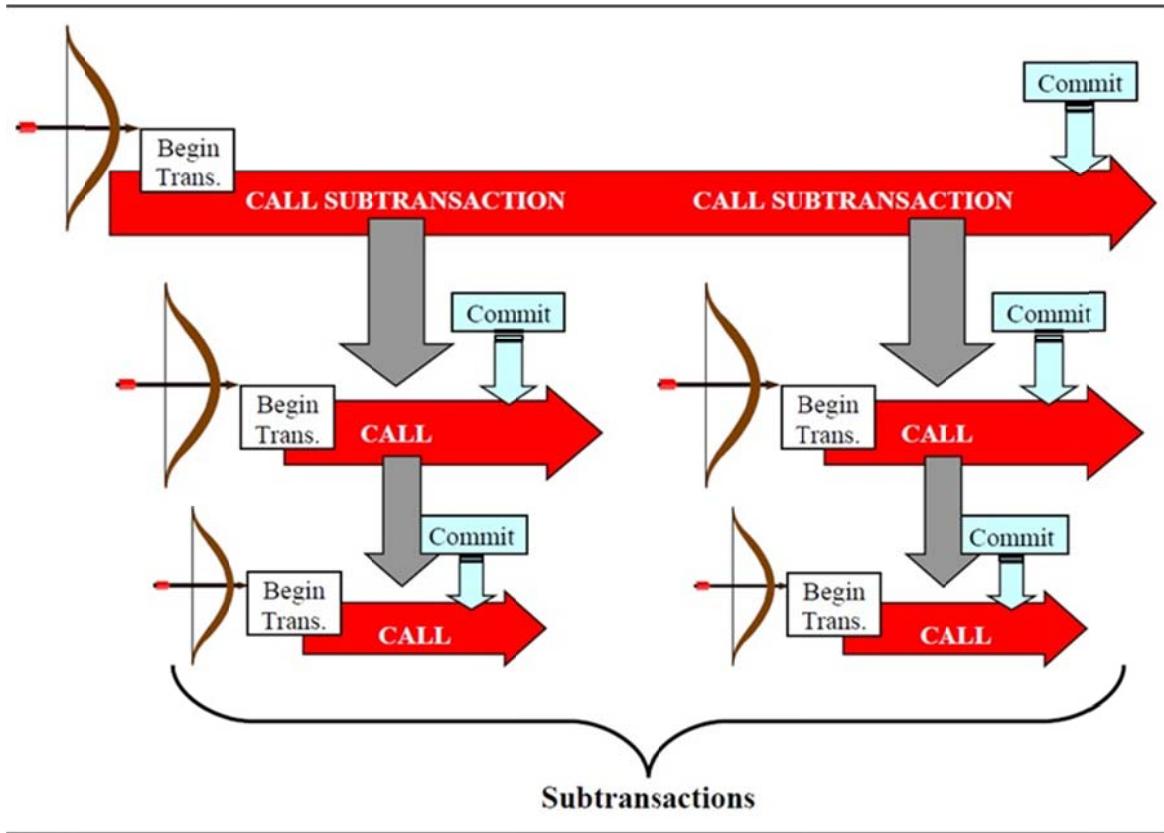


Fig. 55. Funcionamiento de Transacciones Anidadas

6.6. Monitores de Transacciones (TP Monitors). TP-Lite o TP-Heavy

Los monitores de transacciones son un concepto que apareció ya en los viejos mainframes brindando un ambiente de ejecución robusto y permitieron soportar aplicaciones OLTP (On Line Transaction Processing) a gran escala.

Con la migración de las OLTP a plataformas Cliente/Servidor clásico surgió la necesidad de introducir nuevamente la idea de monitores de transacciones y surgieron muchos productos comerciales para tal fin.

Los monitores de transacciones administran las transacciones desde su punto de origen a través de uno o más servidores. Cuando la transacción termina se garantiza un estado consistente del sistema. Además de ejecutar transacciones, los monitores de transacciones encaminan las mismas a través de la red, balancean la carga de su ejecución y las recuperan después de fallas. Se comportan como un sistema operativo para las transacciones y aseguran la posibilidad de contar con transacciones distribuidas.

TP-Lite y TP-Heavy

TP-Lite es simplemente la integración de funciones de un monitor de transacciones en los motores de base de datos (a través de procedimientos almacenados y Triggers).

TP-Heavy son los monitores de transacciones propiamente dichos que realizan la administración, balanceo de carga, sincronización y monitoreo de las transacciones. (Ejemplos: CICS, Tuxedo y Jaguar CTS).

A continuación analizamos distintas características y su comportamiento en TP-Lite y en TP-Heavy.

Alcance del Commit

En TP-Lite los procedimientos almacenados son definidos en un dialecto de SQL y almacenados como un objeto dentro de la base de datos, constituyendo una unidad transaccional pero no pudiendo participar con otras unidades transaccionales respecto de una transacción global. Si A llama a B, B es otra unidad transaccional y luego A muere, los commit hechos por B no pueden ser deshechos luego de la muerte de A, violando el *todo o nada*. Por lo tanto la única solución es que A y B estén en la misma transacción generando una unidad demasiado grande.

Por su parte, los procesos TP-Heavy son escritos usando lenguajes procedurales estándar (no SQL) permitiendo lograr el *todo o nada* y el concepto de transacción global.

Administración de recursos heterogéneos

Un procedimiento almacenado en TP-Lite solo puede persistir transacciones que afecten a recursos propios del vendedor de la base de datos. Los procedimientos TP-Heavy pueden persistir las transacciones sobre múltiples recursos de distinto tipo dentro del alcance de una transacción.

Administración de procesos

Un proceso almacenado en TP-Lite es invocado, se ejecuta transaccionalmente y puede quedar en memoria temporal para reusos futuros.

Un proceso TP-Heavy es manejado como una clase de servidor pudiendo manejar prioridades, utilizar barreras (firewalls) para que otros procesos no los interfieran y si la clase muere, la transacción puede ser reasignada a otra clase.

Invocaciones Cliente/Servidor

Una invocación a un procedimiento es totalmente no estándar, se realiza a través del mecanismo de RPC provisto por el vendedor.

En TP-Heavy el mecanismo de interacción es cualquiera (RPC o mensajes).

Performance

Los procedimientos almacenados poseen un mayor rendimiento en cuanto al tráfico de red respecto del SQL estático o embebido, pero son incapaces de mejorar la performance en otros aspectos, como por ejemplo el balanceo de carga.

TP-Heavy que tiene código precompilado permite multiplexar los requerimientos del cliente.

6.7 - Conclusiones del Capítulo

El capítulo presenta un análisis detallado del uso e impacto de los servidores de base de datos SQL en la construcción de aplicaciones distribuidas.

El lenguaje SQL, además de ser un poderoso lenguaje de consulta se transforma en un mecanismo de comunicación válido hacia los servidores de bases de datos. Estos a su vez han ampliado sus prestaciones hacia modelos distribuidos de más de dos niveles y además han adquirido suficiente versatilidad para comportarse como componentes de ejecución que permite alojar la lógica de negocio que hasta el momento sólo podía ser ejecutada por los clientes.

CAPÍTULO 7

GroupWare

Alejandro Fernández

En la literatura nos referimos con el término groupware (contracción de las palabras grupo y software) a los sistemas de software que incluyen características específicas para asistir a grupos de personas trabajando en conjunto. Ellis, Gibbs y Rein definen groupware como

“Sistemas informáticos que soportan a grupos de personas involucradas en una tarea común y que proveen una interface a un ambiente compartido” (Ellis, 1991). Esta definición plantea dos puntos importantes. Por un lado, que existe una tarea común, u objetivo, que los usuarios del sistema persiguen en conjunto. Por otro, que dicho grupo comparte un ambiente digital al que tiene acceso por medio del sistema. En ese ambiente contiene cosas como los documentos y herramientas que comparten, detalles de su plan de trabajo, información sobre quiénes son los miembros del grupo y que es lo que aportan, y sus mecanismos de comunicación. Los Johnson-Lenz agregan algo más en su definición: “Procesos grupales intencionales más software para darles soporte” (Johnson-Lenz, 1998). El grupo utilizará ciertas estrategias (procesos) para alcanzar su objetivo. El rol del software es asistir a los miembros del grupo en la implementación de dichas estrategias. Las dos partes, procesos grupales y software, hacen al groupware.

La amplia mayoría de los sistemas groupware incluyen aspectos de sistemas distribuidos y por eso tienen su lugar en este libro. Y, por lo que vimos en el párrafo anterior, tienen sus particularidades. En este capítulo vamos a analizar esas particularidades en detalle.

El tipo de estrategia que utilizará el grupo para alcanzar su objetivo determina las características del software que les asistirá¹¹. Veremos primero cuales son las posibles alternativas y que implica en términos de funcionalidad necesaria.

7.1. CSCW: Computer-Supported Cooperative Work

CSCW es el estudio de cómo las actividades de trabajo en grupo¹² (colaboración, cooperación) pueden ser asistidas por sistemas de computadoras. Abarca tanto a la tecnología así

11 A veces, la herramienta disponible determina la estrategia de trabajo: para quien solo tiene un martillo, todo parece un clavo. Para los que hacen groupware, la herramienta y los procesos intencionales son dos elementos de un todo y por lo tanto se diseñan en conjunto.

12 En este capítulo intercalamos el uso de los términos equipo y grupo aunque sabemos que el concepto de equipo involucra vínculos más fuertes.

como su efecto psicológico y sociológico/organizacional. En consecuencia, es un área multidisciplinaria que involucra a profesionales de la informática, sociología, psicología, pedagogía, entre otros. El groupware, en este contexto, es el *Computer-Support* (asistencia que nos da la tecnología). Y qué groupware necesito, o qué necesito del groupware, depende de cómo defina el *Cooperative Work* (trabajo cooperativo).

Algunas formas de trabajo en equipo se ajustan mejor que otras a una meta en particular. Por ejemplo, si nuestra meta es obtener una lista amplia de páginas web que hablan de sistemas distribuidos modernos, podemos buscar en paralelo, todos al mismo tiempo, sin mucha necesidad de coordinarnos. Pero si queremos escribir un único reporte, en equipo, sobre las características más relevantes de dichos sistemas, probablemente queramos armar un plan y tomar turnos para escribir y no pisarnos.

Dada una forma de trabajo en equipo, algunas tecnologías (o funcionalidades) nos asisten mejor que otras. Siguiendo con el ejemplo anterior, si vamos a buscar y recolectar páginas web en paralelo sin mucha coordinación, nos alcanza con la funcionalidad que ofrece una lista de correo electrónico (o grupo de WhatsApp¹³), donde todos publicaremos lo que encontramos. Pero si decidimos trabajar con un plan y tomar turnos para no pisarnos al escribir nuestro reporte elaborado, vamos a necesitar funcionalidad para saber a quién le toca, quién ya terminó con su tarea, y qué tareas quedan pendientes.

Si caracterizamos la forma de trabajo en equipo en términos del objetivo principal, podemos hablar de cuatro situaciones a las que llamaremos: informar, coordinar, colaborar, cooperar¹⁴.

Informar

Hay situaciones de trabajo en equipo donde lo importante es mantener a una parte del grupo informada sobre algo que la otra parte conoce. Veamos algunos ejemplos:

El profesor quiere informar a los alumnos sobre los contenidos de la asignatura (así, entre todos, resuelven la tarea de aprender)

La Facultad quiere informar a los docentes los resultados de la última autoevaluación institucional (así, entre todos, mejoran como espacio de enseñanza).

Una empresa de desarrollo de software quiere comunicar a todos sus programadores las mejores prácticas que ha elaborado (para aumentar la calidad de sus productos).

Un experto programador quiere transmitir su experiencia a otros (para ayudarlos a programar mejor)

Lo que caracteriza a estas situaciones es que la información fluye en un solo sentido (o al menos ese es objetivo principal). Del profesor a los alumnos, de la Facultad a los docentes, de

13 <http://www.whatsapp.com>: aplicación de mensajería instantánea móvil

14 La selección de nombres para estas situaciones respeta la propuesta original de Bair (1989). La separación entre los términos en lenguaje natural no es tan clara (en particular entre colaborar y cooperar)

la empresa a sus programadores, del experto a otros. Además, el que origina la información no necesita conocer a quienes van a recibirla, y tiene poco (o ningún) contacto con ellos.

Que tiene que hacer (al menos) una tecnología para ser útil en esta situación:

1. Debe ayudar al proveedor de información en la preparación de la información a transmitir
2. Debe ofrecer al proveedor una forma de publicar la información para que los receptores puedan accederla
3. Debe asegurar que los receptores tienen acceso a la misma y la encuentran fácilmente
4. Podría incluir mecanismos para que los receptores de información se enteren de que hay nueva información disponible y para que el proveedor sepa si (cuanto) ha sido accedida.

¿Qué tecnologías conocemos que cumplen, al menos, con esos requerimientos? Un editor de texto para crear una página web, y ftp para desplegar la página en un servidor web alcanza para proveedor de información. Si sumamos una URL conocida aseguramos que los receptores la encuentren. O alternativamente, podríamos una lista de correo a la que el proveedor envía la información y a la que los receptores se suscriben. Con eso alcanza. Podríamos agregar mucha más funcionalidad (por ejemplo, la posibilidad de comentar como en los blogs, o de permitir que todos editen la información como en Wikipedia) pero eso requiere más esfuerzo (de quien hace el sistema y de quien tiene que aprender a usarlo) y nuestro escenario no lo requiere.

Coordinar

Hay situaciones de trabajo en grupo donde lo importante es coordinar el uso de elementos compartidos. Puede ser un aula, una herramienta, un documento, o el tiempo disponible de los participantes. Veamos algunos ejemplos:

- Los profesores quieren coordinar el uso de las aulas, saber cuáles aulas están libres y cuáles ocupadas.
- Los programadores quieren coordinar el uso de los archivos de código fuente (para evitar editarlos concurrentemente). Quieren saber si un archivo está siendo editado por otro programador, o si pueden editarlo ellos.
- Los profesores y los alumnos quieren coordinar sus calendarios para que no se solapen los horarios de clase de las asignaturas. Los alumnos quieren saber el horario de cada clase y necesitan que los docentes tengan en cuenta sus horarios al planificar.

Lo que caracteriza estas situaciones es ambas partes deben entrar en contacto a fin de coordinar el uso de lo que planean compartir. Se coordinan porque lo que comparten es importante para que puedan conseguir sus objetivos (aunque no necesariamente deben ser los mismos para todos los involucrados). La información fluye ahora en ambos sentidos.

Que tiene que hacer (al menos) una tecnología para ser útil en esta situación:

1. Debe tener un modelo que represente información sobre los recursos compartidos

2. Debe registrar y mantener consistente el estado de los recursos compartidos
3. Debe permitir a los involucrados consultar el estado de los recursos compartidos
4. Debe permitir a los involucrados negociar el uso de los recursos compartidos

¿Qué tecnologías conocemos que cumplen, al menos, con esos requerimientos? Para coordinar el uso del tiempo se puede utilizar un calendario compartido (por ejemplo Google Calendar¹⁵) o herramientas pensadas específicamente para coordinar reuniones (por ejemplo Meet-o-matic¹⁶ o Doodle¹⁷). Cuando el recurso a compartir o las formas de negociación son más complejos (o específicos) se hace necesario construir aplicaciones a medida. Algunas universidades tienen sistemas de software para la gestión aulas y horarios que ofrecen exactamente la funcionalidad indicada. Otras construyen un sistema combinando tecnologías que no cumplen con todos los requerimientos lo que las fuerza a resolver el resto de ellos con estrategias de trabajo. Por ejemplo, una planilla de cálculo alcanza para almacenar información sobre las aulas y su estado. La negociación de uso y consultas sobre el estado se hacen personalmente o por correo siguiendo alguna política pre-acordada. Mantener la consistencia es responsabilidad de alguna persona.

Colaborar

Hay situaciones de trabajo en equipo donde lo importante es que cada uno haga su parte para que, entre todos, alcancen un objetivo. Veamos algunos ejemplos:

- En una línea de producción automotriz, cada uno en su estación de trabajo debe hacer bien su tarea para que al final de la línea haya un auto listo.
- En un proyecto de desarrollo de software que utiliza procesos tradicionales, los analistas deben hacer bien su análisis, los diseñadores bien su diseño, los programadores bien sus programas y los testers bien sus pruebas, para que al final tengamos una buena aplicación. El líder de proyecto debe saber en todo momento como van.
- En una oficina de reclamos, alguien atiende el teléfono, registra el reclamo y da al interesado un número de reclamo. El reclamo pasa a la oficina que corresponde para ser analizado y derivado a quien debe resolverlo. Quien debe resolverlo es notificado, lo resuelve y lo registra como resuelto. Quien recibió el reclamo ve que está resuelto y llama al interesado para confirmar que el problema no persiste. De confirmarse, el reclamo se da por resuelto.

Lo que caracteriza a estas situaciones es que los involucrados participan en un mismo proceso. Hay un resultado conjunto pero cada uno es responsable de una parte (se lo evalúa principalmente por cómo cumple su tarea). La frecuencia y forma de interacción entre los involucrados es variada, dependiendo de cómo sus tareas están vinculadas. Cada uno debe saber

¹⁵ <http://calendar.google.com>: Aplicación de Calendario de Google

¹⁶ <http://www.meetomatic.com>: Aplicación web para organizar reuniones

¹⁷ <http://doodle.com/es/> : Aplicación web para organizar reuniones

con claridad cuando comenzar, cuando se espera que termine, y debe tener disponible los materiales y herramientas necesarios. Hay alguien que necesita saber, en todo momento, el estado del proceso.

Que tiene que hacer (al menos) una tecnología para ser útil en esta situación:

1. Debe tener un modelo del proceso, que incluya información sobre las tareas que lo componen y las dependencias entre ellas, así como sobre los recursos y las herramientas involucrados en las tareas (por ejemplo, el proceso desde que llaman con un reclamo hasta que se resuelve).
2. Debe permitir iniciar nuevas instancias del proceso (por ejemplo, cada vez que alguien llama con un nuevo reclamo se inicia una nueva instancia del proceso).
3. Debe permitir asignar tareas a los miembros del equipo.
4. Debe tener información sobre las instancias del proceso concluidas y las que están en ejecución.
5. Para cada instancia del proceso en ejecución debe tener información sobre su estado (qué tareas están concluidas, cuales están en ejecución, cuales están pendientes, quienes son los responsables, que tiempos llevan).
6. Debe permitir, a cada involucrado conocer sus tareas pendientes y acceder a los recursos y herramientas que necesita para resolverlas.

Si alguna vez hizo un trámite en una repartición no informatizada habrá visto ejemplos de estas situaciones resueltas con carpetas y con procesos gestionados íntegramente por personas. Es menos frecuente encontrar tecnologías que cubran estos requerimientos (simplemente porque son más complejas). Se pueden utilizar carpetas compartidas (servidores de archivos o servicios como Dropbox¹⁸) para mantener y distribuir los recursos compartidos (por ejemplo documentos). Se puede utilizar el correo electrónico para notificar a alguien cuando debe iniciar su tarea, adjuntando el link a los recursos que necesita y tomar nota en un documento del cambio de estado del proceso. Pero por lo general, este tipo de situaciones requieren el desarrollo de aplicaciones a medida. Por suerte son tan comunes que han dado origen al desarrollo de múltiples librerías reutilizables y generadores de aplicaciones. Se las puede encontrar bajo el nombre de librerías de *workflows* o librerías de para procesos de gestión de negocios (*Business Process Management*).

Cooperar

Finalmente, hay situaciones de trabajo en equipo en las que resulta imposible (o contraproducente) pensar tarea en términos de subtareas, coordinadas, a ser desarrolladas por separado. Veamos algunos ejemplos:

¹⁸ <https://www.dropbox.com/home> : Servicio de alojamiento de archivos en la nube

- Un grupo de alumnos tienen que escribir un reporte sobre los desafíos de seguridad en el desarrollo de aplicaciones distribuidas. El informe debe estar escrito con un estilo homogéneo, evitando redundancia y con un hilo conductor claro. Los alumnos serán evaluados grupalmente por el resultado final.
- Una organización debe decidir entre varias estrategias de acción para el desarrollo de una nueva línea de negocios. Los involucrados pertenecen a varios departamentos (ventas, ingeniería, recursos humanos, finanzas). La decisión debe tener en cuenta los requerimientos y conocimientos de todos los involucrados. La responsabilidad de la decisión es compartida; el resultado ya sea éxito o fracaso, recae sobre todos.

Lo que caracteriza a estas situaciones es la responsabilidad compartida de alcanzar el resultado esperado y la dificultad que llegar a una resolución mediante una estrategia de división de tareas. El objetivo del grupo toma preponderancia sobre los objetivos individuales. La interacción entre los participantes es frecuente y se concentra en recursos compartidos (documentos, herramientas, datos, conocimientos) a los que deben acceder en forma simultánea. Se trata de situaciones que en la mayoría de los casos requieren interacción cara-a-cara.

Que tiene que hacer (al menos) una tecnología para ser útil en esta situación:

- Debe permitir el acceso simultáneo a los recursos compartidos, maximizando la participación (p.e., evitando bloqueos innecesarios), y garantizando la consistencia de los mismos aun cuando se los modifique concurrentemente.
- Debe ofrecer mecanismos ricos para la comunicación frecuente, subsanando los desafíos de la distribución cuando las reuniones cara a cara no son posibles.
- Debe asistir en la toma de decisiones participativa por medio de modelos y mecanismos que permitan representar combinar las distintas perspectivas, sus funciones de valor (como evaluar el costo/beneficio de cada alternativa), y las alternativas de acción disponibles.

En los últimos años hemos visto mucho desarrollo en soluciones que apuntan a este tipo de situaciones. Compañías como Google, Microsoft, y Zoho ofrecen procesadores de textos y planillas de cálculo que cubre una amplia gama de requerimientos para el trabajo cooperativo, fluido, en documentos compartidos. No solo permiten que múltiples usuarios editen el documento en simultáneo sino que incluyen herramientas para asistir en el proceso de creación conjunta (revisiones, comentarios y seguimiento de cambios, conversaciones dentro del documento mismo, sugerencias, etc.). Cada vez es más frecuente encontrar herramientas de comunicación como video conferencia, mensajería, y mensajes de voz, integrados con las herramientas de creación cooperativa de contenidos. Si bien la toma de decisiones participativa es un tema ampliamente explorado (por ejemplo en el área de Investigación Operativa), es menos frecuente encontrar tecnologías que la hagan accesible al público en general. Esto se debe no solo a la complejidad técnica de este tipo de sistemas sino a la dificultad de incorporarlas en los procesos grupales de las organizaciones.

7.2. Características del Groupware

En la sección anterior discutimos, en términos generales, que se espera de una tecnología dada una determinada situación de trabajo en equipo. También hemos planteado que una solución de groupware es una combinación de software y procesos grupales. En algunos casos es necesario construir software a medida que se complemente de forma adecuada con formas de trabajo. En otros casos se pueden utilizar aplicaciones groupware existentes adaptando los procesos de trabajo para complementarlas. Algunos autores, entre ellos Johnson-Lenz [Johnson-Lenz, 1991] y Orlikowski [Orlikowski, 1997] argumentan que el desarrollo de groupware es un proceso de co-evolución. Se adoptan herramientas que dan soporte a ciertos procesos grupales existentes. Los procesos grupales cambian como consecuencia de la incorporación de tecnologías y esto trae aparejado un nuevo ciclo de cambio tecnológico. Con esto en mente, y sabiendo lo costoso que es el desarrollo y adopción del groupware, uno puede comenzar adoptando tecnologías groupware simples y flexibles para, a medida que el uso de las mismas madura, ir incorporando más estructura.

Alcance y estructura

El espacio de las aplicaciones groupware es muy variado. Si pensamos en términos de alcance, encontramos desde sistemas groupware abarcativos, que cubren todos los aspectos de un trabajo en particular, hasta pequeños programas genéricos de mensajería instantánea. Si pensamos en términos de cuánta estructura imponen y cuanto fijan las reglas de trabajo en equipo, tenemos desde sistemas que fuerzan una forma de trabajo particular hasta sistemas muy flexibles que pueden utilizarse en muchas situaciones y de muchas formas. Dos ejemplos extremos nos pueden servir para entender mejor este aspecto.

Abarcativos y controladores: Algunos sistemas cubren todos los aspectos del trabajo que el grupo debe realizar, establecen bien claro la forma en que debe realizarlo, y controlan que así se haga. Los sistemas que informatizan los servicios de atención al cliente suelen entrar en esta categoría. Cubren las situaciones de *información*, con herramientas que fijan los procesos con los cuales la organización informará a sus usuarios sobre los servicios ofrecidos. Cubren las situaciones de *coordinación* con mecanismos que aseguran que el acceso a los documentos compartidos se hace de forma segura y se mantienen consistentes. Enfatizan el soporte a situaciones de *colaboración* con un modelo del proceso que se debe respetar, y funcionalidad para darle seguimiento y visibilidad.

Acotados y maleables: Como contraste encontramos soluciones groupware de foco muy acotado y adaptable a variados escenarios de uso. Meet-o-matic y Doodle son soluciones groupware cuyo único objetivo es asistir a un grupo de personas que intenta encontrar el momento óptimo para una reunión (una clara situación de *coordinación*). Estas herramientas no intentan ofrecer asistencia a lo que pueda suceder antes, durante, o después de las reuniones. Tampoco se enfocan en los recursos (además del tiempo) ni las herramientas que serán utilizados en la reunión. Si bien sugieren una estrategia para su uso (que combina notificaciones

por correo electrónico y reportes de las propuestas más populares) no la imponen. Están pensadas para ser utilizadas en cualquier situación (organizar una reunión de trabajo, un paseo, una reunión de amigos) y sin imponer una forma particular de negociar las alternativas y llegar a una decisión.

Awareness

Para participar eficientemente y productivamente en un grupo es importante saber que se espera de nosotros, que es lo que los otros miembros hacen, que es lo que pasó mientras no estábamos, qué es lo que está pasando ahora. En Groupware utilizamos el término **Awareness** para referirnos al "conocimiento de las actividades de otros que provee contexto para tus propias actividades" (Dourish, 1992).

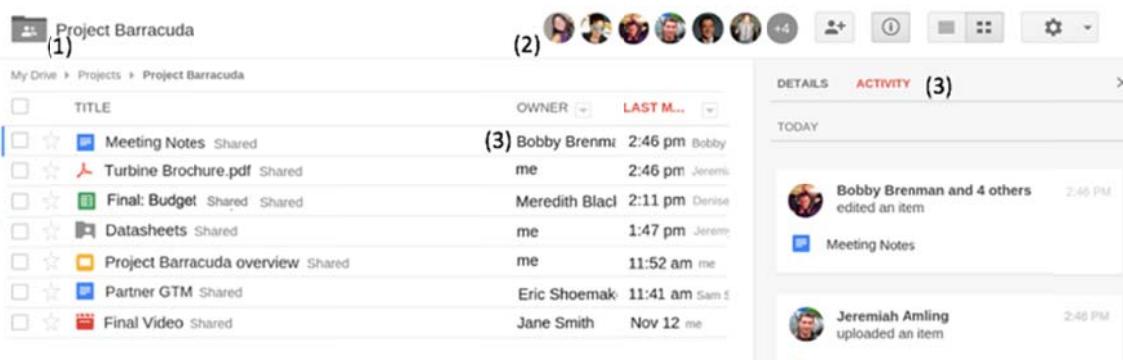


Fig. 56. Captura de pantalla de Google Drive que incluye elementos de awareness

La Figura 56 muestra una captura de una carpeta compartida en Google drive. Es importante saber que el contenido de la carpeta es compartido y otros podrán ver y modificar lo que ahí se encuentre. Google drive utiliza un icono particular para indicar cuando una carpeta está compartida (1). Quienes pueden acceder al contenido compartido (2) nos permite confirmar que solo quienes deben acceder a nuestros documentos lo hacen, y que todos los que necesitan acceder pueden. En ciertas situaciones es importante saber quién es el propietario de un recurso compartido (3). Por lo general el propietario es quien tiene la última palabra al respecto y quien puede dar o quitar permisos de acceso. Google Drive es un espacio en la nube para almacenar documentos y, si queremos, para compartirlos. Los documentos que dejamos en un espacio compartido pueden ser modificados en nuestra ausencia. Saber que paso en el espacio compartido mientras no estábamos mirando (3) nos permite ajustar nuestro accionar.

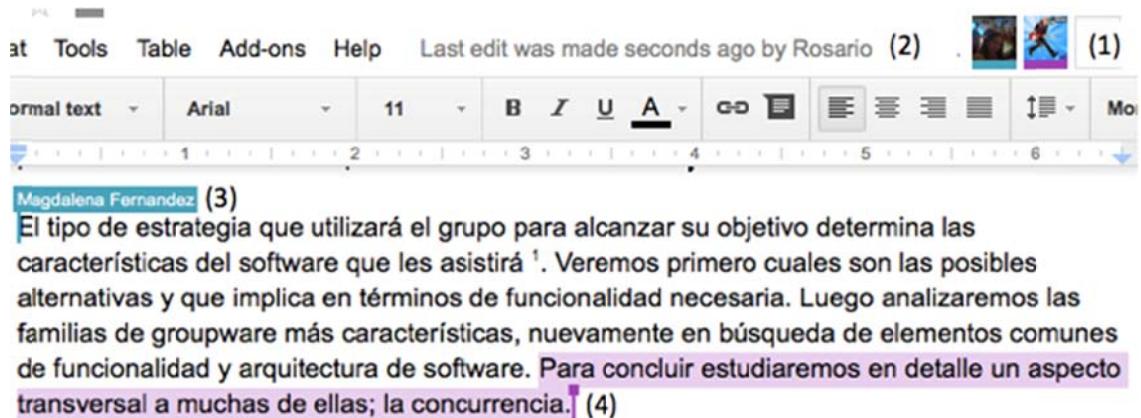


Fig. 57. Captura de pantalla de Google Docs mostrando elementos de Awareness

La Figura 57 muestra una captura de pantalla de Google Docs, uno de los primeros procesadores de texto cooperativo de uso masivo. En el momento que se tomó la captura, tres personas se encontraban trabajando en el documento simultáneamente. Arriba a la derecha de la imagen se observan los avatares de los dos usuarios "remotos" (1). Para todos ellos es muy importante coordinar sus acciones a fin de evitar molestar o duplicar trabajo. Cursos con colores asociados a los avatares de los usuarios indican en qué parte del documento se encuentra escribiendo cada usuario remoto. Un pequeño pop-up muestra el nombre del usuario remoto (3). Si un usuario remoto selecciona texto, el mismo se resalta con el color que identifica al usuario en cuestión.

La funcionalidad de awareness permite a los usuarios *coordinar su trabajo basado en el conocimiento de lo que otros hacen o han hecho*. Si alguien selecciona un párrafo, o veo su cursor en un párrafo, evitaré modificarlo para no romper lo que está haciendo. También, motiva la colaboración espontánea. Si cuando abro el documento compartido veo que mi compañero está trabajando en una sección respecto a la cual tengo observaciones, intentaré contactarlo (por ejemplo vía chat) para conversar con él ahora que está en tema. La información de awareness mantiene a los miembros del equipo mejor informados del estado del proyecto. Esto es particularmente importante en actividades poco estructuradas

Quienes usan con frecuencia redes sociales saben lo duro que es lidiar con la sobrecarga de información que generan las actividades de sus contactos. Muy poca información y nos perdemos oportunidades interesantes (p.e., alguien organiza una fiesta de reencuentro y no nos enteramos). Mucha información y puede suceder lo mismo. Las novedades realmente interesantes quedan sepultadas bajo una montaña de datos. ¿Qué nos preguntamos para diseñar funcionalidad de awareness que resulte útil?

Contexto de awareness

Lo primero a reconocer es cuál es la situación que enmarca el ambiente en el que queremos ofrecer awareness, el contexto. En una situación de "información" (como se definió en la sección 7.1) es importante que los destinatarios de la información se enteren cuando la misma se vuelve disponible o es actualizada. RSS (Real Simple Syndication) es una estrategia utilizada

por muchos sitios web para notificar a sus lectores cuando nueva información está disponible. Es un mecanismo para ofrecer awareness de cambios. Actualmente, el RSS está siendo reemplazado por la difusión en redes sociales. En una situación de información, es importante que el creador de los contenidos conozca el alcance que tuvo. En la actualidad este tipo de awareness se puede obtener integrando en los sitios funcionalidad de web analytics. Un análisis similar puede hacerse para situaciones de *coordinación*, *colaboración*, y *cooperación*. Siempre se debe tener en cuenta que ofrecer mucha información (aunque para el programador implique poco esfuerzo) es tan malo como ofrecer muy poca. Otros elementos que comúnmente se tienen en cuenta para definir el contexto en el que se ofrecerá awareness es si el usuario recién ingresa al ambiente (luego de una ausencia corta, luego de mucho tiempo, etc.), que está haciendo (por ejemplo, si está teniendo una conversación, buscando algo, trabajando en un recurso compartido), y si otros trabajan al mismo tiempo con él.

¿Awareness respecto a qué o quién?

En las redes sociales nos enteramos de lo que dicen y hacen otros, pero no todo el mundo, solo nuestros amigos. Puedo incluso configurar de quienes quiero o no quiero recibir noticias. Si estamos trabajando en una carpeta compartida en red, ¿respecto a qué cosas queremos información? ¿Todos los documentos en la carpeta o solo aquellos en los que hemos trabajado? ¿Es posible definir mi foco de atención en cada momento para solo recibir información respecto a ello? En Google drive, por ejemplo, la vista de actividad se actualiza para mostrar solo las novedades de la carpeta seleccionada (Figura 58, imagen 1).

Privacidad y confianza

¿Cuánta información sobre mí y mis acciones está divulgando el sistema con la excusa de proveer awareness? WhatsApp introdujo la idea del doble tick para indicar que un mensaje habría llegado, y doble tick celeste para indicar que el receptor lo había visto (Figura 58, imagen 2). En comparación con el SMS, popular hasta el momento y poco confiable y predecible, esta nueva funcionalidad (de awareness) fue un diferenciador. Sin embargo, al poco tiempo, muchos usuarios comenzaron a reclamar que les hacía perder su privacidad (ya no podían decir que no les había llegado el mensaje!). Los ticks de WhatsApp hacen que el sistema sea más robusto y confiable a costa de perder un poco de privacidad. El tiempo el balance es el adecuado para sus usuarios. Por ahora, permite a los usuarios desactivar por completo la funcionalidad (Figura 58, imagen 3).

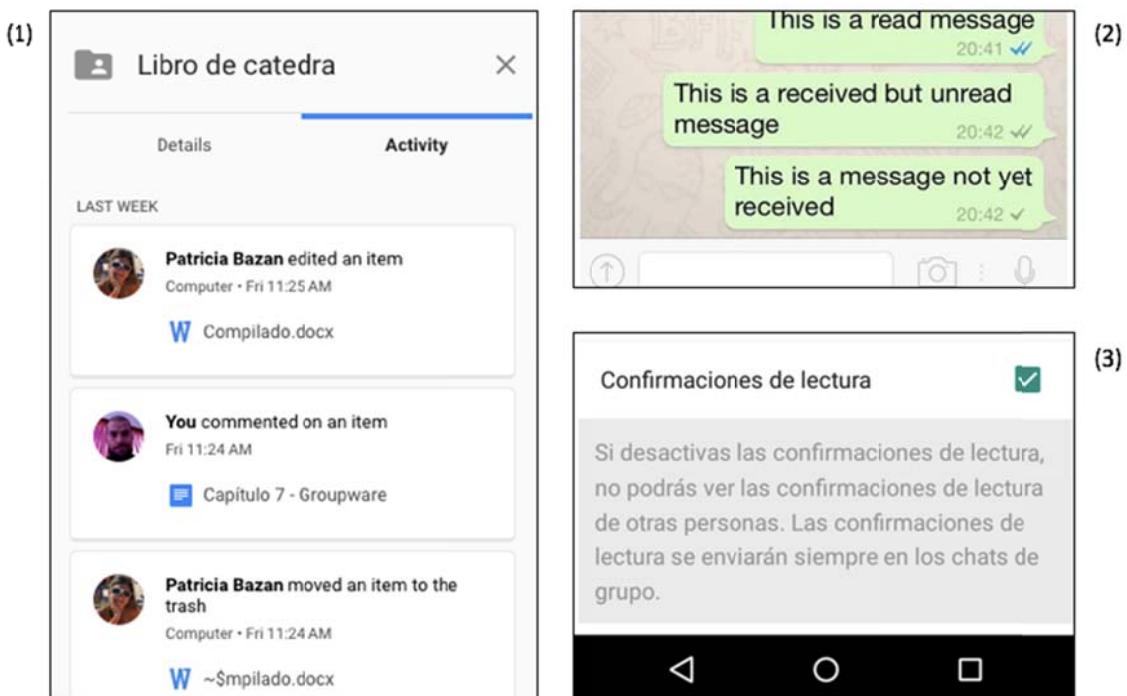


Fig. 58. Awareness en carpetas Google Drive y en WhatsApp

7.3. Conclusiones

El Groupware es una combinación de procesos grupales intencionales y software que asiste a los participantes que llevan a cabo esos procesos. Estudiar Groupware implica, primero, entender en qué tipo de situaciones se da el trabajo en grupo.

En la sección 7.2 vimos que esas situaciones pueden caracterizarse como *informar, coordinar, colaborar, cooperar*. Sabiendo que situación enfrentamos podemos hacernos una idea de lo que se espera del groupware. Lamentablemente, las situaciones del mundo real no son tan fácilmente clasificables. Por lo tanto, esa discusión se ofrece a modo de guía para comenzar el análisis.

Como se discutió en la sección 7.3, el groupware puede abarcar gran parte de la actividad que el grupo debe realizar (como es el caso de un sistema de gestión de expedientes) o solo un aspecto de ella (como es el caso de un sistema de mensajería instantánea). Como consecuencia, a veces es necesario combinar varias soluciones groupware para cubrir todas las necesidades del equipo.

Incluso cuando los miembros del grupo se encuentren en el mismo lugar, al mismo tiempo, el groupware plantea desafíos comunes de sistemas distribuidos como los que discuten en otros capítulos de este libro. Los desafíos más frecuentes tienen que ver con aspectos de distribución de recursos compartidos, manejo de consistencia en el acceso concurrente a los recursos compartidos, autorización y control de acceso, robustez respecto a problemas de conectividad, escalabilidad, e integración.

CAPÍTULO 8

Computación en la Nube

Patricia Bazán

La Computación en la nube (Cloud Computing) es un paradigma que posibilita el acceso ubicuo bajo demanda a servicios TIC accesibles a través de Internet. Se sustenta en la idea consumir sólo lo que se necesita y pagar por lo que se consume, de una manera similar a los servicios de luz o de gas.

El término *cloud* (nube) se refiere a la forma de representar la red (Internet) en los diagramas y es una abstracción de las complejidades de su infraestructura.

El desafío es cómo dimensionar la capacidad de los servidores para ajustarlos a la demanda (servicios estacionales, eventos masivos y puntuales, juegos *on line*).

El origen de este paradigma surge como consecuencia de disponer las infraestructuras privadas de los proveedores de servicios de Internet (ISP), hacia el ámbito global. Esta evolución también ha supuesto un cambio profundo en la forma en que los ISPs gestionan y ofrecen sus recursos añadiendo interfaces de gestión y los mecanismos para implementar el pago por uso (indican cómo medir el consumo de ancho de banda, espacio de almacenamiento, consumo de ciclos de CPU, etc.).

La computación en la nube no constituye una nueva tecnología sino que es un paradigma que combina tecnologías como: SOA, *Web Services*, Virtualización¹⁹ y *Grid Computing*²⁰

El capítulo se organiza de la siguiente manera: en la Sección 1 se detallan las bases conceptuales de la computación en la nube (*cloud computing*), en la Sección 2 se revisan primero los antecedentes tecnológicos que originaron este nuevo modelo de computación. Luego, la Sección 3 se describe el modelo de nube, los modelos de servicio y despliegue. En la Sección 4 se analiza el modelo de nube y su aplicación a BPM. En la Sección 5 se discuten desafíos y métodos de aplicaciones distribuidas en el modelo de nube. Finalmente se presentan algunas conclusiones.

¹⁹ Creación a través de software de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red.

²⁰ Tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado

8.1. Conceptos Generales y Definiciones

Una de las definiciones más referenciadas es la provista por el National Institute of Standards and Technology (NIST) [8.1]:

Definición 1. COMPUTACIÓN EN LA NUBE [NIST]

La computación en la nube es un modelo que habilita, de manera ubicua y bajo demanda, el acceso a la red para compartir un conjunto de recursos de computación configurable (redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser provistos y publicados con un mínimo esfuerzo de administración por parte del proveedor del servicio o de interacción con el mismo.

Otra definición provista por la European Network and Information Security Agency (ENISA), enumera [8.2]:

Definición 2. COMPUTACIÓN EN LA NUBE [ENISA]

La computación en nube es un modelo de servicio bajo demanda para la prestación de TI, a menudo basado en la virtualización y en las tecnologías informáticas distribuidas. Las arquitecturas de computación en nube poseen: 1) recursos con un alto grado de abstracción; 2) escalabilidad y flexibilidad prácticamente instantáneas; 3) prestación casi instantánea; 4) recursos compartidos (hardware, base de datos, memoria, etc.); 5) servicio bajo demanda que suele incluir un sistema de facturación de pago por uso; y 6) gestión programática, como por ejemplo, mediante la API del servicio web.

Si bien las definiciones anteriores son consideradas como las más completas y conocidas, en [8.3] un grupo de reconocidos expertos, presentan más definiciones. Algunas de ellas, se presentan a continuación. Por ejemplo, la definición provista por Furht, Borko, Escalante, y Armando, en su libro “Handbook of Cloud Computing” es la siguiente:

Definición 3. COMPUTACIÓN EN LA NUBE [FURHT ET AL.]

Computación en la Nube puede definirse como un nuevo estilo de computación en el cual recursos dinámicamente escalables y usualmente virtualizados son provistos como servicios a través de Internet.

La definición de UC Berkeley RADLabs indica lo siguiente:

Definición 4. COMPUTACIÓN EN LA NUBE [UCBerkeleyRADLabs.]

Computación en la Nube es un nuevo enfoque de computación caracterizado por: 1) la ilusión de contar con infinitos recursos de cómputos, 2) la eliminación de compromisos de antemano por parte de los usuarios de la Nube, y 3) la habilidad de pagar por el uso, como sea necesario.

Por otra parte, McKinsey & Company enuncia:

Definición 5. COMPUTACIÓN EN LA NUBE [McKinsey]

La Nube son servicios basados en hardware que ofrecen capacidades de cómputos, de redes y de almacenamiento donde la administración del hardware es altamente abstracta de los compradores, los compradores incurren en costos de infraestructura como una variable OPEX (costos operativos), y la capacidad de infraestructura es altamente flexible.

La comparación de las definiciones anteriores se presenta en la Figura 59. Los aspectos comunes incluyen un nuevo modelo de computación, la posibilidad de acceder a recursos ubicuos, escalables y configurables por Internet, la abstracción del hardware y el pago por uso de

DEFINICIÓN	CARACTERÍSTICAS RELEVANTES	ASPECTOS COMUNES
NIST	<ul style="list-style-type: none"> ○ Es un modelo ○ Acceso a recursos configurables ○ Acceso a recursos ubicuos ○ Acceso a recursos compartidos ○ Mínimo esfuerzo de administración ○ Mínimo esfuerzo de interacción 	<ul style="list-style-type: none"> ○ Nuevo modelo de Computación ○ Acceso a recursos ubicuos ○ Recursos escalables ○ Recursos configurables ○ Acceso a recursos por Internet ○ Abstracción del hardware ○ Pago por uso
ENISA	<ul style="list-style-type: none"> ○ Modelo de servicio bajo demanda ○ Recursos con alto grado de abstracción ○ Recursos escalables ○ Recursos flexibles ○ Prestación inmediata ○ Recursos compartidos ○ Servicio bajo demanda ○ Pago de servicios por uso ○ Gestión para programar los servicios 	<p>Modelo abstracción esfuerzo escalables</p> <p>Pago Acceso costos flexibles compartidos bajo programar interacción Prestación alto Nuevo máxima contar idea SERVICIOS accedidos demanda uso Internet configurables Abstracción enfoque Gestión operativos grado Administración hardware Servicio Mínimo servicio Recursos virtualizados Infraestructura provistos flexible</p>
Furht et al.	<ul style="list-style-type: none"> ○ Nuevo estilo de computación ○ Recursos escalables ○ Recursos virtualizados ○ Recursos como servicios ○ Acceso a recursos por Internet 	
UCBerkeley	<ul style="list-style-type: none"> ○ Nuevo enfoque computacional ○ Idea de recursos infinitos ○ Sin compromisos previos ○ Pago por uso 	

recursos computacionales.

Fig. 59. Comparación de Definiciones de Computación en la Nube

On-Premise y Cloud

Un recurso de TI (Tecnología de la Información) que se aloja en un entorno de TI empresarial dentro de los límites físicos de la compañía, se considera que está disponible “bajo las premisas” (*on-premise*) de la misma. Por lo tanto, no es un recurso e TI basado en la nube.

Los términos *on-premise* y *cloud* pueden considerarse antónimos. Un recurso de TI que es *on-premise* no puede ser *cloud* y viceversa [8.6].

Inquilino único (*single-tenant*) o inquilino múltiple (*multi-tenant*)

En una arquitectura *single-tenant* de computación en la nube, cada instancia de una aplicación de software y su infraestructura de soporte, aloja a un solo cliente. Es decir hay una copia del software para cada cliente y sus usuarios y dicho software puede adecuarse a las necesidades de ellos. En contraposición, en *multi-tenant* varios clientes comparten la misma instancia del software de aplicación.

Un claro ejemplo de *multi-tenant* lo constituye por ejemplo Gmail o Yahoo que alojan una sola instancia de aplicación y administran cientos de miles de cuentas de usuarios. Mientras que por ejemplo un cliente de mail de escritorio como Outlook es un ejemplo de *single-tenant*, dado q ejecuta una instancia distinta del aplicativo para cada cliente.

8.2. Antecedentes Tecnológicos

La computación en la nube es un paradigma que posibilita el acceso ubicuo bajo demanda a servicios IT accesibles a través de Internet. El término nube (*cloud*) se refiere a la forma de representar la red (Internet) en los diagramas y es una abstracción de las complejidades de su infraestructura [8.5].

El desafío es cómo dimensionar la capacidad de los servidores para ajustarlos a la demanda (servicios estacionales, eventos masivos y puntuales, juegos *on line*). La Figura 60 representa alguno de los elementos involucrados dentro de “la nube” y que deben funcionar como un sistema único y de manera transparente. La misma muestra distintos usuarios, canales de acceso y dispositivos, y una extensa variedad de servicios existentes.

Una arquitectura de tal complejidad, provoca incertidumbre acerca de la demanda real de recursos exigida por los usuarios y las aplicaciones. Esto puede provocar una mala utilización de los recursos, donde puede haber picos de demanda insatisfecha en ciertos momentos y recursos ociosos en otros.

El paradigma de la nube se basa en la noción de aprovechamiento bajo demanda. Esto significa, consumir lo que se necesite y pagar por lo que se consume.

Los primeros proveedores de servicios en la nube surgen de la evolución de la infraestructura privada de los propios proveedores de servicios de Internet, o *Internet Service Providers* (ISPs) y lo hicieron como una manera de aprovechar recursos ociosos en sus datacenters, abriendo las interfaces de gestión a los usuarios y agregando lo necesario

para implementar un modelo de pago por uso. Estos proveedores incluyen a Amazon, Google, y Microsoft, entre otros.

Las infraestructuras de computación en la nube se encuentran soportadas por servidores tradicionales y virtualizados. En este sentido se puede afirmar que la virtualización es uno de los conceptos tecnológicos que sustentan la infraestructura. A este concepto de virtualización se suman otras tecnologías relacionadas como por ejemplo, *Service Oriented Architecture* (SOA), *Grid Computing* y Servicios Web. Asimismo, cabe destacar que la nube no presenta una nueva tecnología, sino un nuevo paradigma de uso que combina tecnologías existentes. En síntesis, la nube no incorpora aspectos tecnológicos nuevos sino que ofrece un nuevo modelo de servicio combinando lo existente.



Fig. 60. Elementos de Computación en la Nube

8.3. El Modelo de Nube

El modelo de nube enunciado por NIST responde a la metáfora 5-3-4 que definen 5 características esenciales, 3 modelos de servicio y 4 modelos de despliegue.

Características Esenciales

Las cinco características esenciales incluyen:

- 1) Autoservicio a demanda – un consumidor puede aprovisionar unilateral y automáticamente recursos de computación según necesite.
- 2) Amplio acceso a la red – las capacidades están accesibles en la red a través de mecanismos estándar que permiten el acceso desde plataformas heterogéneas de clientes.
- 3) Recursos mancomunados – los recursos de computación provistos son agrupados para servir a múltiples clientes usando un modelo multi-inquilino
- 4) Elasticidad rápida – las capacidades son aprovisionadas y liberadas rápida y elásticamente, y en algunos casos automáticamente.
- 5) Servicio medido – los sistemas en la nube controlan y optimizan automáticamente el uso de los recursos proporcionando alguna capacidad de medición, habitualmente basado en pago por uso o cargo por uso.

Modelos de Servicio

Los tres modelos de servicios incluyen:

- 1) **Infraestructura como Servicio** – en inglés, *Infrastructure as a Service* (IaaS) donde el consumidor aprovisiona recursos de computación (e.g. capacidad de CPU, almacenamiento, red) en los que ejecuta su software, incluidas aplicaciones y sistemas operativos. El consumidor no controla la infraestructura de nube subyacente pero sí los sistemas operativos, el almacenamiento, las aplicaciones desplegadas y a veces la red (firewalls).

Las características de IaaS son:

- Provisión de recursos bajo demanda - Los recursos son asignados a medida que los necesita quien está desplegando su entorno de TI en la nube.
- Elasticidad dinámica de los recursos - Cuando los recursos no son requeridos, se liberan de manera dinámica y transparente.
- Uso eficiente de los recursos - Para que los recursos brinden el máximo de rendimiento, se utilizan técnicas de virtualización y arquitectura *multi-tenant*.
- Modelo de pago por uso - Este modelo permite que la TI se ofrezca como servicio e implica establecer métodos y criterios para poder medir el uso de TI.
- Gestión más eficiente de los *datacenters* - Se obtienen enormes beneficios de la economía de escala, estandarización, se reducen los costos y se hace un uso más eficiente de la energía.

Las funciones de gestión típicas de IaaS son:

- Gestión dinámica de recursos (p.e. inicio/parada de máquinas virtuales, creación/eliminación de volúmenes, asignación de mayor capacidad de cómputo)
- Funciones de automatización (p.e. balance automático de carga, reglas de autoescalado horizontal, manejo de prioridades en la asignación de recursos)

- Monitorización de recursos (cpu, memoria, ancho de banda)
- Gestión de consumos y costos que permite medir los consumos de recursos de TI para poder cobrar al cliente en función de ellos.

2) Plataforma como Servicio – en inglés, Platform as a Service (PaaS) donde el consumidor despliega en la infraestructura provista por el proveedor, aplicaciones tanto propias como adquiridas, desarrolladas usando entornos de programación soportados por el proveedor.

Los tipos de ofertas de PaaS son:

- Desarrollo de extensiones SaaS. Proveedores de SaaS definen APIs para construir por sobre ellos (Ej: SalesForce, Facebook)
- Plataformas propietarias: algunos solo despliegan otros soportan todo el ciclo de vida del software: Amazon Elastic BeansTalk (despliegue de aplicaciones en Amazon EC2), Microsoft Azure (plataforma .NET), Google AppEngine (APIs propias)
- Open Platform: Ofertas que se basan en tecnologías abiertas para evitar la dependencia del vendedor (en inglés, *vendor lock-in*). Son arquitecturas extensibles en las que se puede escoger el entorno de desarrollo, el lenguaje, la base de datos, el servidor de aplicaciones. Ej: Ejemplos: CloudFoundry (VMWare), OpenShift (Red-Hat), CumuLogic, CloudBees

Las prestaciones típicas de PaaS son:

- Gestión del ciclo de vida de las aplicaciones. Esto es muy importante dado que las plataformas de desarrollo basadas en la nube son accedidas por varios usuarios a una misma instancia. Esta instancia, de la misma manera que una instancia de clase en la orientación a objetos, es creada, tiene un tiempo de vida y eventualmente es destruida o muere.
- Herramientas de desarrollo colaborativo. El modelo de trabajo en la nube es una actividad netamente colaborativa dado que varios acceden a la misma instancia y se deben aplicar las reglas de control de acceso y versiones típicas de este modelo.
- Herramientas para el desarrollo de interfaces de usuario. Como cualquier entorno de desarrollo tradicional u on-premise, se debe contar con componentes específicos para la construcción de interfaces de usuario.
- Integración con Web Services (SOAP, REST): *mashups*²¹, composición de servicios. En PaaS es mucho más frecuente y necesario construir aplicaciones como composición de componentes existentes y que puedan interoperar entre ellas aprovechando así un entorno de ejecución global.
- Integración con BD SQL y NoSQL. Al igual que los entornos de desarrollo clásico, se debe contar con manejadores que permitan establecer conexiones con las bases de datos, pero estando estas en un entorno *cloud*.

²¹ Forma de integración y reutilización. Ocurre cuando de una aplicación web es usada o llamada desde otra aplicación, con el fin de reutilizar su contenido y/o funcionalidad

- Mecanismos de extensibilidad (*plugins*) para configurar nuevos frameworks.
- Independencia del proveedor IaaS

3) **Software como Servicio** – en inglés, Software as a Service (SaaS) donde el consumidor utiliza las aplicaciones del proveedor que son ejecutadas en una infraestructura de nube. El consumidor no controla ni la infraestructura de nube subyacente, ni las capacidades de la aplicación, pero puede controlar la configuración personal de dichas aplicaciones.

La Figura 61 muestra los beneficios y desventajas de SaaS.

La adaptabilidad se plantea como una desventaja en el sentido que las aplicaciones desplegadas en la nube no se adaptan a los requisitos específicos de cada usuario, sino que brindan características universales en las que se sustenta el modelo *multi-tenant*.

Respecto de los aspectos legales y de seguridad, se refiere específicamente a que se modifica el modelo acuerdo de nivel de servicio (SLA - *Service Level Agreement*) dado que se definen nuevas responsabilidades propias de un modelo de servicio diferente.

La dependencia del vendedor se ve reflejada en términos que cual es la plataforma donde se alojan las aplicaciones en SaaS y la incertidumbre de su disponibilidad en el tiempo.

BENEFICIOS	DESVENTAJAS
<ul style="list-style-type: none"> • No hay costos de licencias • La Gestión de TI se simplifica • Las actualizaciones son automáticas y constantes • Ubicuidad (accesible desde cualquier dispositivo) • Integración a través de APIs 	<ul style="list-style-type: none"> • Dependencia de la red • Adaptabilidad • Aspectos legales y de seguridad • Dependencia del vendedor • Costos: para pymes es ventajoso.

Fig. 61. Beneficios y desventajas de SaaS

Las diferencias en ASP y SaaS

El término ASP (*Application Service Provider*) define compañías que ofrecen servicios basados en computadoras a los clientes a través de una red y usando el protocolo HTTP. En resumidas cuentas, son compañías que desarrollan sistemas basados en Web.

Si se analizan estos proveedores con los de SaaS se observan muchas similitudes dado que ambas son aplicaciones bajo demanda, acceso ubicuo, pago por uso, el cliente no gestiona ni controla la infraestructura subyacente, configuraciones personalizadas, reducción de costes de instalación y operación.

La principal diferencia es que ASP es un modelo *single-tenant*, mientras que SaaS es un modelo *multi-tenant*, pero desde el punto de vista del usuario es difícil percibir la diferencia.

En la Figura 62 se muestra un modelo ASP single-tenant y sus características, mientras que en la Figura 63, se ve el modelo SaaS *multi-tenant*

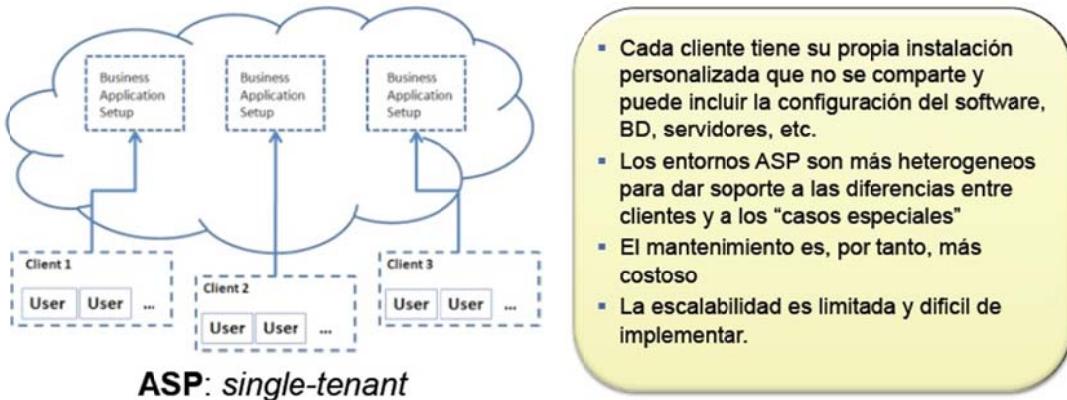


Fig. 62. ASP- Single- tenant

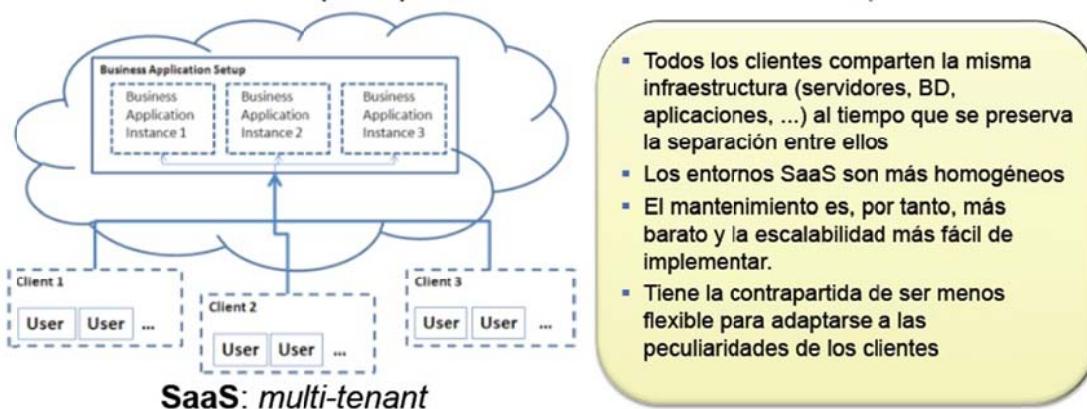


Fig. 62. SaaS multi-tenant

Modelos de Despliegue

Los cuatro modelos de despliegue incluyen:

1) Nube Pública – Ofrece una infraestructura para uso del público o empresas en general. Estas nubes tienen una interface para la gestión interna de la infraestructura virtualizada y exponen otra externa para la gestión de los recursos de los clientes.

2) Nube Privada – Es para uso exclusivo de una organización con múltiples clientes (e.g. departamentos de una empresa). Puede ser de su propiedad (on-premise), alquilada (off-premise) o una combinación de ambas. Permite una gestión flexible y ágil de los recursos de la organización. Los principales proveedores públicos de IaaS comenzaron creando nubes privadas para mejorar la gestión de sus datacenters.

3) Nube Híbrida – Es la combinación de nube privada y pública que permite gestionar los picos de carga obteniendo recursos de la nube pública. El uso de una nube pública es totalmente transparente para usuarios de la nube privada.

4) **Nube Comunitaria** - Es una infraestructura para uso de una comunidad de organizaciones que comparten intereses, donde cada componente conserva su autonomía. La relación entre componentes se hace mediante tecnologías (propietas o estandarizadas) que permitan la portabilidad de datos y aplicaciones. Puede ser propiedad de una o varias de las organizaciones, ser alquilada o una combinación de ambas.

La Figura 64 resume el modelo de nube describiendo cada uno de los modelos que lo componen de manera integrada. Como se observa en la figura, se puede desplegar SaaS sobre IaaS aunque la plataforma de desarrollo no sea "aaS".

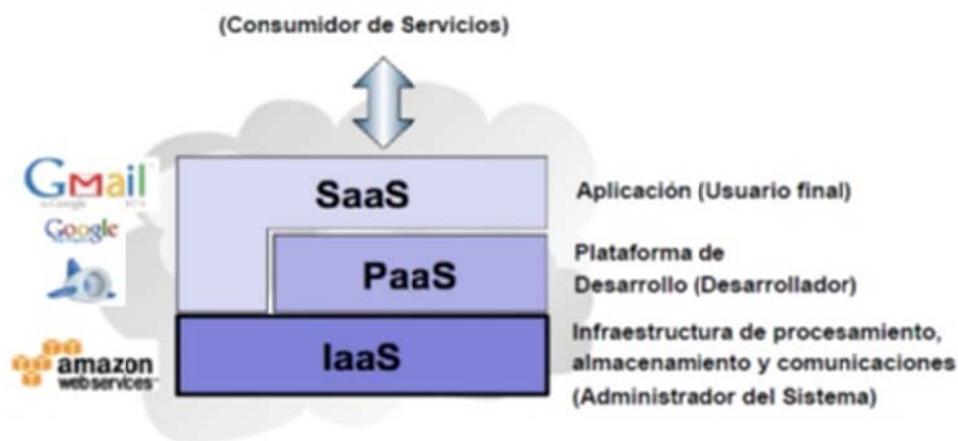


Fig. 63. Modelo de Nube. Visión Integrada

La Figura 65 muestra el modelo de nube provisto por NIST de manera integrada incluyendo también las 5 características esenciales.



Fig. 64. Modelo de Nube [NIST]

8.4- Análisis del Modelo de Nube y su Aplicación en BPM

El modelo de nube presenta ventajas y desventajas en general, que a la hora de ser analizadas a la luz de BPM deben ser consideradas a fin de proponer nuevos modelos de servicio denominados Business Process as a Service (BPaaS) [8.4]. A continuación se analiza el modelo de nube y su aplicación en BPM.

En general, el modelo de nube presenta las siguientes ventajas:

- **Reducción de costos** – el modelo de pago por uso resulta beneficioso en cuanto la inversión requerida para montar una infraestructura tecnológica, sobre todo en el ámbito de pequeñas y medianas empresas que pujan en un mercado competitivo y donde la innovación es un aspecto esencial para su crecimiento [8.4 Sección 2.1.5].
- **Mejora de la eficiencia** – es posible dedicar más recursos a la actividad propia de una organización delegando la responsabilidad de la gestión de TI, además de facilitar la escalabilidad y la elasticidad en el aprovisionamiento de recursos tecnológicos.
- **Flexibilidad** – se evita la inversión en hardware que se pierde vigencia rápidamente, además de permitir las operaciones desde cualquier lugar.
- **Seguridad** – los proveedores de servicios en la nube generalmente poseen mejores prácticas y sistemas de seguridad, así como aseguran el cumplimiento de regulaciones.
- **Continuidad del negocio** – la alta fiabilidad y la tolerancia a fallos, provista por sistemas redundantes, replicación de datos y distribución geográfica permiten a las organizaciones otorgar un servicio 24x7.

Asimismo, el modelo de nube presenta las siguientes desventajas:

- **Seguridad** – así como en un aspecto, la seguridad es una ventaja en el modelo de nube, se transforma en desventaja cuando se observa la seguridad física de los datacenters, las garantías de transmisión cifrada y el acceso a datos sensibles para la organización.
- **Dependencia de la red** – la necesidad de estar siempre conectado para poder trabajar, puede ser una desventaja, así como la dependencia de las velocidades de la red.
- **Dependencia del vendedor** – la fiabilidad del proveedor, así como los fallos masivos y la escasa interoperabilidad por la carencia de estándares, constituyen sin dudas un riesgo en la operación de las compañías.
- **Aspectos legales** – el modelo de nube aún carece de adecuada armonía entre legislaciones de distintos países, más aún cuando se piensa que la muchas de estas arquitecturas operan con modelos globalizados. Asimismo, es poco clara aún la protección al consumidor.
- **Integridad de datos** – la pérdida de control de los datos de las organizaciones pone en alto riesgo la privacidad y confidencialidad de los mismos.

Teniendo en cuenta las ventajas y desventajas descriptas anteriormente, a continuación se discute la aplicación de este modelo a BPM.

BPM basada en la nube da a los usuarios la posibilidad de usar software de una manera de “pago por uso”, en lugar de forzarlos a emprender grandes inversiones en software de BPM, hardware y mantenimiento como se da en el caso de los sistemas de licenciamiento tradicional. Por otra parte, los sistemas pueden escalarse, creciendo o decreciendo, de acuerdo a las necesidades de los usuarios, lo cual significa que los mismos no deben preocuparse acerca del sobre o bajo aprovisionamiento de recursos, gracias a la alta adaptabilidad provista en la actualidad por los prestadores de servicios en la nube.

Sin embargo, existentes puntos débiles: al desplegar un BPMS en la nube, los usuarios pueden llegar a perder control sobre los datos sensibles. Este aspecto resulta no menor al considerar que los procesos de negocio dentro de una organización gestionan información de gran importancia para la misma y sus miembros. Por otro lado, la eficiencia y efectividad de las actividades de los procesos que no son altamente computacionales pueden no incrementarse por ponerlas en la nube, sino al contrario, estas actividades pueden volverse más costosas. Por ejemplo, una actividad que no es intensamente computacional podría necesitar procesar cierta cantidad de datos. La transferencia de los datos a la nube puede tomar más tiempo que transmitirlos a una versión embebida instalada localmente (*on-premise*). Incluso la misma puede resultar aún mayor que la necesidad real de procesamiento. Además, los costos de la actividad pueden incrementarse debido a que la transferencia de datos es uno de los elementos de facturación en un sistema de computación en la nube, debido a la alta disponibilidad de la conexión [8.4].

8.5. Aplicaciones Distribuidas y Cloud: desafíos y métodos

Con el rápido desarrollo de TI en el contexto del lanzamiento y ejecución de arquitecturas basadas en la nube, las compañías se enfrentan con nuevos problemas. En particular, las características distribuidas y colaborativas de las aplicaciones se torna evidente.

El paradigma de computación en la nube puede considerarse un facilitador de la combinación mejorada de arquitecturas orientadas a servicios, donde la construcción de aplicativos como composición de componentes promete ser una realidad. Pero este potencial depende de las condiciones de los distintos *frameworks*, las cuales pueden ser apreciadas desde un aspecto técnico tanto como económico. A continuación, se discuten estos dos aspectos; uno técnico y otro económico.

Vista Técnica

Desde un punto de vista técnico se pueden identificar tres dimensiones para el diseño, implementación y operación exitosos de desarrollos de software en un ambiente de nube: 1) programación, 2) integración y 3) seguridad.

1) Programación – Los sistemas complejos y distribuidos son altamente realizables en el campo de TI actual. En conexión con el objetivo de alcanzar una mayor usabilidad y flexibilidad, esta complejidad representa nuevos requerimientos para la Ingeniería de Software. Para resolver este problema es necesaria la adopción de nuevos lenguajes, recayendo sobre nuevos conceptos y técnicas innovadoras.

2) Integración – La integración puede dividirse en integración de datos e integración de funciones. A la luz de los desafíos involucrados, el tópico de la integración juega un rol fundamental en distintos escenarios. Por ejemplo, un *workflow* basado en la nube puede controlar actividades variables distribuidas más allá de las fronteras de las compañías.

3) Seguridad – La seguridad puede dividirse en tres categorías: seguridad funcional, de la información y de los datos. Todas estas categorías tienen una relevancia significativa para las aplicaciones distribuidas y desplegadas en la nube. La seguridad funcional específica como el estado actual se corresponde con el estado deseado de funcionalidad. La seguridad de información se enfoca en los cambios o extracciones de información no autorizados. La seguridad de los datos se encarga de los datos utilizados por los aplicativos.

Vista Económica

Se pueden mencionar dos dimensiones desde el punto de vista económico:

1) Disponibilidad – Los servicios que son provistos por una infraestructura de nube pueden ser accedidos en cualquier momento. Basados en un alto nivel de abstracción, la personalización e instalación se vuelven significativamente más fáciles. En adición a esta simplificación, el usuario final es capaz de trabajar con el servicio en forma inmediata.

2) Riesgo de inversión – En el contexto de los distintos modelos de facturación variable como el de pago por transacción, el sistema orientado a la nube resulta más accesible que un sistema de licenciamiento tradicional.

8.6. Conclusiones del Capítulo

La idea subyacente en *cloud computing* es que aporta un nivel superior de eficiencia para distribuir y desplegar recursos de IT bajo demanda. La incorporación de conceptos como virtualización, despliegue bajo demanda, distribución de recursos por Internet cambian el enfoque no

sólo para el desarrollo de aplicaciones, sino también para su despliegue, mantenimiento y fundamentalmente su interoperabilidad.

En este sentido, mientras la arquitectura de tres capas de las aplicaciones basadas en Web revolucionó el desarrollo del software en su momento, la virtualización en “nubes” ha dado origen a nuevos niveles como lo son: las aplicaciones, los servicios y la infraestructura. Los conceptos de SaaS (*Software as a Service*), PaaS (*Platform as a Service*) y IaaS (*Infraestructure as a Service*), hacen de la red un recurso escalable y constituyen un medio para desarrollo y despliegue de aplicaciones globales.

Bibliografía

- Arsanjani, A et al. (2006). From Grid Middleware to a Grid Operating System. *Grid and Cooperative Computing, GCC*. (9,16). Recuperado de <https://arxiv.org/ftp/cs/papers/0608/0608046.pdf>.
- Arsanjani,A. (2004) Ph.D. Service-oriented modeling and architecture. Recuperado de <http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/>.
- Bair, J. (1989) Supporting cooperative work with computers: addressing meeting mania. *Digest of Papers. COMPCON Spring 89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage*. (208,217). IEEE Compuing. Soc. Press. doi: 10.1109/CMPCON.1989.301929. Recuperado de <http://ieeexplore.ieee.org/abstract/document/301929/>
- Bazán, P. (2010) Un modelo de integrabilidad con SOA y BPM. Tesis de Maestría en Redes de Datos. Facultad de Informática. Universidad Nacional de La Plata. Recuperado de <http://sedici.unlp.edu.ar/handle/10915/4181>.
- Bazán, P. (2015]. Implementación de procesos de negocio a través de servicios aplicando metamodelos, software distribuido y aspectos sociales. Tesis doctoral. Recuperado de <http://sedici.unlp.edu.ar/handle/10915/45100>.
- Bochenksi, B. (1994). Implementing Production-Quality Client/Server Systems, *Editorial John Wiley & Sons*, Estados Unidos. ISBN:0471585319
- Colouris, G et al. (2000) Distributed Systems Concepts and Design 3e. *Addison-Wesley*. ISBN-13: 978-0132143011
- Dourish, P et al. (1992) . Portholes: supporting awareness in a distributed work group. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Penny Bauersfeld, John Bennett, and Gene Lynch (Eds.). ACM, New York, NY, USA, 541-547. DOI=<http://dx.doi.org/10.1145/142750.142982>
- Edwards, J. et al. (1998). Tier Client/Server at Work. *John Wiley & Sons, Inc.* ISBN 0-471-18443-8.
- Ellis, C et al. (1991). Groupware: some issues and experiences. *Communications of the ACM*, 34(1), 39–58. doi:10.1145/99977.99987.
- Erl, T. (2007) SOA Principles of Service Design. *Prentice Hall*. ISBN-13: 9780132344821. (25, 119).
- Erl, T et al. (2013] Cloud computing. Concepts, Technology and Architectures. *Prentice Hall*. ISBN-13: 978-0-13-338752-0

- Foster, I et al. (2008) Cloud Computing and Grid Computing 360-Degree Compared. *Grid Computing Environments Workshop*. Recuperado de <http://ieeexplore.ieee.org/abstract/document/4738445/>.
- Johnson-Lenz, P. (1998). Post-mechanistic groupware primitives: rhythms, boundaries and containers. *International Journal of ManMachine Studies*, 34(3), (395,417). Recuperado de <http://www.sciencedirect.com/science/article/pii/0020737391900275>.
- Johnson-Lenz, P. (1998) Groupware: Coining and Defining it. *ACM SIGGROUP Bulletin*, 19(2), 34. doi:10.1145/290575.290585.
- Kerrish, M. (2010) The Linux Programming Interface. *A linux and Unix System programming Handbook*. No starch press. ISBN 10: 1-59327-220-0. ISBN 13: 978-1-59327220-3.
- Minor, M et al. (2012) Adaptive Workflow Management in the Cloud – Towards a Novel Platform as a Service. *Business Information Systems II*. University of Trier, Germany. Recuperado de http://www.wi2.uni-trier.de/shared/publications/2011_POCBR_MinorEtal.pdf.
- Orfali, R et al. (1994) Essential Client/Server Survival Guide. *John Wiley & Sons*. ISBN: 978-0-471-31615-2
- Orlikowski, W.J. et al. (1997) An Improvisational Model of Change Management: The Case of Groupware Technologies. *Sloan Management Review*. 382), (11–21).
- Pardo, X. (2013) “Curso Cloud Computing” dictado en el marco de la carrera del Doctorado en Ciencias Informáticas de la Facultad de Informática de la UNLP.
- Popek, G et al. (1994) Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*. Volume 17 Issue 7. (.412-421). Recuperado de <http://dl.acm.org/citation.cfm?id=361073>
- Portnoy, M. (2012). Virtualization Essentials. *Sybex*. ISBN-13: 978-1118176719 ISBN-10: 1118176715.
- Price. (1995) Price Waterhouse Interamerica Consulting Group, Estándares de Sistemas Abiertos de Software, Informe No. 18. *Club de Investigación Tecnológica*. Costa Rica.
- Pulier, E. et al (2006) Understanding Enterprise SOA. *Manning Publications Co*. ISBN 1-932394-59-1. 2006. (1, 73).
- Silberschatz, G. (2009) Operating System Concepts. *Addison Wesley*. 8va Edición. ISBN-10: 1118112733
- Stallings, W. (2001) Sistemas Operativos 4ta. Edición, ISBN: 84-205-3177-4.
- Sun. (2002). Web Component Development With Java Technology. *Sun Microsystem*.
- Sun. (2002) Developing J2EE Compliant Applications. *Sun Microsystem*
- Tanenbaum, A et al. (2007) Distributed Systems: Principles and Paradigms, 2e. *Prentice Hall*. ISBN-13: 978-1530281756.
- Weske, M. (2008) Business Process Management: Concepts, Languages, Architectures. *Springer*. (3, 67). ISBN 978-3-540-73521-2.

Los autores

Banchoff, Tzancoff Matias

Es Licenciado en Informática graduado de la Facultad de Informática UNLP. Se desempeña como Jefe de Trabajos Prácticos en asignaturas vinculadas a redes, sistemas operativos y arquitectura de computadoras, en la Facultad de Informática UNLP y en la Universidad Nacional Arturo Jauretche Florencio Varela. Desarrolla su actividad profesional como administrador de sistemas en CeSPI desde 2006.

Bazán, Patricia

Es Doctora en Ciencias Informáticas graduada en la Facultad de Informática UNLP en 2015. Es Magister en Redes de Datos graduada en la misma unidad académica en 2010. Se desempeña como Profesor Titular con Dedicación Exclusiva en la Facultad de Informática UNLP en la asignatura Desarrollo de Software en Sistemas Distribuidos de la carrera de Licenciatura en Sistemas. Las tareas de investigación las desarrolla en el Laboratorio de Investigación y Formación en Informática Avanzada (LINTI) desde 1995, dirigiendo becarios y tesistas. Ha sido Consultor en proyectos financiados por el Banco Mundial y el BID.

del Rio, Nicolás

Es Analista Computación graduado de la Facultad de Informática UNLP y Magister en Redes de Datos graduado en 2016 en la misma unidad académica. Se desempeña como Profesor Adjunto en la Facultad de Informática de la UNLP en las asignaturas Introducción a los Sistemas Operativos y Sistemas Operativos de las carreras de Licenciatura en Sistemas y Licenciatura en Informática. Es docente de postgrado para la Maestría en Redes de Datos en la asignatura Sistemas Distribuidos. Desarrolla su actividad profesional como Administrador de la Infraestructura de Red y Data Center en la Agencia de Recaudación de la Provincia de Buenos Aires

Fernandez, Alejandro

Licenciado en Informática de la Universidad Nacional de La Plata, y Dr. en Ciencias (Dr. rer. nat.) de la FernUniversität Hagen, de Alemania. Es profesor adjunto en la Facultad de Informática de la Universidad Nacional de La Plata. Es investigador asistente de la Comisión de Investigaciones Científicas de la Provincia de Buenos, en el Centro Asociado LIFIA. Su foco actual de investigación son las tecnologías para la gestión del conocimiento en red.

Molinari, Lía

Es Magister en Redes de Datos, Facultad de Informática, UNLP. Es Profesora Titular de las asignaturas Introducción a los Sistemas Operativos y Sistemas Operativos de las carreras de la Facultad de Informática de la UNLP. También es Profesora en el Master de Redes de Datos de la Facultad de Informática de la Universidad Nacional de La Plata y Docente Investigador en temas de Seguridad Informática y de los Sistemas de Información y Auditoría de TI. Ha obtenido certificaciones CISA (*Certified Information Systems Auditor*), ISACA (*Information Systems Audit and Control Association*).

Perez, Juan Pablo

Es Licenciado en Informática y Analista en Computación, graduado en la Facultad de Informática UNLP. Se desempeña como Profesor Adjunto en la asignatura Sistemas Operativos de las carreras de Licenciatura en Informática y Sistemas, de la Facultad de Informática UNLP. Su labor profesional la desarrolla como Director de la Unidad de Sistemas – Presidencia – Universidad Nacional de La Plata, desde 2014.

Aplicaciones, servicios y procesos distribuidos : una visión para la construcción del software / Patricia Bazán ... [et al.] ; coordinación general de Patricia Bazán. - 1a ed . - La Plata : Universidad Nacional de La Plata, 2017.

Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-950-34-1520-7

1. Diseño de Software. I. Bazán, Patricia II. Bazán, Patricia, coord.
CDD 005.12

Diseño de tapa: Dirección de Comunicación Visual de la UNLP

Universidad Nacional de La Plata – Editorial de la Universidad de La Plata
47 N.º 380 / La Plata B1900AJP / Buenos Aires, Argentina
+54 221 427 3992 / 427 4898
edulp.editorial@gmail.com
www.editorial.unlp.edu.ar

Edulp integra la Red de Editoriales Universitarias Nacionales (REUN)

Primera edición, 2017
ISBN 978-950-34-1520-7
© 2017 - Edulp

e
exactas


Editorial
de la Universidad
de La Plata



UNIVERSIDAD
NACIONAL
DE LA PLATA