

# Arquitecturas, paradigmas y aplicaciones de los sistemas distribuidos

Joan Manuel Marquès i Puig  
Xavier Vilajosana i Guillén  
Pedro A. García López

P07/M2106/02839



Universitat Oberta  
de Catalunya

[www.uoc.edu](http://www.uoc.edu)



# Índice

<b>Introducción</b> .....	5
<b>Objetivos</b> .....	8
<b>1. Conceptos previos de sistemas distribuidos</b> .....	9
<b>2. Estilos arquitectónicos de sistemas distribuidos</b> .....	13
<b>3. Tipos de arquitecturas de los sistemas distribuidos</b> .....	14
3.1. Arquitecturas centralizadas .....	14
3.1.1. Cliente-servidor .....	14
3.2. Arquitecturas descentralizadas .....	19
3.2.1. No estructuradas .....	21
3.2.2. Estructuradas .....	25
3.3. Paradigma de publicación-suscripción .....	34
3.3.1. Sistemas distribuidos basados en eventos .....	37
3.4. Código móvil .....	39
<b>4. Aplicaciones de los sistemas distribuidos</b> .....	43
4.1. Sistemas computacionales distribuidos .....	43
4.1.1. <i>Clusters</i> .....	43
4.1.2. <i>Grid</i> .....	44
4.1.3. Sistemas computacionales usando recursos de los extremos de Internet .....	49
4.2. Sistemas de información distribuidos .....	50
4.3. Sistemas distribuidos omnipresentes .....	51
<b>Resumen</b> .....	54
<b>Actividades</b> .....	57
<b>Solucionari</b> .....	58
<b>Glosario</b> .....	58
<b>Bibliografía</b> .....	59



## Introducción

Internet es un recurso compartido formado por millones de ordenadores repartidos por todo el mundo que están conectados por medio de redes. El número de aplicaciones y usuarios conectados a este megasistema informático no para de crecer. La introducción de la web en 1994 comportó un cambio radical a Internet. Ésta empezó un proceso de generalización frenético que todavía dura hoy. Sin embargo, es durante el año 2000 que mucha gente se da cuenta de que Internet es algo más que correo electrónico, web y servicios comerciales por web. Con la explosión del fenómeno Napster, una nueva visión de Internet se hace realidad: la Red es un entorno idóneo para el intercambio y la compartición. Los usuarios conectados dejan de ser simples elementos pasivos que se conectan y extraen contenidos de Internet, y se convierten en suministradores de contenidos y de recursos. Y lo que es más importante, esta compartición se hace a partir de formar comunidades y colaborar directamente utilizando los ordenadores de los usuarios. Lo que el Napster proporcionó al público de Internet no fue sólo una manera fácil de obtener cualquier canción en formato MP3 sin tener que pagar. Sobre todo, significó un cambio de mentalidad respecto de qué es, cómo se organiza y para qué puede servir Internet.

### Napster

El Napster fue la primera aplicación popular de intercambio de ficheros que seguía el modelo de igual a igual. Permitía el intercambio de ficheros en formato MP3 entre usuarios finales de Internet. Seguía un modelo de igual a igual no puro: un índice centralizado y las canciones repartidas en los ordenadores de los usuarios que las aportaban. Una vez localizada la ubicación de la canción preguntando al índice, la bajada de canciones se hacía directamente entre los ordenadores de los usuarios.

Centrando esta reflexión en torno a los objetivos de este módulo es importante darnos cuenta que tanto el Napster como sus sucesores: Gnutella, KaZaA, eMule, BitTorrent, etc. apuestan por la descentralización –tanto en la aportación de contenidos como en la administración del sistema–, la escala –son sistemas formados por millones de usuarios repartidos por toda Internet–, la autonomía –los recursos para que los sistemas funcionen (o como mínimo la mayor parte de éstos) los aportan los mismos usuarios del sistema–, y la autoorganización –el sistema se adapta de manera automática a los cambios que suceden en el sistema (conexiones, desconexiones, fallos, movilidad, etc.). El modelo de arquitectura que siguen lo denominamos de igual a igual o entre iguales (*peer-to-peer* en inglés), y se diferencia muy claramente de la arquitectura web, la gran dominadora de Internet durante la década de los 90, que es una arquitectura centralizada (en la arquitectura web los nodos servidores aportan contenidos y recursos; y los nodos clientes acceden a los mismos). Antes de empezar a concretar cada una de las arquitecturas que trataremos en este módulo, haremos un breve repaso de la historia de Internet, y esto nos ayudará a entender las diferentes arquitecturas de sistemas distribuidos que han dominado en cada momento.

Internet se creó a finales de los sesenta con el objetivo de compartir recursos de procesamiento que estaban esparcidos por todo Estados Unidos y que hasta aquel momento estaban aislados los unos de los otros. El reto que

tenían delante era conseguir integrar las redes existentes en aquel momento, y también las posibles tecnologías futuras, en una única arquitectura de red que permitiera a todos los nodos tener un papel al mismo nivel (todos los nodos iguales entre sí). Cualquier ordenador podía enviar paquetes a cualquier otro. El uso principal que se daba a la red era que diferentes investigadores colaborasen. Una muestra de este espíritu colaborativo es que hasta finales de los ochenta no empiezan a aparecer los primeros cortafuegos. A pesar de que desde el principio había aplicaciones como el FTP y Telnet, que por naturaleza son cliente-servidor, dado que todos los nodos implementaban tanto la parte cliente como la parte servidora, el uso que se hacía de las mismas era totalmente simétrico. Cualquier nodo podía hacer Telnet o FTP a cualquier otro nodo. Otras aplicaciones de esta primera época son los mensajes Usenet (*Usenet news*), que implementaban un control totalmente descentralizado de los contenidos de los grupos; y el correo SMTP, en el que cada servidor de correo se conecta directamente con cualquier otro para hacer llegar el correo dirigido a este servidor.

**Usenet**

Usenet es un sistema que, sin utilizar un control centralizado, copia ficheros entre ordenadores.

A finales de los ochenta y principios de los noventa, las arquitecturas cliente-servidor se empiezan a imponer como la manera más rápida y económica para dar soporte a grandes cantidades de usuarios no técnicos. Una de las ventajas de estas arquitecturas es que también permitían que se utilizaran PC, que eran ordenadores menos potentes y más económicos.

En el año 1994 se produce la explosión de Internet, que la cambió radicalmente. Millones de personas se incorporan a este medio para enviar correo electrónico, leer páginas web y comprar. Internet deja de ser un reducto de científicos para convertirse en un fenómeno cultural de masas. Una de las principales consecuencias es que cambia radicalmente su arquitectura: el uso de cortafuegos se generaliza; muchos de los usuarios, cada vez que se conectan, utilizan una dirección IP distinta; muchos de los usuarios se conectan con módems o utilizando conexiones asimétricas (como la ADSL o el cable). La arquitectura que se impone en esta nueva etapa es la arquitectura web, basada en el modelo cliente-servidor. Un tercer concepto también muy importante es que Internet se convierte en una oportunidad de negocio, hecho que acelera todavía más este crecimiento. Este carácter comercial de Internet se construye en torno a la centralización que ofrece la arquitectura cliente-servidor de la web.

A finales de los noventa ya hay mucha gente conectada a Internet. En este momento se empieza a extender el uso de aplicaciones que necesitan una arquitectura diferente de la que ofrece el modelo cliente-servidor. El primer ejemplo claro es la mensajería instantánea. Sin embargo, durante el año 2000 la gente empieza a tomar conciencia de que hay algo que está cambiando. Aparece Napster y, en poco tiempo, se populariza como manera de intercambiar directamente ficheros MP3 entre los usuarios finales de Internet. Napster no es sino el ejemplo más popular de una nueva manera descentralizada y autoorganizada de hacer sistemas distribuidos, que se escalan perfectamente a millones de

nodos, que intercambian, comparten o almacenan información y usando sólo los recursos de los propios usuarios.

Otro modelo que trataremos en este módulo son los *grids*. Se empieza a hablar de *grids* a partir de la segunda mitad de los noventa. Como en el caso de igual a igual, el *grid* es una consecuencia del aumento sustancial en el rendimiento de los ordenadores personales y de las redes que ha habido en los últimos diez o quince años. El reto de los *grids* es crear una infraestructura computacional aprovechando los recursos (ordenadores, almacenes de datos, instrumentos científicos, bases de datos, etc.) que pueden aportar las diferentes instituciones que participan en el *grid*. De esta manera, con la combinación de todos estos recursos se pueden resolver problemas utilizando más recursos de los que se dispone individualmente. De la misma manera, en el mundo comercial y social también aparece la oportunidad de utilizar el concepto de *grid* de forma que la capacidad de computación, almacenamiento y los servicios (aplicaciones y su licencia de uso) no se tenga que comprar sino que se pueda obtener cuando se necesita de un proveedor externo, y se pueda pagar por uso en lugar de por propiedad. Eso hace que computación, almacenamiento y servicios se puedan adaptar a las necesidades.

Acabamos esta introducción incidiendo en el hecho de que las aplicaciones o sistemas informáticos cada vez más están formados por componentes que se encuentran por toda Internet. Es importante conocer diferentes maneras de organizar la interacción entre estos componentes a fin de que el sistema se comporte de la manera que más nos interese. En este módulo veremos los diferentes estilos arquitectónicos de los sistemas distribuidos. Las arquitecturas nos permitirán conocer las formas de organizar los componentes que forman una aplicación distribuida. Las clasificaciones siempre son complicadas de hacer, sobre todo teniendo en cuenta que muchas veces los sistemas existentes no siguen al cien por cien ninguno de los modelos genéricos, sea porque implementan más de un modelo, sea porque implementan un modelo que es híbrido. Conscientes de ello, creemos que la clasificación que aquí presentamos puede ser útil para ayudar al diseñador de una aplicación distribuida a la hora de decidir cómo tiene que ser la arquitectura de la aplicación.

## Objetivos

Los objetivos que tiene que conseguir el estudiante con este módulo didáctico son los siguientes:

- 1.** Conocer las características de un sistema distribuido.
- 2.** Conocer diferentes formas de organizar los componentes lógicos de los sistemas distribuidos.
- 3.** Conocer con detalle las arquitecturas más usadas actualmente.
- 4.** Conocer las aplicaciones principales de los sistemas distribuidos.

## 1. Conceptos previos de sistemas distribuidos

Hay muchas definiciones de sistemas distribuidos. A continuación, presentaremos una que creemos que encaja con el enfoque que se ha dado a este módulo.

Un sistema distribuido es una colección de ordenadores **autónomos** enlazados por una **red** de ordenadores y soportados por un **software** que hace que la colección actúe como un servicio **integrado**.

Una arquitectura de *software* determina cómo se identifican y se asignan los componentes del sistema; cómo interactúan los componentes para formar el sistema; la cantidad y la granularidad de la comunicación que se necesita para la interacción; y los protocolos de la interfaz usada por la comunicación.

Cuando se construye un sistema distribuido, no se persigue la creación de una topología de interacciones determinada ni el uso de ningún tipo de componente determinado. Lo que se quiere es construir un sistema que satisfaga las necesidades de la aplicación. En este módulo presentaremos una serie de propiedades o requerimientos que hay que tener en cuenta a la hora de diseñar aplicaciones distribuidas.

Antes de comentar estos aspectos, pero, mencionaremos las características de la escala Internet. 

La primera característica es la **gran cantidad tanto de ordenadores como de usuarios** que hay en Internet. Relacionado con esto, se encuentra la **dispersión geográfica** de éstos. La dispersión geográfica afecta a la manera en la que los componentes del grupo perciben las acciones que pasan en el sistema.

Por ejemplo, los miembros del sistema pueden percibir dos acciones que pasan en dos extremos opuestos del mundo en diferente orden, según la proximidad de cada componente al componente generador de la acción. Según el sistema, necesitaremos o no mecanismos para abordar estas situaciones.

En el módulo 2 se verá con más detalle esta problemática y su solución. 

Una tercera característica es la **autonomía** de los diferentes ordenadores que forman el sistema informático. Es muy habitual que diferentes partes de un sistema distribuido estén administradas por distintos administradores. Relacionada con esta característica está la **seguridad**. Cada organización tiene unas políticas de seguridad diferentes, como cortafuegos. Es necesario que los sistemas distribuidos que diseñamos se puedan ejecutar teniendo en cuenta estas restricciones impuestas por los diferentes administradores.

Una cuarta característica es la **calidad de servicio**. En un sistema distribuido a escala Internet, éste es un aspecto fundamental que vendrá muy condicionado por la latencia de la red, los cortes y otros fenómenos que la puedan afectar.

Finalmente, hay los aspectos relacionados con la **movilidad**: dispositivos que se conectan y desconectan; acceso desde diferentes ubicaciones; y calidad de acceso menor (ancho de banda, fiabilidad, etc.). Este último aspecto, sin embargo, mejorará mucho a medida que estos servicios se generalicen.

Una vez vistas las características de la escala Internet, veremos algunos aspectos que es necesario tener en cuenta a la hora de diseñar un sistema distribuido: 

- **Heterogeneidad.** El sistema distribuido puede estar formado por una variedad de diferentes redes, sistemas operativos, lenguajes de programación o *hardware* del ordenador o del dispositivo. Es necesario que, a pesar de estas diferencias, los componentes puedan interactuar entre sí.
- **Apertura.** Es deseable que el sistema se pueda ampliar. Por ampliar entendemos que se puedan añadir nuevos recursos y servicios compartidos y que éstos se puedan poner a disposición de los diferentes componentes que forman el sistema o aplicación.
- **Seguridad.** Los sistemas distribuidos de gran escala contienen información valiosa para sus usuarios. Por lo tanto, su seguridad es muy importante. La seguridad de la información tiene tres componentes: confidencialidad (protección ante accesos para individuos no autorizados); integridad (protección contra alteración o corrupción); y disponibilidad (protección contra interferencias con la intención de acceder al recurso). Un aspecto crítico relacionado con la seguridad es saber la identidad de los usuarios u otros agentes que intervengan en el sistema distribuido. Se pueden usar técnicas de cifrado para proporcionar una protección adecuada de los recursos compartidos y mantener secreta, mientras se transmite por la red, la información que se crea oportuna.
- **Escalabilidad.** Se dice que un sistema es escalable si se mantiene efectivo cuando hay un incremento significativo en el número de recursos y el número de usuarios.

Internet es un ejemplo de sistema distribuido que ha crecido mucho, tanto en el número de ordenadores como de servicios.

Fecha	Ordenadores	Servidores web
Diciembre 1979	188	0
Julio 1989	139.000	0
Julio 1993	1.776.000	130
Julio 1995	6.642.000	23.500
Julio 1997	19.540.000	1.203.096
Julio 1999	56.218.000	6.598.697

Para garantizar la escalabilidad es necesario:

a) controlar el coste de añadir nuevos recursos físicos (por ejemplo, si en un futuro hay el doble de usuarios, que sea suficiente con el doble de servidores);

b) controlar la pérdida de rendimiento (que el hecho de añadir nuevos recursos no haga que el rendimiento del sistema se resienta);

c) evitar que los recursos de *software* se agoten (por ejemplo, cuando se crearon las direcciones Internet se hicieron de 32 bits. Con el ritmo actual de demanda de direcciones Internet, se agotarán en poco tiempo. Como solución, se propone usar una nueva versión del protocolo que utiliza direcciones de 128 bits);

d) evitar puntos que puedan ser cuellos de botella en el momento en el que se ejecuten (de esta manera, si hay un crecimiento en la demanda, este punto no afecta al rendimiento del sistema).

- **Comportamiento ante fallos del sistema.** Un sistema puede fallar. En el caso de los sistemas distribuidos, puede ser que falle un componente o parte de un componente, pero es posible que el resto del sistema continúe funcionando. Las técnicas para gestionar los fallos son las siguientes:

a) detectar si hay algún fallo;

b) ser capaz de funcionar como si no pasara nada (por ejemplo, algún componente del sistema va registrando qué pasa y cuando se resuelve el fallo, se encarga de comunicar al componente que fallaba todo lo que no se le ha podido comunicar);

c) tolerar los fallos (no es necesario que se enmascaren todos los fallos. En muchos casos es bueno que un componente sepa que un servicio no funciona. De esta manera, en vez de quedarse esperando que el servicio funcione, puede decidir utilizar otro servicio o hacer alguna otra cosa);

d) recuperarse de algún fallo (hay que mantener el estado de manera que se pueda recuperar después de un fallo);

e) la última técnica que comentaremos es la utilización de la redundancia para minimizar el efecto de los fallos del sistema.

- **Concurrencia.** Los distintos componentes de un sistema distribuido pueden demandar acceder a un recurso simultáneamente. Es necesario que el sistema esté diseñado para permitirlo.

### **Ejemplo**

Se puede dar el caso en el que dos componentes lean un dato compartido, lo modifiquen localmente y vuelvan a escribir el resultado de la modificación en el espacio compartido.

En muchos casos nos interesará que primero lea un componente, modifique localmente el dato y después lo escriba. El segundo componente tiene que esperar a que el primero haya acabado su operación; si no es así, el resultado de su cálculo puede ser erróneo, ya que no utiliza el dato que ha modificado el primer componente, sino el dato inicial.

- **Transparencia.** La transparencia procura que ciertos aspectos del sistema sean invisibles a las aplicaciones (actúan pero las aplicaciones no los ven, no les afecta, por eso se dice que son transparentes). Se puede aplicar a diferentes aspectos:
  - **Transparencia de ubicación:** acceso a un recurso sin saber su ubicación.
  - **Transparencia de acceso:** permite que recursos locales y remotos se accedan usando las mismas operaciones.
  - **Transparencia de concurrencia:** permite que diferentes usuarios compartan recursos de manera concurrente.
  - **Transparencia de fallos:** que el sistema continúe funcionando aunque haya algún fallo de un componente o recurso.
  - **Transparencia de movilidad:** que los recursos y los usuarios se puedan mover por el sistema sin que afecte a su funcionamiento.
  - **Transparencia de replicación:** que haya más de una instancia de un recurso.
  - **Transparencia de rendimiento:** permite que a medida que la carga varía el sistema se reconfigure para mejorar el rendimiento.
  - **Transparencia de escalabilidad:** permite al sistema y las aplicaciones aumentar la escala sin cambios en la estructura del sistema o los algoritmos de las aplicaciones.

A la hora de construir un sistema distribuido, es importante encontrar un equilibrio entre un nivel alto de transparencia y el rendimiento del sistema. Por ejemplo, la mayoría de aplicaciones en Internet intentan repetidamente contactar a un servidor antes de abandonar. Este intento de esconder el fallo transitorio de un servidor antes de intentar otro puede moderar el funcionamiento global del sistema. La conclusión que hay que sacar es que la transparencia es buena cuando se diseña e implementa un sistema distribuido, pero hay que considerarla conjuntamente con otras características como el rendimiento.

## 2. Estilos arquitectónicos de sistemas distribuidos

Los estilos arquitectónicos describen la organización lógica de los componentes de un sistema distribuido.

Un **componente lógico** es una unidad modular, sustituible, que define las interfaces requeridas y ofrecidas en otros componentes lógicos. Los sistemas distribuidos se forman a base de componentes lógicos interrelacionados entre sí. Los **conectores** son mecanismos que hacen de mediadores de la comunicación, coordinación o cooperación entre componentes. A partir de interrelacionar componentes lógicos y conectores se pueden construir diferentes estilos de sistemas distribuidos:

1) **Arquitecturas por capas.** Los componentes lógicos se organizan por capas. Un componente lógico de la capa  $L_i$  sólo puede invocar operaciones en la capa de bajo  $L_{i+1}$ . En este estilo, las invocaciones van de las capas superiores a las capas inferiores, mientras que los resultados van de las capas inferiores hacia las capas superiores.

2) **Arquitecturas orientadas a componentes.** Cada objeto representa un componente lógico y los componentes lógicos están interconectados mediante invocaciones remotas (RPC). Este estilo y el estilo de la arquitectura por capas son los más usados en sistemas distribuidos de gran escala.

3) **Arquitectura orientada a los datos.** En los sistemas distribuidos la localidad de los datos es un factor que afecta al rendimiento del sistema. Algunos sistemas de ficheros distribuidos, o las aplicaciones web distribuidas, deben organizar sus componentes lógicos en función de la localidad de los datos. Esta arquitectura también es utilizada cuando se quieren gestionar datos que están distribuidos de una cierta manera, por ejemplo, cuando el volumen de datos es muy grande, puede ser interesante que el componente lógico que procesa los datos esté allí donde estén los datos para no tener que enviarlas mediante la red.

4) **Arquitecturas orientadas a eventos.** Los componentes básicamente se comunican mediante eventos. Este estilo tiene como uno de sus paradigmas más representativos lo que se conoce como paradigma de publicación-suscripción. En estos sistemas los componentes publican eventos que sólo son recibidos por aquellos componentes que están suscritos a ellos. La principal ventaja de estos sistemas es el desacoplamiento de los componentes lógicos.

### 3. Tipos de arquitecturas de los sistemas distribuidos

Una clasificación muy extendida de los sistemas distribuidos es aquella que los clasifica en función de la ubicación, jerarquía o relación entre los componentes lógicos.

#### 3.1. Arquitecturas centralizadas

En las arquitecturas centralizadas la interrelación entre componentes sigue un patrón muy característico en el que hay una jerarquía definida de manera que ciertos componentes requieren información o servicios que ofrecen otros componentes lógicos.

##### 3.1.1. Cliente-servidor\*

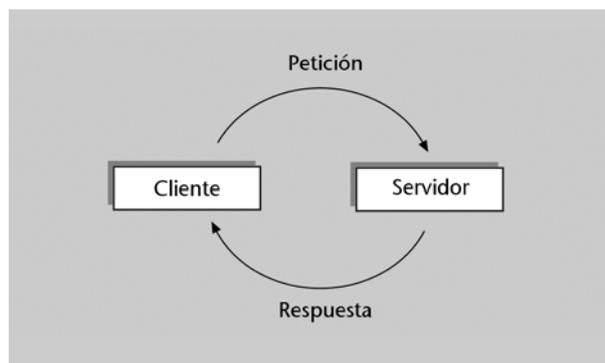
\* En inglés: *client/server*.

Esta arquitectura es la que estamos más acostumbrados a utilizar en entornos distribuidos. Históricamente ha sido la más usada, y todavía lo es hoy en día. La web es un ejemplo de arquitectura cliente-servidor.

En el modelo cliente-servidor hay dos tipos de componentes:

- **Cientes:** hacen peticiones de servicio. Normalmente, los clientes inician la comunicación con el servidor.
- **Servidores:** proveen servicios. Normalmente, los servidores esperan recibir peticiones. Una vez han recibido una petición, la resuelven y devuelven el resultado al cliente.

Figura 1



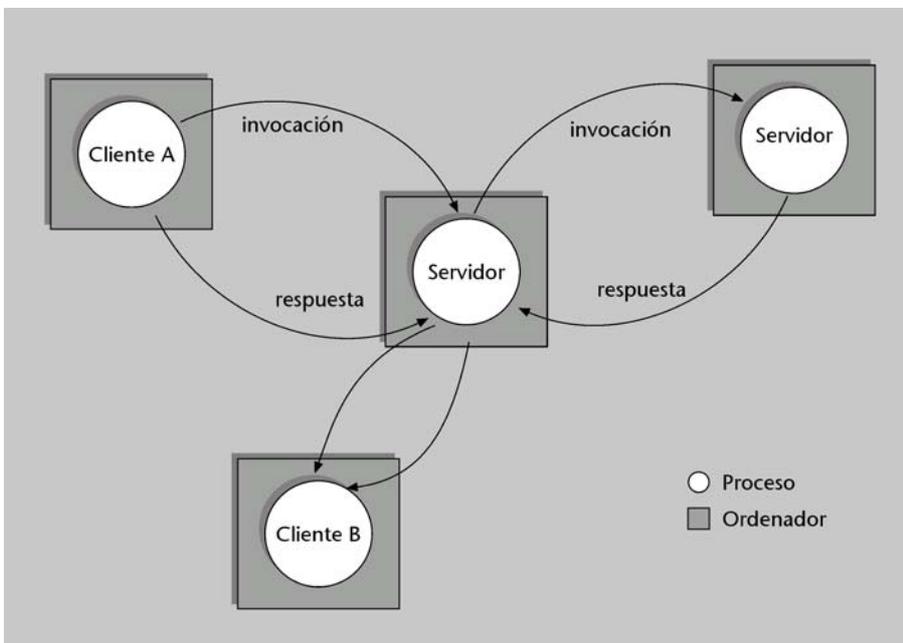
El modelo cliente servidor

Un servidor también puede ser cliente de otros servidores, tal y como se ve en la figura 2. Por ejemplo, una aplicación de correo vía web actúa como servidor para el navegador y como cliente del servidor de correo que gestiona los mensajes del usuario en cuestión. Los servidores web y los otros servicios disponibles en Internet son clientes del servicio de resolución de nombres (DNS). Un tercer ejemplo son los buscadores (*search engines*), que permiten a los usuarios acceder a sumarios de información de páginas web extendidas por muchos sitios web de toda Internet. Un buscador es al mismo tiempo servidor y cliente: responde a peticiones provenientes de los navegadores clientes y ejecuta programas que, actuando como clientes (robots web; en inglés, *web crawlers*), acceden a servidores de Internet buscando información. De hecho, un buscador incluirá diferentes hilos de ejecución, algunos sirviendo al cliente y otros ejecutando robots web.

DNS (*domain name system*, 'sistema de nombres de dominio') traduce nombres de dominio de Internet a direcciones IP.

Por ejemplo, traduce el nombre de dominio `www.MiNegocio.com` a la dirección de Internet `196.156.34.2`.

Figura 2



Servidor cliente de otros servidores

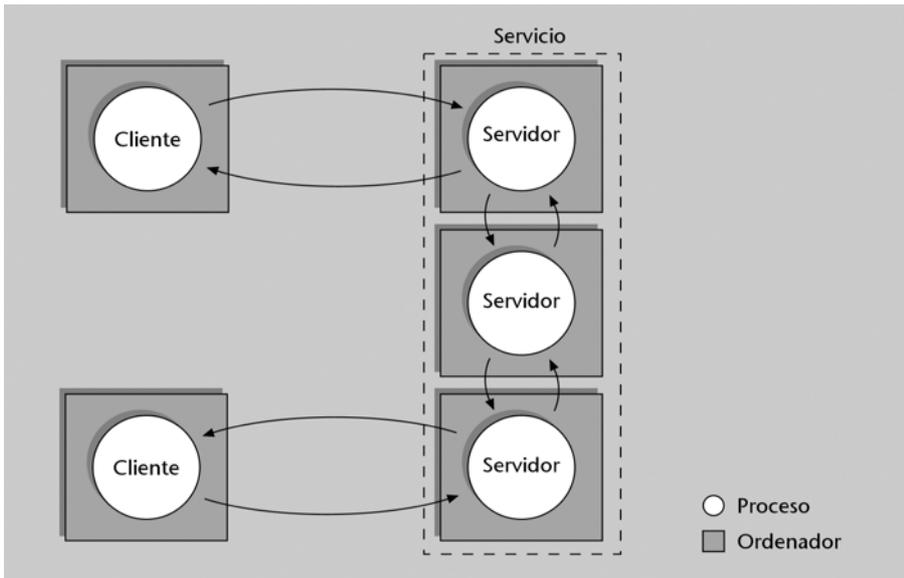
### Servicios proporcionados por múltiples servidores

Los servicios se pueden implementar como diferentes procesos servidores que se ejecutan en distintos ordenadores y que interactúan para proporcionar un servicio a procesos clientes (figura 3). Los servidores se pueden repartir los distintos objetos que componen el servicio que proporcionan, o pueden mantener réplicas de los objetos en diferentes ordenadores.

La web proporciona un ejemplo habitual de partición de los datos, en el que cada servidor gestiona su conjunto de recursos. Un usuario utiliza un navegador para acceder a un recurso situado en cualquiera de los servidores.

La replicación se usa para incrementar el rendimiento y la disponibilidad y para mejorar la tolerancia a fallos. Proporciona múltiples copias consistentes de los datos en procesos que se ejecutan en diferentes ordenadores. Por ejemplo, la web proporcionada en `www.google.com` (o `www.uoc.edu`) está mapeada en diferentes servidores que tienen datos replicados.

Figura 3



Diferentes procesos servidores

**Arquitectura multiestrato\***

\* En inglés: *multitier architecture*.

En las arquitecturas multiestrato, la funcionalidad está distribuida entre distintas plataformas u ordenadores. La interfaz reside en el ordenador del usuario, los servicios funcionales pueden estar en uno o más ordenadores, y los datos o los sistemas propietarios están en plataformas adicionales. Es muy habitual hablar de arquitecturas multiestrato en la bibliografía relacionada con las arquitecturas de sistemas de información. Las arquitecturas multiestrato más habituales son la de dos estratos y la de tres estratos.

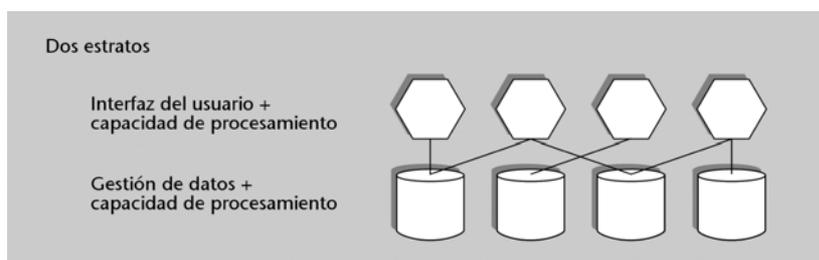
La **arquitectura de dos estratos** está formada per tres componentes distribuidos en dos niveles (nivel cliente y nivel servidor). Los tres componentes son los siguientes:

- 1) Interfaz usuario del sistema
- 2) Capacidad de procesamiento
- 3) Gestión de datos (servicio de datos y ficheros)

**En inglés...**  
 ... la arquitectura de dos estratos se denomina *two tier software architecture*, y la arquitectura de tres estratos se denomina *three tier software architecture*.

La interfaz de usuario está ubicada en el cliente. La gestión de la base de datos está ubicada en el servidor. La capacidad de procesamiento está repartida entre el cliente y el servidor. La figura 4 muestra la arquitectura de dos estratos.

Figura 4

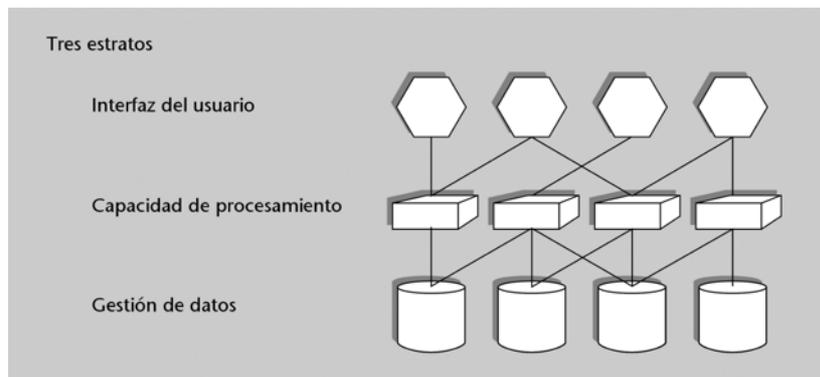


Arquitectura de dos estratos

La arquitectura en dos estratos mejora la usabilidad, la escalabilidad y la flexibilidad de las aplicaciones.

La **arquitectura de tres estratos** es una evolución de la arquitectura de dos estratos y ubica el tercer estrato entre la interfaz de usuario (cliente) y el gestor de datos (servidor). Este tercer estrato proporciona la capacidad de procesamiento. En la figura 5 se muestra esta arquitectura.

Figura 5



Arquitectura de tres estratos

Comparándola con la arquitectura de dos estratos, la arquitectura de tres estratos mejora el rendimiento, la flexibilidad, la facilidad de mantenimiento de la aplicación, la reusabilidad y la escalabilidad.

### Aplicaciones basadas en la web

Un caso particular de aplicaciones cliente-servidor son las aplicaciones que se ejecutan aprovechando la arquitectura de la web. Estas aplicaciones se basan en el hecho de tener toda la capacidad de procesamiento en un servidor web (o conjunto de servidores) a los cuales se accede desde un navegador web. Cuando el usuario hace clic sobre un enlace de la página web que tiene en su navegador, se genera una petición en el servidor que contiene la información. El servidor, al recibir la petición, devuelve la página pedida si la petición estaba en una página, o devuelve el resultado de ejecutar una aplicación si el enlace correspondía a un código que había que ejecutar (por ejemplo, CGI o Servlet). El navegador web proporciona a la aplicación un entorno en el que presentar la información. La comunicación entre cliente y servidor se hace utilizando el protocolo HTTP. El resultado que devuelve el servidor al cliente se envía en formato HTML, de manera que para el cliente web es totalmente transparente si accede a una página web o a una aplicación que genera un resultado formateado en HTML.

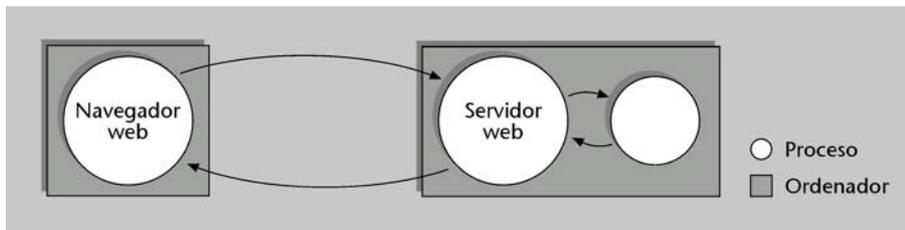
Las aplicaciones basadas en la web tienen la ventaja de que son accesibles desde cualquier ordenador que disponga de un navegador (la práctica totalidad de los ordenadores conectados hoy día a Internet) sin que sea necesario tener nada más instalado en el ordenador local. El uso de estas arquitecturas también facilita el diseño de las aplicaciones, ya que no hay que implementar la comunicación entre el cliente y el servidor (se utiliza el protocolo HTTP); y la parte de

presentación de la aplicación se facilita mucho por el hecho de formatear el documento en HTML y aprovechar las funcionalidades que proporciona el navegador (como los intérpretes de javascript y Java).

La facilidad y universalidad en el acceso a las aplicaciones que proporciona esta arquitectura es la base de los servicios ofrecidos en Internet.

Algunos ejemplos son el campus virtual de la UOC, los servidores de correo web tipo Yahoo o Google y los bancos por Internet.

Figura 6



Aplicaciones web

### Servidores intermediarios\* y memorias caché

\* En inglés: *proxy*.

El modelo cliente-servidor se dota de un conjunto de mecanismos para mejorar su disponibilidad, accesibilidad y tiempo de respuesta. Estos mecanismos son las memorias caché y los intermediarios.

Una memoria caché es un almacén intermedio de objetos usados recientemente entre un cliente y un servidor. Cuando un ordenador recibe un objeto, lo almacena en la memoria caché. Cuando el cliente pide un objeto, el ordenador comprueba primero si está en la memoria caché. Si esta copia está actualizada (normalmente el protocolo entre cliente y servidor incluye un comando para validar si la copia en la memoria caché está actualizada o no), el cliente obtiene la copia. Si no lo está, la va a buscar. La introducción de memorias caché elimina parcialmente o completamente algunas interacciones, y mejora así la eficiencia y la percepción del usuario sobre ésta.

#### Ejemplos de uso de memorias caché son los siguientes:

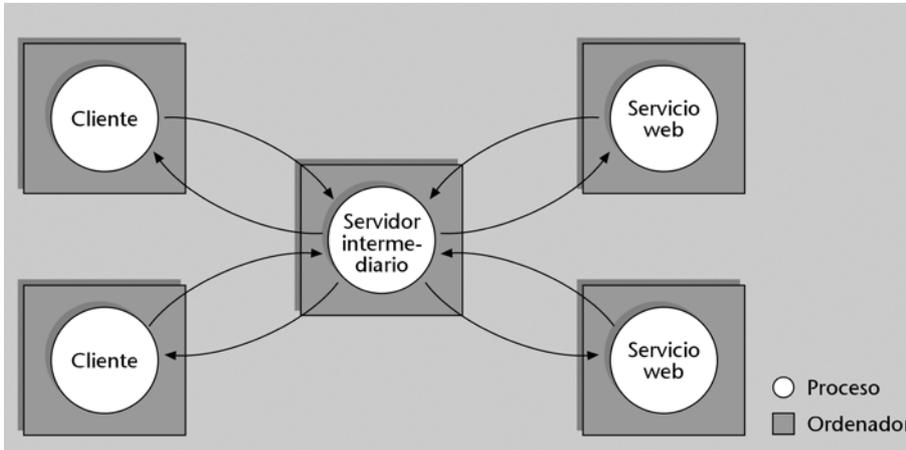
- *Network file system* (NFS) de Sun Microsystems,
- memorias caché locales de los navegadores. Los navegadores web mantienen en el sistema de ficheros local del cliente una memoria caché con las páginas web o los recursos visitados recientemente. Éstos, antes de presentar la página de la memoria caché al usuario, comprueban en el servidor original, utilizando una petición HTTP especial, si la página está actualizada.

Cada cliente puede tener su memoria caché, o éstas pueden estar situadas en un servidor intermediario que puede ser compartido por distintos clientes.

Como se muestra en la figura 7, los servidores intermediarios reciben las peticiones y las reenvían, con posibles traducciones, a los servidores. Los intermediarios actúan como servidores compartidos por uno o más clientes. El propósito de los servidores intermediarios es incrementar la disponibilidad y

el rendimiento de un servicio reduciendo la carga en la red y en los servidores. En el caso de los servidores web, los servidores intermediarios proporcionan una memoria caché compartida de recursos web para los ordenadores clientes. También se pueden usar con otros roles; por ejemplo, para acceder a un servidor web remoto por medio de un cortafuego.

Figura 7



Servidores intermedios

### 3.2. Arquitecturas descentralizadas

En el apartado anterior hemos visto cómo las arquitecturas multiestrato hacían una **distribución vertical** de los componentes, es decir, distribuían *lógicamente* los componentes en diferentes máquinas según sus funcionalidades. La distribución vertical no es la única manera de separar funcionalidades de un sistema distribuido. La **distribución horizontal** consiste en distribuir en partes lógicamente equivalentes las funcionalidades de un cliente o de un servidor, de manera que cada parte mantenga todas las funcionalidades, pero que la carga sobre el sistema quede balanceada entre las diferentes partes.

Uno de los paradigmas más representativos de la distribución vertical son los sistemas de igual a igual\*. En un sistema de igual a igual, la mayor parte de la interacción entre los componentes es simétrica, es decir, los componentes actúan como clientes y servidores al mismo tiempo. Los nodos que forman un sistema o aplicación de igual a igual se organizan formando lo que se denomina una *red superpuesta* (*overlay network*, en inglés) que funciona sobre la red física que conecta los nodos. Hay diferentes modos de organizar esta red superpuesta, que entre otros condicionará el determinismo del sistema. Principalmente las dividimos en tres categorías: estructuradas, no estructuradas e híbridas.

#### Motivación

En Internet, desde sus orígenes, ha habido sistemas y aplicaciones que se han comportado siguiendo la filosofía de igual a igual. Estos sistemas se han caracterizado por ser totalmente descentralizados, de gran escala y con capacidad para autoorganizarse. El ejemplo paradigmático son los mensajes Usenet.

\* En inglés: *peer-to-peer* o *P2P*.

#### Los mensajes Usenet

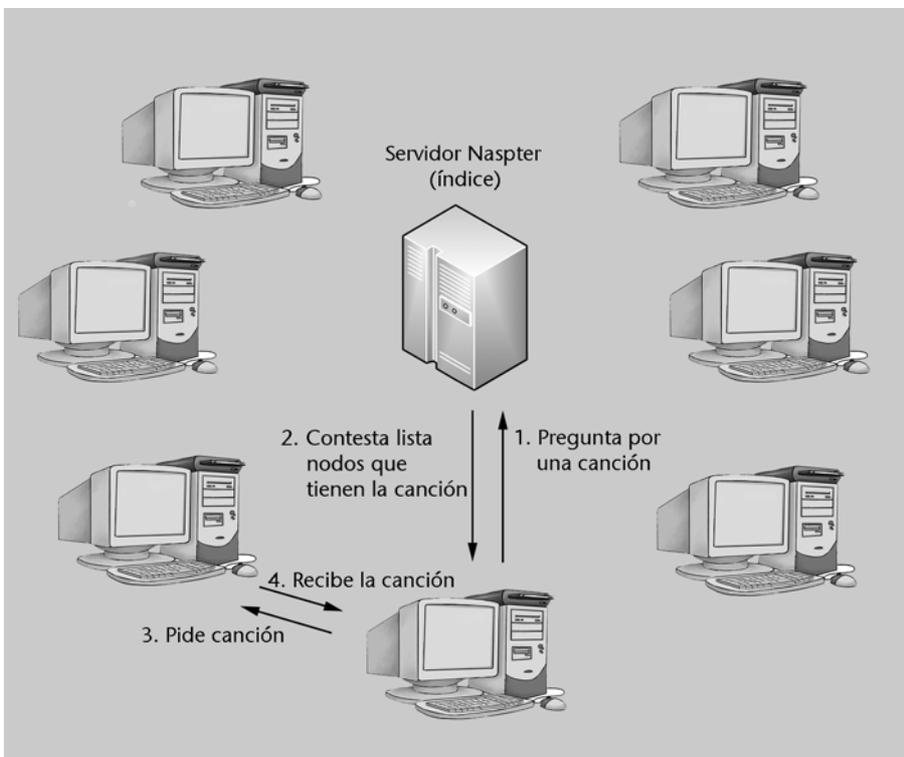
Los mensajes Usenet (*usenet news*, en inglés) es un sistema global y descentralizado de grupos de discusión en Internet. Los mensajes se distribuyen entre un gran número de servidores, que almacenan y se reenvían mensajes los unos a los otros. Los usuarios se bajan los mensajes y ponen otros nuevos en uno de los servidores. El intercambio de mensajes entre los servidores hace que a la larga los mensajes lleguen a todos los servidores.

Aun así, el fenómeno de igual a igual empieza como tal a finales de los noventa de la mano del Napster. En la época de la explosión de Internet –a partir de 1994– la vertiente de colaboración que había dominado Internet hasta aquel momento empezó a perder protagonismo frente a la vertiente más comercial que imponía la arquitectura cliente-servidor, liderada por la web como arquitectura estrella.

Dentro de este contexto dominado por la centralización de la arquitectura cliente-servidor, los usuarios del Naster descubrieron que el efecto agregado de poner cada individuo canciones al servicio de una comunidad era que los participantes de la comunidad encontraban con facilidad las canciones que les interesaban.

El funcionamiento del Napster era muy sencillo. Usaba un servidor (o índice) para proporcionar un servicio de directorio. Cuando un usuario arrancaba un nodo del Naster, éste conectaba al servidor y publicaba allí la lista de canciones que el nodo local hacía pública. De esta manera, el servicio de directorio sabía, por cada igual, qué objetos tenía disponibles para compartir. Cuando alguien buscaba una canción hacía una petición de la canción al servidor y éste le contestaba la lista de nodos que tenían un título similar. El usuario escogía a uno y se descargaba la canción directamente de este nodo.

Figura 8



La aplicación Napster

Los grandes cambios que aporta el Napster, tanto desde el punto de vista de la arquitectura como del funcionamiento del sistema, con respecto a las soluciones centralizadas (cliente-servidor) que predominaban en aquel momento son:

- Los ficheros que hay disponibles son los que los usuarios, de manera individual, deciden aportar al sistema (autonomía de los usuarios).
- La disponibilidad de un fichero depende de si los usuarios que lo tienen están conectados al sistema (conectividad puntual o *ad-hoc* en inglés).
- Hay muchos usuarios que proporcionan un mismo fichero (tolerancia a fallos).
- Los recursos necesarios para almacenar los ficheros los aportan los mismos usuarios (coste del sistema).
- El sistema evoluciona y se adapta a medida que los usuarios se conectan o desconectan (autoorganización).
- El sistema soporta a muchos usuarios (escalabilidad). De hecho, llegó a haber millones de usuarios conectados al Napster.

- Al haber muchas réplicas de una canción, la carga está repartida (mejora rendimiento).

Aunque hubiera un servidor o índice, se considera que el Napster era un sistema de igual a igual porque los ficheros se encontraban en los ordenadores de los usuarios y la bajada se hacía directamente entre el ordenador que buscaba la canción y lo que le ofrecía.

Esta manera de organizar grandes cantidades de información a escala Internet para facilitar su compartición resultó ser muy eficaz. La prueba de esto la encontramos tanto en el hecho de que el sistema llegó a tener millones de usuarios como en el hecho de que muchas empresas discográficas lo vieron como una amenaza. Precisamente, el motivo de que el Napster dejara de funcionar fue que denunciaron a los propietarios del servicio de directorio por infringir las leyes del *copyright*. El caso Napster acabó con una sentencia judicial que forzó el cierre del servidor índice.

A raíz del éxito de la solución, mucha gente se animó a hacer propuestas más descentralizadas y que superaran las limitaciones del Naster. Gnutella es un ejemplo de ello. Gnutella se basa en un algoritmo de inundación para localizar el fichero que se busca. De esta manera elimina el punto único de fallo que supone tener un servicio centralizado de directorio. Una vez localizado el fichero, la descarga se hace directamente entre quien quiere el fichero y quien lo proporciona. Esta solución tiene el inconveniente de que no es determinista.

#### Determinismo

Por determinismo entendemos el fenómeno por el cual diferentes ejecuciones de una misma operación dan el mismo resultado. Sistemas tipos Gnutella no son deterministas. El algoritmo que utilizan para localizar ficheros dentro del sistema no garantiza que si un fichero está en alguno de los iguales la encuentre. Puede ser que, según el camino que haya seguido la consulta, nos diga que no lo ha encontrado, cuando sí que está.

### 3.2.1. No estructuradas

Un sistema de igual a igual que utilice una red superpuesta tipo no estructurado es un sistema que está compuesto de iguales que se conectan a la red sin conocer su topología. Estos sistemas usan mecanismos de inundación para enviar consultas a través de la red superpuesta. Cuando un igual recibe la pregunta, envía al igual que lo ha originado una lista de todo el contenido que encaja con la pregunta. Mientras que las técnicas basadas en la inundación son útiles para localizar objetos altamente replicados y son resilientes ante las conexiones y desconexiones de los nodos, no tienen un comportamiento muy bueno cuando se hacen búsquedas de objetos poco replicados. De esta manera, las redes superpuestas no estructuradas tienen fundamentalmente un problema: cuando tienen que gestionar un ritmo elevado de consultas o cuando hay crecimientos repentinos del tamaño del sistema, se sobrecargan y tienen problemas de escalabilidad.

A pesar de que el sistema de direccionamiento basado en claves que utilizan los sistemas de igual a igual estructurados puede localizar de manera eficiente objetos y, además, es escalable, sufre sobrecargas significativamente más grandes que los sistemas no estructurados por contenidos populares. Por eso, los sistemas descentralizados no estructurados se usan más.

El ejemplo más relevante de los sistemas no estructurados lo encontramos con el sistema Gnutella.

#### a) Búsquedas en sistemas no estructurados

El rendimiento de un sistema distribuido no estructurado está condicionado, entre otros, por el coste de localizar un objeto en el sistema. Las técnicas de busca han sido sobradamente estudiadas, ya que son mecanismos que deter-

minan tanto la probabilidad de localizar un objeto como la carga que debe soportar el sistema. En este módulo estudiaremos las siguientes:

- **Inundación (*flooding*)**. La técnica de inundación consiste en encaminar las búsquedas mediante todos los nodos vecinos. Cada nodo retransmite la petición a sus nodos vecinos de manera recursiva hasta que algún nodo encuentra el objeto buscado y responde al nodo emisor. Una de las ventajas de esta técnica es su resiliencia ante entradas y salidas de nodos al sistema. Una de las desventajas de la busca por inundación es su escalabilidad, ya que los costes de buscar un objeto se incrementan de modo exponencial a medida que el número de nodos aumenta. A pesar de ello, sistemas como Gnutella han demostrado su viabilidad en redes de tamaño medio.
- **Caminos aleatorios (*random walks*)**. Podemos considerar las búsquedas basadas en caminos aleatorios como procesos estocásticos que consisten en una secuencia de cambios determinados de modo aleatorio. En una red no estructurada, esta técnica en lugar de encaminar las búsquedas mediante todos los vecinos lo hace mediante un número determinado y aleatorio de éstos.

Formalmente, podemos definir un camino aleatorio como:

Dado un grafo conexo  $G(V,E)$  con  $|V| = n$ ,  $|E| = m$  un **paso** aleatorio en  $G$  es el movimiento desde algún nodo  $u$  hacia otro nodo vecino  $v$  aleatoriamente seleccionado. Un camino aleatorio es una secuencia de estos pasos aleatorios empezando desde algún nodo inicial.

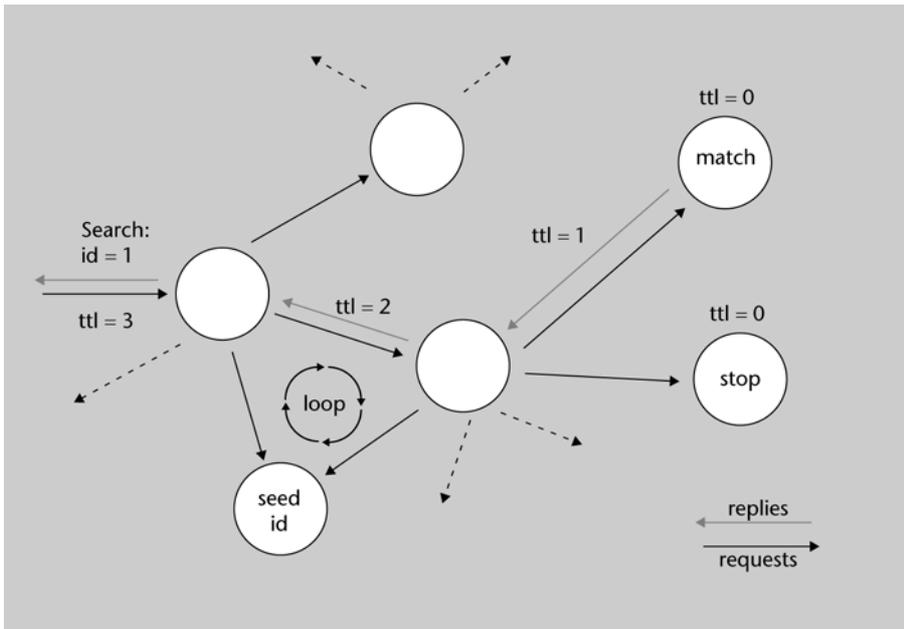
La aplicación de los caminos aleatorios como mecanismos de busca en sistemas no estructurados ha demostrado comportamientos más escalables que las técnicas de inundación en sistemas donde la red superpuesta forma estructuras clusterizadas o en sistemas donde una busca se repite sin que haya demasiados cambios en la topología de la red\*.

\* Christos Gkantsidis; Milena Mihail; Amin Saberi (2004). "Random walks in peer-to-peer networks". Proceedings of IEEE INFOCOM.

- **Caminos aleatorios adaptativos**. Recientemente se han introducido mejoras en la técnica de los caminos aleatorios. Una de estas mejoras consiste en controlar el radio de busca mediante el TTL (*time to live*) del mensaje de busca. La técnica inicia la busca con un TTL con valor 1; si no se encuentra el objeto buscado, vuelve a iniciar una con un TTL con valor 2 y así sucesivamente hasta que se encuentra el objeto buscado o bien se abandona la busca. Esta mejora permite controlar la carga sobre la red. Otras técnicas orientan la busca mediante ecuaciones que seleccionan el siguiente paso en función de ciertos parámetros, como puede ser la popularidad estimada del siguiente nodo que hay que consultar, es decir, seleccionan aquel nodo que tiene más probabilidad de tener el objeto. Esta técnica ha demostrado ser eficiente para búsquedas de objetos populares.

En la figura siguiente vemos una busca que ya ha sido iniciada por el nodo con id = 1 y que todavía no ha encontrado el objeto buscado habiendo probado el ttl = 2. La nueva busca se inicia con ttl = 3. Ésta se encamina hacia algunos de los nodos vecinos del nodo con id = 1 y se va decrementando el ttl a cada salto. Al cabo de tres saltos la retransmisión del mensaje se detiene (el nodo marcado con *stop* ya no retransmitirá el mensaje a sus vecinos). El nodo marcado con *match* contiene el objeto buscado y responde a la petición.

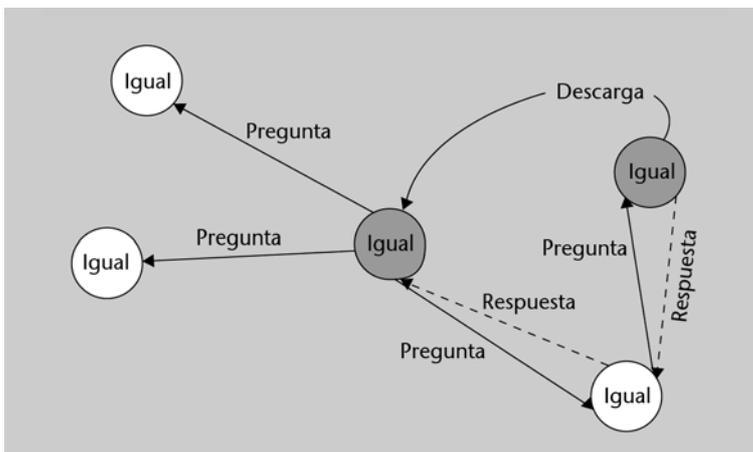
Figura 9



Random Walks

**Gnutella:** protocolo totalmente descentralizado para hacer búsquedas sobre una topología de iguales totalmente plana. Ha sido (y aún lo es) muy usado. Para localizar un objeto, un igual pregunta a sus vecinos. Éstos inundan a sus vecinos y así hasta un cierto radio. Este mecanismo es extremadamente resiliente ante entradas y salidas de nodos, pero los mecanismos actuales de busca no son escalables y generan cargas no esperadas en el sistema. La figura 10 muestra un ejemplo de búsqueda.

Figura 10



## b) Mecanismos para el mantenimiento de la información

Los sistemas distribuidos no estructurados requieren mecanismos para mantener de modo consistente la información distribuida, como puede ser información sobre el estado del sistema, información sobre la topología de la red superpuesta, etc.

La red superpuesta que forman los sistemas distribuidos no estructurados puede ser comparada a un grafo aleatorio. Cada componente del sistema distribuido representa un nodo del grafo y las aristas representan relaciones de vecindad, de manera que un nodo A está unido con una arista al nodo B si, y solo si, el nodo A es vecino\* del nodo B.

\* Las relaciones de vecindad son dependientes de la aplicación. Se pueden definir en términos de proximidad geográfica, topológica, o bien definidas según las necesidades de la aplicación.

Cada nodo del grafo conoce una lista de vecinos a la que denominaremos **vista parcial**. Los mecanismos más usados para construir y mantener la **vista parcial** de los componentes del sistema distribuido son los mecanismos epidémicos.

Los **mecanismos epidémicos** son aquellos en los que cuando dos nodos se comunican intercambian su información local así como la información que han recibido de otros. Estos mecanismos, análogamente a una epidemia, propagan la información mediante los nodos vecinos llegando en “infectar” al sistema completo. Los mecanismos epidémicos se usan para resolver problemas complejos en sistemas donde la topología de la red es no estructurada, la escala del sistema es muy grande, o bien son la solución más eficiente de las posibles.

Un mecanismo epidémico satisface las condiciones siguientes:

- Los componentes interactúan en parejas de manera periódica.
- La información que intercambian los componentes es de medida pequeña y limitada.
- El estado de uno o de los dos componentes cambia después de la interacción.
- La comunicación es no fiable.
- La frecuencia de las interacciones es pequeña en comparación con la latencia de los mensajes. Esto provoca que los costes del mecanismo sean negligibles.
- Hay una cierta aleatoriedad en la selección de los componentes con los que interactuar. La selección se puede efectuar entre los vecinos o bien teniendo en cuenta todos los nodos del sistema.

La dirección en la que se transmite la información, así como la proactividad de los componentes, diferencia dos modos de operación:

- **Modo de envío inducido (en inglés, *pull mode*)**. En este modo, los componentes esperan pasivos a recibir información de otro nodo del sistema.

Un componente sólo actualiza su información cuando otro nodo vecino se pone en contacto con él.

- **Modo de envío automático (en inglés, *push mode*).** En este modo, los componentes de manera periódica toman la iniciativa de comunicarse con algún otro nodo con el fin de actualizar su información.

Como ya se ha dicho antes, los mecanismos epidémicos o de “chismorreo” (en inglés, *gossip*) se usan entre otras cosas para mantener la topología de la red superpuesta, por ejemplo, utilizando el **modo de envío inducido**, el **modo de envío automático** o bien una combinación de ambos modos los componentes de un sistema consiguen mantener la **vista parcial**.

### c) Aplicaciones de igual a igual en red superpuesta no estructurada

Las aplicaciones más representativas que se han construido haciendo uso del paradigma de igual a igual no estructurado (también conocido como de igual a igual puro) son Gnutella, que ya hemos visto anteriormente, y Freenet.

Freenet es una red descentralizada para intercambio de información cuyo objetivo es mantener el anonimato de sus usuarios, ya sea de los que aportan información a la red o de los que la utilizan. Es descentralizada porque no hay ningún servidor central que almacene datos o coordine la red, sino que cada “nodo” de la red guarda información y aporta información para toda la red. La ideología de Freenet es preservar el anonimato de los que lo usan y permitir la libre expresión en la red sin que exista ningún tipo de censura. Una vez un fichero es insertado en la red se hace casi imposible saber de dónde proviene o quién lo ha originado; tampoco es posible eliminar el contenido de la red (éste va desapareciendo a medida que deja de ser accedido).

Otra aplicación con una finalidad bastante diferente a la de Freenet y Gnutella es Groove. Groove es una aplicación de igual a igual que ofrece funcionalidades para el trabajo colaborativo mediante la red. La idea básica es la de acceso a un espacio virtual de almacenaje en el que todos los iguales de un grupo pueden tener acceso a él. Groove permite a sus usuarios la sincronización de documentos, entre otras actividades colaborativas. Con el fin de mantener la información sincronizada y detectar conflictos, los iguales de un grupo hacen uso de mecanismos epidémicos. Esta aplicación además de gestionar la colaboración entre iguales de manera totalmente no estructurada también permite hacer uso de servidores que facilitan de manera distribuida la sincronización de la información.

### 3.2.2. Estructuradas

La topología de la red superpuesta sobre la que se construyen estos sistemas está fuertemente controlada y el contenido no va a cualquier lugar, sino a uno determinado que hace que las consultas sean más eficaces. Estos sistemas utilizan tablas de *hash* distribuidas (*distributed hash tables* o DHT en

inglés) como sustratos, en los que la ubicación de los objetos (o valores) se hace de manera determinista. Los sistemas que se basan en DHT tienen la propiedad de asignar los identificadores de nodos de una manera consistente a los iguales dentro de un espacio con muchos identificadores. A los objetos de datos se les asigna un identificador, al que se denomina clave, escogido dentro del mismo espacio de nombres. El protocolo de la red superpuesta mapea las claves a un único nodo entre los conectados. Estas redes superpuestas soportan el almacenamiento y la recuperación de pares {clave,valor} en la red superpuesta.

Cada igual mantiene una tabla de direccionamiento pequeña. Los mensajes se dirigen de una manera progresiva hacia los iguales a través de caminos de superposición. Cada nodo envía el mensaje al nodo de su tabla de direccionamiento que tiene un identificador más próximo a la clave en el espacio de identificadores. Los diferentes sistemas basados en DHT tienen diferentes esquemas de organización para objetos de datos y su espacio de claves y estrategias de direccionamiento. En teoría, los sistemas basados en DHT garantizan que, como media, se puede localizar cualquier objeto en  $O(\log N)$  saltos en la red superpuesta, donde  $N$  es el número de iguales en el sistema. El camino entre dos nodos en la red física puede ser muy diferente del camino en la red superpuesta DHT. Esto puede provocar que la latencia en las búsquedas en un sistema de igual a igual basado en una red DHT sea bastante grande y pueda afectar negativamente al rendimiento de la aplicación que funcione por encima.

### **Redes superpuestas estructuradas**

Can, Chord, Tapestry, Pastry, Kademlia, DKS o Viceroy son ejemplos de redes superpuestas estructuradas.

Podéis encontrar más información de estos sistemas en:

#### **CAN**

S. Ratnasamy y otros (2001). "A Scalable Content Addressable Network". *Proc. ACM SIGCOMM* (pág. 161-72).

#### **Chord**

I. Stoica; R. Morris y otros (2003). "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications". *IEEE/ACM Trans. Net.* (vol. 11, núm. 1, pág. 17-32).

#### **Tapestry**

B. Y. Zhao y otros (2004, enero). "Tapestry: A Resilient Global-Scale Overlay for Service Deployment". *IEEE JSAC* (vol. 22, núm. 1, pág. 41-53).

#### **Pastry**

A. Rowstron; P. Druschel (2001). "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems". *Proc. Middleware*.

#### **Kademlia**

P. Maymounkov; D. Mazieres (2002, febrero). "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". *Proc. IPTPS* (pág. 53-65). Cambridge, MA, EUA.

#### **DKS**

DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications.

#### **Viceroy**

D. Malkhi; M. Naor; D. Ratajczak (2002, julio). "Viceroy: A Scalable and Dynamic Emulation of the Butterfly". *Proc. ACM PODC 2002* (pág. 183-92). Monterey, CA, EUA.

## Tablas de *hash* distribuidas (*distributed hash tables*)

Las DHT surgen en el ámbito de la investigación en sistemas *peer-to-peer* como una evolución de los modelos de índice centralizado de Napster y totalmente descentralizado por inundación como Gnutella.

### Historia

En entornos académicos de investigación sobre sistemas distribuidos a finales de los noventa se intenta resolver el problema de la localización de recursos descentralizada que resuelva los cuellos de botella y problemas de escalabilidad de un índice centralizado. Un problema clave es conseguir que todos los nodos sean al mismo tiempo encaminadores de información y que se produzca un reparto balanceado de la carga entre todos los participantes. En este contexto, los sistemas descentralizados no estructurados como Gnutella se basan en sistemas de encaminamiento por inundación que mediante algoritmos de réplica adecuados permiten la localización de los datos. Se denominan *no estructurados* porque las conexiones entre los nodos son más o menos aleatorias sin que haya así una estructura global en el sistema.

Sin embargo, este tipo de sistemas generan mucho tráfico innecesario y no aseguran que podamos localizar un recurso aunque exista en la red. Un primer paso para solucionar estas limitaciones es el dado por Freenet situando los índices en zonas más o menos conocidas de la red. De esta manera, un nodo puede encaminar hacia la zona donde se supone que se encuentra el índice que está buscando. Éste es un primer paso hacia el denominado *greedy routing* (*routing* egoísta o ansioso), por el que cada nodo puede tomar una decisión local de hacia dónde encaminar partiendo de lo que se busca. Esto supone un gran avance frente al modelo de Gnutella, en el que cada nodo no sabe cuál de sus conexiones debe utilizar para encaminar y por ello ha de propagar en todas direcciones.

Las tablas de *hash* distribuidas (DHT) son un tipo de sistemas distribuidos que permiten la localización eficiente de datos por medio de un índice descentralizado y uniformemente repartido entre los nodos del sistema. Se les denomina DHT porque cada nodo es análogo a una celda de una tabla *hash* que permite almacenamiento y recuperación de información (PUT (K,V), GET(K)) de manera eficiente en un entorno distribuido.

Así, a comienzos del siglo XXI, se proponen las primeras redes *peer-to-peer* estructuradas como Chord, CAN, Pastry o Tapestry entre otras. En estas redes, los nodos se conectan entre ellos siguiendo una estructura predeterminada como un anillo, un hipercubo o un árbol. Las estructuras establecidas aseguran que cada nodo necesita pocas conexiones ( $\log N$ ), que la red tiene un diámetro pequeño y que se puede encaminar información de un nodo a otro en pocos saltos ( $\log N$ ). Además, se utiliza el *greedy routing*, por lo que cada nodo sabe por dónde es mejor enviar la información.

Un descubrimiento clave que propicia la aparición de estas redes estructuradas en el concepto de *hashing* consistente (*consistent hashing*). Aunque no fue pensado en

principio para entornos descentralizados *peer-to-peer*, esta contribución sirvió de inspiración para resolver el problema del reparto balanceado de los recursos (*load balancing*) entre los nodos de la red. Así, el *hashing* consistente utiliza una función determinística que asigna identificadores en un rango de manera uniforme, asegurando así el reparto equitativo de la carga.

## Chord

Estudiaremos el caso de Chord como ejemplo paradigmático y clásico de las redes *peer-to-peer* estructuradas o también denominadas *tablas de hash distribuidas* (DHT). Como vemos en la figura, en Chord los nodos forman un anillo con identificadores comprendidos entre  $[0 \text{ y } 2^M]$  módulo  $2^M$  siendo  $M$  el número de bits del identificador. En este ejemplo,  $M$  es 6, por lo que los identificadores están comprendidos entre el 0 y el 63.

El identificador de cada nodo se obtiene utilizando una función de *hashing* consistente que asegura que los identificadores estarán uniformemente repartidos. Un ejemplo de esto es la función SHA (*secure hash algorithm*), que genera un identificador de 160 bits que después se pueden truncar para manejar identificadores más reducidos (el tamaño suele estar entre 32 y 160 bits, lo que permitirá de esta manera un gran número de nodos).

Cuando un nodo entra en la red y obtiene un identificador, se debe situar en la zona del anillo adecuada entre el predecesor y el sucesor de su identificador. Es lo que se conoce como *proceso de entrada a la red* (*join, bootstrapping*) y lo puede hacer mediante cualquier nodo. Por ejemplo, si en esta red entra el nodo 4, se situará entre el nodo N1 y el N8.

En Chord, cada nodo es responsable de las claves o recursos situados entre su predecesor y él mismo (incluido). De esta manera, el N8 es responsable de las claves 2, 3, 4, 5, 6, 7 y 8. Cuando se inserta un recurso en la red, se aplica la función de *hashing* consistente para obtener su clave y así se asegura que los recursos están uniformemente repartidos. Al calcular el *hashing*, por ejemplo del nombre del fichero, (SHA(ASD.doc)) obtenemos una clave que nos dice, además, qué nodo de la red es su responsable. De esta manera, al insertar la clave 17 (PUT(17)) se almacenará en el sucesor de 17, que es el nodo N21.

Si cada nodo sólo tuviera los enlaces de sucesor, el encaminador sería  $O(N)$ . Es decir, en el peor de los casos si estamos en el nodo cero y sólo podemos ir hacia la derecha, para llegar al nodo 63 deberemos pasar por todos los nodos del anillo. Para mejorar esto, Chord tiene una tabla de enlaces a otros nodos denominada *finger table*. Esta tabla tiene  $\log N$  enlaces siendo  $N = 2^M$ , por lo cual en nuestro ejemplo la medida es 6. En la *finger table*, cada nodo tiene conexiones logarítmicas a otros nodos siguiendo el patrón:  $n + (2^i \text{ módulo } 2^M)$ , donde  $n$  es el identificador del nodo, e  $i$  corresponde a cada entrada de la *finger table* (ejemplo, 0...6). Como vemos en el ejemplo, el nodo N8 debería tener conexiones a los nodos 9, 10, 12, 16, 24 y 40 si la red fuera completa. Al no estarlo, se conectará a los nodos sucesores de estos identificadores que sí se encuentran en la red.

El encaminamiento en Chord es ansioso (*greedy routing*), en el sentido de las agujas del reloj (*clock-wise*) y se suele denominar *encaminamiento basado en claves* (*key based routing*, KBR). Cuando desde un nodo se intenta localizar un recurso con una clave concreta, el nodo busca en su *finger table* la conexión que más lo acerca a su destino. Por ejemplo, si en el N8 se busca la clave 54 (GET(54)), se enviará la solicitud al nodo N42, que es el que más se aproxima a la clave 54. Éste, a su vez, lo enviará al 51, que finalmente preguntará al 56, que es el responsable de la clave 54. Como podemos observar, en muy pocos saltos encontramos cualquier recurso de manera eficiente y así se evitan las inundaciones del modelo Gnutella.

## Discusión

Las ventajas clave de Chord y otros sistemas similares son la descentralización, la escalabilidad y el balanceo de la carga entre los participantes.

Las DHT permiten construir sistemas completamente descentralizados en los que todos los nodos son iguales entre ellos y altamente escalables a millones de participantes. Además, la ventaja esencial de estos sistemas es que permiten localizar información con un orden de saltos logarítmico ( $O(\log N)$ ) y manteniendo un pequeño número de conexiones a otros nodos (tablas de encaminamiento de orden constante  $O(1)$  o logarítmico  $O(\log N)$ ).

Los principales problemas son el mantenimiento del sistema, la entrada y salida de nodos constante (*churn*), la proximidad en red y la limitación a búsquedas exactas (*exact match*).

En Chord, por ejemplo, es necesario mantener en cada nodo una lista de sucesores de medida logarítmica para asegurar que no se rompe el anillo. Además, es necesario un protocolo de mantenimiento de enlaces que los actualice correctamente a medida que entran o salen nodos. Esto implica un coste que limita su aplicación para sistemas muy dinámicos y todavía se está estudiando en círculos de investigación académicos.

Además, en estas redes descentralizadas es esencial tener en cuenta la proximidad en red o las latencias entre nodos. Si un DHT necesita pocos saltos, pero éstos pasan por nodos muy lejanos en términos de latencia (España-Japón-Australia-Francia), el sistema será muy ineficiente. Por ello, muchos DHT han intentado organizarse partiendo de información de proximidad como CAN, Pastry o CORAL para mejorar el encaminamiento. Sin embargo, eso continúa siendo un problema abierto que se sigue investigando en círculos científicos.

Por otra parte, las DHT ofrecen un sistema de busca exacta GET(K) que las hacen menos flexibles que otros sistemas con búsquedas abiertas o más complejas. Además, cuando un nodo se va, sus claves han de migrar al nodo sucesor que ahora es el responsable de estas claves. Por ello se deben replicar estas claves para hacer el sistema tolerante a fallos y asegurar que

las claves no se pierden. Diferentes trabajos de investigación continúan estudiando hoy en día cómo se deben adaptar o construir aplicaciones sobre las DHT que permitan búsquedas más avanzadas.

De cualquier modo, las DHT han supuesto una auténtica revolución en entornos descentralizados y ya se las considera la tercera generación de sistemas *peer-to-peer* después de Napster y Gnutella. Gracias a las considerables cualidades de las DHT se han desarrollado muchos servicios distribuidos que se construyen sobre éstas.

Entre las aplicaciones existentes hoy en día que se basan en DHT destacamos las siguientes:

- **eMule KAD**: el programa de descarga de ficheros P2P eMule utiliza una DHT denominada Kademlia (KAD) como evolución del sistema. KAD mejora la localización de fuentes en la red y la hace menos frágil a ataques sobre los servidores centrales.
- **BitTorrent/Azureus**: los clientes de bitTorrent utilizan una implementación del DHT Kademlia para conseguir la descentralización del componente *tracker*. Esto los hace menos vulnerables a la caída de éste.
- **OpenDHT**: es un DHT (Tapestry) públicamente accesible que ofrece las funcionalidades de PUT y GET mediante interfaces Sun RPC y XMLRPC. Diferentes aplicaciones se han basado en OpenDHT como middleware base. <http://opendht.org/>

En conclusión, en la actualidad las DHT se consideran un sustrato distribuido esencial sobre el que construir servicios descentralizados. Su integración con mecanismos de proximidad en red de ámbito Internet permitirán aplicaciones eficientes en el contexto de la computación Grid de gran escala o en redes de distribución de contenidos (CDN). Finalmente, cabe destacar que las DHT no son la solución ideal a todos los problemas en sistemas distribuidos descentralizados (*silver bullet*). Por ejemplo, en entornos móviles con un gran dinamismo e inestabilidad de los nodos, las DHT no son la solución adecuada y sí que lo son, en cambio, los sistemas desestructurados o híbridos con menos coste de mantenimiento de la red.

## Híbridos

Los sistemas híbridos solucionan problemas que no pueden ser resueltos de modo eficiente por ninguno de los modelos anteriores. Por ejemplo, la localización de ciertos datos en sistemas descentralizados se convierte en problemática cuando la red crece. Los sistemas no estructurados usan técnicas de inundación no deterministas que no nos aseguran el éxito en las búsquedas. Los sistemas estructurados tienen problemas de sobrecarga cuando se hacen búsquedas sobre rangos de objetos, ya que éstos sólo permiten realizar búsquedas indexadas de modo eficiente sobre objetos concretos.

Por otra parte, los sistemas centralizados acaban convirtiéndose en un “cuello de botella” a medida que la escala del sistema aumenta.

**a) Super-iguales (*super-peers*)**. Uno de los tipos de sistemas híbridos más usados son los basados en super-iguales. Un super-igual es un nodo que actúa como índice o gestor de la información del sistema. Tal como su nombre indica, los super-iguales también se organizan en una red de igual a igual, formando una estructura con dos o más niveles. En muchos casos, la relación igual-super-igual es fija durante el tiempo de conexión del igual, ya que se

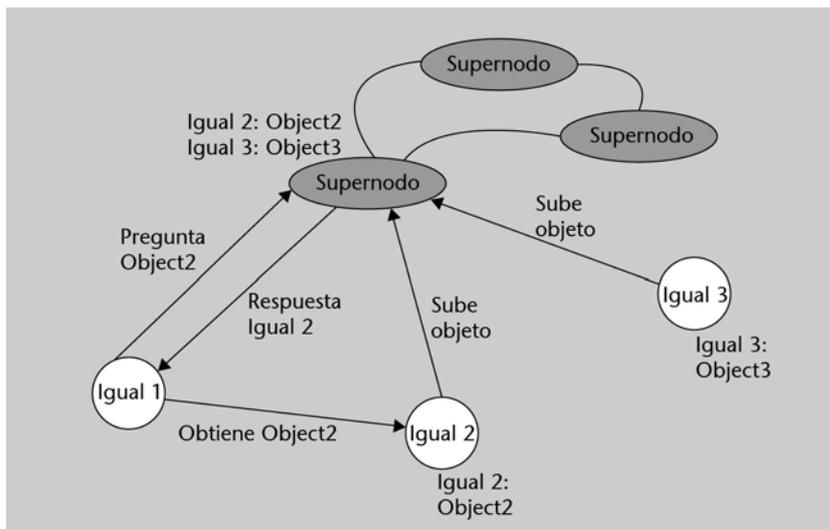
### Ejemplos de servicios...

... serían los sistemas de ficheros distribuidos, los sistemas de compartimentación de archivos P2P, el almacenamiento cooperativo en web, servicios de *multicast* o *anycast* en el ámbito de aplicación, o incluso servicios de nombres y de DNS descentralizados.

mantiene la relación mientras dura la conexión. Los super-iguales tienden a ser nodos que se mantienen conectados y sufren de poco dinamismo durante la vida del sistema. Los sistemas basados en super-iguales ofrecen facilidades en las búsquedas y gestión de la información, ya que ésta puede estar indexada. Un nuevo problema que aparece con estos sistemas es el de la elección de los nodos que actuarán como super-iguales, cuestión muy relacionada con los *problemas de elección de un líder* que estudiaremos en otro módulo de este curso.

- **KaZaA:** es un sistema de ficheros descentralizado en el cual los nodos se agrupan alrededor de super-iguales (*super-peers* en inglés) para hacer las búsquedas más eficaces tal como se muestra en la figura 11. La comunicación entre los iguales en KaZaA se hace utilizando el protocolo Fast Track, que es un protocolo propietario. Los super-iguales son iguales del sistema que se han escogido para que mantengan metainformación que hará las búsquedas más eficaces. En el momento de una búsqueda, el igual pregunta al super-igual en el que está conectado. Éste, de manera similar a lo que hace Gnutella, hace un *broadcast* a los otros super-iguales.

Figura 11



Búsqueda en KaZaA

Los iguales se conectan a un super-igual. Las consultas se encaminan hacia los super-iguales. Las bajadas se hacen entre iguales.

- **eDonkey:** es un sistema de igual a igual híbrido organizado en dos niveles para el almacenamiento de información. Está formado por clientes y servidores. Los servidores actúan como concentradores para los clientes y permiten a los usuarios localizar los ficheros que hay en la red. Esta arquitectura proporciona bajada concurrente de un fichero desde varias ubicaciones, uso de funciones resumen (*hash*, en inglés) para la detección de ficheros corruptos, compartición parcial de ficheros mientras se bajan, y métodos expresivos para hacer búsquedas de ficheros. Para que un nodo se pueda conectar al sistema, hace falta que conozca un igual que actúe como servidor. En el proceso de conexión, el cliente proporciona al servidor la información sobre los ficheros que comparte. Cuando un cliente busca un fichero, los servidores proporcionan

las ubicaciones de éstos. De esta manera los clientes pueden bajar los ficheros directamente de las ubicaciones indicadas.

eMule es una aplicación cliente de la red eDonkey (híbrida), aunque también puede utilizar la red Kad (estructurada, ya que es una DHT).

- **Skype** es otro sistema que proporciona telefonía mediante Internet. Utiliza un protocolo propietario de cuya implementación se conocen pocos detalles. Funciona siguiendo una organización en super-iguales tal como lo hace KaZaA. De hecho, Skype fue fundada por los fundadores de KaZaA. Un aspecto que conviene destacar es que consigue superar los problemas (entre otros la incapacidad de comunicación) que tienen los iguales que están detrás de un cortafuegos o problemas derivados del NAT (*network address translation*). La idea básica para hacerlo es la siguiente: los cortafuegos y los sistemas de NAT sólo dejan pasar aquellos paquetes que pertenecen a una dirección de confianza, conocida o con la que el ordenador se haya comunicado antes. Skype lo que hace es “persuadir” al cortafuegos haciéndole creer que ya se ha establecido una conexión con anterioridad. A partir de aquí, como la comunicación se hace mediante paquetes UDP, de los cuales el cortafuegos sólo puede extraer información sobre las direcciones IP y puertos de los participantes en la comunicación, el cortafuegos no puede verificar más información de los paquetes y permite su paso.

**b) Trackers:** Los Trackers son servidores dedicados a mantener información sobre un conjunto de iguales que comparten o almacenan un contenido determinado. Cada Tracker es responsable de mantener información sobre los iguales que almacenan uno o más de estos contenidos. El sistema más representativo que hace uso de los Trackers es BitTorrent.

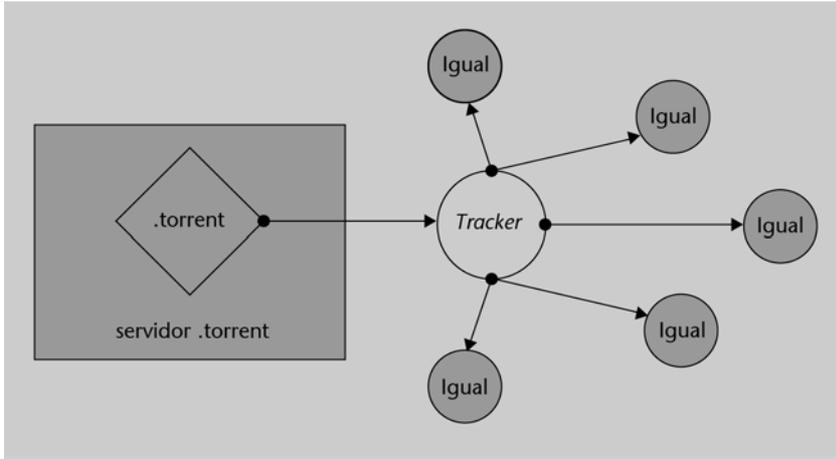
- **BitTorrent:** es un sistema de igual a igual para distribuir grandes volúmenes de datos sin que el originador de la información tenga que soportar todo el coste de los recursos necesarios para servir el contenido. Este tipo de soluciones son útiles para distribuir contenidos que son muy populares. BitTorrent utiliza servidores para gestionar las descargas. Estos servidores almacenan un fichero que contiene información sobre el fichero: longitud, nombre, información de resumen (*hashing information*, en inglés) y la URL del *tracker*. El *tracker* (mirad la figura 12) conoce todos los iguales que tienen el fichero (tanto totalmente como parcialmente) y hace que los iguales se conecten los unos con los otros para bajar o subir los ficheros. Cuando un nodo quiere bajar un fichero, envía un mensaje al *tracker*, que le contesta con una lista aleatoria de nodos que están descargando el mismo fichero. BitTorrent parte los ficheros en trozos (de 256 Kbytes) para poder saber qué tiene cada uno de los iguales. Cada igual que está bajando el fichero anuncia a sus iguales los trozos que tiene. El protocolo tiene mecanismos para penalizar a los usuarios que obtienen información sin proporcionarla. De esta manera, a la hora de subir información, un igual escogerá otro igual del que haya recibido datos.

#### Bibliografía complementaria

Para más información sobre el funcionamiento de BitTorrent, podéis consultar el artículo siguiente:

**B. Cohen** (junio, 2003). “Incentives Build Robustness in BitTorrent”. *Proc. First Workshop the Economics of Peer-to-Peer Systems*. USA: Berkeley.

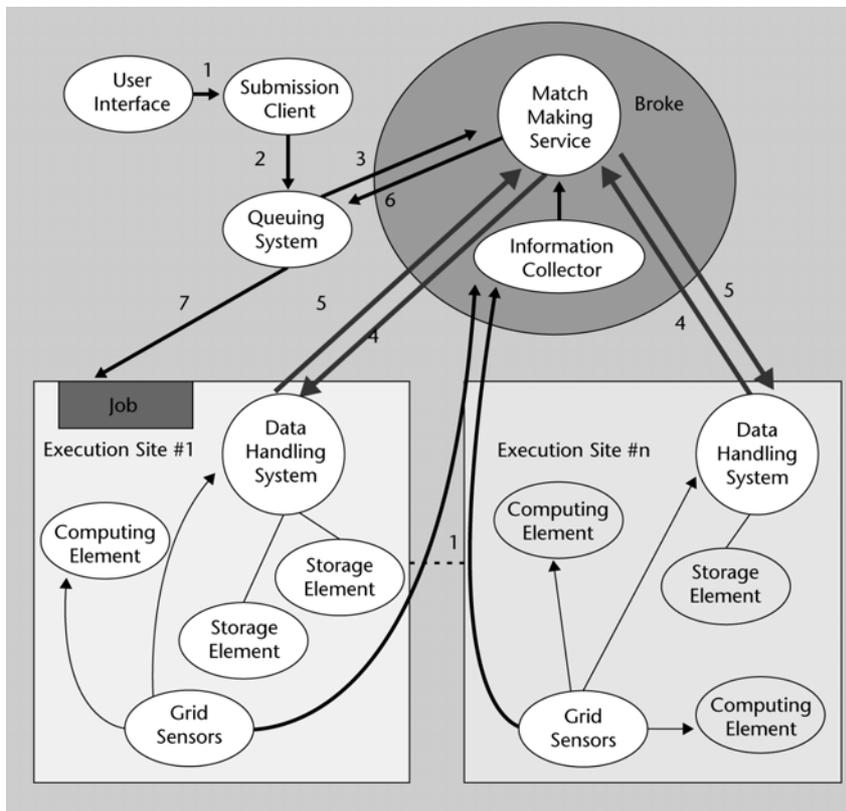
Figura 12



Tracker de BitTorrent

c) **Brokers:** Los Brokers son componentes dedicados a hacer de mediadores o intermediarios del resto de componentes del sistema. Por ejemplo, en Globule, una red de distribución de contenidos colaborativa formada por servidores de páginas web conectados en una red de igual a igual, los brokers actúan como registros de los servidores permitiendo que éstos sean descubiertos por los otros iguales. Otro ejemplo de esto lo encontramos en algunos sistemas computacionales distribuidos por computación GRID como Globus, en el que los gestores (diferentes posibles) de tareas hacen uso de un Broker que permite programar la ejecución de tareas de modo paralelo en un sistema heterogéneo y que en algunos casos se intenta garantizar un tiempo de finalización.

Figura 13



Secuencia típica de acciones para la ejecución de una tarea en un sistema de computaciones distribuido en el que la gestión es llevada a cabo por un broker.

## Aplicaciones de igual a igual

Los sistemas y aplicaciones de igual a igual se han hecho populares de la mano de las aplicaciones de compartición de ficheros, pero hay otros tipos de aplicaciones. Skype es otro sistema tipo de igual a igual que es muy popular. Skype proporciona telefonía en Internet. Utiliza un protocolo propietario del cual se conocen pocas cosas sobre su implementación. Funciona siguiendo una organización con super-iguales tal como lo hace KaZaA. De hecho, Skype fue fundada por los fundadores de KaZaA. Un aspecto a destacar es que consigue superar los problemas que tienen los iguales cuando están detrás de un cortafuegos o los problemas derivados del NAT (*network address translation*).

También hay otros sistemas de igual a igual para la comunicación síncrona (como la mensajería instantánea), juegos, sistemas de procesamiento distribuido (como seti@home) o software para la colaboración (como Groove).

### **SETI@home (<http://setiathome.berkeley.edu>)**

Es un proyecto que tiene como objetivo detectar vida inteligente fuera de la Tierra. Distribuye procesamiento entre muchos ordenadores personales que están suscritos al proyecto. Analiza datos de radiotelescopios aprovechando las grandes cantidades de tiempo de procesamiento que los PC desperdician porque no hacen nada.

### **Groove (<http://www.groove.net>)**

Groove es un sistema de igual a igual para facilitar la colaboración y comunicación en grupos pequeños. Proporciona herramientas para la compartición de ficheros, la mensajería instantánea, calendario, gestión de proyectos, etc.

## 3.3. Paradigma de publicación-suscripción

El paradigma de la publicación-suscripción\* se puede implementar haciendo uso tanto de arquitecturas centralizadas como de descentralizadas o híbridas. Si toman, por ejemplo, el modelo cliente-servidor que acabamos de ver, cada vez que el cliente necesita alguna información activamente debe hacer una petición al servidor. Éste es un buen modelo para muchas situaciones, pero en otras este método es poco eficiente. Pensemos, por ejemplo, en el caso de una agencia de bolsa que quiere mantener informados a sus clientes de la evolución en tiempo real de las cotizaciones de las acciones; o el caso de una agencia de noticias que distribuye información al momento. En estos casos, los receptores tendrían que ir consultando continuamente el servidor para tener la última cotización o la última noticia, con la sobrecarga que esto significa tanto a la red como al cliente y al servidor.

La manera como el paradigma publicación-suscripción aborda estas situaciones es haciendo que un productor de información anuncie la disponibilidad de un cierto tipo de información, un consumidor interesado se suscriba a esta información y el productor periódicamente vaya publicando información.

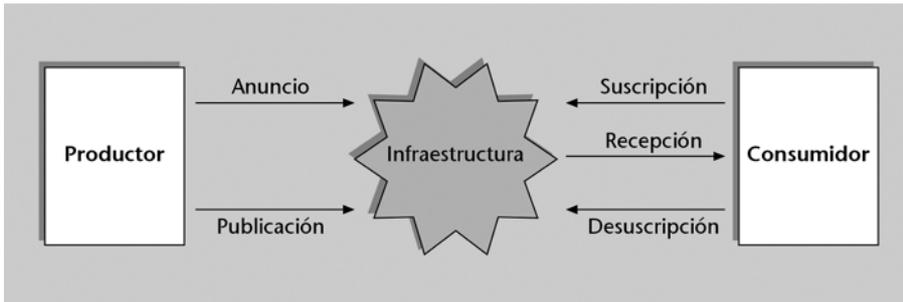
### **Bibliografía complementaria**

Encontraréis más información sobre el funcionamiento interno de Skype en:

**S. A. Baset; H. Schulzrinne** (abril, 2006). "An analysis of the Skype peer-to-peer internet telephony protocol". *Proceedings of IEEE INFOCOM 2006*. Barcelona.

\* En inglés: *publish/subscribe*.

Figura 14



Paradigma de publicación-suscripción

Para poder tener el comportamiento descrito, la arquitectura publicación-suscripción está formada por los componentes siguientes:

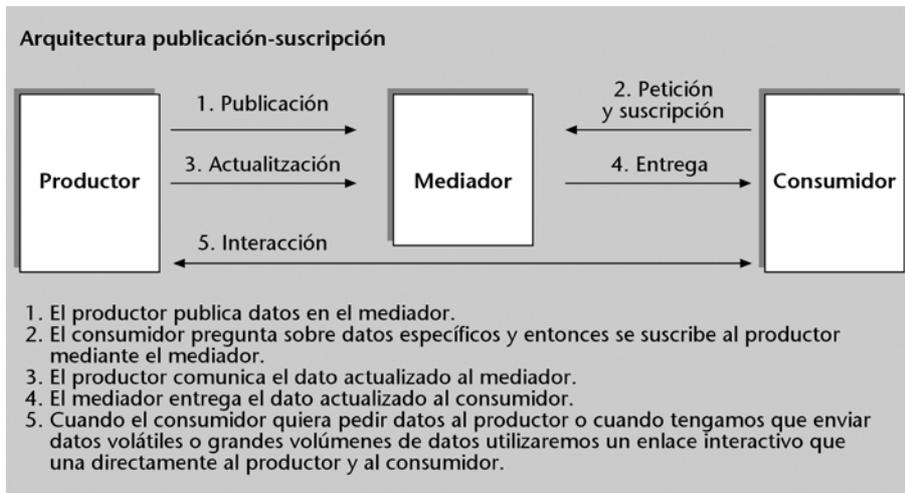
- **Productor de información.** Aplicación que tiene la información que hay que difundir. El productor publica esta información sin tener que saber quién está interesado en recibirla. Envía la información a través de canales.
- **Consumidor de información.** Aplicación interesada en recibir información. El consumidor se suscribe a los canales que diseminan la información que le interesa. Recibe esta información por los canales a los que está suscrito.
- **Mediador (broker).** Está entre el productor y el consumidor de información. Recibe información de los productores y peticiones de suscripción de los consumidores. También se encarga de encaminar la información publicada a los destinatarios suscritos al canal. Este mediador puede estar distribuido. En este caso, es necesario que los diferentes mediadores se organicen para proveer los canales.
- **Canal.** Son los conectores (lógicos) entre los productores y los consumidores de información. El canal determina varias de las propiedades de la información que hay que diseminar y de las funcionalidades soportadas: tipo de información; formato de los datos; posibilidades de personalización del canal por parte del usuario (por ejemplo, selección de contenidos, modos de operación); si el contenido expira o es persistente; estrategia que se seguirá para hacer las actualizaciones; si los datos se entregan sólo una vez (al ocurrir, como TV) o si, en cambio, garantizamos que se puede recibir el contenido independientemente del momento en el que se generó; modo de operación (si se da apoyo por el modo de operación en desconectado), pago (cuál es la política de pago que se utiliza: pagar por ver, por tiempo, por contenido, etc.).

Los canales modelan una relación de uno a muchos entre productores y consumidores. Habitualmente también son necesarios canales para que los consumidores se puedan relacionar de uno a uno con los productores. Esto se acostumbra a hacer en un estilo cliente-servidor y, por lo tanto, desde el punto de vista conceptual estaría fuera del sistema publicación-suscripción.

En la primera de las dos figuras que ponemos a continuación (figura 15), vemos la relación entre los distintos componentes de una arquitectura publica-

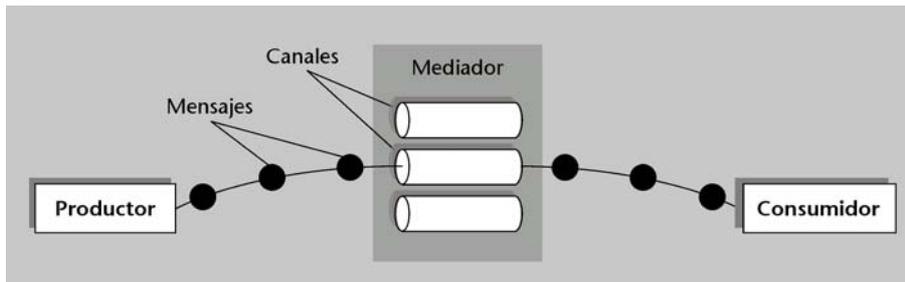
ción-suscripción. En la otra figura (figura 16) vemos el detalle de cómo se hace la comunicación entre un productor y un consumidor.

Figura 15



Detalle de la comunicación del paradigma de publicación-suscripción

Figura 16



Paradigma de publicación-suscripción

Tal y como hemos visto, los sistemas publicación-suscripción permiten una distribución asíncrona de información. A continuación indicamos algunas situaciones y aspectos en los que estos sistemas pueden ser una alternativa apropiada:

- **Localización:** para los usuarios es un problema saber dónde está la información que les interesa. Aunque haya buenas herramientas de búsqueda, muchas veces la información obtenida no es de la calidad deseada. En los sistemas publicación-suscripción, el usuario se suscribe a unos canales y ahora es el proveedor de información quien asume el rol activo de hacer llegar su información a los interesados.
- **Focalización:** puesto que el usuario dice explícitamente cuáles son sus preferencias, es fácil proporcionar la información centrada en sus intereses.
- **Personalización:** el usuario puede especificar que, antes de que los datos y sus propiedades se entreguen, se apliquen ciertos requerimientos. Por ejemplo, formato de los datos, prioridad, palabras clave, etc.
- **Actualidad:** los datos se pueden diseminar a medida que están disponibles. El proveedor de información puede invalidar los datos obsoletos.

- **Adaptación (*tailoring*):** el proveedor también puede decidir qué información ve el receptor y cuál no.
- **Reducción del tráfico:** puesto que el sistema disemina la información a quien está interesado en recibirla, se reduce mucho el tráfico en la red. Intentar localizar la información puede provocar mucho tráfico. Además, si se utiliza una infraestructura de transporte apropiada, aún se puede reducir más la ocupación de la red.

Las arquitecturas publicación-suscripción están pensadas para proporcionar tres tipos de servicios: coordinación de procesos, replicación de contenidos e informar a personas.

**Algunos de los campos en los que se utilizan aplicaciones publicación-suscripción son los siguientes:**

- Grupos de noticias y listas de distribución de correo. Los mensajes Usenet y las listas de distribución de correo se pueden considerar como sistemas de publicación-suscripción un poco primitivos. Por ejemplo, los mensajes Usenet diseminan artículos por todo Internet. Un servidor de mensajes se suscribe a otros servidores de mensajes y recibe los mensajes de los grupos a los cuales está suscrito. Cuando en un grupo se genera un nuevo artículo, el servidor en el que se ha generado el artículo se encarga de que este artículo se disemine hacia otros servidores.
- Bolsa y noticias: los sistemas que informan sobre la evolución de las acciones en la bolsa o las agencias de noticias son otro ejemplo de sistemas publicación-suscripción. En estos sistemas, los usuarios especifican unos intereses y el sistema debe garantizar que los usuarios dispongan en todo momento de la información tan actualizada como sea posible.
- Sistemas de información de tráfico. Como en las aplicaciones para la bolsa y las noticias, es necesario que la información se envíe la mayoría de las veces en tiempo real. La información también se distribuye por medio de ordenadores o dispositivos móviles.
- Distribución de *software*. Muchos sistemas requieren que el *software* se actualice frecuentemente. Por ejemplo, es necesario que el *software* de los bancos de inversiones, por necesidades de seguridad, se actualice frecuentemente y extensivamente. Utilizando una aplicación publicación-suscripción se consigue que el sistema esté funcionando continuamente y actualizado a la última versión sin problemas de seguridad para las actualizaciones.
- Servicios de alerta, monitorizaciones, vigilancia y control.

Algunos ejemplos de aplicaciones publicación-suscripción son Castanet, PointCast, BackWeb, WebCasting, WebCanal e Intermind.

### 3.3.1. Sistemas distribuidos basados en eventos\*

\* En inglés: *event-based systems*

Los sistemas distribuidos basados en eventos consiguen reducir el acoplamiento entre los diferentes componentes que forman un sistema a partir de eliminar la necesidad de saber la identidad de la interfaz con la cual se tienen que conectar. En lugar de invocar otro componente directamente, un componente puede anunciar (difundir) uno o más eventos. Otros componentes del sistema pueden registrar que están interesados en este tipo de eventos y, cuando un evento se anuncia, el sistema mismo invoca todos los componentes interesados que están registrados.

Este tipo de tecnología ha estado muy desarrollada en un ámbito de redes de área local. En los últimos años también se ha convertido en una tecnología muy apropiada para los sistemas a escala Internet. En este apartado nos centraremos en comentar los aspectos más interesantes de los sistemas distribuidos basados en eventos a escala Internet.

A escala Internet, además de tener componentes poco acoplados, nos encontraremos con que los componentes que forman el sistema pueden ser muy heterogéneos. Una arquitectura que se base en la generación, la observación y la notificación de eventos es muy apropiada.

El uso de eventos permite que un objeto pueda reaccionar a cambios que han ocurrido en otro objeto. La notificación de eventos es asíncrona y determinada por los receptores (deben mostrar interés en recibir un determinado tipo de evento). Los eventos y las notificaciones se pueden usar en una amplia variedad de aplicaciones diferentes. Por ejemplo, para comunicar que se ha añadido una figura a un dibujo, que se ha hecho una modificación a un documento, que una persona entra o sale de un espacio virtual, o que un dispositivo está en una nueva ubicación.

De todo esto extraemos que los sistemas distribuidos basados en eventos tienen dos características principales:

- **Son heterogéneos.** Utilizando la notificación de eventos para comunicar objetos distribuidos, conseguimos que componentes del sistema distribuido que no están diseñados para que interoperen trabajen conjuntamente. Lo único que es necesario es que los objetos que generan eventos publiquen los tipos de eventos que ofrecen, y que los otros objetos se suscriban a eventos y proporcionen una interfaz para recibir las notificaciones.
- **Son asíncronos.** Los objetos que generan los eventos los envían asíncronamente a todos los objetos que se han suscrito. Esto ahorra a los objetos que publican eventos tener que sincronizarse con los suscriptores.

Los sistemas diseñados siguiendo los principios de las arquitecturas basadas en eventos son muy apropiados para entornos distribuidos sin autoridad central; para construir sistemas orientados a componentes; para dar apoyo a aplicaciones que tienen que monitorizar o reaccionar a cambios en el entorno, en los intereses para alguna información o en el estado de procesos.

Muchos de los sistemas distribuidos basados en eventos usan el paradigma publicación-suscripción para diseminar los eventos. Aunque los sistemas de distribución basados en eventos usen el paradigma publicación-suscripción, son dos enfoques muy distintos a la hora de construir sistemas distribuidos. Las diferencias más significativas son:

- a) El propósito de los sistemas publicación-suscripción es la distribución de datos en el momento apropiado, mientras que los sistemas distribuidos basados en eventos se centran en la notificación de eventos.

b) Los roles de los participantes son muy diferentes. En los sistemas publicación-suscripción los productores y los consumidores están claramente diferenciados, mientras que en los sistemas basados en eventos todo el mundo puede producir y consumir eventos.

c) El número y la frecuencia de información que se disemina en los sistemas publicación-suscripción estará limitado por las ratios de transmisión de los contenidos, y esto hará que sea moderada. En cambio, los sistemas distribuidos basados en eventos pueden llegar a ratios de eventos muy elevadas.

d) El último elemento diferenciador que mencionaremos es que el tamaño de los contenidos transmitidos en los sistemas de publicación-suscripción puede llegar a ser grande (ya que son orientados a la información), y en los sistemas de eventos uno de los objetivos es que los eventos sean lo más pequeños posibles.

Los sistemas basados en eventos facilitan la extensibilidad, la reusabilidad y la evolución del sistema. La **extensibilidad** es dada por la facilidad de añadir un nuevo componente que escuche eventos. La **reusabilidad**, gracias a la potenciación de una interfaz general de eventos y un mecanismo de integración. La **evolución** se consigue por el hecho de que se pueden sustituir componentes sin que afecten a la interfaz de otros componentes.

Como en todos los modelos de arquitecturas distribuidas que vemos en este módulo, un aspecto clave es conseguir escalabilidad. Un aspecto muy relacionado con la escalabilidad en los sistemas distribuidos a escala Internet basados en eventos es la expresividad del mecanismo de selección de eventos. Por expresividad queremos decir la capacidad del servicio de notificación de eventos de proporcionar un modelo potente que permita capturar información sobre los eventos, expresar filtros y patrones de intereses de notificación y usar este modelo como base para optimizar la entrega de notificaciones.

#### Ejemplos de sistemas distribuidos...

... basados en eventos: OMG CORBA Event Service, TIB/Rendezvous de TIBCO, JEDI (Java Event-based Distribution Infrastructure), TINA Notification Service, SIENA.

### 3.4. Código móvil

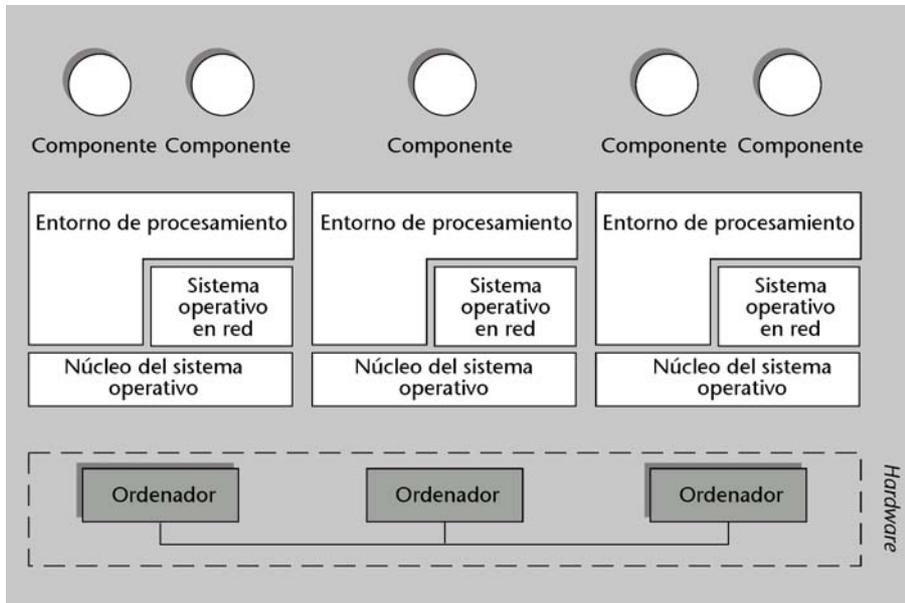
Los paradigmas de código móvil, por su parte, pretenden usar la movilidad para cambiar dinámicamente la distancia entre el procesamiento y la fuente de los datos o la destinación de los resultados. De esta manera, cambiando de ubicación, un componente puede mejorar la proximidad y la calidad de las interacciones, reducir el coste de las interacciones y, así, mejorar la eficiencia y la percepción del usuario sobre el rendimiento.

#### Máquina virtual

La tecnología de código móvil incluye lenguajes de programación y sus plataformas de ejecución. Es necesario que el código se ejecute controladamente en estas

plataformas o entornos, tanto para satisfacer las necesidades de seguridad como para estar seguro de que se podrá ejecutar. Esto es lo que proporciona la máquina virtual.

Figura 17



La máquina virtual

Los lenguajes de *scripting* son los ejemplos más comunes de uso de máquinas virtuales (por ejemplo, lenguajes de propósito general como Perl o lenguajes orientados a tareas como PostScript). Los navegadores web han contribuido a popularizar las máquinas virtuales al introducir la JVM (*Java virtual machine*) como parte de sus funcionalidades. De esta manera, cualquier navegador se puede conectar a una página web y, aparte de bajarse texto, bajar un programa (miniaplicación\*) que se ejecutará localmente en el ordenador cliente.

\* En inglés: *applet*.

## Paradigmas de código móvil

### a) Evaluación remota

#### Ejemplo

Pepe quiere preparar un plato de canalones. Dispone de la receta, pero en casa no tiene ni los ingredientes ni el horno. Sabe que su amiga Rosa tiene tanto el horno como los ingredientes en su casa, pero ella no sabe cómo se hacen los canalones. Con el objetivo de hacer los canalones, Pepe llama a Rosa y le dicta la receta por teléfono. Rosa hace los canalones siguiendo la receta de Pepe, y se los lleva.

En este paradigma, un cliente tiene el conocimiento necesario para realizar un servicio, pero no dispone de los recursos (potencia de cálculo, datos, etc.) necesarios, que se encuentran en un ordenador remoto. Por este motivo, el cliente envía el conocimiento al servidor ubicado en un ordenador remoto. Éste ejecuta el código con los recursos que tiene allí. Los resultados de la ejecución se devuelven al cliente. Este paradigma de evaluación remota presupone que el código que se proporciona se ejecutará en un entorno protegido, de manera que no impacte a otros clientes del mismo servidor aparte del impacto que

pueda significar el hecho de tener que compartir recursos. Por este motivo, es muy importante que el servidor pueda confiar en los clientes.

Algunos ejemplos muy conocidos de evaluación remota son rsh de Unix, que permite ejecutar archivos de comandos (*scripts*) en un ordenador remoto. Otro ejemplo es la interacción entre un procesador de textos y una impresora PostScript. En este caso, la impresora es el recurso y el código es el fichero PostScript. Un intérprete de PostScript situado en la impresora ejecuta el código.

Aunque a simple vista pueda parecer que la evaluación remota sea un caso particular de cliente-servidor, la diferencia es significativa. En el paradigma cliente-servidor, un servidor pone a disposición de los clientes un conjunto limitado de funcionalidades que éstos pueden invocar. En el caso de la evaluación remota, el ordenador que ejecuta el código remoto ofrece un servicio programable con un lenguaje de programación completo.

Otro ejemplo son gusanos (*worms* en inglés), que son unos tipos de virus informáticos en que un programa envía copias de él mismo a otros nodos.

## b) Código bajo demanda

### **Ejemplo**

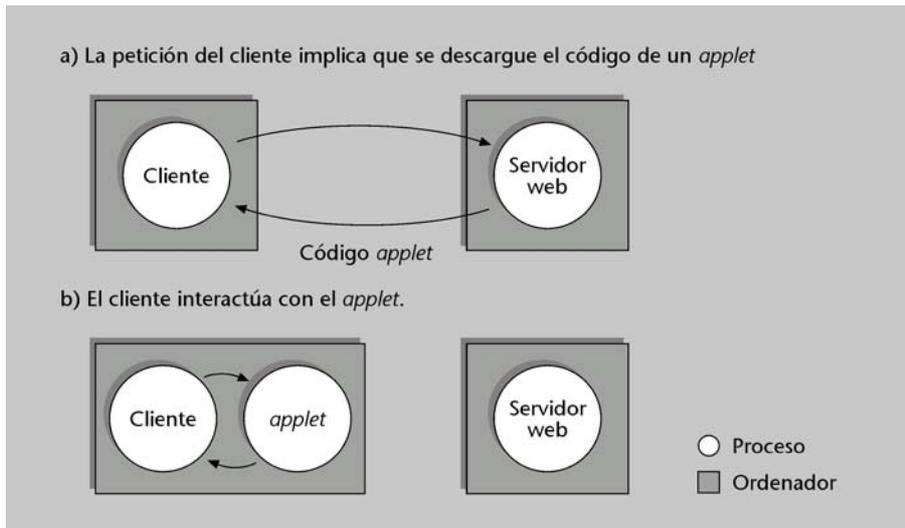
Pepe quiere preparar un plato de canalones. En casa tiene tanto los ingredientes como el horno, pero no dispone de la receta. Sabe que su amiga Rosa dispone de la receta. Pepe le llama y se la pide. Rosa le dicta la receta y Pepe prepara los canalones en su casa.

El paradigma de código bajo demanda se da cuando un cliente tiene acceso a un conjunto de recursos, pero no tiene el conocimiento necesario para procesarlos. Para poder hacer la ejecución, el cliente envía una petición a un servidor remoto para que le envíe el código necesario. Una vez recibido el código, lo ejecuta localmente.

Entre las ventajas de código bajo demanda está la posibilidad de añadir funcionalidades a un cliente sin tener que modificarlo. Además, la ejecución local puede proporcionar un buen nivel de interactividad, ya que no se sufren ni los retrasos ni la variabilidad de ancho de banda asociada a las comunicaciones en red. Esto no quiere decir que el código bajado tenga que interactuar únicamente con código local. En muchas situaciones se comunicará con otros programas extendidos en Internet. En este paradigma es muy importante que el cliente pueda confiar en los servidores de donde se baja el código.

Los ejemplos más conocidos de código móvil son las miniaplicaciones –cuando el usuario selecciona (en el navegador web que está utilizando) un enlace que hace referencia a una miniaplicación (que está almacenada en un servidor web), el código se baja al navegador. Éste ejecuta la miniaplicación localmente (podéis ver la figura 18).

Figura 18



### c) Agentes móviles

Pepe quiere preparar un plato de canalones. Tiene los ingredientes y dispone de la receta, pero en casa no tiene horno. Sabe que su amiga Rosa tiene horno en su casa. Pepe prepara los canalones y va a casa de su amiga Rosa a cocerlos.

En el paradigma de agentes móviles, una unidad de computación se mueve a un ordenador remoto, y se lleva su estado, la parte de código que necesite y, si es el caso, los datos necesarios para llevar a cabo la tarea.

Este paradigma es una combinación de los dos anteriores, ya que funciona en los dos sentidos. En comparación con los otros dos paradigmas, aporta un dinamismo mayor por el hecho de poder decidir cuándo hay que mover el código de un ordenador a otro y, así, mejorar el rendimiento global. Una aplicación puede estar a medio procesar una información en una ubicación y decidir cambiar a otra ubicación para reducir la distancia entre el código y el próximo conjunto de datos que quiere procesar.

Será interesante utilizar este tipo de aplicaciones cuando el volumen de datos que hay que procesar sea muy grande y estos datos estén en ubicaciones distintas. En estas situaciones, resulta más eficiente mover el código que procesa los datos que llevar todos los datos a un lugar –con el coste de comunicación que esto representa.

Como en los casos anteriores, la seguridad es un aspecto importante. Un agente móvil accederá a datos locales del ordenador en el que se ejecuta y, por lo tanto, es necesario que el entorno confíe en el agente. Por otra parte, también es necesario que el agente se proteja contra funcionamientos parciales o malfuncionamientos del entorno en el cual se ejecuta con el fin de no dar resultados incorrectos; o que una aplicación del entorno en el que se ejecuta el agente no pueda obtener datos del agente a los cuales no tiene derecho de acceso.

## 4. Aplicaciones de los sistemas distribuidos

Hasta ahora hemos estudiado los sistemas distribuidos desde un punto de vista arquitectónico que nos ha permitido conocer los componentes del sistema distribuido y las formas que tienen de organizarse e interrelacionarse. En este apartado veremos los sistemas distribuidos desde un punto de vista funcional que nos dará una idea de las aplicaciones que tienen este tipo de sistemas.

### 4.1. Sistemas computacionales distribuidos

Son sistemas de computación de alto rendimiento. Están formados por conjuntos de computadores interconectados mediante una red que ofrecen funcionalidades de supercomputación, tales como la computación paralela. Diferenciamos principalmente tres tipos de sistemas computacionales distribuidos:

#### 4.1.1. *Clusters*

Están formados por colecciones de computadores de similares características interconectados mediante una red de área local. Los computadores hacen uso de un mismo sistema operativo y un *middleware* que se encarga de abstraer y virtualizar los diferentes computadores del sistema dando la visión al usuario de un sistema operativo único. Los *clusters* son sistemas dedicados a la supercomputación. El sistema operativo de un *cluster* es estándar y, por lo tanto, es el *middleware* quien provee de librerías que permiten la computación paralela.

Uno de los problemas más habituales en los *clusters* es el de la gestión de los procesos que hay que ejecutar. Las soluciones más usadas son las colas de procesos gestionadas por un nodo denominado *master*. Como alternativa, el sistema MOSIX propuso una solución simétrica en la cual no había una jerarquía maestro-esclavo, como en las soluciones propuestas hasta entonces. MOSIX ofrecía una visión del sistema no sólo como un único sistema operativo, sino como una única máquina (Single System Image, en inglés SSI). Los sistemas SSI permiten la migración de tareas a otros nodos de manera transparente y preemptiva. La migración permite a un usuario iniciar una aplicación en cualquier nodo (conocido como nodo hogar, *home node*, en inglés) y, de modo transparente, ésta puede ser migrada a otro nodo para hacer un uso más eficiente de los recursos.

En el año 2002 el proyecto MOSIX pasó a ser un proyecto de software propietario, lo cual lo condenó a la desaparición. A pesar de ello, la investigación

hecha hasta el momento continuó con el proyecto openMosix hasta el año 2008, en el que el proyecto se daba por finalizado dada la disminución de la demanda de sistemas SSI a causa del abaratamiento de los sistemas computacionales multiprocesador.

#### 4.1.2. *Grid*

Mientras que los *clusters* están orientados a dar servicios computacionales de uso local o con una función concreta, los sistemas Grid tienen como objetivo la “meta-computación”, es decir, capacidades computacionales a escala Internet. No se pueden hacer asunciones sobre el tipo de hardware, sistemas operativos, interconexiones de red, dominios administrativos, políticas de seguridad, de este sistema.

Cuando se habla de *grid* se hace referencia a una infraestructura que comporta el uso integrado y colaborativo de ordenadores, redes, bases de datos e instrumentos científicos que son propiedad y están gestionados por diferentes organizaciones. Las aplicaciones *grid* a menudo trabajan con grandes cantidades de datos y/o recursos computacionales que requieren de una compartición segura de recursos atravesando diferentes límites organizativos o dominios de administración. Por su parte, idealmente el usuario tiene una visión del *grid* como si fuera un único sistema informático, ya que éste le proporciona un acceso uniforme a los recursos.

En el momento de escribir este módulo, el concepto de *grid* todavía es un concepto abierto. Todavía tiene que pasar un poco más tiempo y se verá cómo evoluciona su implantación, tanto en usos científicos como comerciales y domésticos. Sin embargo, las definiciones siguientes pueden ayudar a entender mejor qué se quiere decir cuando se habla de *grid*:

a) Plaszczak/Wellner definen tecnología *grid* como “the technology that enables resource virtualization, on-demand provisioning, and service (resource) sharing between organizations”.

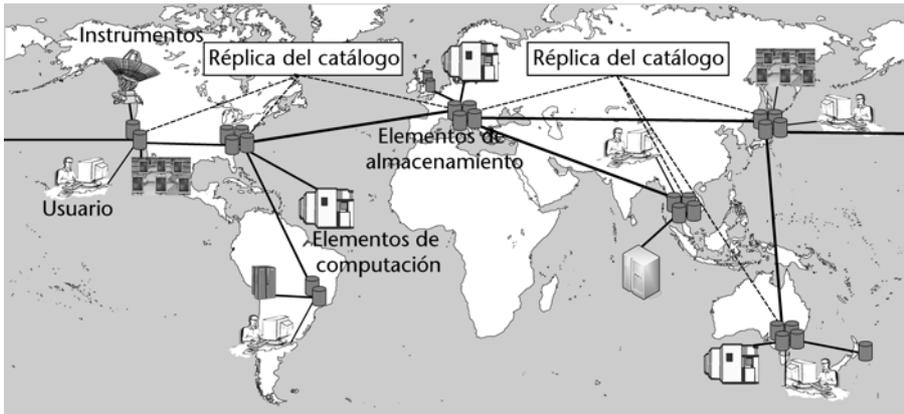
b) IBM define Grid Computing como “the ability, using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across 'multiple' administrative domains based on their (resources) availability, capacity, performance, cost and users' quality-of-service requirements”.

“IBM Solutions Grid for Business Partners: Helping IBM Business Partners to Grid-enable applications for the next phase of e-business on demand”  
([http://www-304.ibm.com/jct09002c/isv/marketing/emerging/grid\\_wp.pdf](http://www-304.ibm.com/jct09002c/isv/marketing/emerging/grid_wp.pdf)).

c) Buyya define *Grid* como “a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements”.

A Gentle Introduction to Grid Computing and Technologies (<http://www.buyya.com/papers/GridIntro-CSI2005.pdf>).

Figura 19

Ejemplo de una infraestructura *grid*

La figura 19 muestra el ejemplo de un *grid* donde hay un instrumento que genera datos y unos nodos del *grid* procesan estos datos. Para hacerlo, se dividen los datos en diferentes trozos y se envían a los diferentes centros. Estos datos se combinan con datos procedentes de otras fuentes y que están repartidos en diferentes ubicaciones. Otros nodos piden partes de estos datos para procesarlos y obtener resultados. Una vez procesados, se podrían enviar a un centro de visualización o al ordenador de un usuario para que examine los resultados. Diferentes entidades o instituciones que deciden colaborar para conseguir un objetivo determinado aportan los nodos que forman el *grid*. Esta colaboración puede ser consecuencia de unos acuerdos de colaboración entre los participantes o a cambio de compensaciones económicas.

Se empieza a hablar de *grids* a partir de la segunda mitad de los noventa. Como en el caso de igual a igual, el *grid* es una consecuencia del aumento sustancial en el rendimiento de los ordenadores personales y de las redes que ha habido en los últimos diez o quince años. Por otra parte, gracias a la combinación entre el abaratamiento de los ordenadores personales y su aumento de potencia, proliferaron sistemas de altas prestaciones a bajo coste que permitieron a muchos colectivos disponer de suficiente potencia de cómputo para solucionar problemas que requieren un uso intensivo de recursos sin tener que disponer de superordenadores. En particular, el mundo científico se ha beneficiado de esta potencia de cómputo para hacer simulaciones y experimentos mucho más exhaustivos y para los cuales antes había que disponer de grandes superordenadores.

Las redes más rápidas han permitido compartir datos de los instrumentos y resultados de los experimentos con colaboradores de todo el mundo casi instantáneamente. En este contexto, los *grids* nacen como un paso más en este esfuerzo de colaboración y compartición. El reto es crear una infraestructura computacional aprovechando los recursos (ordenadores, almacenes de datos, instrumentos científicos, bases de datos, etc.) que pueden aportar las diferentes instituciones que participan en el *grid*. De esta manera, con la combinación de todos estos recursos se pueden resolver problemas utilizando más recursos de los que se dispone individualmente. Igualmente, en el mundo comercial y social también aparece la oportunidad de utilizar el concepto de *grid* de modo que la capacidad de computación, almacenamiento y los servicios (aplicaciones y su licencia de uso) no se tenga que comprar, sino que se pueda obtener cuando hace falta un proveedor exter-

#### Grid

A esta infraestructura se la llamó *grid* haciendo una analogía con la red eléctrica (en inglés dicen *electrical power grid*) que proporciona un acceso universal, fiable, compatible y transparente a la energía eléctrica con independencia de su origen.

El modelo de *utility computing* se describe con más detalle al final del módulo "Arquitectura de aplicaciones web".

no, y se pueda pagar por el uso en lugar de por la propiedad. Eso hace que computación, almacenamiento y servicios se puedan adaptar a las necesidades: es el modelo de *utility computing*.

Los *grids* son un marco conceptual donde hay proveedores y consumidores de recursos y donde hay que definir de manera muy clara qué se comparte, quién está autorizado a compartir, y las condiciones en que ocurre la compartición. Un conjunto de individuos y/o de instituciones definidas por estas reglas de compartición forman lo que se denomina una organización virtual.

El despliegue y la gestión de aplicaciones en los *grid* es una tarea compleja. Eso ha hecho que se hayan desarrollado diferentes *middlewares grid* que proporcionan a los usuarios la capacidad de integrar la computación y el acceso uniforme a los recursos en un entorno *grid* heterogéneo. Estos *middleware* gestionan la complejidad de la distribución, administración, virtualización, planificación, etc.

### Arquitectura del *grid*

Los componentes de un *grid* se pueden organizar en capas. Cada capa se construye utilizando los servicios ofrecidos por la capa inferior, así como interactuando y cooperando con componentes de la misma capa. A continuación describimos una manera típica de organizar una arquitectura *grid* en cuatro capas:

- **Fábrica:** son los recursos, como ordenadores (pueden ser tanto *clusters*, como superordenadores o PC, y pueden ejecutar diferentes sistemas operativos), entornos de ejecución, redes, dispositivos de almacenamiento e instrumentos científicos.
- **Núcleo del *Middleware grid*:** ofrece servicios como gestión de procesos remotos, coasignación de recursos, acceso al almacenamiento, descubrimiento en registro de información, seguridad, y apoyo para dar calidad de servicio (QoS) como reserva y adquisición de recursos. Abstrae la complejidad y heterogeneidad del nivel de fábrica por el hecho de proporcionar un método consistente para acceder a los recursos distribuidos.
- **Nivel usuario del *Middleware grid*:** proporciona abstracciones y servicios de más alto nivel. Estos servicios pueden ser entornos de desarrollo de aplicaciones y herramientas de programación.
- **Aplicaciones *grid* y portales.**

#### Globus Toolkit

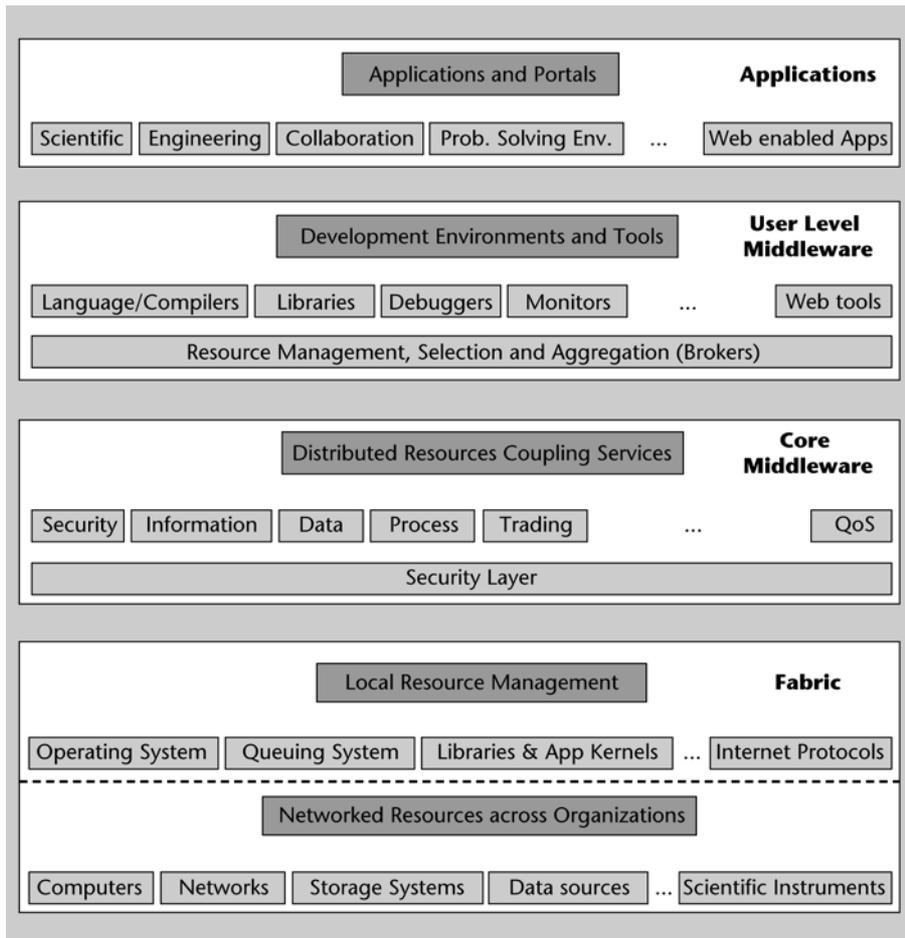
Globus Toolkit (<http://www.globusconsortium.org/>) es un conjunto de herramientas de código abierto muy popular entre la comunidad *grid* para construir infraestructuras *grid*.

#### Webs recomendadas

Algunos entornos de ejecución populares en los *grid* son:

- Condor (<http://www.cs.wisc.edu/condor/>),
- Sun Grid Engine (<http://gridengine.sunsource.net/> para la versión libre y <http://www.sun.com/software/gridware/> por la comercial)
- Torque (<http://www.clusterresources.com/pages/products/torque-resource-manager.php>)

Figura 20

Arquitectura del *grid*

Un aspecto muy importante en este tipo de sistemas es la interoperabilidad. Eso ha hecho que actualmente dentro de la comunidad *grid* se hagan muchos esfuerzos destinados a usar estándares con el fin de facilitar esta interoperabilidad.

#### Ejemplo de estándares basados en Web Services

- OGSA: pueden modelar y utilizar recursos de Web Services (<http://www.globus.org/ogsa/>)
- WSRF: usa Web services con estado ([http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsrf](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf) i <http://www.globus.org/wsrf/>)

#### Globus

Globus es un proyecto que proporciona un conjunto de herramientas de código abierto que sirven para construir infraestructuras *grid*. Globus permite la compartición de capacidad de proceso, bases de datos y otros recursos de manera segura, atravesando diferentes límites corporativos, institucionales o geográficos, sin sacrificar la autonomía local. Es decir, los usuarios pueden acceder a recursos remotos preservando el control local sobre quién y cuándo puede acceder a los recursos. La figura 21 presenta la arquitectura de

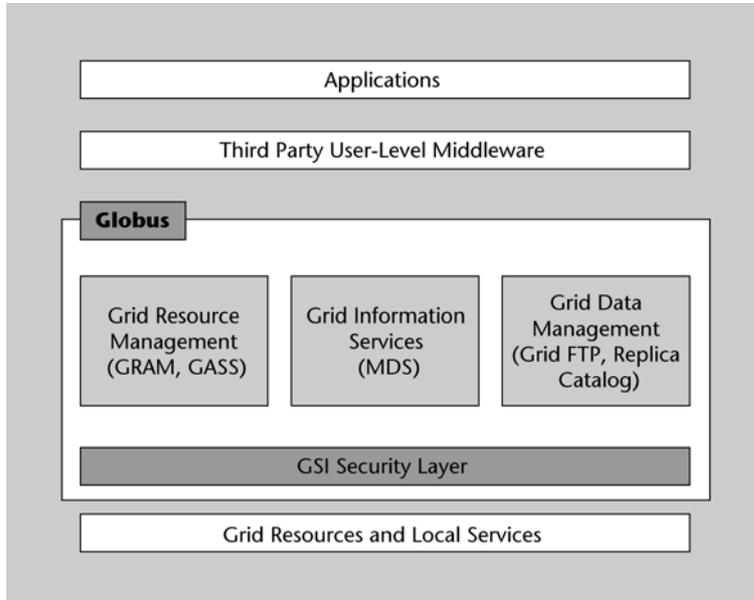
#### Webs complementarias

Para saber un poco más de *grid*, podéis ver:

- Open Grid Forum (<http://www.ogf.org/>) y
- Grid café (<http://gridcafe.web.cern.ch/gridcafe/>)

Globus. Como se puede ver, tiene tres grupos de servicios accesibles a través de un nivel de seguridad: gestión de recursos, gestión de datos y servicios de información.

Figura 21



Arquitectura de Globus

La capa de recursos y servicios locales contiene los servicios del sistema operativo, los servicios de red (p. ej. TCP/IP), y los servicios de planificación de *clusters* –que proporciona, entre otros, el envío de tareas y la consulta de colas.

La capa que contiene el núcleo de Globus está formada por:

- GSI Security Layer: proporciona los métodos para autenticar a los usuarios y hacer las comunicaciones seguras.
- Grid Resource Management: se encarga de la asignación de recursos, que comprende el envío de trabajos a ejecutar, monitorización de trabajos y la recogida de los resultados.
- Grid Information Services: proporciona propiedades dinámicas y estáticas de los nodos que están conectados al *grid*.
- Grid Data Management: proporciona utilidades y librerías para transmitir, almacenar y gestionar grandes volúmenes de datos que son necesarios para las aplicaciones que se ejecutan en el *grid*.

La capa que hay por encima contiene herramientas que integran los servicios de la capa inferior o implementa funcionalidades que ésta no tiene.

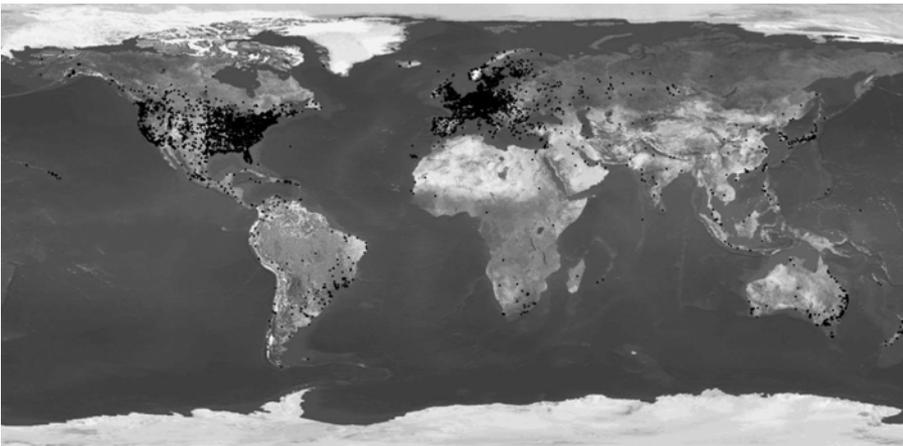
### 4.1.3. Sistemas computacionales usando recursos de los extremos de Internet

Este tipo de sistemas se caracterizan por agregar la capacidad computacional de los extremos de Internet. También se conocen como *sistemas computacionales voluntarios* o *grid de escritorio (desktop grid)*. Por extremos de Internet entendemos todos aquellos ordenadores de usuarios conectados a la red.

La tecnología más relevante en esta área es BOINC (Berkeley Open Infrastructure for Network Computing), que fue desarrollada por la Universidad de Berkeley. BOINC es un *middleware* que permite a los ordenadores de los extremos de Internet interconectarse y crear una red que agrega las capacidades de computación no usadas. La arquitectura de BOINC consiste en un sistema servidor y un conjunto de clientes que se comunican entre ellos con el fin de distribuir, procesar y devolver resultados de computación.

Para que nos hagamos una idea de la capacidad de computación de BOINC, en el año 2007 había más de 430.000 computadores activos por todo el mundo procesando en media 663 TFLOPS. Esta capacidad de computación superaba con creces la capacidad del super-computador más potente en el momento (BlueGene/L de IBM), que ofrecía 360 TFLOPS. A pesar de ello, esta comparación no es del todo real, ya que no se ha tenido en cuenta que los procesadores de BlueGene están dedicados exclusivamente a esta tarea.

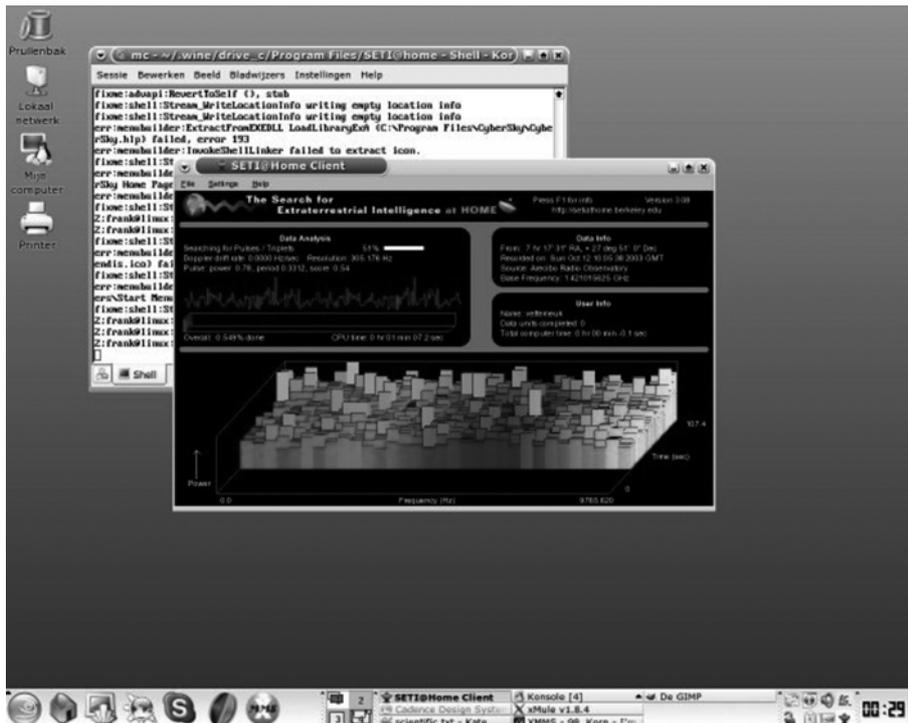
Figura 22



Distribución de los clientes de BOINC por todo el mundo

La aplicación de BOINC más representativa es Seti@Home. Un sistema de igual a igual que consiste en el procesamiento de señales de radio para buscar una prueba de inteligencia extraterrestre. Es el primer intento de computación distribuida realizado con éxito y en el que participan voluntarios de todo el mundo.

Figura 23



Aplicación seti@home

## 4.2. Sistemas de información distribuidos

Un sistema de información es un software que administra datos de algún aspecto del mundo real con una finalidad específica. Como aspecto del mundo real entendemos, por ejemplo, el proceso de cadenas genéticas, sistemas bancarios, contenidos de páginas web y, como finalidad específica, el almacenaje, la computación, la extracción de información, etc.

Los aspectos más relevantes que hay que tener en cuenta en el desarrollo de un sistema con estas características son los siguientes:

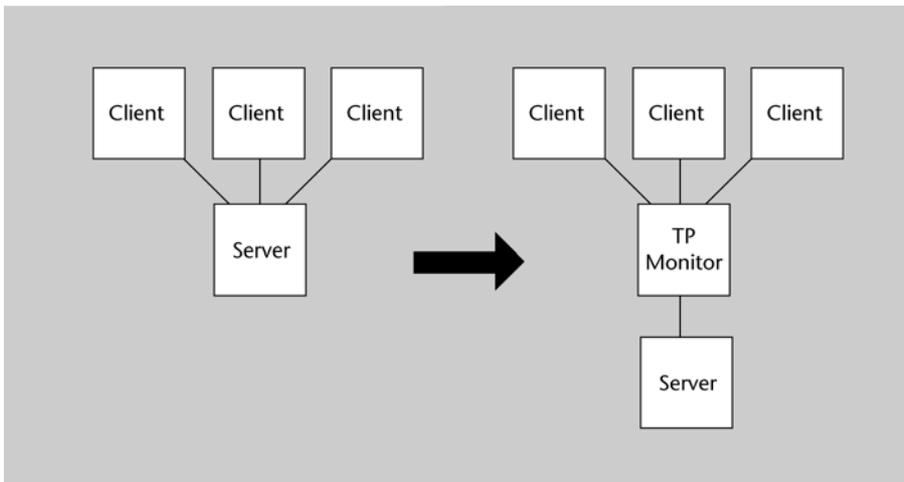
- El almacenaje de los datos: los datos deben ser almacenados durante períodos de tiempo. Además, se ha de tener en cuenta que las operaciones sobre los datos pueden ser complejas y que la cantidad de datos suele ser grande.
- Los sistemas deben ofrecer funcionalidades para permitir la interpretación de los datos y la extracción de conocimiento.

Los principales enfoques para desarrollar sistemas que administran gran cantidad de datos y soportan multitud de usuarios son los sistemas basados en transacciones.

Los **sistemas basados en transacciones** se han convertido en claves a la hora de permitir distribuir la información en sistemas de gran escala. Estos sistemas

normalmente se organizan en componentes con funcionalidades determinadas. El componente más representativo es el gestor de transacciones (*transaction processing monitor* en inglés), que se encarga de la gestión de las operaciones sobre los datos. El gestor de transacciones, aunque en la siguiente figura se muestra como un único componente, puede estar distribuido en un *cluster*, por ejemplo.

Figura 24



Evolución del modelo cliente servidor clásico al modelo basado en transacciones

Las transacciones anidadas (*nested transactions*, en inglés) permiten distribuir las transacciones en una base de datos distribuida, repartiendo la carga entre diferentes nodos y permitiendo computación paralela. La necesidad de gestores de las transacciones se hace evidente y es una de las problemáticas que deben afrontar los diseñadores de estos sistemas.

En muchos casos la integración de aplicaciones se hace a este nivel, es decir, diseñando un gestor capaz de distribuir las transacciones en diferentes bases de datos o en una distribuida. La integración de aplicaciones hace aparecer necesidades de intercomunicación e interoperabilidad entre aplicaciones. Fruto de esta necesidad nacen *middlewares* de comunicación como los conocidos RPC o RMI, que facilitan la comunicación entre aplicaciones. Uno de los inconvenientes de RPC y RMI es que se requiere que las aplicaciones que se comunican estén en funcionamiento durante la comunicación y que, además, cada una conozca cómo referirse a la otra. Estos inconvenientes han hecho que el paradigma de la publicación-suscripción también haya sido usado a la hora de integrar aplicaciones.

### 4.3. Sistemas distribuidos omnipresentes\*

\*Pervasive Distributed Systems,  
en inglés

Los sistemas distribuidos omnipresentes o sistemas computacionales ubicuos son sistemas heterogéneos formados por computadores y otros dispositivos electrónicos como sensores, equipos multimedia y otros sistemas electrónicos con funcionalidades específicas. Estos sistemas se caracterizan por su omnipre-

sencialidad a diferencia de los sistemas computacionales estudiados hasta ahora, que se caracterizan por su estabilidad.

Los avances de las redes inalámbricas, el uso de los dispositivos móviles y los sistemas integrados han permitido desarrollar aplicaciones que dan solución a problemas hasta hoy no abordables. Los campos principales de estudio han sido: la física, la medicina, las telecomunicaciones, la protección civil y la industria armamentística.

Los sistemas distribuidos omnipresentes están formados por conjuntos de nodos con capacidades computacionales y sensitivas que permiten obtener y procesar información del entorno. Los nodos del sistema están interconectados mediante redes inalámbricas que sufren en muchos casos de altos niveles de dinamismo y movilidad de sus nodos y en las que generalmente no hay un administrador.

Los nodos de un sistema ubicuo tienden a tener comportamientos autónomos y autogestionados que requieren mecanismos para detectar cambios en el entorno y poder reaccionar en consecuencia.

Los sistemas distribuidos omnipresentes se caracterizan por los requisitos siguientes:

- Detectar cambios en el entorno.
- Favorecer la formación de redes *ad-hoc*.
- Compartir información.

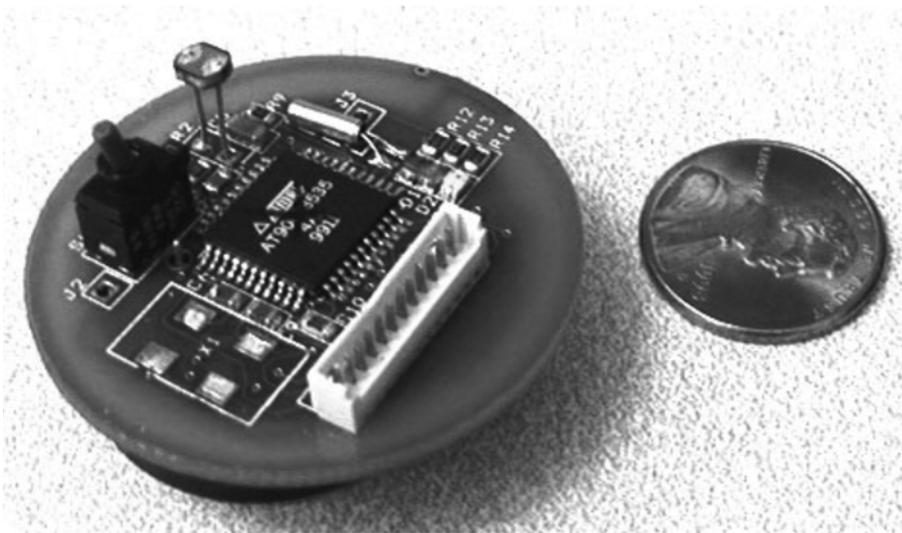
Una aplicación de los sistemas ubicuos la encontramos en el ámbito de la salud (*Electronic Health Care Systems*, en inglés). Las aplicaciones desarrolladas tienen como funcionalidad monitorizar y cuidar de un paciente con el fin de prevenir su hospitalización. Normalmente están formados por conjuntos de sensores distribuidos por el cuerpo del paciente en lo que se denomina en inglés *body-area networks* (BAN). Los sensores monitorizan a los pacientes y envían los datos mediante una conexión inalámbrica a un centro de proceso. En muchos casos, estos sistemas requieren el proceso de la información dentro de la propia red, ya que por una parte la cantidad de información obtenida por los sensores puede ser muy grande y, por otra, la capacidad de transmisión puede estar limitada entre otros por el ancho de banda o las limitaciones energéticas de los dispositivos.

Un segundo ejemplo de aplicaciones lo encontramos en la domótica y los sistemas multimedia en el hogar. Estos sistemas normalmente usan las redes de área local del hogar y están integrados por ordenadores personales, dispositivos móviles como teléfonos y PDA y sistemas de audio y vídeo como televisiones y dispositivos de juego.

Finalmente, las redes de sensores (del inglés, *sensor networks*) son redes de dispositivos electrónicos con capacidad de computación (denominados nodos), equipados con sensores, que colaboran en una tarea común. Las redes de sensores están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación sin hilos que permiten formar redes *ad hoc* sin infraestructura física preestablecida ni administración central. Su principal funcionalidad es la de adquirir y tratar datos.

Éstas se caracterizan por su facilidad de despliegue y por ser autoconfigurables. Los nodos de la red se comportan en todo momento como sistemas emisores y receptores, ofreciendo servicios de encaminamiento a otros nodos sin visión directa, así como registran datos capturados por los sensores locales de cada nodo. Cabe destacar la necesidad de la gestión eficiente de la energía, que les debe permitir una alta tasa de autonomía y ha de evitar que su carencia las haga poco operativas.

Figura 25



Mica Dot. Un nodo desarrollado por la Universidad de Berkeley.

La investigación actual en el ámbito de las redes de sensores se centra básicamente en la minimización del consumo de energía en la transmisión de datos. Como ya se ha dicho anteriormente, la transmisión de datos es costosa energéticamente y la energía es el recurso más escaso. Técnicas como la agregación de información y el proceso de la información en cada nodo de manera autónoma son muy usadas con el fin de minimizar la cantidad de información transmitida. Además, en algunos casos, el mantenimiento de una topología de red, aunque costoso, permite reducir la energía consumida en el proceso de transmisión de la información.

## Resumen

En este módulo hemos explicado algunos modelos para clasificar los sistemas distribuidos. Hemos empezado presentando los estilos arquitectónicos más usados a la hora de diseñar sistemas y aplicaciones distribuidas, y hemos visto que los estilos nos permiten describir el modo como se organizan los componentes lógicos de un sistema distribuido.

También hemos visto otro tipo de clasificación basada en la relación entre los componentes lógicos de los sistemas distribuidos. Hemos observado que sus componentes pueden organizarse de manera centralizada, donde el paradigma principal es el de cliente-servidor (hoy en día todavía sigue siendo el más usado en Internet); descentralizadas, donde hemos estudiado tanto los sistemas no estructurados, como los estructurados y los híbridos. De los sistemas descentralizados no estructurados se han explicado los mecanismos de busca, así como mecanismos para el mantenimiento de la información del sistema. Con respecto a los sistemas estructurados hemos visto cómo se organizan los nodos a fin de que la topología de la red esté estructurada. Finalmente, hemos tratado los sistemas híbridos que aprovechan algunas de las ventajas de los modelos anteriores.

Se han tratado otros enfoques como el modelo de publicación-suscripción, que es muy útil cuando los clientes necesitan recibir la información a medida que se va produciendo. En esta solución, el servidor es quien toma la iniciativa de hacer llegar los datos nuevos al cliente. También hemos estudiado las diferentes variantes del código móvil. Estas técnicas son muy útiles para reducir la distancia entre datos y código. Así, haciendo que viaje el código en lugar de los datos, conseguimos que se minimice el uso de la Red. Otra ventaja del código móvil es que si lo combinamos con cliente-servidor conseguimos aumentar las funcionalidades de un cliente o un servidor sin tener que reinstalar ni reconfigurar nada.

Para acabar, hemos visto algunos de los diferentes ámbitos de aplicación de los sistemas distribuidos. Los sistemas computacionales pueden ser *cluster* caracterizados por tener una función dedicada. Sistemas basados en aprovechar los recursos de los extremos de Internet, y como principal ejemplo la aplicación Seti@Home basada en la tecnología BOINC. Los sistemas Grid, de los cuales hemos visto que son sistemas que pretenden que diferentes organizaciones compartan recursos (ordenadores, almacenes de datos, instrumentos científicos, bases de datos, etc.). También que el mundo comercial y social se puede beneficiar de esta oportunidad que ofrece el *grid*, de manera que la capacidad de computación, almacenaje y los servicios (aplicaciones y su licencia de uso) no se deban comprar, sino que se puedan obtener cuando sea necesario un

proveedor externo, y se pueda pagar por el uso en lugar de por la propiedad. Del *grid*, hemos visto una organización típica en cuatro capas y la arquitectura de Globus, que es el conjunto de herramientas de código abierto más utilizado actualmente para construir infraestructuras *grid*.

Los sistemas de información distribuidos son también aplicaciones muy extendidas en el mundo empresarial, tanto para sistemas bancarios, como para sistemas de comercio electrónico. Hemos visto que en estos sistemas es importante garantizar la consistencia de los datos y que hacen uso de monitores de transacciones con el fin de garantizar la consistencia de la información de las bases de datos. También hemos visto que los monitores de transacciones son muy útiles a la hora de integrar diferentes aplicaciones.

También hemos tratado los sistemas distribuidos omnipresentes, de los cuales hemos destacado sus aplicaciones en medicina y domótica. Otro tipo de sistemas que se han tratado son las redes de sensores, de las cuales se hacen usos tan diferentes como la de prevención de riesgos naturales, usos militares, monitorización y control del tráfico, etc. Hemos visto que un aspecto muy importante para estas redes es la minimización del gasto energético.

Ya para acabar, al comienzo de este módulo, en el apartado 1, hemos dado una pincelada sobre algunos de los aspectos que hay que considerar cuando se diseña una aplicación distribuida: heterogeneidad, apertura, seguridad, escalabilidad, comportamiento ante fallos, concurrencia y transparencia.



## Actividades

Para la red Chord de la figura, haced lo siguiente:

- a) Mostrad las tablas de encaminamiento de los nodos N8, N12, N1 y N56 cuando entre el N12.
- b) Mostrad los saltos dados cuando en el nodo N56 queremos obtener la clave K13.
- c) Mostrad los saltos dados cuando en el nodo N56 queremos obtener la clave K37.
- d) Mostrad los saltos dados cuando en el nodo N56 queremos obtener la clave K21.

## Solucionari

a)

N8+1 12  
 N8+2 12  
 N8+4 12  
 N8+8 21  
 N8+16 32  
 N8+32 42  
 N12+1 14  
 N12+2 14  
 N12+4 21  
 N12+8 21  
 N12+16 32  
 N12+32 48  
 N1+1 8  
 N1+2 8  
 N1+4 8  
 N1+8 12  
 N1+16 21  
 N1+32 38  
 N56+1 1  
 N56+2 1  
 N56+4 1  
 N56+8 1  
 N56+16 8  
 N56+32 32

b) 56->8->12->14

c) 56->32->38

d) 56->8->21

## Glosario

**agentes móviles** *m pl* Paradigma de código móvil en el que una unidad de computación se mueve a un ordenador remoto, y se lleva su estado, la porción de código que necesite y, si es el caso, los datos necesarios para hacer la tarea.

**apertura** *f* Capacidad que tiene un sistema distribuido para ampliarse (añadir nuevos recursos y servicios que estén disponibles para los componentes que forman el sistema o aplicación).

**arquitectura multiestrato** *f* Variante de arquitectura cliente-servidor en la cual la interfaz reside en el ordenador del usuario y los servicios funcionales (capacidad de procesamiento y gestión de los datos) están en plataformas adicionales. Nosotros hemos visto arquitecturas de dos y de tres estratos.

*en* multitired architecture

**caché** *f* Almacén de objetos usados recientemente que actúa como mediador entre un cliente y un servidor.

*en* cache

**código bajo demanda** *m* Paradigma de código móvil en el que un cliente se baja de un ordenador remoto el código necesario para hacer una operación. La ejecución se lleva a cabo en el ordenador del cliente.

**código móvil** *m* Conjunto de paradigmas que usan la movilidad para cambiar dinámicamente la distancia entre el procesamiento y la fuente de los datos o el destino de los resultados.

**cliente-servidor** *m* Tipo de arquitectura distribuida formada por dos tipos de componentes: los clientes que hacen peticiones de un servicio, y los servidores que proveen este servicio.

**de igual a igual** *m* Tipo de arquitectura distribuida que se caracteriza por el hecho de que todos los nodos que forman el sistema o aplicación tienen las mismas capacidades y responsabilidades, y en la que toda la comunicación es simétrica.

*en* peer-to-peer

*sin.* entre iguales

**distributed hash table** (DHT) Véase **tabla de hash distribuida**

**escalabilidad** *f* Un sistema es escalable si la adición de usuarios y recursos no provoca una pérdida de rendimiento ni un incremento de complejidad de la administración.

**evaluación remota** *f* Paradigma de código móvil en el que un cliente envía un código para ejecutar en un ordenador remoto que dispone de los recursos necesarios para realizar el servicio.

**evento** *m* Ocurrencia de algún cambio de estado en un componente que se hace visible al mundo externo.  
*en event*

**heterogeneidad** *f* Propiedad de un sistema distribuido que indica que está formado por una variedad de diferentes redes, sistemas operativos, lenguajes de programación o *hardware* del ordenador o del dispositivo.

**igual** *m* Cada uno de los nodos que forman un sistema de igual a igual.  
*en peer*

**máquina virtual** *f* Entorno seguro y fiable que permite garantizar que el código móvil se ejecutará de la manera como se espera que se ejecute.

**Napster** *m* Aplicación muy popular para el intercambio de ficheros en formato MP3 que seguía el modelo de una arquitectura de igual a igual.

**Overlay network** Véase **red superpuesta**.

**peer** *m* Véase **igual**.

**peer-to-peer** *m* Véase **de igual a igual**.

**proxy** *m* Véase **servidor intermediario**.

**publicación-suscripción** *f* Tipo de arquitectura distribuida formada por consumidores que están interesados en recibir un tipo de información; productores que envían la información activamente; y mediadores que se encargan de poner en contacto a productores y consumidores de manera transparente para ellos.

**red superpuesta** *m* Red a nivel de aplicación que forman los nodos de un sistema o aplicación de igual a igual. Esta red funciona sobre la red física que conecta los nodos.  
*en overlay network*

**servidor intermediario** *m* Servidor que acepta peticiones de clientes u otros intermediarios y genera peticiones hacia otros intermediarios o hacia el servidor destino.  
*en proxy*

**sistema distribuido** *m* Colección de ordenadores autónomos enlazados por una red de ordenadores y soportados por un *software* que hace que la colección actúe como un servicio integrado.

**sistemas distribuidos basados en eventos** *m pl* Tipo de arquitectura distribuida en el que se consigue reducir el acoplamiento entre los componentes que forman el sistema, por medio de diseminar eventos entre sus componentes.

**tabla de hash distribuida** *f* Mecanismo distribuido que permite la localización eficaz de datos en sistemas de gran escala a través de un índice descentralizado y uniformemente repartido entre todos los nodos del sistema.  
*en distributed hash table*

**topología** *f* Maneras de interrelacionarse que tienen los componentes de un sistema distribuido en función de los flujos de información que intercambian.

**transparencia** *f* Calidad que procura que ciertos aspectos del sistema estén ocultos a las aplicaciones.

## Bibliografía

Coulouris, G.; Dollimore, J.; Kindberg, T. (2005). *Distributed systems. Concepts and design. Sistemas Distribuidos. Conceptos y Diseño* (3.ª ed.). Londres: Pearson Education / Addison Wesley, 2001.

Es un libro que trata los principios y el diseño de los sistemas distribuidos, incluyendo los sistemas operativos distribuidos. La versión en inglés va por la cuarta edición, mientras que la versión castellana va por la tercera edición.

**Eng Keong Lua; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S.** (2005). "A survey and comparison of peer-to-peer overlay network schemes". *IEEE Communications Surveys & Tutorials* (vol. 7, núm. 2).

En este artículo encontraréis un resumen de cómo funcionan las redes superpuestas de igual a igual más populares, así como una comparativa entre ellas. También encontraréis más detalle de los sistemas explicados en este apartado.

**Oram, A.** (ed.) (2001). *Harnessing the power of Disruptive Technologies*. Editorial O'Reilly.

**Steinmetz, R.; Wehrle, K.** (eds.) (setiembre, 2005). "Peer-to-Peer Systems and Applications". *Lecture Notes on Computer Science* (vol. 3485). Berlín: Springer Publishing.

Este libro es una recopilación de artículos que tratan diferentes aspectos relacionados con los sistemas de igual a igual.

**Tanenbaum A.; Steen, M.** (2007). *Distributed Systems: Principles and Paradigms, 2*. E. Prentice Hall.

Este libro es una buena ayuda para programadores, desarrolladores e ingenieros con el fin de entender los principios y paradigmas básicos de los sistemas distribuidos. Relaciona los conceptos explicados con aplicaciones reales basadas en estos principios. Es la segunda edición de un libro que ha tenido mucho de éxito tanto por los aspectos que trata como por el tratamiento que hace de ellos.

#### Artículos

**Fielding, R. T.** "Software architecture styles for Network-based applications".

**Fuggetta, A.; Picco, G. P.** (1998). "Giovanni Vigna. Understanding code mobility". *IEEE transactions on software engineering*.

**Lua, E. K. y otros** (2005). "A survey and comparison of peer-to-peer overlay network schemes". *IEEE Communications Surveys&Tutorials* (vol. 7, núm. 2).

Es un artículo en el que encontraréis un resumen de cómo funcionan las DHT más populares, así como una comparativa entre ellas.

**Milojicic, D. S.; Kalogeraki, V.; Lukose, R.; Nagaraja, K.; Pruyne, J.; Richard, B.; Rollins, S.; Xu, Z.** (2002). "Peer-to-peer computing".