

CARACTERIZACIÓN DE LOS SISTEMAS DISTRIBUIDOS



- 1.1. Introducción
- 1.2. Ejemplos de sistemas distribuidos
- 1.3. Recursos compartidos y Web
- 1.4. Desafíos
- 1.5. Resumen

Un sistema distribuido es aquel en el que los componentes localizados en computadores, conectados en red, comunican y coordinan sus acciones únicamente mediante el paso de mensajes. Esta definición lleva a las siguientes características de los sistemas distribuidos: concurrencia de los componentes, carencia de un reloj global y fallos independientes de los componentes. Proporcionamos tres ejemplos de sistemas distribuidos:

- Internet.
- Una Intranet, que es una porción de Internet gestionada por una organización.
- La computación móvil y ubicua.

Compartir recursos es uno de los motivos principales para construir sistemas distribuidos. Los recursos pueden ser administrados por servidores y accedidos por clientes o pueden ser encapsulados como objetos y accedidos por otros objetos clientes. Se analiza el Web como un ejemplo de recursos compartidos y se introducen sus principales características.

Los desafíos que surgen en la construcción de sistemas distribuidos son la heterogeneidad de sus componentes, su carácter abierto, que permite que se puedan añadir o reemplazar componentes, la seguridad y la escalabilidad, que es la capacidad para funcionar bien cuando se incrementa el número de usuarios, el tratamiento de los fallos, la concurrencia de sus componentes y la transparencia.

Sistemas distribuidos

1.1 INTRODUCCION

Existen redes de computadores en cualquier parte. Una de ellas es Internet, como lo son las muchas redes de las que se compone. Las redes de teléfonos móviles, las redes corporativas, las de las empresas, los campus, las casas, redes dentro del coche, todas, tanto separadas como combinadas, comparten las características esenciales que las hacen elementos importantes para su estudio bajo el título de *sistemas distribuidos*. En este libro se pretenden explicar las características de los computadores en red que deben considerar los diseñadores e implementadores de sistemas y presentar los conceptos y técnicas fundamentales que han sido desarrolladas para ayudar en las tareas de diseño e implementación de sistemas que se basan en dichas características.

Definimos un sistema distribuido como aquel en el que los componentes hardware o software, localizados en computadores unidos mediante red, comunican y coordinan sus acciones sólo mediante paso de mensajes. Esta definición sencilla cubre el rango completo de sistemas en los que se utilizan normalmente computadores en red.

Los computadores que están conectados mediante una red pueden estar separados espacialmente por cualquier distancia. Pueden estar en continentes distintos, en el mismo edificio o en la misma habitación. Nuestra definición de sistemas distribuidos tiene las siguientes consecuencias significativas:

Concurrencia: en una red de computadores, la ejecución de programas concurrentes es la norma. Yo puedo realizar mi trabajo en mi computador, mientras tú realizas tu trabajo en la tuya, compartiendo recursos como páginas web o ficheros, cuando es necesario. La capacidad del sistema para manejar recursos compartidos se puede incrementar añadiendo más recursos (por ejemplo, computadores) a la red. Describiremos formas en las que esta capacidad extra puede ser usada de forma útil, en muchos puntos de este libro. La coordinación de programas que comparten recursos y se ejecutan de forma concurrente es también un tema importante y recurrente.

Inexistencia de reloj global: cuando los programas necesitan cooperar coordinan sus acciones mediante el intercambio de mensajes. La coordinación estrecha depende a menudo de una idea compartida del instante en el que ocurren las acciones de los programas. Pero resulta que hay límites a la precisión con lo que los computadores en una red pueden sincronizar sus relojes, no hay una única noción global del tiempo correcto. Esto es una consecuencia directa del hecho que la única comunicación se realiza enviando mensajes a través de la red. En el Capítulo 10 se describen ejemplos de estos problemas de temporización y soluciones a los mismos.

Fallos independientes: todos los sistemas informáticos pueden fallar y los diseñadores de sistemas tienen la responsabilidad de planificar las consecuencias de posibles fallos. Los sistemas distribuidos pueden fallar de nuevas formas. Los fallos en la red producen el aislamiento de los computadores conectados a él, pero eso no significa que detengan su ejecución. De hecho, los programas que se ejecutan en ellos pueden no ser capaces de detectar cuando la red ha fallado o está excesivamente lenta. De forma similar, la parada de un computador o la terminación inesperada de un programa en alguna parte del sistema (*crash*) no se da a conocer inmediatamente a lo demás componentes con los que se comunica. Cada componente del sistema puede fallar independientemente, permitiendo que los demás continúen su ejecución. Las consecuencias de esta característica de los sistemas distribuidos serán un tema recurrente a lo largo de este libro.

La motivación para construir y utilizar sistemas distribuidos tiene su origen en un deseo de compartir recursos. El término *recurso* es un poco abstracto, pero caracteriza bien el rango de cosas que pueden ser compartidas de forma útil en un sistema de computadores conectados en red. Éste se extiende desde los componentes hardware como los discos y las impresoras hasta las entidades

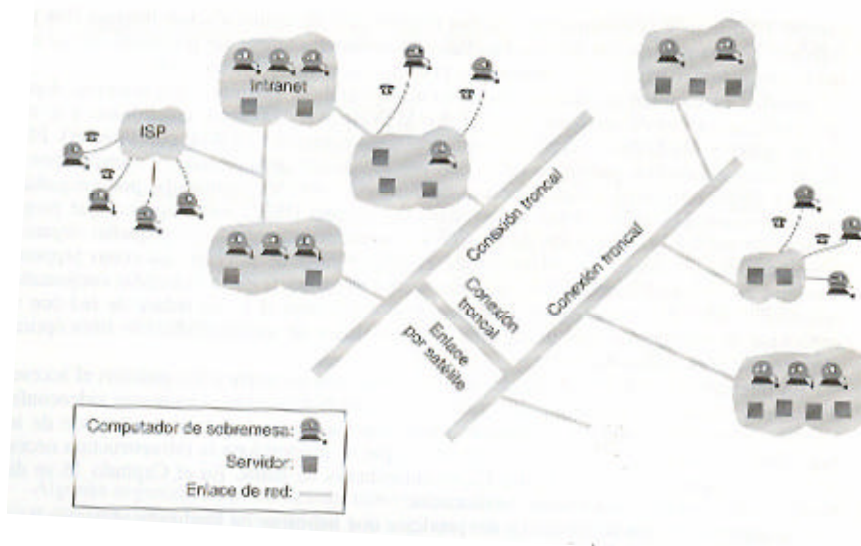


Figura 1.1. Una porción típica de Internet.

de software definidas como ficheros, bases de datos y objetos de datos de todos los tipos. Incluye la secuencia de imágenes que sale de una cámara de vídeo digital y la conexión de audio que representa una llamada de teléfono móvil.

El propósito de este capítulo es transmitir una visión clara de la naturaleza de los sistemas distribuidos y de los retos que deben ser considerados para asegurar que se alcanzan con éxito. La Sección 1.2 presenta algunos ejemplos fundamentales de sistemas distribuidos, los componentes de los que están formados y sus objetivos. La Sección 1.3 explora el diseño de sistemas de recursos compartidos en el contexto del World Wide Web. La Sección 1.4 describe los desafíos fundamentales a los que deben enfrentarse los diseñadores de sistemas distribuidos: heterogeneidad, carácter abierto, seguridad, escalabilidad, gestión de fallos, concurrencia y la necesidad de transparencia.

Nuestros ejemplos están basados en redes de computadores conocidos y utilizados ampliamente: Internet, intranets y la tecnología emergente basada en dispositivos móviles. Se han elegido para proporcionar ejemplos del amplio rango de servicios y aplicaciones que son soportados por redes de computadores y para comenzar la discusión de las cuestiones técnicas que entraña su implementación.

1.2.1. INTERNET

Internet es una vasta colección de redes de computadores de diferentes tipos interconectados. La Figura 1.1 muestra una porción típica de Internet. Programas ejecutándose en los computadores conectados a ella interactúan mediante paso de mensajes, empleando un medio común de comunicación. El diseño y la construcción de los mecanismos de comunicación Internet (los protocolos Internet) es una realización técnica fundamental, que permite que un programa que se está ejecutando en cualquier parte dirija mensajes a programas en cualquier otra parte.

Internet es también un sistema distribuido muy grande. Permite a los usuarios, donde quiera que estén, hacer uso de servicios como el World Wide Web, el correo electrónico, y la transferencia de ficheros (de hecho, a veces se confunde incorrectamente el Web con Internet). El conjunto de servicios es abierto, puede ser extendido por la adición de servidores y nuevos tipos de servicios.

La figura nos muestra una colección de intranets, subredes gestionadas por compañías y otras organizaciones. Los proveedores de servicios de Internet (ISP¹) son empresas que proporcionan enlaces de módem y otros tipos de conexión a usuarios individuales y pequeñas organizaciones, permitiéndolas el acceso a servicios desde cualquier parte de Internet, así como proporcionando servicios como correo electrónico y páginas web. Las intranets están enlazadas conjuntamente por conexiones troncales (*backbones*). Una conexión o red troncal es un enlace de red con una gran capacidad de transmisión, que puede emplear conexiones de satélite, cables de fibra óptica y otros circuitos de gran ancho de banda.

En Internet hay disponibles servicios multimedia, que permiten a los usuarios el acceso a datos de audio y vídeo, incluyendo música, radio y canales de televisión y mantener videoconferencias. La capacidad de Internet para mantener los requisitos especiales de comunicación de los datos multimedia es actualmente bastante limitada porque no proporciona la infraestructura necesaria para reservar capacidad de la red para flujos individuales de datos. En el Capítulo 15 se discute la necesidad de sistemas distribuidos multimedia.

La implementación de Internet y los servicios que mantiene ha implicado el desarrollo de soluciones prácticas para muchas cuestiones de sistemas distribuidos (incluyendo la mayoría de las definidas en la Sección 1.4). Nosotros destacaremos esas soluciones a lo largo del libro, señalando su ámbito y sus limitaciones cuando sea apropiado.

1.2.2. INTRANETS

Una intranet es una porción de Internet que es, administrada separadamente y que tiene un límite que puede ser configurado para hacer cumplir políticas de seguridad local. La Figura 1.2 muestra una intranet típica. Está compuesta de varias redes de área local (LANs) enlazadas por conexiones backbone. La configuración de red de una intranet particular es responsabilidad de la organización que la administra y puede variar ampliamente, desde una LAN en un único sitio a un conjunto de LANs conectadas perteneciendo a ramas de la empresa u otra organización en diferentes países.

Una intranet está conectada a Internet por medio de un encaminador (*router*), lo que permite a los usuarios hacer uso de servicios de otro sitio como el Web o el correo electrónico. Permite también acceder a los servicios que ella proporciona a los usuarios de otras intranets. Muchas organizaciones necesitan proteger sus propios servicios frente al uso no autorizado por parte de usuarios maliciosos de cualquier lugar. Por ejemplo, una empresa no querrá que la información segura esté accesible para los usuarios de organizaciones competidoras, y un hospital no querrá que los datos sensibles de los pacientes sean revelados. Las empresas también quieren protegerse a sí mismas de que programas nocivos, como los virus, entren y ataquen los computadores de la intranet y posiblemente destruyan datos valiosos.

El papel del *cortafuegos* es proteger una intranet impidiendo que entren o salgan mensajes no autorizados. Un cortafuegos se implementa filtrando los mensajes que entran o salen, por ejemplo de acuerdo con su origen o destino. Un cortafuegos podría permitir, por ejemplo, sólo aquellos mensajes relacionados con el correo electrónico o el acceso web para entrar o salir de la intranet que protege.

ISP es el acrónimo de Internet Service Providers.

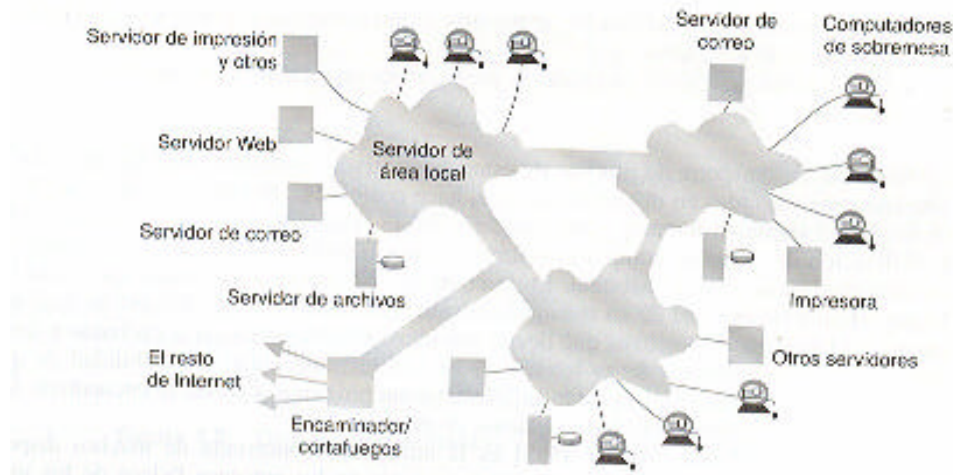


Figura 1.2. Una intranet típica.

Algunas organizaciones no desean conectar sus redes internas a Internet. Por ejemplo, la policía y otras agencias para la seguridad y la vigilancia de la ley prefieren disponer de algunas redes internas que están aisladas del mundo exterior, y el Servicio Nacional de Salud del Reino Unido ha tomado la opción de que los datos médicos sensibles relacionados con los pacientes sólo pueden ser protegidos adecuadamente manteniéndolos en una red interna separada físicamente. Algunas organizaciones militares desconectan sus redes internas de Internet en tiempos de guerra. Pero incluso dichas organizaciones desearán beneficiarse del amplio rango de aplicaciones y software de sistemas que emplean los protocolos de Internet. La solución que se adopta en tales organizaciones es realizar una intranet como se ha indicado, pero sin conexiones a Internet. Tal intranet puede prescindir de cortafuegos, o, dicho de otro modo, dispone del cortafuegos más efectivo posible, la ausencia de cualquier conexión física a Internet.

Los principales temas relacionados con el diseño de componentes para su uso en intranets son:

- Los servicios de ficheros son necesarios para permitir a los usuarios compartir datos, el diseño de éstos se discutirá en el Capítulo 8.
- Los cortafuegos tienden a impedir el acceso legítimo a servicios, cuando se precisa compartir recursos entre usuarios externos e internos, los cortafuegos deben ser complementados con el uso de mecanismos de seguridad más refinados, que se verán en el Capítulo 7.
- El coste de instalación y mantenimiento del software es una cuestión importante. Estos costes pueden ser reducidos utilizando arquitecturas de sistema como redes de computadores y clientes ligeros, descritos en el Capítulo 2.

1.2.3. COMPUTACIÓN MÓVIL Y UBICUA

Los avances tecnológicos en la miniaturización de dispositivos y en redes inalámbricas han llevado cada vez más a la integración de dispositivos de computación pequeños y portátiles en sistemas distribuidos. Estos dispositivos incluyen:

- Computadores portátiles.
- Dispositivos de mano (*handheld*), entre los que se incluyen asistentes digitales personales (PDA), teléfonos móviles, buscapersonas y videocámaras o cámaras digitales.
- Dispositivos que se pueden llevar puestos, como relojes inteligentes con funcionalidad semejante a la de los PDAs.

- Dispositivos insertados en aparatos, como lavadoras, sistemas de alta fidelidad, coches y frigoríficos.

La facilidad de transporte de muchos de estos dispositivos, junto con su capacidad para conectarse adecuadamente a redes en diferentes lugares, hace posible la *computación móvil*. Se llama computación móvil (también llamada *computación nómada* [Kleinrock 1997, www.cooltown.hp.com]) a la realización de tareas de cómputo mientras el usuario está en movimiento o visitando otros lugares distintos de su entorno habitual. En la computación móvil, los usuarios que están fuera de su *hogar intranet* (la intranet de su trabajo o su casa) disponen de la posibilidad de acceder a los recursos mediante los dispositivos que llevan con ellos. Pueden continuar accediendo a Internet; pueden acceder a los recursos de su intranet; y se está incrementando la posibilidad de que utilicen recursos, como impresoras, que están suficientemente próximos a donde se encuentren. Esto último se conoce como *computación independiente de posición*.

Computación ubicua [Weiser 1993] es la utilización concertada de muchos dispositivos de computación pequeños y baratos que están presentes en los entornos físicos de los usuarios, incluyendo la casa, la oficina y otros. El término *ubicuo* está pensado para sugerir que los pequeños dispositivos llegarán a estar tan extendidos en los objetos de cada día que apenas nos daremos cuenta de ellos. O sea, su comportamiento computacional estará ligado con su función física de forma íntima y transparente.

La presencia de computadores en cualquier parte sólo será útil cuando se puedan comunicar entre sí. Por ejemplo, podría ser conveniente para los usuarios controlar su lavadora y su equipo de alta fidelidad desde un dispositivo de *control remoto universal* en su casa. Del mismo modo, la lavadora podría avisar al usuario a través de una tarjeta inteligente o reloj cuando el lavado hubiera finalizado.

La computación ubicua y móvil se solapan, puesto que un usuario moviéndose puede beneficiarse, en principio, de los computadores que están en cualquier parte. Pero, en general, son distintas. La computación ubicua podrá beneficiar a los usuarios mientras permanecen en un entorno sencillo como su casa o un hospital. De forma similar, la computación móvil tiene ventajas si sólo se consideran computadores convencionales y dispositivos como computadores portátiles e impresoras.

La Figura 1.3 muestra a un usuario que está visitando una organización. En la figura se aprecia la intranet de la casa del usuario y la de la organización anfitriona en el lugar que está visitando dicho usuario. Ambas intranets están conectadas al resto de Internet.

El usuario tiene acceso a tres formas de conexión inalámbrica. Su computador portátil tiene un medio para conectarse a la red de área local (LAN) inalámbrica de su anfitrión. Esta red proporciona una cobertura de unos pocos cientos de metros (por ejemplo, una planta de un edificio). Se conecta con el resto de la intranet del anfitrión mediante una pasarela. El usuario dispone además de un teléfono móvil (celular), que está conectado a Internet utilizando el protocolo de acceso inalámbrico (WAP) a través de una pasarela (véase el Capítulo 3). El teléfono da acceso a páginas de información sencilla que se presenta en la pantalla del mismo. Por último, el usuario lleva una cámara digital, que puede comunicar a través de un enlace de infrarrojos cuando apunta al correspondiente dispositivo como una impresora.

Con una infraestructura adecuada del sistema, el usuario puede realizar algunas tareas sencillas en el lugar del anfitrión utilizando los dispositivos que lleva. Mientras viaja hacia el lugar, el usuario puede buscar las últimas cotizaciones en un servidor web utilizando el teléfono móvil. Durante las reuniones con sus anfitriones, el usuario puede mostrarles una fotografía reciente enviándola directamente desde la cámara digital a una impresora adecuada en la sala de la reunión. Esto sólo precisa el enlace de infrarrojos entre la cámara y la impresora. Y pueden enviar un documento

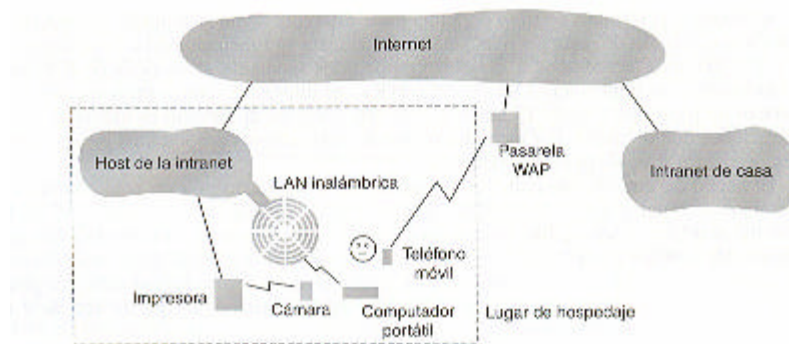


Figura 1.3. Dispositivos portátiles y de mano en un sistema distribuido.

desde un computador portátil a la misma impresora utilizando la red inalámbrica y enlaces cableados de Internet a la impresora.

La computación móvil y ubicua plantea temas significativos sobre el sistema [Milojicic y otros 1999, p. 266, Weiser 1993]. La Sección 2.2.3 presenta una arquitectura para computación móvil y esboza los temas que se plantean desde ella, incluyendo cómo realizar el descubrimiento de recursos en un entorno anfitrión, eliminando la necesidad de que los usuarios reconfiguren sus dispositivos móviles a medida que se mueven, ayudándoles a arreglárselas con la conectividad limitada cuando viajan, y proporcionándoles garantías de privacidad y seguridad a ellos y a los entornos que visitan.

1.3 RECURSOS COMPARTIDOS Y WEB

Los usuarios están tan acostumbrados a los beneficios de compartir recursos que pueden pasar por alto su significado. Normalmente compartimos recursos hardware como impresoras, recursos de datos como ficheros, y recursos con una funcionalidad más específica como máquinas de búsqueda.

Considerado desde el punto de vista de la provisión de hardware, se comparten equipos como impresoras y discos para reducir costes. Pero es mucho más significativo para los usuarios compartir recursos de alto nivel que forman parte de sus aplicaciones y su trabajo habitual y sus actividades sociales. Por ejemplo, los usuarios están preocupados con compartir datos en forma de una base de datos compartida o un conjunto de páginas web, no los discos o los procesadores sobre los que están implementados. Igualmente, los usuarios piensan en términos de recursos como una máquina de búsqueda o un conversor de monedas, sin considerar el servidor o servidores que los proporcionan.

En la práctica, los patrones de compartir recursos varían mucho en su alcance y cuan estrechamente trabajan juntos los usuarios. En un extremo, una máquina de búsqueda en el Web proporciona una función para los usuarios en todo el mundo, los usuarios no necesitan establecer contacto con los demás directamente. En el otro extremo, en un sistema de trabajo *cooperativo mantenido por computador* (CSCW, *computer-supported cooperative working*), un grupo de usuarios que colaboran directamente entre ellos comparte recursos como documentos en un grupo pequeño y cerrado. El patrón de compartir y la distribución geográfica de los usuarios particulares determina qué mecanismos debe proporcionar el sistema para coordinar sus acciones.

Utilizamos el término *servicio* para una parte diferente de un sistema de computadores que gestiona una colección de recursos relacionados y presenta su funcionalidad a los usuarios y aplicaciones. Por ejemplo, accedemos a ficheros compartidos mediante el servicio de ficheros, enviamos documentos a las impresoras a través del servicio de impresión, compramos regalos a través de un servicio de pago electrónico. El único acceso que tenemos al servicio es mediante un

conjunto de operaciones que él ofrece. Por ejemplo, un servicio de ficheros proporciona las operaciones de *lectura, escritura y borrado* en los ficheros.

El hecho de que los servicios limiten el acceso a los recursos a un conjunto bien definido de operaciones es una práctica habitual en la ingeniería de software. Pero también refleja la organización física de los sistemas distribuidos. Los recursos en un sistema distribuido están encapsulados físicamente con los computadores y sólo pueden ser accedidos desde otros computadores a través de comunicación. Para que se compartan de forma efectiva, cada recurso debe ser gestionado por un programa que ofrece una interfaz de comunicación permitiendo que se acceda y actualice el recurso de forma fiable y consistente.

El término *servidor* es probablemente familiar para la mayoría de los lectores. Se refiere a un programa en ejecución (un *proceso*) en un computador en red que acepta peticiones de programas que se están ejecutando en otros computadores para realizar un servicio y responder adecuadamente. Los procesos solicitantes son llamados *clientes*. Las peticiones se envían a través de mensajes desde los clientes al servidor y las contestaciones se envían mediante mensajes desde el servidor a los clientes. Cuando un cliente envía una petición para que se realice una operación, decimos que el cliente *invoca una operación* del servidor. Se llama *invocación remota* a una interacción completa entre un cliente y un servidor, desde el instante en el que el cliente envía su petición hasta que recibe la respuesta del servidor.

El mismo proceso puede ser tanto un cliente como un servidor, puesto que los servidores a veces invocan operaciones en otros servidores. Los términos *cliente* y *servidor* se aplican a los roles desempeñados en una única solicitud. Ambos son distintos, en muchos aspectos, los clientes son activos y los servidores pasivos, los servidores se están ejecutando continuamente, mientras que los clientes sólo lo hacen el tiempo que duran las aplicaciones de las que forman parte.

Hay que señalar que por defecto los términos *cliente* y *servidor* se refieren a *procesos* no a los computadores en las que se ejecutan, aunque en lenguaje coloquial dichos términos se refieren también a los propios computadores. Otra distinción que se discutirá en el Capítulo 5, es que en un sistema distribuido escrito en un lenguaje orientado a objetos, los recursos pueden ser encapsulados como objetos y accedidos por objeto clientes, en cuyo caso hablaremos de un *objeto cliente* que invoca un método en un *objeto servidor*.

Muchos sistemas distribuidos, aunque no todos, pueden ser construidos completamente en forma de clientes y servidores que interaccionan. El World Wide Web, el correo electrónico y las impresoras en red concuerdan con esta modelo. En el Capítulo 2 se verán alternativas a los sistemas cliente-servidor.

Un navegador (*browser*) es un ejemplo de cliente. El navegador se comunica con el servidor web para solicitarle páginas. A continuación se examina el Web con más detalle.

1.3.1. EL WORLD WIDE WEB

El World Wide Web [www.w3.org I, Berners-Lee 1991] es un sistema en evolución para publicar y acceder a recursos y servicios a través de Internet. Utilizando el software de un navegador web, fácilmente disponible como Netscape o Internet Explorer, los usuarios utilizan el Web para recuperar y ver documentos de muchas clases, para escuchar secuencias de audio y ver secuencias de vídeo, y para interaccionar con un conjunto ilimitado de servicios.

El Web comenzó su vida en el centro europeo para la investigación nuclear (CERN), Suiza, en 1989 como un vehículo para el intercambio de documentos entre una comunidad de físicos, conectados a Internet [Berners-Lee 1999]. Una característica fundamental del Web es que proporciona una estructura *hipertexto* entre los documentos que almacena, reflejando los requisitos de los usuarios para organizar su conocimiento. Esto significa que los documentos tienen *enlaces*, referencias a otros documentos y recursos, también almacenados en la red.

Es fundamental para la experiencia del usuario en el Web que cuando encuentra una imagen determinada o una parte de texto en un documento, esto estará acompañado de enlaces a

documentos relacionados y otros recursos. La estructura de los enlaces puede ser arbitrariamente compleja y el conjunto de recursos que puede ser añadido es ilimitado, la «telaraña» (web) de enlaces está por consiguiente repartida por todo el mundo (world-wide). Bush [1945] concibió las estructuras hiper-textuales hace 50 años; el desarrollo de Internet fue lo que propició la manifestación de esta idea en una escala mundial.

El Web es un sistema *abierto*: puede ser ampliado e implementado en nuevas formas sin modificar su funcionalidad existente (véase la Sección 1.4.2). Primero, su operación está basada en estándares de comunicación y en documentos estándar que están publicados libremente e implementados ampliamente. Por ejemplo, existen muchos tipos de navegador, cada uno de ellos implementados en muchos casos sobre diferentes plataformas; y existen muchas implementaciones de servidores web. Cualquier navegador conforme puede recuperar recursos de cualquier servidor conforme. Por lo tanto los usuarios pueden tener acceso a los navegadores en la mayoría de los dispositivos que utilizan, desde un PDA a computadores portátiles.

Segundo, el Web es abierto respecto a los tipos de recursos que pueden ser publicados y compartidos en él. En su forma más simple, un recurso es una página web o algún otro tipo de *contenido* que puede ser almacenado en un fichero y presentado al usuario, como ficheros de programa, de imágenes, de sonido y documentos en formato PostScript o PDF. Sí alguien inventa, por ejemplo, un nuevo formato de almacenamiento de imágenes, las imágenes en dicho formato pueden ser publicadas inmediatamente en el Web. Los usuarios necesitan un medio de ver imágenes en este nuevo formato, pero los navegadores están diseñados para acomodar la nueva funcionalidad de presentación en forma de aplicaciones *colaboradoras* y conectores (*plug-ins*).

El Web se ha desarrollado mas allá de estos recursos de datos sencillos para abarcar servicios como la compra electrónica de regalos. Ha evolucionado sin cambiar su arquitectura básica. El Web está basado en tres componentes tecnológicos de carácter estándar básicos:

- El lenguaje de etiquetado de hipertexto (HTML, *Hypertext Markup Language*) es un lenguaje para especificar el contenido y el diseño de las páginas que son mostradas por los navegadores.
- Localizadores Uniformes de Recursos (URL, *Uniform Resource Locaíor*) que identifican documentos y otros recursos almacenados como parte del Web. El Capítulo 9 examina otros identificadores web.
- Una arquitectura de sistema cliente-servidor, con reglas estándar para interacción (el protocolo de transferencia hipertexto-HTTP, *HyperText Transfer Protocol*) mediante la cual los navegadores y otros clientes obtienen documentos y otros recursos de los servidores web. La Figura 1.4 muestra algunos servidores web, y navegadores que les hacen peticiones. Una característica importante es que los usuarios puedan localizar y gestionar sus propios servidores web en cualquier parte de Internet.

A continuación se detallan estos componentes y se explica la operación de los navegadores y servidores web cuando un usuario localiza páginas web y selecciona los enlaces que contiene.

✓ **HTML.** El lenguaje de etiquetado de hipertexto [www.w3.org II] se utiliza para especificar el texto e imágenes que forman el contenido de una página web, y para especificar como serán formateados para la presentación al usuario. Una página web contiene elementos estructurados como cabeceras, párrafos, tablas e imágenes. HTML se utiliza también para especificar enlaces y qué recursos están asociados con ellos.

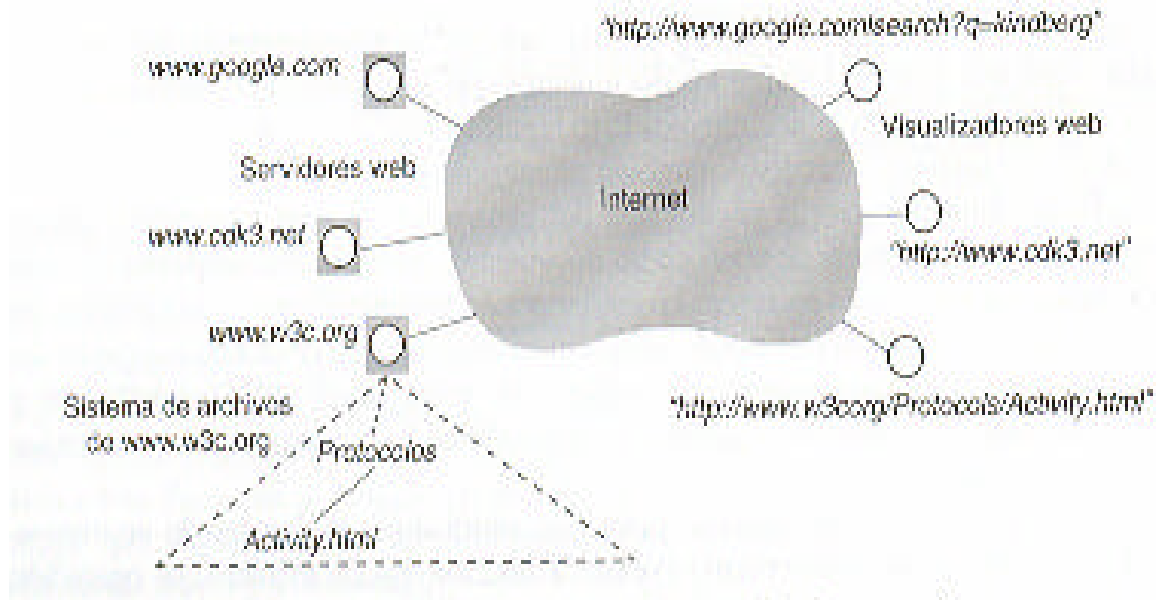


Figura 1.4. Servidores web y visualizadores web.

Los usuarios producen HTML a mano, utilizando un editor de texto estándar, o pueden utilizar a un editor *wysiwyg*² que genera HTML a partir de un diseño que pueden crear gráficamente. Un ejemplo típico de texto HTML puede ser:

```

<IMG SRC = «http://www.cdk3.net/WebExample/Images/earth.jpg»>           1
<P>                                                                       2
¿Bienvenido a la Tierra! Los visitantes pueden estar interesados también en echar un
vistazo a la                                                                3
<A HREF = «http://www.cdk3.net/WebExample/moon.html»>Luna</A>           4
<P>                                                                       5
(etcétera)                                                                    6
  
```

Este texto HTML se almacena en un fichero al que puede acceder un servidor web, supongamos que es el fichero *earth.html*. Un navegador recupera el contenido de este fichero desde un servidor web, en este caso un servidor en un computador llamado *www.cdk3.net*. El navegador lee el contenido devuelto por el servidor y lo organiza en texto formateado y en imágenes presentadas en una página web en la forma habitual. Sólo el navegador, no el servidor, interpreta el texto HTML. Pero el servidor debe informar al navegador sobre el tipo de contenido que devuelve, para distinguirlo de, por ejemplo, un documento en PostScript. El servidor puede deducir el tipo de contenido de la extensión del fichero *.html*.

Hay que señalar que las directivas HTML, conocidas como *etiquetas*, están encerradas entre ángulos como *<P>*. La línea 1 del ejemplo identifica un fichero que contiene una imagen para la presentación. Su URL es *http://www.cdk3.net/WebExample/Images/earth.jpg*. Las líneas 2 y 5 son una directiva de comienzo de un nuevo párrafo cada una. Las líneas 3 y 6 contienen texto que será mostrado por el navegador en el formato estándar de párrafo.

La línea 4 especifica un enlace en la página. Contiene la Palabra «Luna» rodeada por dos etiquetas HTML relacionadas *<A HREF...>* y **. El texto comprendido entre dichas etiquetas es lo que aparece en el enlace tal como se presenta en la página web. La mayoría de los navegadores están configurados para mostrar el texto de los enlaces subrayado, así que lo que el usuario verá en el párrafo es:

¡Bienvenido a la Tierra! Los visitantes pueden estar interesados en echar un vistazo a la Luna.

wysiwyg es el acrónimo de la frase inglesa «what you see is what you get» lo que se ve es lo que se obtiene (N. del T.).

El navegador registra la asociación entre el texto mostrado del enlace y el URL contenido en la etiqueta `<A HREF... >`, en este caso:

`http://www.cdk3.net/WebExam.ple/moon.html`

Cuando el usuario selecciona el texto, el navegador localiza el recurso identificado por el correspondiente URL y se lo presenta. En el ejemplo, el recurso es un fichero HTML que especifica una página sobre la Luna.

✓ **URLs.** El propósito de un URL [www.w3.org III] es identificar un recurso de tal forma que permita al navegador localizarlo. Los navegadores examinan los URLs con el fin de buscar los recursos correspondientes de los servidores web. A veces el usuario teclea un URL en el navegador. Habitualmente, el navegador busca el URL correspondiente cuando el usuario hace clic en un enlace o selecciona uno de sus enlaces anotados (*bookmarks*), o cuando el navegador busca un recurso insertado en una página web, como una imagen. Cada URL, en su forma global, tiene dos componentes:

esquema: localización-específica-del-esquema.

El primer componente, el *esquema*, declara qué tipo de URL es. Se precisan los URLs para especificar posiciones de una variedad de recursos y también para especificar una variedad de protocolo de comunicación para recuperarlos. Por ejemplo, `mailto:joe@anISP.net` identifica la dirección de correo de un usuario; `ftp://ftp.downloadli.com/software/aProg.exe` identifica un fichero que será recuperado utilizando el Protocolo de Transferencia de Ficheros (FTP, *File Transfer Protocol*) en lugar de HTTP, utilizado con más frecuencia. Otros ejemplos de esquemas son *nntp* (utilizado para especificar un grupo de noticias de Usenet), y *telnet* (utilizado para hacer log in en un computador).

El Web es abierto con respecto a los tipos de recursos que pueden ser utilizados para acceder, mediante los designadores de esquema de los URLs. Si alguien inventa un nuevo tipo de recurso llamémoslo *artefacto*, quizá con su propio esquema de direccionamiento para localizar artefactos y su propio protocolo para acceder a ellos, entonces la gente puede comenzar a utilizar URLs de la forma *artefacto:...* Como es lógico, los navegadores deben disponer de la capacidad para utilizar el nuevo protocolo *artefacto*, pero esto se puede conseguir añadiendo una aplicación colaboradora o un conector.

Los URLs de HTTP son los más utilizados para localizar recursos utilizando el protocolo estándar HTTP. Un URL HTTP tiene dos tareas importantes que hacer: identificar qué servidor Web mantiene el recurso, e identificar cuál de los recursos del servidor es el solicitado. En la Figura 1.4 se ven tres servidores haciendo peticiones por recursos gestionados por tres servidores web. El web en la parte superior está realizando una consulta a una máquina de búsqueda. El del medio solicita la página por defecto de otro sitio web. El de la parte inferior solicita una página que está especificada completamente, con la inclusión de un nombre de recorrido relativo para el servidor. Los ficheros para un servidor web determinado se localizan en uno o más subdirectorios del sistema de ficheros del servidor, y cada recurso es identificado por el nombre del recorrido del fichero (*path ñame*) relativo al servidor.

En general, los URLs de HTTP son de la forma:

`http://nombredelservid.or [-.puerto] [/nombredelpathdelservidor] ['Cargamentos]`

en el que los elementos entre corchetes son opcionales. Un URL de HTTP siempre comienza con `«http://»` seguido por un nombre del servidor, expresado como un nombre del Servicio de Nombres de Dominio (DNS, *Domain Name Service*) (véase la Sección 9.2). El nombre DNS del servidor está seguido opcionalmente por el nombre del «puerto» en el que el servidor escucha las solicitudes (véase el Capítulo 4). Después viene un nombre de recorrido opcional del recurso del servidor. Si éste no aparece se solicita la página web por defecto del servidor. Por último, el URL finaliza opcionalmente con un conjunto de argumentos, por ejemplo, cuando un usuario envía las

entradas en una forma como una página de consulta de una máquina de búsqueda. Considérense los URLs:

<http://www.cdk3.net/>

<http://www.w3.org/Protocols/Activity.html> <http://www.google.com/search?q=kindberg>

Pueden ser separados de la forma siguiente:

<i>Nombre del servidor de DNS</i>	<i>Ruta en el servidor</i>	<i>Argumentos</i>
www.cdk3.net	(por defecto)	(ninguno)
www.w3.org	Protocols/Activity.html	(ninguno)
www.google.com	search	q = kindberg

El primer URL indica la página por defecto proporcionada por *www.cdk3.net*. El siguiente identifica un fichero en el servidor *www.w3.org*, cuyo nombre de recorrido es *Protocols;Activity.html*. El tercer URL especifica una consulta a una máquina de búsqueda. El recorrido identifica un programa llamado *search* y la cadena de caracteres después del carácter ? codifica los argumentos para este programa, en este caso especifica la cadena de búsqueda. Se discutirán los URL que indican programas con más detalle cuando se analicen después características más avanzadas.

Los lectores pueden haber observado también la presencia de un ancla al final de un URL, un nombre precedido por un # como *¿Preferencias*, que indica un punto dentro de un documento. Las anclas no son parte de los URLs sino que están definidas como una parte de la especificación HTML. Únicamente los navegadores interpretan anclas, para colocar en pantalla páginas web a partir de un punto determinado. Los navegadores siempre recuperan páginas web completas de los servidores, no partes de ellas indicadas por anclas.

Publicación de un recurso: Mientras el Web tiene un modelo claro para recuperar un recurso a partir de su URL, el método para publicar un recurso en el Web todavía es difícil de manejar y normalmente precisa intervención humana. Para publicar un recurso en el Web, un usuario debe colocar, en primer lugar, el correspondiente fichero en un directorio al que pueda acceder el servidor web. Conocido el nombre de un servidor *S* y un nombre de recorrido *P* que el servidor puede reconocer, el usuario puede construir el URL de la forma *http://S/P*. El usuario coloca este URL en un enlace de un documento existente o distribuye el URL a otros usuarios, por ejemplo mediante correo electrónico.

Existen unas ciertas convenciones para nombres de recorrido que los servidores reconocen. Por ejemplo, un nombre de recorrido comenzando por *~juan* está por convención en un subdirectorío *public-html* del usuario Juan. De forma similar, un nombre de recorrido que termina en un nombre de directorío en lugar de un único fichero se refiere a un fichero en ese directorío llamado *index.html*.

Huang y otros [2000] proporcionan un modelo para insertar contenido en el Web con la mínima intervención humana. Esto es particularmente relevante allí donde los usuarios necesitan extraer contenido de una variedad de dispositivos, como cámaras, para su publicación en páginas web.

<> **HTTP.** El protocolo de transferencia hipertexto [www.w3.org IV] define las formas en las que

los navegadores y otros tipos de clientes interaccionan con los servidores web. En el Capítulo 4 se verá HTTP con más detalle, pero aquí se esbozan sus principales características (restringiendo la discusión a la recuperación de recursos en ficheros):

Interacciones petición-respuesta: HTTP es un protocolo de petición-respuesta. El cliente envía un mensaje de petición al servidor que contiene el URL del recurso solicitado. (El servidor sólo precisa la parte del URL que sigue al propio nombre DNS del servidor.) El servidor localiza el nombre de recorrido y, si existe, devuelve el contenido del fichero en un mensaje de respuesta al cliente. En caso contrario, devuelve un mensaje de error.

Tipos de contenido: los navegadores no son necesariamente capaces de manejar o hacer buen uso de cualquier tipo de contenido. Cuando un navegador hace una petición, incluye una lista de los tipos de contenido que prefiere, por ejemplo, en principio puede ser capaz de sacar en pantalla imágenes en formato *GIF* pero no en *JPEG*. El servidor puede ser capaz de tener esto en cuenta cuando devuelve el contenido al navegador. El servidor incluye el tipo de contenido en el mensaje de respuesta de forma que el navegador sabrá cómo procesarlo. Las cadenas de caracteres que indican el tipo de contenido se llaman tipos MIME, y están estandarizados en el RFC 1521 [Borenstein y Freed 1993]. Por ejemplo, si el contenido es de tipo *text/html* entonces el navegador interpretará el texto como HTML y lo mostrará en pantalla; si el contenido es de tipo *image/GIF* el navegador lo tratará como una imagen en formato *GIF*, si el contenido es de tipo *application/zip* entonces los datos están comprimidos en formato *zip* y el navegador lanzará una aplicación externa para descomprimirlos. El conjunto de acciones que un navegador tomará para un tipo de contenido dado es configurable, y los lectores deben preocuparse de comprobar estos defectos en sus propios navegadores.

Un recurso por solicitud: en la versión 1.0 de HTTP (que es la versión más utilizada en el momento en que se escribe esto), el cliente solicita un recurso por cada petición HTTP. Si una página web contiene nueve imágenes, por ejemplo, el navegador realizará un total de diez peticiones separadas para obtener el contenido completo de la página. Los navegadores normalmente hacen varias peticiones concurrentemente, para reducir el retardo total para el usuario.

Control de acceso simple: por defecto, cualquier usuario con una conexión de red a un servidor web puede acceder a cualquiera de los recursos publicados. Si los usuarios desean restringir el acceso a un recurso, pueden configurar el servidor para plantear un desafío a cualquier usuario que lo pida. Los usuarios correspondientes deben probar entonces que tienen derecho para acceder al recurso, por ejemplo tecleando una contraseña (*password*).

✓ **Características más avanzadas, servicios y páginas dinámicas.** Hasta ahora hemos descrito cómo los usuarios pueden publicar páginas Web y otros contenidos almacenados en ficheros en el Web. El contenido puede cambiar en el tiempo, pero es lo mismo para cualquiera. Sin embargo, mucha de la experiencia de los usuarios del Web es la de los servicios con los que el usuario puede interactuar. Por ejemplo, cuando se compra algo en una tienda electrónica, el usuario rellena con frecuencia *un formulario web* para proporcionar sus detalles personales o para especificar exactamente lo que se desea comprar. Un formulario web es una página que contiene instrucciones para el usuario y elementos para la introducción de datos como campos de texto y cajas de comprobación. Cuando un usuario envía el formulario (normalmente pulsando un botón o una tecla de *retorno* (*return*), el navegador envía una petición HTTP a un servidor web, que contiene los valores enviados por el usuario.

Puesto que el resultado de la petición depende de los datos introducidos por el usuario, el servidor debe *procesar* dichos datos. Por tanto, el URL o su componente inicial representa un *programa* en el servidor, no un archivo. Si los datos introducidos por el usuario son pocos, entonces son enviados como el componente final del URL, siguiendo a un carácter ? (en el resto de los casos son enviados como datos adicionales en la petición). Por ejemplo, una solicitud que contenga el URL siguiente llama a un programa llamado *search* en www.google.com y especifica una consulta para *Kindberg*: <http://www.google.com/search?q=kindberg>.

El programa «*search*» produce como resultado texto HTML, y el usuario verá una lista de páginas que contienen la palabra *kindberg*. (El lector puede realizar una consulta en su buscador favorito y darse cuenta del URP que el navegador saca en pantalla cuando se devuelve el resultado.) El servidor devuelve el texto HTML que genera el programa del mismo modo que si hubiera sido obtenido de un fichero. Dicho de otro modo, la diferencia entre el contenido estático localizado en un fichero y el contenido que es generado dinámicamente es transparente para el navegador.

Un programa que se ejecuta en los servidores web para generar contenido para sus clientes se suele llamar, a menudo, programa de Interfaz de Pasarela Común (COI, *Common Gateway Interfa-ce*). Un programa CGI puede tener una funcionalidad específica de la aplicación, ya que puede analizar los argumentos que el cliente le proporciona y producir un resultado del tipo requerido (normalmente texto HTML). El programa consultará o modificará una base de datos cuando procesa la solicitud.

Código descargado: Un programa CGI se ejecuta en el servidor. A veces los diseñadores de servicios Web precisan algún código relacionado con el servicio para ejecutar en el navegador, en el computador del usuario. Por ejemplo, código escrito en Javascript [www.netscape.com] se descarga a menudo con un formulario web para proporcionar una interacción con el usuario de mejor calidad que la proporcionada por los artefactos estándar de HTML. Una página mejorada con Javascript puede dar al usuario información inmediata sobre entradas inválidas (en lugar de forzar al usuario a comprobar los valores en el servidor, lo que precisaría mucho más tiempo). Javascript puede ser utilizado también para modificar partes del contenido de una página web sin que sea preciso traer una nueva versión completa de la página y reformatearla.

Javascript tiene una funcionalidad bastante limitada. Como contraste, un *applet* es una pequeña aplicación que descarga automáticamente el navegador y se ejecuta cuando se descarga la página correspondiente. Los applets pueden acceder a la red y proporcionar interfaces de usuario específicas, utilizando las posibilidades del lenguaje, Java [java.sun.com, Flanagan 1997]. Por ejemplo, aplicaciones de tipo «chat» están implementadas, a veces, como applets que corren en los navegadores de los usuarios, junto con un programa de servidor. Los applets envían el texto al que lo distribuye a todos los applets para su presentación al usuario. Se discutirán los applets con más detalle en la Sección 2.2.3.

✓ **Discusión sobre el Web.** El extraordinario éxito del Web se basa en la facilidad con la que pueden publicarse recursos, una estructura de hipertexto apropiada para la organización de muchos tipos de información, y la extensibilidad arquitectónica del sistema. Los estándares en que se basa su arquitectura son simples y fueron difundidos ampliamente desde un principio. Esto ha posibilitado que se integraran muchos tipos nuevos de recursos y servicios.

El éxito del Web contradice algunos principios de diseño. Primero, su modelo de hipertexto es deficiente en algunos aspectos. Si se borra o mueve algún recurso, ocurre que los llamados enlaces *descolgados* a este recurso permanecen, causando cierta frustración a los usuarios. También aparece el problema habitual de los usuarios *perdidos en el hiperespacio*. Los usuarios a menudo se encuentran a sí mismos siguiendo confusamente un montón de vínculos, que apuntan a páginas de una colección dispar de enlaces, en algunos casos de dudosa fiabilidad. Los motores de búsqueda son un complemento útil para seguir enlaces con el fin de buscar información en el Web, pero son claramente imperfectos a la hora de degenerar lo que quiere concretamente el usuario. Una aproximación a este problema, ejemplificada en el Marco de Descripción de Recursos (*Resource Description Framework*) [www.w3.org V], sería estandarizar el formato de metadatos acerca de los recursos web. Los metadatos describirían los atributos de los recursos web, y serían leídos por herramientas que asistirían a los usuarios que procesaran recursos web *masivamente*, tal como ocurre al buscar y recopilar listas de enlaces relacionados.

Existe una necesidad creciente de intercambio de muchos tipos de datos estructurados en el Web, pero HTML se encuentra limitado en que no es extensible a aplicaciones más allá de la «inspección» de la información. HTML consta de un conjunto estático de estructuras tales como los párrafos, que se limitan a indicar la forma en que se presentan los datos a los usuarios. Más recientemente se ha diseñado el Lenguaje de Marcado Extensible (*Extensible Markup Language*, XML) [www.w3.org VI] como medio de representar datos en formularios estándar, estructurados, y específicos para cada aplicación. Pongamos por ejemplo que XML se emplee

para describir las características de ciertos dispositivos y para describir información personal almacenada acerca de los usuarios. XML es un metalenguaje de descripción de datos, lo cual hace que los datos sean intercambiables entre aplicaciones. El Lenguaje Extensible de Hojas de Estilo (*Extensible Stylesheet Language, XSL*) [www.w3.org VII] se emplea para declarar cómo serán presentados a los usuarios los datos almacenados en el formato XML. Por ejemplo, empleando dos hojas de estilo diferentes, la misma información sobre un usuario concreto podría presentarse en una página web bien gráficamente bien como una simple lista.

Corno arquitectura del sistema, el Web plantea problemas de escala. Los servidores web más populares pueden experimentar muchos *accesos* por segundo, y como resultado la respuesta a los usuarios se ralentiza. El Capítulo 2 describe el empleo de memorias (caché) en los visualizadores y de servidores *proxy* para aliviar estos efectos. A pesar de ello la arquitectura cliente-servidor del Web implica que no hay medios eficientes para mantener a los usuarios al día con las últimas versiones de las páginas. Los usuarios tienen que presionar el botón de *recarga* de su navegador para asegurar que poseen la última información, y éstos se ven forzados a comunicarse con los servidores para comprobar si la copia local del recurso es válida aún.

Finalmente, una página web no siempre es una interfaz de usuario satisfactoria. Los elementos de diálogo definidos para HTML son limitados, y los diseñadores incluyen a menudo en la páginas web, pequeñas aplicaciones, o applets, o bien muchas imágenes para darles una función y apariencia más aceptable. Consecuentemente el tiempo de carga se incrementa.

1.4 DESAFIOS

Los ejemplos de la Sección 1.2 pretenden ilustrar el alcance de los sistemas distribuidos y sugerir las cuestiones que aparecen en su diseño. Aunque se encuentran sistemas distribuidos por todas partes, su diseño es aún bastante simple y quedan todavía grandes posibilidades de desarrollar servicios y aplicaciones más ambiciosas. Muchos de los desafíos que se discuten en esta sección están ya resueltos, pero los futuros diseñadores necesitan estar al tanto y tener cuidado de considerarlas.

1.4.1. HETEROGENEIDAD

Internet permite que los usuarios accedan a servicios y ejecuten aplicaciones sobre un conjunto heterogéneo de redes y computadores. Esta heterogeneidad (es decir, variedad y diferencia) se aplica a todos los siguientes elementos:

- Redes.
- Hardware de computadores.
- Sistemas operativos.
- Lenguajes de programación.
- Implementaciones de diferentes desarrolladores.

A pesar de que Internet consta de muchos tipos de redes diferentes (como vimos en la Figura 1.1), sus diferencias se encuentran enmascaradas dado que todos los computadores conectados a éste utilizan los protocolos de Internet para comunicarse una con otra. Por ejemplo, un computador conectado a Ethernet tiene una implementación de los protocolos de internet sobre Ethernet, así un computador en un tipo de red diferente necesitará una implementación de los protocolos de Internet para esa red. El Capítulo 3 explica cómo se implementan los protocolos de internet sobre una variedad de redes diferentes.

Los tipos de datos, como los enteros, pueden representarse de diferente forma en diferentes clases de hardware por ejemplo, hay dos alternativas para ordenar los bytes en el caso de los enteros. Hay que tratar con estas diferencias de representación si se va a intercambiar mensajes entre programas que se ejecutan en diferente hardware.

Aunque los sistemas operativos de todos los computadores de Internet necesitan incluir una implementación de los protocolos de Internet, no todas presentan necesariamente la misma interfaz de programación para estos protocolos. Por ejemplo, las llamadas para intercambiar mensajes en UNIX son diferentes de las llamadas en Windows NT.

Lenguajes de programación diferentes emplean representaciones diferentes para caracteres y estructuras de datos como cadenas de caracteres y registros. Hay que tener en cuenta estas diferencias si queremos que los programas escritos en diferentes lenguajes de programación sean capaces de comunicarse entre ellos.

Los programas escritos por diferentes programadores no podrán comunicarse entre sí a menos que utilicen estándares comunes, por ejemplo para la comunicación en red y la representación de datos elementales y estructuras de datos en mensajes. Para que esto ocurra es necesario concertar y adoptar estándares (como así lo son los protocolos de Internet).

✓ **Middleware.** El término *middleware* se aplica al estrato software que provee una abstracción de programación, así como un enmascaramiento de la heterogeneidad subyacente de las redes, hardware, sistemas operativos y lenguajes de programación. CORBA, el cual se describe en los Capítulos 4, 5 y 17, es un ejemplo de ello. Algún middleware, como Java RMI (véase el Capítulo 5) sólo se soporta en un único lenguaje de programación. La mayoría de middleware se implementa sobre protocolos de Internet, enmascarando éstos la diversidad de redes existentes. Aun así cualquier middleware trata con las diferencias de sistema operativo y hardware. El cómo se obtiene esto es el tema principal del Capítulo 4.

Además de soslayar los problemas de heterogeneidad, el middleware proporciona un modelo computacional uniforme al alcance de los programadores de servidores y aplicaciones distribuidas. Los posibles modelos incluyen invocación sobre objetos remotos, notificación de eventos remotos, acceso remoto mediante SQL y procesamiento distribuido de transacciones. Por ejemplo, CORBA proporciona invocación sobre objetos remotos, lo que permite que un objeto en un programa en ejecución en un computador invoque un método de un objeto de un programa que se ejecuta en otro computador. La implementación oculta el hecho de que los mensajes se transmiten en red en cuanto al envío de la petición de invocación y su respuesta.

✓ **Heterogeneidad y código móvil.** El término *código móvil* se emplea para referirse al código que puede ser enviado desde un computador a otro y ejecutarse en éste, por eso los applets de Java son un ejemplo de ello. Dado que el conjunto de instrucciones de un computador depende del hardware, el código de nivel de máquina adecuado para correr en un tipo de computador no es adecuado para ejecutarse en otro tipo. Por ejemplo, los usuarios de PC envían a veces archivos ejecutables agregados a los correos electrónicos para ser ejecutados por el destinatario, pero el receptor bien pudiera no ser capaz de ejecutarlo, por ejemplo, sobre un Macintosh o un computador con Linux.

La aproximación de *máquina virtual* provee un modo de crear código ejecutable sobre cualquier hardware: el compilador de un lenguaje concreto generará código para una máquina virtual en lugar de código apropiado para un hardware particular, por ejemplo el compilador Java produce código para la máquina virtual Java, la cual sólo necesita ser implementada una vez para cada tipo de máquina con el fin de poder lanzar programas Java. Sin embargo, la solución Java no se puede aplicar de modo general a otros lenguajes.

1.4.2. EXTENSIBILIDAD

La extensibilidad de un sistema de cómputo es la característica que determina si el sistema puede ser extendido y reimplementado en diversos aspectos. La extensibilidad de los sistemas distribuidos se determina en primer lugar por el grado en el cual se pueden añadir nuevos servicios de compartición de recursos y ponerlos a disposición para el uso por una variedad de programas cliente.

No es posible obtener extensibilidad a menos que la especificación y la documentación de las interfaces software clave de los componentes de un sistema estén disponibles para los desarrolladores de software. Es decir, que las interfaces clave estén *publicadas*. Este procedimiento es similar a una estandarización de las interfaces, aunque a menudo puentea los procedimientos oficiales de estandarización, que por lo demás suelen ser lentos y complicados.

Sin embargo, la publicación de interfaces sólo es el punto de arranque de la adición y extensión de servicios en un sistema distribuido. El desafío para los diseñadores es hacer frente a la complejidad de los sistemas distribuidos que constan de muchos componentes diseñados por personas diferentes.

Los diseñadores de los protocolos de Internet presentaron una serie de documentos denominados «Solicitudes de Comentarios» (*Request For Comments*), o RFC, cada una de las cuales se conoce por un número.

Las especificaciones de los protocolos de Internet fueron publicados en esta serie a principios de los años ochenta, seguido por especificaciones de aplicaciones que corrieran sobre ellos, tales como transferencia de archivos, correo electrónico y *telnet* a mediados de los años ochenta. Esta práctica continúa y forma la base de la documentación técnica sobre Internet. Esta serie incluye discusiones así como especificaciones de protocolos. Se puede obtener copias en [www.jetf.org]. Así la publicación de los protocolos originales de comunicación de Internet ha posibilitado que se construyera una enorme variedad de sistemas y aplicaciones sobre Internet. Los documentos RFC no son el único modo de publicación. Por ejemplo, COREA está publicado a través de una serie de documentos técnicos, incluyendo una especificación completa de las interfaces de sus servicios, Véase [www.omg.org].

Los sistemas diseñados de este modo para dar soporte a la compartición de recursos se etiquetan como *sistemas distribuidos abiertos (open distributed systems)* para remarcar el hecho de ser extensibles. Pueden ser extendidos en el nivel hardware mediante la inclusión de computadores a la red y en el nivel software por la introducción de nuevos servicios y la reimplementación de los antiguos, posibilitando a los programas de aplicación la compartición de recursos. Otro beneficio más, citado a menudo, de los sistemas abiertos es su independencia de proveedores concretos.

En resumen:

- Los sistemas abiertos se caracterizan porque sus interfaces están publicadas.
- Los sistemas distribuidos abiertos se basan en la providencia de un mecanismo de comunicación uniforme e interfaces públicas para acceder a recursos compartidos.
- Los sistemas distribuidos abiertos pueden construirse con hardware y software heterogéneo, posiblemente de diferentes proveedores. Sin embargo, la conformidad con el estándar publicado de cada componente debe contrastarse y verificarse cuidadosamente si se desea que el sistema trabaje correctamente.

1.4.3. SEGURIDAD

Entre los recursos de información que se ofrecen y se mantienen en los sistemas distribuidos, muchos tienen un alto valor intrínseco para sus usuarios. Por esto su seguridad es de considerable importancia. La seguridad de los recursos de información tiene tres componentes: confidencialidad (protección contra el descubrimiento por individuos no autorizados); integridad (protección contra la alteración o corrupción); y disponibilidad (protección contra interferencia con los procedimientos de acceso a los recursos).

La Sección 1.1 apuntaba que a pesar de que Internet permite a un programa de un computador comunicarse con un programa en otro computador sin mencionar su ubicación, el permitir un acceso libre a todos los recursos de una intranet lleva asociados riesgos contra la seguridad. Aunque se pueda emplear un cortafuegos para disponer una barrera alrededor de una intranet, restringiendo el tráfico que pudiera entrar y salir, no pretende asegurar el uso apropiado

de los recursos por usuarios del interior de la intranet, o del uso apropiado de los recursos en Internet, no protegidos por el cortafuegos.

En un sistema distribuido, los clientes envían peticiones de acceso a datos administrados por servidores, lo que trae consigo enviar información en los mensajes por la red. Por ejemplo:

1. Un médico puede solicitar acceso a los datos hospitalarios de un paciente o enviar modificaciones sobre ellos.
2. En comercio electrónico y banca, los usuarios envían su número de tarjeta de crédito a través de Internet.

En ambos casos, el reto se encuentra en enviar información sensible en un mensaje, por la red, de forma segura. Pero la seguridad no sólo es cuestión de ocultar los contenidos de los mensajes, también consiste en conocer con certeza la identidad del usuario u otro agente en nombre del cual se envía el mensaje. En el primer ejemplo, el servidor necesita conocer que el usuario es realmente un médico y en el segundo, el usuario necesita estar seguro de la identidad de la tienda o del banco con el que está tratando. El segundo reto consiste en identificar un usuario remoto u otro agente correctamente. Ambos desafíos pueden lograrse a través de técnicas de encriptación desarrolladas al efecto. Éstas se utilizan ampliamente en Internet y se discutirán en el Capítulo 7.

Sin embargo, aún existen dos desafíos de seguridad que no han sido cumplimentados completamente:

Ataques de denegación de servicio: otro problema de seguridad ocurre cuando un usuario desea obstaculizar un servicio por alguna razón. Esto se obtiene al bombardear el servicio con un número suficiente de peticiones inútiles de modo que los usuarios serios sean incapaces de utilizarlo. A esto se le denomina ataque de *denegación de servicio*. En el momento de escribir esto (primeros meses del año 2000), han ocurrido, recientemente, varios ataques de denegación de servicio sobre conocidos servicios web. Por el momento tales ataques se contrarrestan intentando atrapar y castigar a los perpetradores con posterioridad al suceso, aunque no es una solución general al problema. Se encuentran en desarrollo contramedidas basadas en mejorar la administración de las redes, éstas se comentarán en el Capítulo 3.

Seguridad del código móvil: el código móvil necesita ser tratado con cuidado. Suponga que alguien recibe un programa ejecutable adherido a un correo electrónico: los posibles efectos al ejecutar el programa son impredecibles; por ejemplo, pudiera parecer que presentan un interesante dibujo en la pantalla cuando en realidad están interesados en el acceso a los recursos locales, o quizás pueda ser parte de un ataque de denegación de servicio. En el Capítulo 7 se bosquejarán algunas medidas para asegurar el código móvil.

<i>Fecha</i>	<i>Computadores</i>	<i>Servidores web</i>
Diciembre de 1979	188	0
Julio de 1989	130.000	0
Julio de 1999	56.218.000	5.560.866

Figura 1.5. Computadores en Internet.

1.4.4. ESCALABILIDAD

Los sistemas distribuidos operan efectiva y eficientemente en muchas escalas diferentes, desde pequeñas intranets a Internet. Se dice que un sistema es *escalable* si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios. Internet proporciona un ejemplo de sistema distribuido en el que el número de computadores y

servicios experimenta un dramático incremento. La Figura 1.5 muestra el número de computadores y servicios en Internet durante 20 años, desde 1979 hasta 1999, y la Figura 1.6 muestra el creciente número de computadores y servidores *web* durante los 16 años de historia del Web hasta 1999, véase [info.isoc.org].

El diseño de los sistemas distribuidos escalables presenta los siguientes retos:

Control del coste de los recursos físicos: según crece la demanda de un recurso, debiera ser posible extender el sistema, a un coste razonable, para satisfacerla. Por ejemplo, la frecuencia con la que se accede a los archivos de una intranet suele crecer con el incremento del número de usuarios y computadores. Debe ser posible añadir servidores para evitar el embotellamiento que aparece cuando un solo servidor de archivos ha de manejar todas las peticiones de acceso a éstos. En general, para que un sistema con n usuarios fuera escalable, la cantidad de recursos físicos necesarios para soportarlo debiera ser como máximo $O(\ll)$, es decir proporcional a n . Por ejemplo, si un solo servidor de archivos pudiera soportar 20 usuarios, entonces 2 servidores del mismo tipo tendrán capacidad para 40 usuarios. Aunque parezca una meta obvia, no es tan fácil lograrlo en la práctica, según se mostrará en el Capítulo 8.

Control de las pérdidas de prestaciones: considere la administración de un conjunto de datos cuyo tamaño es proporcional al número de usuarios o recursos del sistema, sea por ejemplo la tabla con la relación de nombres de dominio de computadores y sus direcciones Internet sustentado por el Sistema de Nombres de Dominio (*Domain Name System*), que se emplea principalmente para averiguar nombres DNS tales como *www.amazon.com*. Los algoritmos que emplean estructuras jerárquicas se comportan mejor frente al crecimiento de la escala que los algoritmos que emplean estructuras lineales. Pero incluso con estructuras jerárquicas un incremento en tamaño traerá consigo pérdidas en prestaciones: el tiempo que lleva acceder a datos estructurados jerárquicamente es $O(\log n)$, donde n es el tamaño del conjunto de datos. Para que un sistema sea escalable, la máxima pérdida de prestaciones no debiera ser peor que sistemas distribuidos operan efectiva y eficientemente en muchas escalas diferentes, desde pequeñas intranets a Internet. Se dice que un sistema es *escalable* si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios. Internet proporciona un ejemplo de un sistema distribuido en el que el número de computadores y esta medida.

<i>Fecha</i>	<i>Computadores</i>	<i>Servidores web</i>	<i>Porcentaje (%)</i>
Julio de 1993	1.776.000	130	0,008
Julio de 1995	6.642.000	23.500	0,4
Julio de 1997	19.540.000	1.203.096	6
Julio de 1999	56.218.000	6.598.697	12

Figura 1.6. Comparación entre Computadores y Servidores web en Internet.

Prevención de desbordamiento de recursos software: un ejemplo de pérdida de escalabilidad se muestra en el tipo de número usado para las direcciones Internet (direcciones de computadores en Internet). A finales de los años setenta, se decidió emplear para esto 32 bits, pero como se explicará en el Capítulo 3 el suministro de direcciones para Internet se desbordará probablemente al comienzo de la década del año 2000. Por esta razón, la nueva versión del protocolo empleará direcciones Internet de 128 bits. A pesar de ello, para ser justos con los primeros diseñadores de Internet, no hay una solución idónea para este problema. Es difícil predecir la demanda que tendrá que soportar un sistema con años de anticipación. Además, sobredimensionar para prever el crecimiento futuro pudiera ser peor que la adaptación a un cambio cuando se hace necesario; las direcciones Internet grandes ocupan espacio extra en los mensajes, y en la memoria de los computadores.

Evitación de cuellos de botella de prestaciones: en general, para evitar cuellos de botella de prestaciones, los algoritmos deberían ser descentralizados. Ilustramos este punto aludiendo al predecesor del Sistema de Nombres de Dominio en el cual la tabla de nombres se alojaba en un solo archivo maestro que podía descargarse a cualquier computador que lo necesitara. Esto funcionaba bien cuando sólo había unos cientos de computadores en Internet, pero pronto se convirtió en un serio cuello de botella de prestaciones y de administración. El Sistema de Nombres de Dominio eliminó este cuello de botella particionando la tabla de nombres entre servidores situados por todo Internet y siendo administrados localmente, véanse los Capítulos 3 y 9.

Algunos recursos compartidos son accedidos con mucha frecuencia; por ejemplo, puede que muchos usuarios accedan a la misma página web, causando un declive de las prestaciones. En el Capítulo 2 veremos que el empleo de *caché* y replicación puede mejorar las prestaciones de los recursos que estén siendo muy fuertemente utilizadas.

Idealmente, el software de sistema y aplicación no tiene por qué cambiar cuando la escala del sistema se incrementa, pero esto es difícil de conseguir. La cuestión del escalado de un sistema es un tema dominante en el desarrollo de sistemas distribuidos. Las técnicas que han demostrado éxito se describen extensamente en este libro. Éstas incluyen el uso de datos replicados (Capítulos 8 y 14), la técnica asociada de caché (Capítulos 2 y 8) y la implementación de múltiples servidores para tratar tareas frecuentes, permitiendo varias tareas similares concurrentemente.

1.4.5. TRATAMIENTO DE FALLOS

Los sistemas computacionales a veces fallan. Cuando aparecen fallos en el hardware o el software, los programas pueden producir resultados incorrectos o pudieran parar antes de haber completado el cálculo pedido. En el Capítulo 2 discutiremos y clasificaremos un rango de tipos de fallos posibles que pueden acaecer en los procesos y redes de que consta un sistema distribuido.

Los fallos en un sistema distribuido son parciales; es decir, algunos componentes fallan mientras otros siguen funcionando. Consecuentemente, el tratamiento de fallos es particularmente difícil. A lo largo del libro se discuten las siguientes técnicas para tratar fallos:

Detección de fallos: algunos fallos son detectables. Por ejemplo, se pueden utilizar sumas de comprobación (*checksums*) para detectar datos corruptos en un mensaje o un archivo. El Capítulo 2 explica que es difícil o incluso imposible detectar algunos otros fallos como la caída de un servidor remoto en Internet. El reto está en arreglárselas en presencia de fallos que no pueden detectarse pero que sí pueden esperarse.

Enmascaramiento de fallos: algunos fallos que han sido detectados pueden ocultarse o atenuarse. Dos ejemplos de ocultación de fallos son:

1. Los mensajes pueden retransmitirse cuando falla la recepción.
2. Los archivos con datos pueden escribirse en una pareja de discos de forma que si uno está deteriorado el otro seguramente está en buen estado.

Simplemente eliminar un mensaje corrupto es un ejemplo de atenuar un fallo (pudiera retransmitirse de nuevo). Probablemente el lector se dará cuenta de que las técnicas indicadas para ocultar los fallos no tienen garantía de funcionamiento las peores situaciones; por ejemplo, los datos en el segundo disco pudieran también estar corrompidos, o el mensaje bien pudiera no llegar a tiempo no importa cuantas veces se retransmita.

Tolerancia de fallos: la mayoría de los servicios en Internet exhiben fallos; es posible que no sea práctico para ellos pretender detectar y ocultar todos los fallos que pudieran aparecer en una red tan grande y con tantos componentes. Sus clientes pueden diseñarse para tolerar

ciertos fallos, lo que implica que también los usuarios tendrán que tolerarlos generalmente. Por ejemplo, cuando un visualizador web no puede contactar con un servidor web no hará que el cliente tenga que esperar indefinidamente mientras hace sucesivos intentos; informará al usuario del problema, dándole la libertad de intentarlo más tarde.

Recuperación frente a fallos: la recuperación implica el diseño de software en el que, tras una caída del servidor, el estado de los datos pueda reponerse o retractarse (*roll back*) a una situación anterior. En general, cuando aparecen fallos los cálculos realizados por algunos programas se encontrarán incompletos y al actualizar datos permanentes (archivos e información ubicada en almacenamiento persistente) pudiera encontrarse en un estado inconsistente. La recuperación se describirá en el Capítulo 13.

Redundancia: puede lograrse que los servicios toleren fallos mediante el empleo redundante de componentes. Considere los siguientes ejemplos:

1. Siempre deberá haber al menos dos rutas diferentes entre cualesquiera dos *routers* (enrutadores) en Internet.
2. En el Sistema de Nombres de Dominio, cada tabla de nombres se encuentra replicada en dos servidores diferentes.
3. Una base de datos puede encontrarse replicada en varios servidores para asegurar que los datos siguen siendo accesibles tras el fallo de cualquier servidor concreto; los servidores pueden diseñarse para detectar fallos entre sus iguales; cuando se detecta algún error en un servidor se redirigen los clientes a los servidores restantes.

El diseño de técnicas eficaces para mantener réplicas actualizadas de datos que cambian rápidamente sin una pérdida excesiva de prestaciones es un reto; en el Capítulo 14 se discutirán varias aproximaciones a él.

Los sistemas distribuidos proporcionan un alto grado de disponibilidad frente a los fallos del hardware. La *disponibilidad* de un sistema mide la proporción de tiempo en que está utilizable. Cuando falla algún componente del sistema distribuido sólo resulta afectado el trabajo relacionado con el componente defectuoso. Así como cuando un computador falla el usuario puede desplazarse a otro, también puede iniciarse un proceso de servicio en otra ubicación.

1.4.6. CONCURRENCIA

Tanto los servicios como las aplicaciones proporcionan recursos que pueden compartirse entre los clientes en un sistema distribuido. Existe por lo tanto una posibilidad de que varios clientes intenten acceder a un recurso compartido a la vez. Por ejemplo, una estructura de datos que almacena licitaciones de una subasta puede ser accedida muy frecuentemente cuando se aproxima el momento de cierre.

El proceso que administra un recurso compartido puede atender las peticiones de cliente una por una en cada momento, pero esta aproximación limita el ritmo de producción del sistema (*throughput*). Por esto los servicios y aplicaciones permiten, usualmente, procesar concurrentemente múltiples peticiones de los clientes. Más concretamente, suponga que cada recurso se encapsula en un objeto y que las invocaciones se ejecutan en hilos de ejecución concurrentes (*threads*). En este caso es posible que varios *threads* estuvieran ejecutando concurrentemente el contenido de un objeto, en cuyo caso las operaciones en el objeto pueden entrar en conflicto entre sí y producir resultados inconsistentes. Por ejemplo, sean dos ofertas que concurren a una subasta como «Pérez: 122\$» y «Rodríguez: 111\$» y las operaciones correspondientes se entrelazan sin control alguno, estas ofertas se pueden almacenar como «Pérez: 111\$» y «Rodríguez: 122\$».

La moraleja de esta historia es que cada objeto que represente un recurso compartido en un sistema distribuido debe responsabilizarse de garantizar que opera correctamente en un entorno concurrente. De este modo cualquier programador que recoge una implementación de un objeto que no está concebido para su aplicación en un entorno distribuido deberá realizar las modificaciones necesarias para que sea seguro su uso en un entorno concurrente.

Para que un objeto sea seguro en un entorno concurrente, sus operaciones deben sincronizarse de forma que sus datos permanezcan consistentes. Esto puede lograrse mediante el empleo de técnicas conocidas como los semáforos, que se usan en la mayoría de los sistemas operativos. Esta cuestión y su extensión a conjuntos de objetos compartidos se discutirá en los Capítulos 6 y 12.

1.4.7. TRANSPARENCIA

Se define transparencia como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes en un sistema distribuido, de forma que se perciba el sistema como un todo más que como una colección de componentes independientes. Las implicaciones de la transparencia son de gran calado en el diseño del software del sistema.

El Manual de Referencia ANSA (*ANSA Reference Manual*) [ANSA 1989] y el Modelo de Referencia para el Procesamiento Distribuido Abierto (RM-ODP: *Reference Model for Open Distributed Processing*) de la Organización Internacional de Estándares [ISO 1992] identifican ocho formas de transparencia. Hemos reproducido las definiciones originales de ANSA, reemplazando su transparencia de migración por nuestra propia transparencia de movilidad, de mayor alcance:

Transparencia de acceso que permite acceder a los recursos locales y remotos empleando operaciones idénticas.

Transparencia de ubicación que permite acceder a los recursos sin conocer su localización.

Transparencia de concurrencia que permite que varios procesos operen concurrentemente sobre recursos compartidos sin interferencia mutua.

Transparencia de replicación que permite utilizar múltiples ejemplares de cada recurso para aumentar la fiabilidad y las prestaciones sin que los usuarios y los programadores de aplicaciones necesiten su conocimiento.

Transparencia frente a fallos que permite ocultar los fallos, dejando que los usuarios y programas de aplicación completen sus tareas a pesar de fallos del hardware o de los componentes software.

Transparencia de movilidad que permite la reubicación de recursos y clientes en un sistema sin afectar la operación de los usuarios y los programas.

Transparencia de prestaciones que permite reconfigurar el sistema para mejorar las prestaciones según varía su carga.

Transparencia al escalado que permite al sistema y a las aplicaciones expandirse en tamaño sin cambiar la estructura del sistema o los algoritmos de aplicación.

Las dos más importantes son la transparencia de acceso y la transparencia de ubicación; su presencia o ausencia afecta principalmente a la utilización de recursos distribuidos. A veces se les da el nombre conjunto de *transparencia de red*.

Como ilustración de la transparencia de acceso, considere una interfaz gráfica de usuario basada en carpetas, donde los contenidos de las carpetas se observan igual ya sean éstas locales o remotas. Otro ejemplo pudiera ser el de una interfaz de programación de aplicaciones [API] para archivos que emplea las mismas operaciones para acceder a éstos ya sean locales o remotos (véase el Capítulo 8). Como ejemplo de carencia de transparencia de acceso, considere un sistema distribuido que no permite acceder a los archivos de un computador remoto a menos que se emplee el programa «ftp».

Los nombres de recursos web o URLs son transparentes a la ubicación dado que la parte del URL que identifica el nombre del dominio del servidor web se refiere a un nombre de computador en un dominio, más que a una dirección en Internet. Sin embargo, un URL no es transparente a la movilidad, porque una página web dada no puede moverse a un nuevo lugar en un dominio diferente sin que todos los enlaces anteriores a esta página sigan apuntando a la página original.

En general, los identificadores como los URLs que incluyen los nombres de dominio en los computadores contravienen la transparencia de replicación. Aunque el DNS permite que un nombre de dominio se refiera a varios computadores, sólo se escoge uno de ellos cuando se utiliza un nombre. Ya que un esquema de replicación generalmente necesita ser capaz de acceder a todos los computadores del grupo, sería necesario acceder a cada entrada del DNS por nombre.

Como ilustración de la presencia de transparencia de red, considere el uso de una dirección de correo electrónico como *Pedro.Picapedra@piedradura.com*. La dirección consta de un nombre de usuario y un nombre de dominio. Observe que a pesar de que los programas de correo aceptan nombres de usuario para usuarios locales, añaden el nombre del dominio. El envío de correo a un usuario no implica el conocimiento de su ubicación física en la red. Tampoco el procedimiento de envío de un mensaje de correo depende de la ubicación del receptor. En resumen, el correo electrónico en Internet proporciona ambas cosas: transparencia de ubicación y transparencia de acceso (en definitiva, transparencia de red).

La transparencia frente a fallos puede ilustrarse también en el contexto del correo electrónico, el cual eventualmente se envía, incluso aunque los servidores o los enlaces de comunicaciones fallen. Los fallos se enmascaran intentando retransmitir los mensajes hasta que se envían satisfactoriamente, incluso si lleva varios días. El middleware convierte generalmente los fallos de redes y procesos en excepciones del nivel de programación (para una explicación véase el Capítulo 5).

Para ilustrar la transparencia a la movilidad, considere el caso de los teléfonos móviles. Supongamos que ambos, el emisor y el receptor, viajan en tren por diferentes partes del país, moviéndose de un entorno (célula) a otro. Veamos al terminal del emisor como un cliente y al terminal del receptor como un recurso. Los dos usuarios telefónicos no perciben el desplazamiento de sus terminales (el cliente y el recurso) entre dos células.

La transparencia oculta y difumina anónimamente los recursos que no son relevantes directamente para la tarea entre manos de los usuarios y programadores de aplicaciones. Por ejemplo, en general es deseable que el uso de ciertos dispositivos físicos sea intercambiable. Por ejemplo, en un sistema multiprocesador, la identificación del procesador en que se ejecuta cada proceso es irrelevante. Aún puede que la situación sea otra: por ejemplo, un viajero que conecta un computador portátil a una red local, en cada oficina que visita hace uso de servicios locales como el correo, utilizando diferentes servidores en cada ubicación. Incluso dentro de un edificio, es normal preparar las cosas para imprimir cada documento en una impresora concreta: generalmente la más próxima. También para un programador que desarrolla programas paralelos, no todos los procesadores son anónimos. El o ella pudiera estar interesado en qué procesadores utilizar para la tarea, o al menos cuántos y su topología de interconexión.

Los sistemas distribuidos están por todas partes. Internet permite que los usuarios de todo el mundo accedan a sus servicios donde quiera que estén situados. Cada organización administra una intranet, que provee servicios locales y servicios de Internet a los usuarios locales y habitualmente proporciona servicios a otros usuarios de Internet. Es posible construir pequeños sistemas distribuidos con computadores portátiles y otros dispositivos computacionales pequeños conectados a una red inalámbrica.

La compartición de recursos es el principal factor que motiva la construcción de sistemas distribuidos. Recursos como impresoras, archivos, páginas web o registros de bases de datos se administran mediante servidores del tipo apropiado. Por ejemplo los servidores web administran páginas y otros recursos web. Los recursos son accedidos por clientes, por ejemplo, los clientes de los servidores web se llaman normalmente visualizadores o navegadores web. La construcción de los sistemas distribuidos presenta muchos desafíos:

Heterogeneidad: debe construirse desde una variedad de diferentes redes, sistemas operativos, hardware de computador y lenguajes de programación. Los protocolos de comunicación de Internet enmascaran las diferencias entre redes y el middleware puede tratar con las diferencias restantes.

Extensibilidad: los sistemas distribuidos deberían ser extensibles, el primer paso es la publicación de las interfaces de sus componentes, pero la integración de componentes escritos por diferentes programadores es un auténtico reto.

Seguridad: se puede emplear encriptación para proporcionar una protección adecuada a los recursos compartidos y mantener secreta la información sensible cuando se transmite un mensaje a través de la red. Los ataques de denegación de servicio son aún un problema.

Escalabilidad: un sistema distribuido es escalable si el coste de añadir un usuario es una cantidad constante en términos de recursos que se deberán añadir. Los algoritmos empleados para acceder a los datos compartidos deberían evitar cuellos de botella y los datos deberían estar estructurados jerárquicamente para dar los mejores tiempos de acceso. Los datos frecuentemente accedidos pudieran estar replicados.

Tratamiento de fallos: cualquier proceso, computador o red puede fallar independientemente de los otros. En consecuencia cada componente necesita estar al tanto de las formas posibles en que pueden fallar los componentes de los que depende y estar diseñado para tratar apropiadamente con cada uno de estos fallos.

Concurrencia: la presencia de múltiples usuarios en un sistema distribuido es una fuente de peticiones concurrentes a sus recursos. Cada recurso debe estar diseñado para ser seguro en un entorno concurrente.

Transparencia: el objetivo es que ciertos aspectos de la distribución sean invisibles al programador de aplicaciones de modo que sólo necesite ocuparse del diseño de su aplicación particular. Por ejemplo, no debe ocuparse de su ubicación o los detalles sobre cómo se accede a sus operaciones por otros componentes, o si será replicado o migrado. Incluso los fallos de las redes y los procesos pueden presentarse a los programadores de aplicaciones en forma de excepciones, aunque deban de ser tratados.

- 1.1. Proponga cinco tipos de recursos hardware y cinco tipos de recursos software o de datos que puedan compartirse útilmente. Proponga ejemplos de su uso compartido tal y como ocurre en la práctica en los sistemas distribuidos.
- 1.2. ¿Cómo podría sincronizarse los relojes de dos computadores unidos por una red local, sin hacer uso de una referencia temporal externa? ¿Qué factores limitarían la precisión del procedimiento propuesto? ¿Cómo podrían sincronizarse los relojes de un mayor número de computadores conectados a Internet? Discuta la precisión de este procedimiento.
- 1.3. Un usuario llega a una estación de ferrocarril que no conoce, portando un PDA capaz de conectarse a una red inalámbrica. Sugiera cómo podría proporcionársele al usuario información sobre los servicios locales y las comodidades en la estación, sin necesidad de insertar el nombre de la estación o sus características. ¿Qué dificultades técnicas hay que superar?
- 1.4. ¿Cuáles son las ventajas y desventajas de HTML, URL y HTTP como tecnologías de base para la consulta y visualización de información? ¿Son algunas de estas tecnologías adecuadas como plataforma de cómputo cliente-servidor en general?
- 1.5. Tome World Wide Web como ejemplo para ilustrar el concepto de compartición de recursos, cliente y servidor.
Los recursos en World Wide Web y otros servicios se direccionan mediante URL. ¿Qué significan la siglas URL? Proporcione ejemplos de tres tipos de recursos web a los que pueda darse un nombre URL.
- 1.6. Dé un ejemplo de URL.
Enumere los tres componentes principales de un URL, indicando cómo se delimitan e ilustre cada uno a partir de un ejemplo.
¿Hasta qué límite es transparente a la ubicación en URL?
- 1.7. Un programa servidor escrito en un lenguaje (por ejemplo C++) proporciona un objeto BURBUJA al que se pretende que accedan clientes que pudieran estar escritos en un lenguaje diferente (por ejemplo Java). Los computadores clientes y servidores pueden tener un hardware diferente, pero todas están conectadas a Internet. Describa los problemas debidos a cada uno de los cinco aspectos de la heterogeneidad que necesitan resolverse para posibilitar que un objeto cliente invoque un método sobre el objeto servidor.
- 1.8. Un sistema distribuido abierto permite la adición de nuevos servicios de compartición de recursos como el objeto BURBUJA del Ejercicio 1.7 y que sean accesibles por una variedad de programas cliente. Discuta en el contexto de este ejemplo, hasta dónde las necesidades de extensibilidad difieren de las de heterogeneidad.
- 1.9. Suponga que las operaciones del objeto BURBUJA están separadas en dos categorías: operaciones públicas disponibles para todos los usuarios y operaciones protegidas disponibles sólo para ciertos usuarios conocidos por un nombre concreto. Presente todos los problemas relacionados con la operación de garantizar que sólo los usuarios con nombre conocido puedan acceder a la operación protegida. Suponiendo que el acceso a una operación protegida da información que no debiera revelarse al resto de los usuarios. ¿Qué más problemas aparecen?
- 1.10. El servicio INFO admite un conjunto de recursos potencialmente muy grande, cada uno de los cuales puede ser accedido por usuarios de Internet mediante una clave (en forma de

«string»). Discuta una aproximación al diseño de los nombres de los recursos que logra la mínima pérdida de prestaciones según crece el número de recursos en el servicio. Sugiera cómo puede implementarse el servicio INFO para evitar cuellos de botella en las prestaciones cuando el número de usuarios se vuelve muy grande.

- 1.11.** Enumere los tres componentes software principales que pueden fallar cuando un proceso cliente invoca un método en un objeto servidor, proporcionando un ejemplo del fallo de cada clase. Sugiera cómo pueden construirse los componentes para que toleren sus fallos mutuamente.
- 1.12.** Un servidor mantiene un objeto de información compartida tal como el objeto BURBUJA del Ejercicio 1.7. Argumente en pro y en contra de si admitir que las peticiones de los clientes se ejecuten concurrentemente en el servidor. En este caso, dé un ejemplo de posible «interferencia» que pudiera aparecer entre las operaciones de diferentes clientes. Sugiera cómo puede prevenirse tal interferencia.
- 1.13.** Varios servidores implementan cierto servicio. Explique el porqué pueden transferirse los recursos entre ellos. ¿Sería satisfactorio para los clientes la multidifusión de todas las peticiones al grupo de servidores como un medio de obtener la transparencia de movilidad para los clientes?