# 1 Introducción a los sistemas distribuidos

# Contenido

# 1.1 Motivación

Los sistemas distribuidos suponen un paso más en la evolución de los sistemas informáticos, entendidos desde el punto de vista de las necesidades que las aplicaciones plantean y las posibilidades que la tecnología ofrece. Antes de proporcionar una definición de sistema distribuido resultará interesante presentar, a través de la evolución histórica, los conceptos que han desembocado en los sistemas distribuidos actuales, caracterizados por la distribución física de los recursos en máquinas interconectadas.

Utilizaremos aquí el término **recurso** con carácter general para referirnos a cualquier dispositivo o servicio, hardware o software, susceptible de ser compartido.

## 1.1.1 Tipos de sistemas

Desde una perspectiva histórica se puede hablar de diferentes modelos que determinan la funcionalidad y la estructura de un sistema de cómputo, las características del sistema operativo como gestor de los recursos, y su campo de aplicación y uso:

- **Sistemas de lotes**. Son los primeros sistemas operativos, que permitían procesar en diferido y secuencialmente datos suministrados en paquetes de tarjetas perforadas. Hoy en día, sobre sistemas multiprogramados con interfaces de usuario interactivas, el proceso por lotes tiene sentido en aplicaciones de cálculo intensivo, por ejemplo en supercomputación.

- **Sistemas** centralizados **de tiempo compartido**. Fue el siguiente paso, a mediados de los 60. El objetivo es incrementar la eficiencia en el uso de la CPU, un recurso entonces caro y escaso, disminuyendo los tiempos de respuesta de los usuarios, que operan interactivamente. Los recursos están centralizados y se accede al sistema desde terminales.

- **Sistemas de teleproceso**. Se diferencian del modelo anterior en que los terminales —más recientemente, sistemas personales— son remotos y acceden a un sistema central utilizando una infraestructura de red (por ejemplo la telefónica) y un protocolo de comunicaciones normalmente de tipo propietario. El sistema central monopoliza la gestión de los recursos. Ejemplos de aplicaciones que resolvía este modelo son los sistemas de reservas y de transacciones bancarias.

- **Sistemas personales**. La motivación de este tipo de sistemas estuvo en proporcionar un sistema dedicado para un único usuario, lo que fue posible gracias al abaratamiento del hardware por la irrupción del microprocesador a comienzos de los 80. Precisamente, el coste reducido es su característica fundamental. El sistema operativo de un ordenador personal (PC) es, en un principio, monousuario: carece de mecanismos de protección. Aunque, por simplicidad, los primeros sistemas operativos eran monoprogramados (MS-DOS), la mejora del hardware pronto permitió soportar sistemas multitarea (Macintosh, OS/2, Windows 95/98), e incluso sistemas operativos diseñados para tiempo compartido, como UNIX y Windows NT[1]. Un sistema personal posee sus propios recursos locales. Inicialmente, éstos eran los únicos recursos accesibles, pero hoy en día la situación ha cambiado. Por otra parte, la evolución hardware ha llevado a los ordenadores personales hacia versiones móviles (PC portátiles y otros dispositivos como PDAs y teléfonos móviles).

- **Sistemas en red**. En la evolución del teleproceso, los terminales fueron ganando capacidad de cómputo y funcionalidad hasta convertirse en sistemas autónomos. El concepto de computador central desaparece; ahora hay que hablar de un conjunto de computadores que se conectan entre sí utilizando una infraestructura de red. Una máquina que proporciona el acceso a un determinado recurso es el **servidor** de ese recurso. Los **clientes**, que pueden disponer de **recursos locales**, acceden a un **recurso remoto** mediante solicitud al servidor correspondiente. Existen protocolos de red propietarios, que restringen la interoperatividad a máquinas del mismo tipo que conforman una **red local**, como Novell para PCs, o AppelTalk para Macintosh. Sin embargo, el desarrollo de protocolos comunes, como TCP/IP, ha permitido interconectar las máquinas independientemente de sus características y sistema operativo (interoperatividad), extendiendo el ámbito de estos sistemas a **redes de área amplia** y posibilitando el surgimiento de **Internet**.

[1] Este tipo de sistemas se conocía hacia 1990 como **estaciones de trabajo** o *work-stations*. Hoy en día cualquier ordenador personal es capaz de soportarlos y ya no se hace está distinción. El PC se ha convertido en un componente estándar que, como veremos, se usa indistintamente como sistema personal, como servidor de recursos compartidos, o incluso como elemento de un sistema de alto rendimiento (*cluster computing*).

- **Sistemas distribuidos**. Los recursos de diferentes máquinas en red se *integran* de forma que desaparece la dualidad local/remoto. La diferencia fundamental con los sistemas en red es que la ubicación del recurso es *transparente* a las aplicaciones y usuarios, por lo que, desde este punto de vista, no hay diferencia con un sistema de tiempo compartido. El usuario accede a los recursos del sistema distribuido a través de una interfaz gráfica de usuario desde un terminal, despreocupándose de su localización. Las aplicaciones ejecutan una interfaz de llamadas al sistema como si de un sistema centralizado se tratase, por ejemplo POSIX. Un servicio de invocación remota (por ejemplo a procedimientos, RPC, o a objetos, RMI) resuelve los accesos a los recursos no locales utilizando para ello la interfaz de red. Los sistemas distribuidos proporcionan de forma transparente la compartición de recursos, facilitando el acceso y la gestión, e incrementando la eficiencia y la disponibilidad.

El modelo de sistema distribuido es el más general, por lo que, aunque no se ha alcanzado a nivel comercial la misma integración para todo tipo de recursos, la tendencia es clara a favor de este tipo de sistemas. La otra motivación es la relación de costes a la que ha llevado la evolución tecnológica en los últimos años. Hoy en día existe un hardware estándar de bajo coste, los ordenadores personales, que son los componentes básicos del sistema. Por otra parte, la red de comunicación, a no ser que se requieran grandes prestaciones, tampoco constituye un gran problema económico, pudiéndose utilizar infraestructura cableada ya existente (Ethernet, la red telefónica, o incluso la red eléctrica) o inalámbrica.

### 1.1.2 Una definición de sistema distribuido

Podemos aventurar ahora una definición de sistema distribuido:

(1) un conjunto de computadores

(2) interconectados

(3) que comparten un estado,

(4) ofreciendo una visión de sistema único.

Dejando aparte la discusión sobre el significado del término *computador*, la característica (2), que les diferencia de los sistemas personales, es compartida por sistemas distribuidos y sistemas en red. La característica (3), en cambio, es privativa de los sistemas distribuidos, siendo su consecuencia (4) la visión de

sistema único[2], que muestra los recursos de manera homogénea, ocultando su distribución: *el usuario y las aplicaciones no ven una red, sino un sistema indistinguible de uno centralizado*. Mientras que un sistema en red puede definirse como un conjunto de sistemas con estados independientes, en un sistema distribuido se define un **estado global**.

La topología y los atributos físicos de la red están ocultados por los protocolos de red, mientras que la arquitectura de cada máquina está ocultada por el sistema operativo. Como los componentes de un sistema distribuido pueden ser heterogéneos, se requiere una capa de software (a menudo llamado *middleware*) para proporcionar la visión de sistema único. Volveremos sobre este punto al hablar de la estructura del sistema distribuido.

### 1.1.3 Perspectivas

El desarrollo tecnológico en un conjunto de áreas –procesadores, almacenamiento de información, redes (en particular inalámbricas), almacenamiento de energía, pantallas– está introduciendo nuevos dispositivos que abren perspectivas para nuevas aplicaciones en el mundo de los sistemas distribuidos. Una evolución, iniciada en los años noventa del siglo pasado a partir de la introducción de equipos portátiles, nos lleva a lo que denominamos **informática móvil**. El equipo móvil sale de la red local para descubrir nuevos recursos. Ha de adaptarse a condiciones cambiantes de red y eventualmente funcionar en modo desconectado. Pueden constituirse espontáneamente redes *ad-hoc* que requieren protocolos específicos, como Mobile IP [PER98]. Se hacen patentes las ventajas de las comunicaciones inalámbricas.

Un paso más allá lo constituyen los **sistemas ubicuos** (*pervasive systems*) [COU05 §16]. El vertiginoso desarrollo de la tecnología hardware está llevando la miniaturización de los dispositivos de cómputo a nuevos productos. Los computadores portátiles (que se pueden transportar) se convierten en ubicuos cuando deja de percibirse su presencia (*tecnología que desaparece* [WEI91]). Desde otro punto de vista, cualquier dispositivo adquiere la capacidad de cómputo y de comunicaciones de un computador. Ejemplos actuales o de un futuro cercano de este tipo de dispositivos son los teléfonos móviles, los sistemas de navegación de los automóviles, las tarjetas inteligentes, y otros muchos dispositivos que irán surgiendo. Un entorno ubicuo es por naturaleza cambiante. Los recursos y servicios se descubren y configuran dinámicamente, y las aplicaciones e interfaces de usuario se adaptan al entorno.

---

[2] *Single System Image*, SSI [BAK00].

Además de los problemas planteados por los sistemas distribuidos y la informática móvil, la ubicua ofrece nuevos retos que involucran a numerosos campos tecnológicos: hardware; redes y protocolos de comunicación (en particular para descubrimiento de dispositivos, como Jini[3] y UPnP[4]; interfaces de usuario (con nuevos dispositivos y formas de interacción), y seguridad. Los sistemas ubicuos tienen un amplio rango de aplicación en entornos muy diversos: hogar (domótica), comercio e industria, servicios asistenciales, educación, ocio y turismo, automoción, robótica…

## 1.2 Propiedades de los sistemas distribuidos

Un sistema distribuido que pretenda ofrecer una *visión de sistema único* deberá cumplir las propiedades que se presentan a continuación[5].

### 1.2.1 Transparencia

El objetivo esencial de un sistema distribuido es proporcionar al usuario y a las aplicaciones una visión de los recursos del sistema como gestionados por una sola máquina virtual. La distribución física de los recursos es transparente. Pueden describirse diferentes aspectos de la transparencia:

- De **identificación**. Los **espacios de nombres** de los recursos son independientes de la topología de la red y de la propia distribución de los recursos. De esta forma, una aplicación puede referirse a un recurso con un nombre independientemente de en qué nodo se ejecute (véase la columna derecha de la Figura 1.1). Por ejemplo, en NFS un sistema de ficheros remoto se monta en un punto del sistema de ficheros local. Que una instalación de NFS proporcione transparencia de identificación a las aplicaciones depende de que todos los nodos cliente tengan montado el sistema remoto en el mismo punto de montaje, lo que es generalmente responsabilidad del administrador.

[3] http://:www.jini.org

[4] http://www.upnp.org

[5] Deben entenderse como propiedades deseables. Como veremos reiteradamente, el compromiso con el rendimiento provoca en la práctica que algunas de las propiedades se relajen.

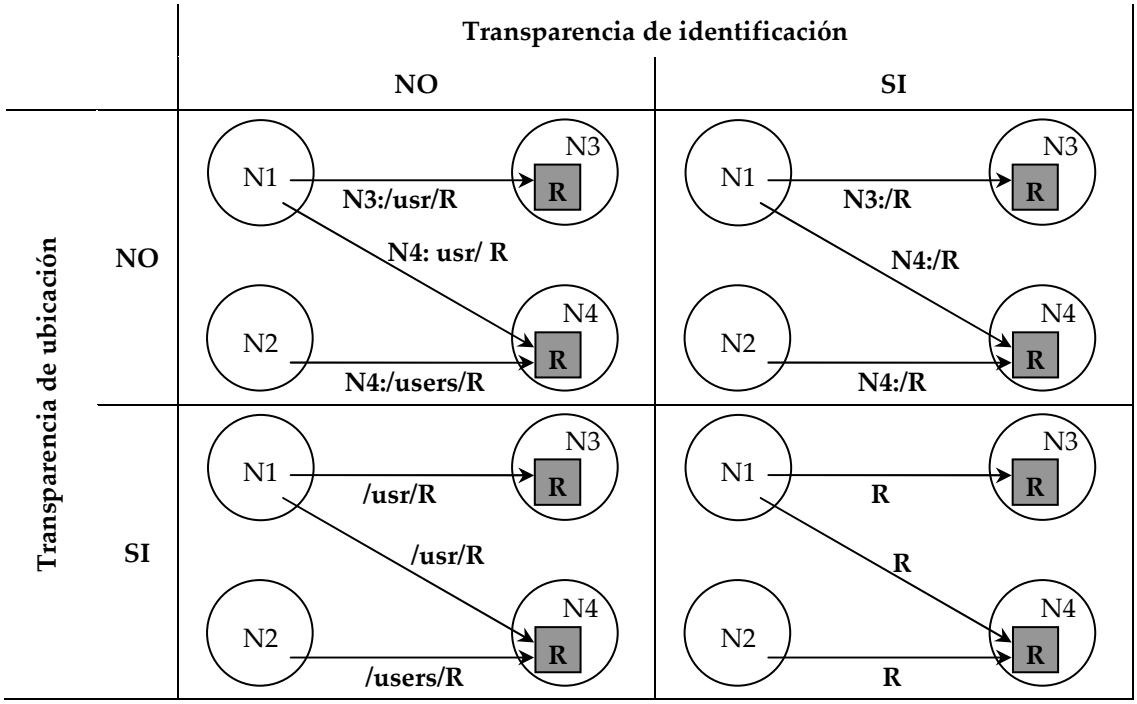


*Figura 1.1. Transparencia de identificación y transparencia de ubicación*

- De la **ubicación** física de los recursos. Ni los usuarios ni las aplicaciones conocen en qué nodo reside el recurso accedido, o si éste es local o remoto. Esto implica también que los recursos pueden **migrar** entre nodos sin que las aplicaciones se vean afectadas. La diferencia entre transparencia de ubicación e identificación se resume en la Figura 1.1. En NFS, podemos migrar un sistema de ficheros remoto de un nodo a otro y la transparencia de la ubicación se preservará si se actualizan convenientemente las tablas de montaje de los nodos cliente.

- De **replicación**. Ni los usuarios ni las aplicaciones conocen cuántas unidades hay de cada recurso, ni si se añaden o eliminan copias del recurso. En NFS los clientes gestionan caches locales de los ficheros remotos. El *caching* es una forma restringida de replicación que requiere alguna forma de validación (en NFS mediante encuesta) por los clientes para preservar la consistencia[6]. Aún así, la semántica UNIX que NFS pretende ofrecer a las aplicaciones puede verse a veces comprometida cuando varios clientes escriben sobre un mismo fichero. Otras formas más generales de replicación, como se verá, son más complejas de gestionar.

[6] Hablaremos más adelante de la *consistencia* como una propiedad específica.

- De **paralelismo**. Una aplicación puede ejecutarse en paralelo, sin que la aplicación tenga que especificarlo, y sin consecuencias sobre la ejecución, salvo por cuestiones de rendimiento. Esta propiedad afecta a los sistemas que permiten distribuir procesos y memoria. En el caso de un sistema de ficheros, sólo es relevante cuando las aplicaciones bloquean temporalmente el acceso a ficheros, lo que se especifica de forma explícita (mediante primitivas de *lock* y *unlock*). Las últimas versiones de NFS incluyen este mecanismo.

- De **compartición**. El que un recurso compartido intente ser accedido simultáneamente desde varias aplicaciones no tiene efectos sobre la ejecución de la aplicación. Como hemos comentado, en NFS esta propiedad puede verse afectada por la existencia de caching, en particular si los periodos de la encuesta de validación son elevados.

- De **rendimiento**. Inevitablemente, implementar las propiedades de los sistemas distribuidos será a costa de una pérdida de rendimiento. Por lo tanto, generalmente es necesario buscar soluciones de compromiso. Así, en NFS la minimización del periodo de validación de la cache permitiría hacer casi totalmente transparente la existencia de caches, a costa de incrementar el tráfico de la red y en consecuencia las latencias.

## 1.2.2 Escalabilidad

Una de las características de los sistemas distribuidos es su **modularidad**, lo que le permite una gran flexibilidad y posibilita su **escalabilidad**, definida como la capacidad del sistema para crecer sin aumentar su complejidad ni disminuir su rendimiento. Uno de los objetivos del diseño de un sistema distribuido es extender la escalabilidad a la integración de servicios.

La escalabilidad presenta dos aspectos. El sistema distribuido debe (1) proporcionar **espacios de nombres** suficientemente amplios, de forma que no supongan una limitación inherente, y (2) mantener un buen nivel de rendimiento en el acceso a los recursos cuando el sistema crece.

- **Espacios de nombres**. Los espacios de nombres, al igual que en los sistemas centralizados, pueden identificar objetos de diferente naturaleza, como ficheros, procesos, variables, o incluso direcciones de memoria (en los sistemas de *memoria compartida distribuida*, *DSM*). En el caso de los espacios lineales, como la memoria, existe una limitación inherente asociada al tamaño del nombre, de forma que hoy en día es razonable plantear la insuficiencia de los espacios de direcciones de 32

bits. En otros casos, los espacios de nombres son jerárquicos y por lo tanto escalables por naturaleza.

- **Complejidad/rendimiento**.[7] El crecimiento de un sistema distribuido puede introducir cuellos de botella y latencias que degradan su rendimiento. Además del incremento de los costes de comunicación por el aumento de la distancia física entre los componentes del sistema, la complejidad estructural de los algoritmos distribuidos es a menudo más que lineal con respecto al tamaño del sistema, como iremos comprobando a lo largo del curso. Es necesario, por tanto, establecer compromisos entre tamaño del sistema, rendimiento y complejidad.

### 1.2.3 Fiabilidad y tolerancia a fallos

La fiabilidad de un sistema puede definirse como su capacidad para realizar correctamente y en todo momento las funciones para las que se ha diseñado. La fiabilidad se concreta en dos aspectos:

- **Disponibilidad**. Es la fracción de tiempo que el sistema está operativo. El principal parámetro para medir la disponibilidad es el tiempo medio entre fallos (MTBF), pero hay que considerar también el tiempo de reparación. La disponibilidad se puede incrementar de dos formas: (a) utilizando componentes de mayor calidad, y/o (b) con un diseño basado en la replicación de componentes que permita al sistema seguir operando aún cuando alguno(s) de ellos falle(n). Ambas alternativas incrementan el coste del sistema; sin embargo, en el estado tecnológico actual, la replicación resulta, en general, menos costosa. Los sistemas distribuidos proporcionan inherentemente la replicación de algunos recursos (por ejemplo, unidades de proceso), mientras que otros normalmente compartidos (por ejemplo, un servidor de ficheros) pueden replicarse para aumentar la disponibilidad. Por otra parte, la ausencia de fallos en los componentes de un sistema, tanto hardware como software, nunca puede garantizarse, de modo que, más allá de unos límites, la replicación es necesaria para seguir incrementando la disponibilidad, ya que la probabilidad de fallo disminuye como una función exponencial de la replicación. Por ejemplo, dada una probabilidad de fallo de un 1% en un componente (en un periodo de tiempo dado), si montamos un sistema

[7] Se refiere tanto a la estructura del sistema como a la de los algoritmos.

replicado con cuatro componentes idénticos, la probabilidad de que fallen en ese periodo los cuatro componentes disminuiría a 0,000001%[8].

- **Tolerancia a fallos**. Aún con una alta disponibilidad, un fallo en un momento determinado puede tener consecuencias desastrosas. Piénsese en sistemas de tiempo real críticos que controlan dispositivos vitales (por ejemplo en medicina, centrales nucleares...). Es decir, aunque la replicación aumenta la disponibilidad, no garantiza por sí sola la continuidad del servicio de forma transparente. La tolerancia a fallos expresa la capacidad del sistema para seguir operando correctamente ante el fallo de alguno de sus componentes, enmascarando el fallo al usuario o a la aplicación. Por lo tanto, la tolerancia a fallos implica (1) detectar el fallo, y (2) continuar el servicio, todo ello de forma transparente para la aplicación (**transparencia de fallos**).

Sobre nuestro ejemplo cabe decir que, al carecer de replicación del servicio, NFS no proporciona tolerancia a fallos. En cambio, se han propuesto otros sistemas de fichero que sí tienen la tolerancia a fallos como objetivo de diseño, como es el caso de AFS [SAT90].

### 1.2.4 Consistencia

La distribución de recursos introduce importantes beneficios. Por una parte, contribuye al incremento del rendimiento a través del paralelismo y promoviendo el acceso a copias locales del recurso (disminuyendo los costes de comunicación). Por otra, como se acaba de ver, la replicación aumenta la disponibilidad, siendo la base para proporcionar tolerancia a fallos. No obstante, distribuir recursos acarrea algunos problemas. Por una parte, la red de interconexión es una nueva fuente de fallos. Además, la seguridad del sistema es más vulnerable ante accesos no permitidos. Pero el problema de mayor complejidad es el de la **gestión del estado global** para evitar situaciones de inconsistencia entre los componentes del sistema. Este es un aspecto fundamental en el diseño del sistema distribuido, por lo que lo comentaremos a continuación.

El problema radica en la necesidad de mantener un estado global consistente en un sistema con varios componentes, cada uno de los cuales posee su propio

[8] Esto debe matizarse: por una parte, puede haber correlación en el fallo de los componentes individuales debida a varias causas, como errores de diseño o catástrofes naturales; por otra parte, la interconexión de los componentes es una fuente de fallos adicional.

estado local. Los nodos del sistema se hallan físicamente distribuidos, por lo que la gestión del estado global depende fuertemente de los mecanismos de comunicación, a su vez soportados por una red sujeta a fallos. Como veremos, la gestión de la consistencia puede basarse en una buena sincronización entre los relojes de los nodos o en mecanismos de ordenación de eventos (relojes lógicos). La distribución física hace, en general, inviable la utilización de un reloj global que aporte referencias absolutas de tiempo, lo que permitiría una ordenación total de los eventos y, por lo tanto, de las transiciones de estado en cada nodo.

Así pues, el mantenimiento de una consistencia estricta requiere un fuerte soporte que implica gran carga de comunicación adicional entre los nodos del sistema, por lo que muchas veces es preferible relajar la consistencia para mantener el rendimiento en un nivel aceptable, de acuerdo a las necesidades de las aplicaciones.

## 1.3 Aplicaciones distribuidas

En primer lugar, hay que distinguir entre aplicaciones distribuidas y aplicaciones paralelas. Una aplicación paralela es aquélla que puede dividirse en tareas que se ejecutan concurrentemente en diferentes elementos de proceso, con el objetivo de disminuir el tiempo de finalización. La mayoría de las aplicaciones pueden ejecutarse en paralelo, ateniéndose a determinados esquemas de cómputo (por ejemplo, siguiendo un modelo *pipeline*) que dependen de la naturaleza de la aplicación y del hardware sobre el que se va a ejecutar. Las tareas de una aplicación paralela se distribuyen entre los elementos de proceso siguiendo criterios como la carga de cada elemento y los costes de comunicación. Puede decirse que la ejecución simultánea de tareas es el objetivo esencial de una aplicación paralela. Las aplicaciones distribuidas presentan motivaciones más diversas y se aplican en entornos más variados.

### 1.3.1 Objetivos de las aplicaciones distribuidas

Si en las aplicaciones paralelas el rendimiento es el objetivo fundamental, una aplicación distribuida puede presentar muy diversas motivaciones:

- **Alto rendimiento**. Una aplicación paralela puede ser también distribuida. Por ejemplo, puede utilizarse una red local para distribuir los procesos de la tarea entre los nodos de la red con el fin de aprovechar los recursos de cómputo disponibles (generalmente PCs de bajo coste) para reducir el tiempo de finalización. Precisamente, este tipo de esquema de cómputo (*computación en cluster*) ofrece hoy una excelente

relación rendimiento/coste y se encuentra en expansión frente a los tradicionales supercomputadores. Aunque hoy en día se siguen utilizando sistemas basados en paso de mensajes (por ejemplo, MPI) con mecanismos de distribución no transparentes, la tendencia es clara hacia la integración transparente de los recursos de cómputo, y en ese sentido están apareciendo nuevos productos (por ejemplo, MOSIX[9]). La memoria compartida distribuida (DSM) es uno de los retos para proporcionar la integración completa. En redes de área amplia se habla de *computación en grid*. En este caso, la disponibilidad de recursos para la aplicación es abierta y abarca unidades de cómputo dispersas que en ese momento están ociosas. Un ejemplo de software para soportar computación en *grid* es el *Globus Toolkit*[10].

- **Tolerancia a fallos**. En otras aplicaciones la distribución viene dictada por criterios como la integridad de la información. Así, en un sistema bancario es preciso mantener replicada la información acerca del estado de las cuentas de los clientes en diferentes servidores, pues el riesgo de perder información por el fallo de una máquina resulta inaceptable por las consecuencias que acarrearía. En estos sistemas es crítico conseguir una actualización consistente de las réplicas. Hoy en día, fundamentalmente por motivos de rendimiento, los sistemas comerciales utilizan técnicas muy conservadoras y poco transparentes, pero, como veremos, las bases teóricas para una gestión transparente de la replicación están bien establecidas.

- **Alta disponibilidad**. Hay aplicaciones donde la distribución se realiza para acercar la información al usuario y disminuir los tiempos de respuesta. En los casos más simples, se utilizan técnicas de replicación que tienen en cuenta la distribución geográfica (*caching* y *mirroring*). La consistencia en la actualización no suele ser un aspecto crítico; en cambio importa mucho la escalabilidad. Hoy en día están muy extendidos los sistemas *peer-to-peer*, caracterizados por su gran escalabilidad al evitar los cuellos de botella del servidor, ofreciendo disponibilidad de recursos de manera prácticamente indiscriminada. Un ejemplo son las redes de distribución de contenidos, como BitTorrent.

- **Movilidad**. La abundancia de dispositivos físicos (ordenadores personales y portátiles, tabletas, teléfonos móviles, etc) introduce una

[9] http://www.mosix.cs.huji.ac.il/

[10] http://www.globus.org

dificultad adicional para el acceso a la información del usuario, de forma que este no tenga que gestionar la actualización de la información en cada dispositivo. Por ejemplo, un mensaje de correo borrado desde el teléfono móvil debería aparecer como borrado cuando posteriormente el usuario acceda a su correo desde un ordenador personal. Se hace imprescindible desligar la información de su soporte, gestionando convenientemente las actualizaciones. Cada vez más se trabaja sobre espacios virtuales de información en vez de sobre dispositivos físicos concretos, que se convierten en meras *caches* del espacio de información del usuario. Así, el usuario se mueve desde un dispositivo a otro y y accede al espacio de su información de forma actualizada y consistente. Ejemplos de productos actuales son *Gmail* de Google para el correo electrónico y *Dropbox* para documentos.

- **Ubicuidad**. A veces los recursos están inherentemente distribuidos. El usuario se mueve en un entorno con recursos (ubicuos) no previstos a priori, y la aplicación trata de ofrecer un comportamiento *inteligente* en función de las necesidades del usuario y la naturaleza y disponibilidad de los recursos. Es el caso de las aplicaciones de *Inteligencia Ambiental* (*AmI*), un campo que está creando enormes expectativas.

## 1.3.2 Entornos para las aplicaciones distribuidas

La disponibilidad de infraestructuras como las citadas anteriormente ofrece diversos escenarios donde pueden desplegarse aplicaciones distribuidas. Puede decirse que, en la actualidad, Internet proporciona el entorno más general, pero hay aplicaciones que tienen sentido en entornos más específicos.

### 1.3.2.1 Internet

Indudablemente, la *World-Wide-Web* (para abreviar, *WWW* o *Web*) es la aplicación estrella de Internet, hasta el punto de que hoy en día es también la forma de acceso común a otras aplicaciones de Internet, como el correo electrónico y la transferencia de ficheros. Se basa en la existencia de una fuerte estructura de comunicaciones, que incluye una infraestructura de enlaces troncales de gran capacidad (*backbones*) donde se conectan las subredes y los ISP (proveedores de servicios de Internet), que proporcionan el acceso a esta infraestructura a los usuarios particulares utilizando medios diversos (por ejemplo, el ubicuo cableado telefónico tradicional). La Web ha impuesto HTTP como protocolo de acceso común.

Las aplicaciones distribuidas que se pueden desplegar en Internet son muy variadas, aunque limitadas en la práctica por aspectos de rendimiento y seguridad. Las llamadas aplicaciones *peer-to-peer* son un ejemplo claro de aplicaciones de alta disponibilidad que se despliegan en Internet. Otro ejemplo son los sistemas para computación en *grid*, que sobre la base de una estructura peer-to-peer ofrecen una gestión transparente del conjunto de recursos distribuidos disponibles en la red. Finalmente, están ganando importancia los servicios de *cloud computing*, que ofrecen recursos en Internet (en *la nube*) para soportar de manera transparente entornos y aplicaciones hasta hace poco se entendían ligados a sistemas locales, como es el caso de repositorios de documentos, agenda o correo electrónico. El objetivo es la *virtualización* de los espacios de información y la movilidad de los usuarios con independencia del dispositivo, como ya hemos comentado.

### 1.3.2.2 Intranets

Una intranet es básicamente un entorno Internet restringido. Se utilizan los mismos protocolos y los mismos medios de acceso que en Internet, si bien el acceso se circunscribe a un dominio administrativo concreto o un conjunto de ellos (por ejemplo, en una empresa). Una intranet puede estar compuesta por varias subredes, y estas pueden estar integradas en Internet, si bien el acceso desde el exterior o la salida desde la intranet pueden estar restringidos por servidores específicos (*firewalls*), que actúan como filtros aplicando criterios a diferentes niveles de la pila de protocolos.

Una intranet admite en principio el mismo tipo de aplicaciones que las que se puedan desplegar en Internet. Además, las aplicaciones de alto rendimiento, como es el caso del la computación en cluster, o las de tolerancia a fallos, suelen tener más sentido sobre una intranet, (o sobre una LAN de la intranet), ya que, en general, los costes de comunicación pueden acotarse.

### 1.3.2.3 Entornos ubicuos

En un sistema ubicuo las aplicaciones no están sujetas a ámbitos administrativos o de red concretos, como las intranets, pero tampoco operan de manera ilimitada en el ámbito de Internet. Así, el dispositivo de un usuario (por ejemplo el teléfono móvil de un futuro cercano) opera en el entorno *físico* concreto en el que se encuentra (por ejemplo, el domicilio del usuario o un aeropuerto) de acuerdo a los servicios que descubre en ese entorno, y se adapta a la infraestructura disponible de la forma más eficiente posible. Por ejemplo, una misma llamada telefónica de un usuario que sale desde su domicilio al aeropuerto podría utilizar infraestructuras diferentes: la línea ADSL en el

domicilio a la que el teléfono se conecta mediante un punto de acceso al que accede por Bluetooth; telefonía móvil ofrecida por un operador durante el trayecto al aeropuerto, o un servicio de acceso a Internet ofrecido por otro operador mediante WiFi, más barato que el de telefonía móvil, una vez que llega al aeropuerto. Los modos de comunicación conmutarían dinámicamente, optimizando la calidad y el coste de la comunicación. Obsérvese que para conseguir este tipo de operación un prerrequisito es que el dispositivo del usuario soporte varios modos de comunicación inalámbrica entre él y la infraestructura ofrecida por el entorno físico. Sin embargo, el mayor reto es soportar los cambios de modo de forma transparente[11].

Por supuesto, un entorno ubicuo también puede proporcionar acceso a servicios de Internet o de intranet, por lo que este tipo de entornos puede considerarse el más general, y, como se ha comentado, plantean grandes retos en cuanto a infraestructuras, dispositivos hardware (miniaturización, autonomía…) y estandarización, y ofrecen grandes expectativas (aplicaciones *AmI*).

# 1.4 Soporte hardware

Distinguiremos aquí entre los dispositivos que soportan el cómputo en un sistema distribuido y la infraestructura de red que permite su interconexión.

## 1.4.1 Elementos de cómputo

La definición de sistema distribuido que hemos proporcionado antes (conjunto de computadores interconectados que comparten un estado) es lo suficientemente ambigua como para plantear discusiones, por ejemplo (1) qué se entiende por computador, y (2) cómo se definen los estados. Para responder a estas preguntas necesitamos conocer las alternativas en materia de arquitecturas de computadores. Atendiendo a los criterios clásicos de Flynn, las arquitecturas se pueden clasificar según permitan paralelismo de datos y/o instrucciones, obteniéndose los cuatro grupos de la Tabla 1.1.

Sobre esta clasificación, cabría matizar que los procesadores actuales (segmentados y superescalares) realizan también de alguna forma proceso paralelo. Pero, de acuerdo a nuestro propósito de concretar qué arquitecturas pueden soportar *distribución* (de cualquier tipo de recurso), hemos de centrarnos únicamente en la categoría MIMD, máquinas con varias unidades de

[11] A los sistemas que proporcionan este tipo de transparencia se les denomina *sin costuras* (*seamless*).

proceso[12]. Por generalidad, introduciremos el término **nodo** para referirnos a cada una de las unidades de proceso.

| | un dato simultáneamente | múltiples datos simultáneamente |
|---|---|---|
| una instrucción simultáneamente | SISD<br>arquitecturas Von Neumann clásicas | SIMD<br>procesadores vectoriales |
| múltiples instrucciones simultáneamente | MISD<br>no hay implementaciones | MIMD<br>multiprocesadores, multicomputadores, redes |

*Tabla 1.1. Clasificación de Flynn*

La categoría MIMD suele subdividirse en subgrupos atendiendo a dos criterios: el grado de acoplamiento (si cada nodo cuenta con su propio espacio de direcciones de memoria física o no), y cómo se conectan los nodos (bus compartido o red de interconexión). La Tabla 1.2 muestra a qué tipos de sistemas conduce esta clasificación.

| | Grado de acoplamiento | |
|---|---|---|
| | espacio de memoria física compartido | espacios de memoria física independientes |
| bus compartido | servidores multiprocesador típicos | multicomputadores, redes LAN |
| red de interconexión | multiprocesadores UMA y NUMA | multicomputadores, redes WAN |

*Tabla 1.2. Tipos de arquitecturas MIMD*

De nuevo, sobre esta clasificación cabe plantear si todos los subgrupos admiten en general distribución de recursos. Para afinar un poco más, identificaremos tres grupos de elementos susceptibles de distribuirse entre los nodos:

- Proceso
- Espacio de memoria

[12] Entenderemos como unidad de proceso el elemento que es capaz de ejecutar instrucciones máquina, es decir, el *procesador* en terminología clásica.

- Espacio de E/S (ficheros y dispositivos)

En los sistemas multiprocesador la memoria y la E/S son recursos por definición centralizados. En estas máquinas sólo tendría sentido hablar de proceso distribuido. Los nodos comparten el estado de la E/S y de la memoria, la distancia física entre procesadores es pequeña y el reloj es único. En los multiprocesadores actuales, cada procesador posee su memoria cache propia, pero la coherencia está garantizada por protocolos hardware. Un **sistema operativo multiprocesador** no introduce aparentemente grandes novedades[13]. La memoria, la E/S y los ficheros se gestionan como en los sistemas operativos tradicionales; sólo la gestión de los procesadores y la asignación de procesos plantean nuevas necesidades. Por todo ello, hoy en día los multiprocesadores no se consideran sistemas distribuidos.

Por el contrario, en multicomputadores y redes, todos los recursos, incluido el reloj, son (o pueden ser) distribuidos[14]. Es necesario implementar mecanismos y servicios que aseguren estados globales consistentes y proporcionen la visión de sistema único en un grado aceptable. En la práctica, la mayoría de los sistemas distribuidos de hoy en día se basan en elementos de cómputo de consumo, fundamentalmente PCs, y la tendencia es, como hemos visto, que los elementos de cómputo se integren en todo tipo de dispositivos, como teléfonos o cámaras (o, desde otro punto de vista, que todo tipo de dispositivo adquiera capacidad para servir de nodo en un sistema distribuido), y que estos elementos se integren en el sistema mediante algún tipo de infraestructura de red estándar.

## 1.4.2 Infraestructura de red

Como se ha comentado más arriba, hoy en día los sistemas distribuidos se basan en la existencia de una red de comunicaciones que conecta dispositivos de cómputo estándares. Actualmente, el mundo industrializado cuenta con amplias infraestructuras troncales (*backbones*), normalmente de fibra óptica, lo que ha permitido el despliegue de Internet. También se utilizan enlaces por satélite para el acceso a lugares recónditos. El problema de acceso a los puntos de usuario (redes de área metropolitana, MAN) se ha estado solucionando mediante la utilización eficiente del viejo cableado telefónico. Para redes locales

[13] Esto es sólo en apariencia. Un soporte eficiente para el multiproceso tiene implicaciones importantes en la estructura del sistema operativo.

[14] En multicomputadores, el espacio de E/S puede ser tanto único como distribuido, pero por generalidad asumiremos lo último.

específicas se ha venido utilizando Ethernet, que permite velocidades muy altas (hasta 1 Gbps).

En los últimos años han experimentado un gran avance las tecnologías inalámbricas, hasta el punto de que pueden competir (es el caso de WiFi IEEE 802.11) con Ethernet en las redes locales. También hay soluciones para ámbitos geográficos de mayor rango. Finalmente, en sistemas ubicuos adquieren importancia las redes inalámbricas de corto alcance (redes de ámbito personal, PAN). La Tabla 1.3 ilustra algunas características de las tecnologías de red actuales.

| | Tipo | Ejemplo | Rango típico | Ancho de banda |
|---|---|---|---|---|
| Redes cableadas | PAN | USB | 1 m | 12 – 480 Mbps |
| | LAN | Ethernet | 1 Km | 10 – 1000 Mbps |
| | MAN | ATM | 10 Km | 1 – 150 Mbps |
| | WAN | Internet | Todo el mundo | 0,5 – 600 Mbps |
| Redes inalámbricas | PAN | Bluetooth | 10 m | 0,5 – 2 Mbps |
| | LAN | WiFi | 100 m | 2 – 54 Mbps |
| | MAN | WiMAX | 10 Km | 1,5 – 20 Mbps |
| | WAN | HDSPA | Todo el mundo | hasta 14 Mbps |

*Tabla 1.3. Ejemplos de tecnologías de red*

# 1.5 Soporte del sistema operativo

Hemos acotado el concepto de sistema distribuido como aplicable a aquellos sistemas compuestos por nodos con espacios propios de memoria y E/S. Cada nodo posee su propio sistema operativo y los servicios de red básicos. Esta descripción concuerda con la de las redes de computadores, pero también puede incluir los multicomputadores.

En general, los nodos serán heterogéneos en cuanto al hardware y al sistema operativo. Un primer paso es proporcionar un conjunto de especificaciones que permitan definir interfaces comunes para construir sobre ellas los servicios del sistema distribuido y proporcionar la visión de sistema único.

En lo que se refiere a la distribución de recursos, cabe pedir del sistema operativo (1) compatibilidad, para permitir el desarrollo de los servicios distribuidos, y (2) que proporcione la flexibilidad adecuada para ubicar y gestionar los servicios eficientemente. Sin embargo, los sistemas clásicos, como UNIX, monolíticos, no están diseñados de acuerdo a estos objetivos. Ofrecen

una interfaz de llamadas al sistema única, integran en el núcleo todo tipo de servicios y proporcionan políticas de gestión predeterminadas.

El enfoque clásico para construir un sistema distribuido en una red es instalar sobre el sistema operativo de cada máquina los mecanismos necesarios para proporcionar el acceso a recursos remotos según el esquema cliente-servidor. Este enfoque proporciona escasa flexibilidad, ya que el servidor de un recurso requiere habitualmente ejecutar un sistema operativo determinado. Por otra parte, en el lado del cliente, habitualmente se requiere la ejecución de aplicaciones que deben ser soportadas por plataformas específicas. Un esquema así de rígido conduce a asociar máquinas concretas a servicios y aplicaciones específicos, en detrimento de la flexibilidad y la transparencia en la ubicación.

Una solución para permitir que cualquier nodo de una red soporte indistintamente cualquier servicio o aplicación es la **emulación** de un sistema huésped sobre un sistema operativo anfitrión. Esto proporciona **personalidad múltiple** a cada nodo[15], permitiendo la coexistencia de aplicaciones y servicios, bien en modo nativo, bien emulados. Existen productos para Windows y Linux, como *VMware, VirtualPC* y *win4lin*, que proporcionan la emulación de un entorno hardware sobre el que se instala el sistema operativo huésped. El éxito de estos productos reside en que utilizan una técnica, la *virtualización*, que permite ejecutar la mayor parte del código en modo nativo, emulando únicamente las llamadas al sistema operativo huésped, por lo que la fuerte pérdida de rendimiento inherente a la emulación puede ser asumida por las máquinas de hoy en día.

Sin embargo, en los primeros años de los sistemas distribuidos la emulación hardware era prácticamente inasumible, y el propio hecho de tener que almacenar y cargar varios sistemas operativos completos, demasiado oneroso. Todo ello condujo hace unos 25 años a una revisión de la estructura del sistema operativo. La premisa era sacar del espacio del sistema operativo los servicios susceptibles de ser distribuidos, junto con sus políticas de gestión, dejando dentro, para su ejecución en modo protegido, únicamente las funciones básicas de control del hardware (gestión básica de memoria e interrupciones) y el soporte para los cambios de contexto y la comunicación. Este núcleo de sistema operativo se denomina **micronúcleo**, y proporciona la base adecuada para la distribución de servicios, al permitir la implementación selectiva sobre él, en el espacio de usuario y de acuerdo al modelo cliente-servidor, de los servicios

[15] En la práctica esto implica que se pueden abrir varias ventanas con diferentes sistemas operativos en un mismo terminal.

específicos que ofrece el sistema, como módulos independientes. La interfaz de llamadas al sistema se implementa como un subsistema en espacio de usuario, lo que permite, de manera natural y con escaso overhead, ofrecer personalidad múltiple y soportar concurrentemente aplicaciones y servicios de diferentes sistemas operativos. El ejemplo más extendido de micronúcleo es el Mach 3.0, base de algunos sistemas comerciales tipo UNIX, como OSF/1, o el actual OS X de Apple para Macintosh.

Cada vez más frecuentemente los dispositivos móviles actuales (en particular los denominados *smart phones*) soportan versiones reducidas de los sistemas operativos comerciales (Mac, Windows), o sistemas específicos (como Android, de Google) con prestaciones nada desdeñables. Actualmente estos sistemas tienen capacidad para soportar una máquina virtual Java, un navegador Web, aplicaciones multimedia, y controlan una variedad de dispositivos, como una pantalla táctil, una cámara, un receptor GPS, y todo tipo de interfaces de comunicación inalambrica lo que los hace extraordinariamente versátiles. Este sector es sin duda el escenario más candente para el desarrollo de la tecnología de los sistemas operativos.

Finalmente, en el extremo de la miniaturización, se están proponiendo dispositivos autónomos dotados de sensores y capacidad de comunicación inalámbrica pensados para configurarse en *redes de sensores*. Cada uno de estos dispositivos (*motas*), diseminados de manera más o menos aleatoria en entornos diversos (por ejemplo, un ecosistema), participan tanto de la recolección de información como del encaminamiento de los mensajes de las motas cercanas. Existen sistemas operativos específicos para soportar estas funciones con un consumo de energía mínimo, como TinyOS[16].

## 1.6 Soporte para la comunicación

Hay que diferenciar entre distribución física de la memoria[17] y **modelo de comunicación**. El grado de acoplamiento determinará el soporte necesario para implementar el modelo de comunicación, pero éste puede estar basado tanto en memoria compartida como en paso de mensajes. El modelo elegido determina

[16] http://www.tinyos.net

[17] La generalización de las memorias cache locales a cada procesador hace que, hoy en día, todas las máquinas MIMD presenten alguna forma de distribución física de la memoria, si bien los multiprocesadores, donde la coherencia cache está garantizada por el hardware, los estamos considerando como de memoria compartida.

parte de las características del sistema distribuido, pero la distribución física de la memoria queda ocultada, salvo a efectos del rendimiento.

## 1.6.1 Modelos de comunicación

Sobre memoria compartida, los mecanismos de variables compartidas cuyo acceso se sincroniza mediante cerrojos de exclusión mutua u otras primitivas (variables condición, cerrojos de lectores-escritores), han sido estudiados desde hace mucho tiempo y están perfectamente establecidos. Lo mismo cabe decir sobre el paso de mensajes, que implementa colas FIFO sobre buffers en memoria, a menudo integrados en el sistema de ficheros (*pipes* de UNIX).

Sobre sistemas débilmente acoplados (habitualmente redes), el paso de mensajes parece el mecanismo de comunicación *natural*. En un sistema distribuido, las necesidades de comunicación conducen a utilizar esquemas específicos de gestión de los recursos para los que el paso de mensajes resulta adecuado. Un recurso estará a cargo de un proceso gestor del recurso, con quien deberá comunicarse cualquier proceso que pretenda acceder al recurso siguiendo un esquema **cliente-servidor**, que permite expresar el acceso a servicios mediante un protocolo de petición-respuesta. El modelo cliente-servidor se puede implementar mediante un mecanismo de paso de mensajes específico, como por ejemplo la interfaz de *sockets* de UNIX, que se apoya en los protocolos TCP/IP o UDP/IP para comunicar procesos en un sistema en red.

Pero para espacios de memoria físicos independientes también se han diseñado interfaces que permiten manejar un espacio de direcciones común sobre un sistema de memoria física distribuida: sistemas de **memoria compartida distribuida** (DSM). La idea es ofrecer a las aplicaciones diseñadas según un modelo de memoria compartida un soporte para su ejecución en un entorno distribuido.

Con el objetivo de disminuir la diferencia semántica entre programas que usan servicios locales (mediante llamadas al sistema) y programas que usan servicios remotos, se han desarrollado mecanismos que encapsulan los detalles de direccionamiento y sincronización del envío/recepción de la petición/respuesta. Así, en los lenguajes procedimentales, las **llamadas a procedimientos remotos**, **RPC**s, proporcionan una sintaxis de llamada a función como interfaz para el acceso a servicios. Se incluye como Apéndice una breve descripción del mecanismo de RPCs. En los lenguajes orientados a objetos, cada recurso es un objeto identificado unívocamente en el sistema distribuido, al que se accede invocando los métodos de acceso definidos para esa clase de objeto. El acceso a

los objetos se realiza mediante una interfaz de **invocación de métodos remotos** (por ejemplo, RMI de Java). A RMI se le dedica también un Apéndice.

La Tabla 1.4 resume los mecanismos que implementan los modelos de comunicación sobre una u otra arquitectura.

La Figura 1.2 resume la estructura de la comunicación en un sistema distribuido. Los diferentes esquemas de programación se soportan por la interfaz de paso de mensajes característica de los sistemas en red. A continuación repasaremos brevemente los conceptos básicos relacionados con la comunicación basada en paso de mensajes.

| | Grado de acoplamiento | |
|---|---|---|
| Modelo de comunicación | Espacio de memoria física compartido | Espacios de memoria física independientes |
| Memoria compartida | Variables compartidas | Memoria compartida distribuida (DSM) |
| Paso de mensajes | Pipes tipo UNIX | Sockets |
| Semánticas de lenguajes de alto nivel | Llamadas a funciones o procedimientos, invocación de métodos sobre objetos | RPC, RMI |

*Tabla 1.4. Mecanismos de comunicación.*

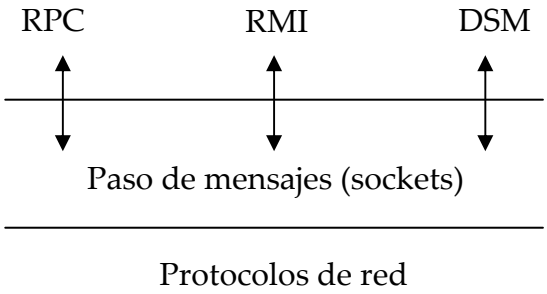


*Figura 1.2: Estructura de la comunicación en un sistema distribuido.*

## 1.6.2 Comunicación basada en paso de mensajes

El paso de mensajes comprende un conjunto de mecanismos que permiten comunicar procesos mediante un enlace o canal de comunicación, bien

directamente, identificando el proceso origen o destino respectivamente en las primitivas de *recibir* o *enviar*, bien a través de un buzón, que identifica explícitamente el canal de comunicación. En sistemas tipo UNIX, el mecanismo básico de paso de mensajes dentro de un nodo es el de *pipes*, con o sin nombre. Para comunicar procesos entre nodos surge el problema del direccionamiento. A este respecto, los *sockets* de UNIX soportan dos formas de comunicación alternativas: identificando un socket dentro del sistema de ficheros (entorno UNIX) o asociando un **puerto** de comunicación en un nodo concreto (entorno Internet). En este último caso se hace precisa la gestión explícita del direccionamiento.

Aunque el modelo de programación del paso de mensajes para comunicación entre nodos no difiere del de paso de mensajes para comunicación dentro de un nodo, el hecho de tener que confiar en los protocolos de red introduce dependencias derivadas del rendimiento y la fiabilidad del canal de comunicación entre nodos.

Las características principales asociadas a un canal de comunicación son las siguientes:

- **Modo de sincronización**. Por una parte, si el canal de comunicación está ocupado, la primitiva de enviar puede bloquear al proceso hasta que se libere el canal (lo que depende también de si permite **buffering** o no) y pueda depositar el mensaje (modo **bloqueante** o síncrono). La alternativa (modo **no bloqueante**) permite que el proceso continúe aunque el mensaje no se haya podido enviar, transfiriendo a la aplicación la responsabilidad de gestionar la sincronización en el uso del buffer de usuario donde se ubica el mensaje enviado. Estos modos se aplican también a la primitiva de recibir. Los sockets de UNIX son en principio bloqueantes, pero pueden configurarse como no bloqueantes.

- **Fiabilidad**. En lo referente a la fiabilidad de la comunicación, el mecanismo de paso de mensajes depende del soporte que le proporcione la red. Si se implementa sobre un protocolo de transporte *seguro*, la comunicación se considera **fiable**, en el sentido de que el emisor puede confiar en que el receptor acabe por recibir el mensaje correctamente o sea informado de lo contrario. Esto es a costa de una cierta sobrecarga por la necesidad que tiene el protocolo de confirmar las recepciones. En cambio, un mecanismo **no fiable** permitirá una comunicación menos costosa, pero delega en la aplicación la responsabilidad de verificar la corrección de la comunicación. UNIX ofrece dos formas de comunicación

con sockets: comunicación *orientada a conexión*, basada en TCP/IP, fiable, y comunicación por *datagramas*, basada en UDP/IP, no fiable.

- **Modo de comunicación**. En sistemas distribuidos es de gran interés el soporte de primitivas de paso de mensajes de 1:N. El **broadcast** o difusión permite enviar un mensaje a todas las direcciones accesibles por el emisor, y se usa en redes locales. Un caso particular de broadcast, el **multicast**, permite seleccionar un subconjunto de direcciones a las que enviar el mensaje. El soporte para multicast es muy útil en sistemas replicados, como se verá más adelante. Como ejemplo, el protocolo IP reserva un conjunto de direcciones para multicast.

El desarrollo de Internet está forzando la evolución de los protocolos de nivel de red. Así, el protocolo IPv4, cuyas direcciones de 32 bits suponen un problema de escalabilidad, está dejando paso al IPv6, que especifica direcciones de 128 bits.

Internet y los nuevos paradigmas de cómputo están imponiendo nuevos estilos de comunicación. De este modo, se aprecia una tendencia general a la utilización de HTTP como protocolo de acceso a servicios, como evolución del esquema RPC clásico. Por otra parte, la distribución de servicios ha provocado una evolución del esquema cliente-servidor clásico hacia estructuras *peer-to-peer*, donde los roles de cliente y servidor son intercambiables. En tales sistemas, habitualmente dinámicos, los nodos indistintamente ofrecen y solicitan servicios, y eventualmente cooperan en la búsqueda de servicios y en el encaminamiento de peticiones.

## 1.7 Sistemas abiertos

Un problema fundamental de la integración de los recursos en un sistema distribuido es la **heterogeneidad** de los sistemas que lo componen, que afecta a la comunicación de datos y a la compatibilidad de aplicaciones y usuarios. Una base esencial para la construcción de sistemas distribuidos sobre sistemas heterogéneos es la concepción de los componentes como **sistemas abiertos**. Un sistema abierto es aquél que ofrece una **especificación pública** de su interfaz, que además debe ser asumida por los fabricantes. La difusión del sistema, normalmente sujeta a las leyes del mercado, determina en gran medida que el sistema abierto se considere efectivamente como tal o no. En este sentido, han fracasado propuestas de sistemas abiertos *oficiales*, definidos por instituciones y grandes consorcios, y, en cambio, otros sistemas de concepción más modesta han llegado a convertirse en **estándares de hecho**, como por ejemplo es el caso del protocolo TCP/IP. Por el contrario, para los **sistemas cerrados**, el fabricante

no proporciona la especificación de sus interfaces, por lo que están limitados a configuraciones propietarias.

Los sistemas abiertos poseen las siguientes propiedades [QUA93]:

- **Interoperabilidad**. Capacidad de mover información entre máquinas a través de la red. Proporcionada por los protocolos de comunicación (p. ej. TCP/IP) y lenguajes de especificación (p. ej. XML) estándares.

- **Transportabilidad de aplicaciones**. Capacidad de mover programas de aplicación entre máquinas del sistema. Los sistemas operativos abiertos proporcionan una interfaz común basada en estándares oficiales (por ejemplo, POSIX) o de hecho, de forma que las aplicaciones son transportables a nivel de código fuente[18].

- **Transportabilidad de usuarios**. Capacidad de permitir que un usuario pueda acceder al sistema desde diferentes máquinas sin necesidad de conocer características particulares de cada máquina. Las interfaces gráficas de usuario, por su propia naturaleza, proporcionan en gran medida esta propiedad, aun pudiendo ser diferentes en diferentes máquinas.

Así pues, los protocolos de red y el sistema operativo determinan fundamentalmente la característica de sistema abierto. Gracias a que las especificaciones son públicas, sobre sistemas de componentes heterogéneos los desarrolladores pueden implementar una o más capas de software (a menudo llamado *middleware*) que proporcione la visión de sistema único. Como vamos a ver, se han diseñado esquemas de comunicación y estructuras de sistemas operativos adecuadas para definir sistemas abiertos y soportar eficientente la distribución de recursos.

Según hemos definido, los sistemas abiertos no garantizan la transportabilidad de aplicaciones a nivel de código ejecutable. El código fuente debe compilarse para cada arquitectura particular. Existe, sin embargo, la alternativa de la **interpretación** directa del código fuente, a costa de una notable pérdida de rendimiento. En este sentido, el lenguaje Java se ha convertido en un estándar de hecho.

[18] Obsérvese que las aplicaciones Java son transportables con independencia del sistema operativo, gracias a que una capa de middleware sobre el sistema operativo (la JVM) proporciona la interfaz estándar a las aplicaciones.

También se ha afrontado el problema de comunicar componentes software escritos en diferentes lenguajes, mediante la definición de interfaces en cada extremo y proporcionando un formato común. Los **lenguajes de definición de interfaces** (IDL) ofrecen una notación para definir interfaces mediante la especificación del formato de entrada y salida de cada argumento. Ejemplos de este tipo de lenguajes son XDR de Sun para llamadas a procedimientos remotos y CORBA IDL para invocación de métodos remotos. En los últimos años se están imponiendo como estándares de hecho lenguajes como HTML y XML para la descripción de dispositivos y servicios y la especificación de interfaces.

Internet está imponiendo nuevos estándares y estilos de comunicación. De este modo, se aprecia una tendencia general a la utilización de HTTP como protocolo de acceso a servicios, como evolución del esquema RPC clásico. Por otra parte, la distribución de servicios ha provocado una evolución del esquema cliente-servidor clásico hacia estructuras *peer-to-peer*, donde los roles de cliente y servidor son intercambiables. En tales sistemas, habitualmente dinámicos, los nodos indistintamente ofrecen y solicitan servicios, y eventualmente cooperan en la búsqueda de servicios y en el encaminamiento de peticiones.

# 1.8 Resumen

El objetivo de un sistema distribuido es integrar los recursos y servicios conectados por una red de comunicación. Desde el punto de vista del usuario y de las aplicaciones, un sistema distribuido proporciona una visión de máquina única y no difiere de uno centralizado (Figura 1.3). En cambio, el punto de vista del diseñador (el sistema como gestor de los recursos) la estructura interna está condicionada por la distribución física de los recursos (Figura 1.4).

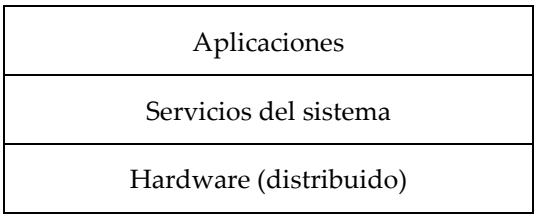


*Figura 1.3: Un sistema distribuido desde el punto de vista del usuario.*

Lo habitual es que el sistema operativo integre los servicios de red, que ofrecen protocolos abiertos de comunicación, como es el caso de TCP y UDP. Sobre estos se disponen los soportes adicionales para la comunicación distribuida, como es el caso de RPC, RMI o DSM, y los servicios específicos que proporcionan las propiedades del sistema distribuido (servicios *middleware*),

como es el caso de la gestión de tiempos, eventos y estado global, sobre los que se asientan las aplicaciones.

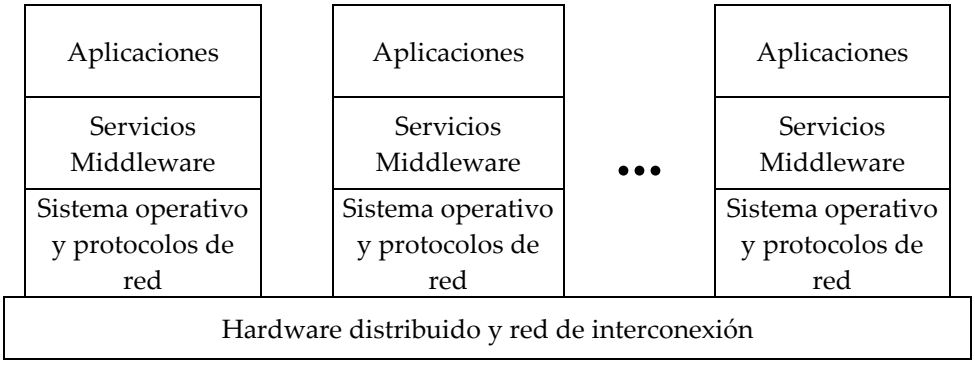


*Figura 4: Estructura de un sistema distribuido.*

## 1.9 Ejercicios

**1** La herramienta *ftp* ¿proporciona transparencia en la identificación y/o ubicación de los ficheros? Explica por qué. Pon algún ejemplo de herramientas y comandos o llamadas al sistema que no proporcionen estas propiedades de transparencia. Pon ejemplos de herramientas que sí las proporcionen.

**2** El protocolo IPv4 es muy limitado en cuanto a número de direcciones que proporciona. Documéntate y comenta cómo IPv4 consigue extender la escalabilidad en base a convenciones de uso.

**3** Explica mediante un ejemplo cómo mediante virtualización se puede obtener transparencia en la ubicación de un servicio.

**4** Enumera los protocolos estándar que han tenido que adoptarse para que un usuario que usa su PC en la cafetería del campus pueda acceder a una página web cualquiera. Utiliza Internet para documentarte.

# Apéndice A: **Llamadas a procedimientos remotos**

El paso de mensajes permite expresar el modelo cliente-servidor explicitando un protocolo de petición-respuesta sobre el servicio al que se desea acceder. Nótese que este protocolo se basa en una semántica orientada a la entrada-salida. En cambio, el acceso a recursos en los sistemas operativos tradicionales, mediante la interfaz de llamadas al sistema, se basa en una semántica de llamada-retorno a funciones. Siendo la transparencia en la ubicación de los recursos un objetivo fundamental en los sistemas distribuidos, resulta evidente que el paso de mensajes no proporciona la base semántica adecuada para el acceso a los recursos remotos.

El mecanismo de **llamada a procedimiento remoto** (RPC), desarrollado en 1984 por Sun Microsystems, trata de eliminar ese salto semántico. Un proceso cliente especifica el acceso a un servicio mediante una sintaxis de llamada a función, por ejemplo:

*resultado= servicio_rpc (parámetros)*

El nombre de la función puede identificar el servicio o ser genérico, en cuyo caso la interfaz de RPCs reconocerá el servicio por alguno de los parámetros. Análogamente, el servidor que detenta el servicio utiliza el mecanismo de RPCs para servir las peticiones mediante una función que le permite bloquearse a la espera de peticiones. El procedimiento completo para la ejecución de una RPC es el siguiente (Figura 1.A1):

(1) Identificado como remoto el servicio solicitado, el mecanismo de RPC[19] invocado por el proceso cliente localiza el servidor que lo soporta.

(2) Se empaquetan los argumentos en un formato estándar (por ejemplo, XDR[20]) para formar el cuerpo de un mensaje. Este proceso se denomina **serialización** (*marshaling*).

(3) Se utiliza la interfaz de red del sistema operativo para enviar un mensaje al servidor que contiene la petición del servicio (por ejemplo, mediante UDP/IP).

[19] Se le denomina *stub*. Viene a ser la función sustituta o representante en el cliente del procedimiento remoto.

[20] EXternal Data Representation. Se ha convertido en un estándar de hecho.

(4) El proceso cliente queda bloqueado en la recepción de la respuesta.

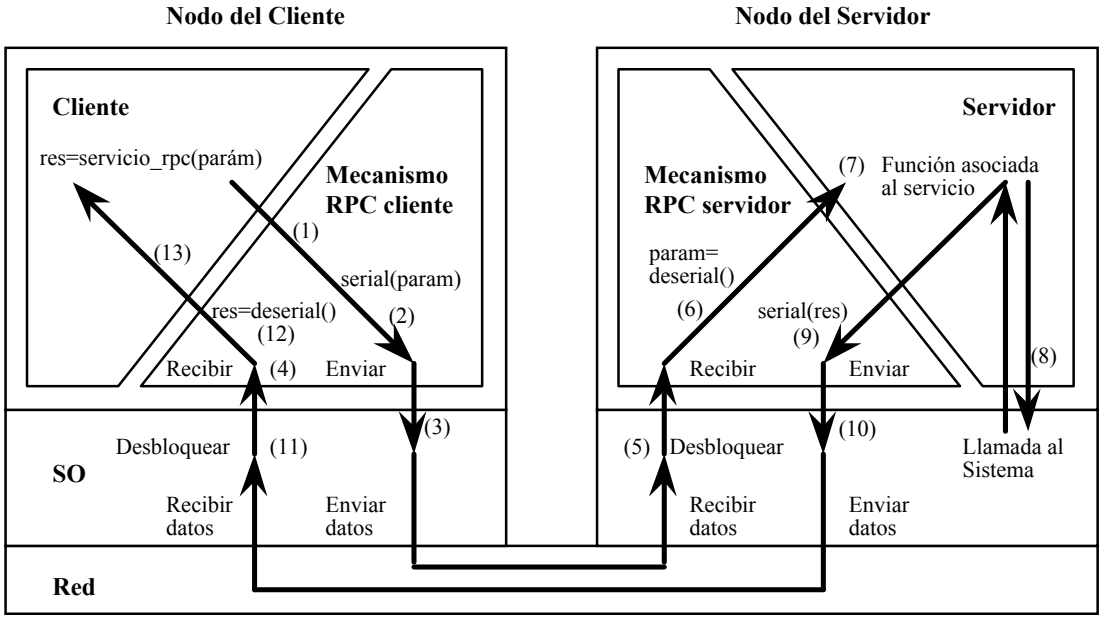


*Figura 1.A1. Ejecución de una RPC.*

(5) En el nodo destino, con la recepción del mensaje, el sistema operativo desbloquea al servidor. Si éste es multithread, un hilo de ejecución se hará cargo de la petición.

(6) Se ejecuta el mecanismo de RPCs de la función de espera del servidor, que desempaqueta el mensaje (**deserialización**) para obtener los parámetros de la petición de servicio y la identificación del origen.

(7) El mecanismo de RPCs ejecuta la función asociada al servicio solicitado, que habrá sido instalada por el servidor en su inicialización.

(8) La función del servidor se ejecuta mediante llamadas al sistema local.

(9) Cuando termina la función asociada al servicio, el mecanismo de RPCs serializa el resultado.

(10) Se envía el mensaje con el resultado al cliente.

(11) El sistema operativo del nodo del cliente desbloquea a éste en la recepción del resultado.

(12) El mecanismo de RPCs del cliente ejecuta la deserialización del resultado.

(13) Devuelve el valor al programa.

Nótese que el servidor estará bloqueado en la ejecución de un procedimiento de espera. La interfaz de RPCs suministra una función específica para ello. En la fase de inicialización, el servidor habrá instalado la(s) función(es) asociadas al (a los) servicio(s).

El nivel semántico de una librería de RPCs puede ser más o menos alto. RPC de Sun distingue tres niveles de acceso. En el nivel superior se suministran una serie de funciones y programas *normales*. A nivel intermedio se suministra una función genérica de llamada para el cliente, *callrpc()*, y dos para el desarrollo de servidores. En el nivel de acceso inferior se pueden encontrar funciones específicas que permiten configurar aspectos básicos de la comunicación, como llamadas asíncronas (no bloqueantes), y conexión permanente entre cliente y servidor (*persistent binding*).

# Apéndice B: **Invocación de métodos remotos**

La invocación de métodos remotos (RMI) es a la programación orientada a objetos lo que las RPC a la programación procedimental. Vamos a revisar primero los conceptos que se manejan en la invocación de métodos del modelo de objetos general, centrándonos después en el modelo de objetos distribuidos.

## El modelo de objetos

En los lenguajes de programación orientada a objetos, como C++ y Java, un **objeto**, que encapsula un conjunto de datos y de **métodos**, se comunica con otros objetos invocando los métodos de éstos, que en general aceptan argumentos y devuelven resultados. Aunque estos lenguajes proporcionan formas para permitir referenciar directamente las variables de los objetos, en el modelo de objetos distribuidos, que veremos luego, las variables sólo pueden accederse a través de métodos.

Los objetos se acceden por referencia[21]. En la invocación de un método de un objeto (que denominaremos objeto *receptor*), hay que proporcionar la referencia al receptor, el método y los argumentos de la invocación. Las referencias a objetos pueden asignarse a variables, pasarse como argumentos o devolverse como resultados de una invocación.

Una **interfaz** define el formato de acceso a un conjunto de métodos, es decir, sus nombres, tipos de los argumentos, valores de retorno y excepciones (pero no la implementación de los métodos).

En la invocación de un método, el objeto receptor ejecuta el método y devuelve el control al objeto invocante, devolviendo eventualmente un resultado. Como consecuencia, el estado del receptor puede cambiar. También pueden desencadenarse invocaciones a otros objetos. Durante la ejecución del método pueden producirse condiciones de error (**excepciones**).

## El modelo de objetos distribuidos

En un sistema distribuido, los objetos de las aplicaciones pueden estar repartidos entre los nodos del sistema. Como ocurre en las RPC, la **invocación de métodos remotos** sigue habitualmente una arquitectura cliente-servidor,

[21] Este concepto de *referencia* no debe confundirse con el de *puntero a memoria* de C.

donde los objetos receptores se gestionan por los servidores de estos objetos. La invocación mantiene la transparencia en la ubicación del objeto.

En el modelo de objetos distribuidos, los procesos constan de objetos que se comunican mediante invocaciones locales o remotas. Las locales son invocaciones a métodos del mismo objeto, mientras que las invocaciones remotas son a métodos de objetos de otros procesos, ya estén en el mismo nodo o en otros nodos del sistema. Algunos objetos sólo pueden recibir invocaciones locales, mientras otros, los **objetos remotos**, pueden recibir tanto invocaciones locales como remotas. Para invocar un método de un objeto remoto, un objeto debe tener acceso a la **referencia de objeto remoto** del receptor, que es un identificador único en el sistema distribuido

Cada objeto remoto posee una **interfaz remota** que especifica cuáles de sus métodos pueden invocarse remotamente. Estos métodos estarán implementados por la clase del objeto remoto. En *Java RMI*, las interfaces remotas se definen como cualquier otra interfaz de Java, sin más que extender la interfaz *Remote*. *CORBA* proporciona un lenguaje de definición de interfaces (*CORBA IDL*) que permite que las clases de los objetos remotos y los programas clientes estén programados en lenguajes diferentes.

En una invocación remota pueden producirse, además de las del modelo general, excepciones relacionadas con la naturaleza remota de la invocación, como time-outs.

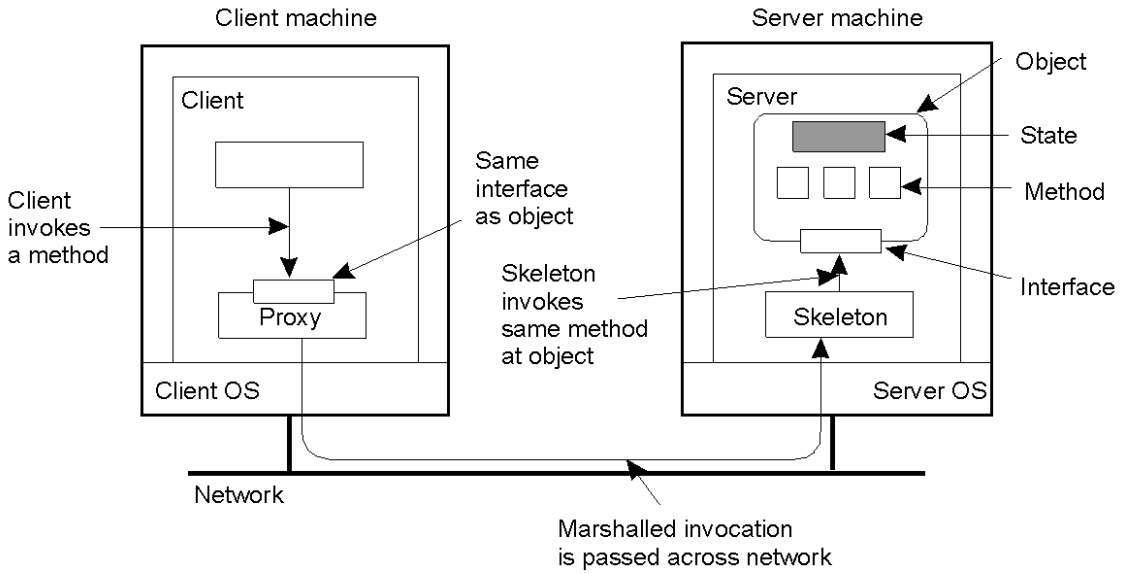


*Figura 1.A2. Invocación de métodos remotos (tomado de [TAN02]).*

La implementación del modelo de invocaciones de objetos remotos es análoga al de RPCs (Figura 1.A2). En el cliente, un representante local de la clase del objeto invocado (*proxy*) implementa de manera transparente la serialización y la comunicación con el servidor que contiene el receptor. En el servidor, la implementación de la clase del receptor (*skeleton*) gestiona la comunicación con el cliente y la serialización, e invoca el método correspondiente, tratando las eventuales excepciones.

# Java RMI

*Java RMI* extiende el modelo de objetos de Java para soportar objetos distribuidos de forma integrada en el lenguaje, haciendo la invocación de métodos remotos transparente salvo por el hecho de que el cliente tenga que tratar las excepciones remotas y el servidor definir como *Remote* la interfaz del objeto remoto.

Las interfaces remotas se definen extendiendo la interfaz *Remote*, proporcionada por el paquete *java.rmi*. Los parámetros de un método son de entrada, y la salida se proporciona en el resultado de la invocación. Los objetos remotos se pasan por referencia y los locales por valor, mediante serialización. Cuando el receptor no dispone de la implementación del objeto que se le pasa por valor, la máquina virtual Java proporciona la descarga automática de la clase correspondiente.

Los nodos que albergan objetos remotos proporcionan un servicio de nombres que almacena las referencias de objetos remotos, el *registro de objetos remotos* (*rmiregistry*).

En una aplicación distribuida en Java RMI, hay que definir las interfaces remotas e implementar los objetos remotos y los clientes. Una vez compilados los ficheros fuente, la aplicación se monta de la siguiente forma:

1. Se crean los *proxies* (*stubs*, en terminología Java-RMI) de las clases remotas mediante el compilador de RMI (*rmic*).

2. Se especifica el acceso a las clases para su descarga y se establece una política de seguridad.

3. Se pone en marcha el registro de objetos remotos como un proceso del servidor (*rmiregistry*).

4. Se pone en marcha el servicio remoto y se lanzan los clientes.