Capítulo 2
Modelos de Sistemas

Sistemas Distribuidos

Universidad Nacional de Asunción
Facultad Politécnica
Ingeniería Informática

Profesor: Ing. Fernando Mancía
Prof. Auxiliar: Ing. José Arzamendia

Modelos Físicos: es la forma más explícita de describir un sistema, incluye la composición de hardware de un sistema en términos de los ordenadores (y otros dispositivos, como teléfonos móviles) y sus redes de interconexión.

Modelos Arquitectónicos: describen un sistema en términos de las tareas de cálculo y de comunicación realizadas por sus elementos computacionales. Los elementos pueden ser ordenadores individuales o conjuntos de ellos con sus interconexiones.

Modelos Fundamentales: perspectivas abstractas con el fin de examinar los aspectos individuales de un SD. Se presentan modelos fundamentales que analizan aspectos importantes como:

- ❑ estructura y secuencia de la comunicación entre los elementos del sistema.
- ❑ las formas en que un sistema puede no funcionar correctamente.
- ❑ las formas en que un sistema está protegido contra los intentos de interferir con su correcto funcionamiento o de robar sus datos.

Modelos Físicos *(Sección 2.2 - 5ta Edición. DISTRIBUTED SYSTEMS- Concepts and Design)*

Un **Modelo Físico** es una representación de los elementos de hardware subyacentes de un sistema distribuido que abstrae los detalles específicos de las tecnologías informáticas y las redes empleadas.

Resulta útil identificar en tres generaciones de sistemas distribuidos.

- **Primeros Sistemas Distribuidos:** emergieron entre los 70' y 80' en respuesta a la emergente tecnología LAN, específicamente Ethernet. Típicamente entre 10 a 100 nodos conectados por medio de una LAN.

- **Sistemas Distribuidos en la escala de Internet:** surgieron en la década de los 90' en respuesta al impresionante crecimiento de Internet. Consiste en un conjunto de nodos interconectado por una red de redes. El nivel de Heterogeneidad comprende Redes, Arquitectura de Computadores, S.O.

- **Sistemas Distribuidos Contemporáneos:** aparición de la computación móvil en la cual los nodos comprenden computadores personales, asistentes digitales, teléfonos móviles. La aparición de la computación en Nube, computación ubicua y tecnologías variadas de red, así como infinidad de aplicaciones hacen que la cantidad se expanda increíblemente.

Figure 2.1 Generations of distributed systems

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Modelos Arquitectónicos

Un modelo arquitectónico de un sistema distribuido trata sobre la colocación de sus partes y las relaciones entre ellas. Ejemplos: modelo cliente-servidor y el modelo de procesos «de igual a igual» *(peer to peer).*

El modelo cliente-servidor admite varias modificaciones debido a:

- ❑ La partición de datos o la replicación en servidores cooperativos.
- ❑ El uso de caché para los datos en clientes y servidores proxy.
- ❑ El uso de código y agentes móviles.
- ❑ Los requisitos para añadir o eliminar dispositivos móviles de forma conveniente.

No hay un tiempo global en un sistema distribuido.
Los relojes en los computadores no tienen necesariamente el mismo tiempo.

Toda comunicación entre procesos se realiza por medio de mensajes. La comunicación en una red puede verse afectada por: retrasos, puede sufrir variedad de fallos y es vulnerable a los ataques a la seguridad.

Estas cuestiones se consideran en tres modelos:

- ***El modelo de interacción*** trata de las prestaciones y de la dificultad de poner límites temporales en un sistema distribuido, por ejemplo para la entrega de mensajes.

- ***El modelo de fallos*** intenta dar una especificación precisa de los fallos que se pueden producir en los procesos y en los canales de comunicación. Define comunicación fiable y procesos correctos.

- ***El modelo de seguridad*** discute sobre las posibles amenazas para los procesos y los canales de comunicación. Introduce el concepto de canal seguro, que lo es frente a dichas amenazas.

Modelos Arquitectónicos

La arquitectura de un sistema es su estructura en términos de componentes especificados por separado.

El objetivo global es asegurar que la estructura satisfará las demandas presentes y previsibles sobre él.

Un modelo arquitectónico de un sistema distribuido **simplifica y abstrae**, inicialmente, las funciones de los componentes individuales de dicho sistema y posteriormente considera:

- La ubicación de los componentes en una red.
- Las interrelaciones entre los componentes.

Capas de software

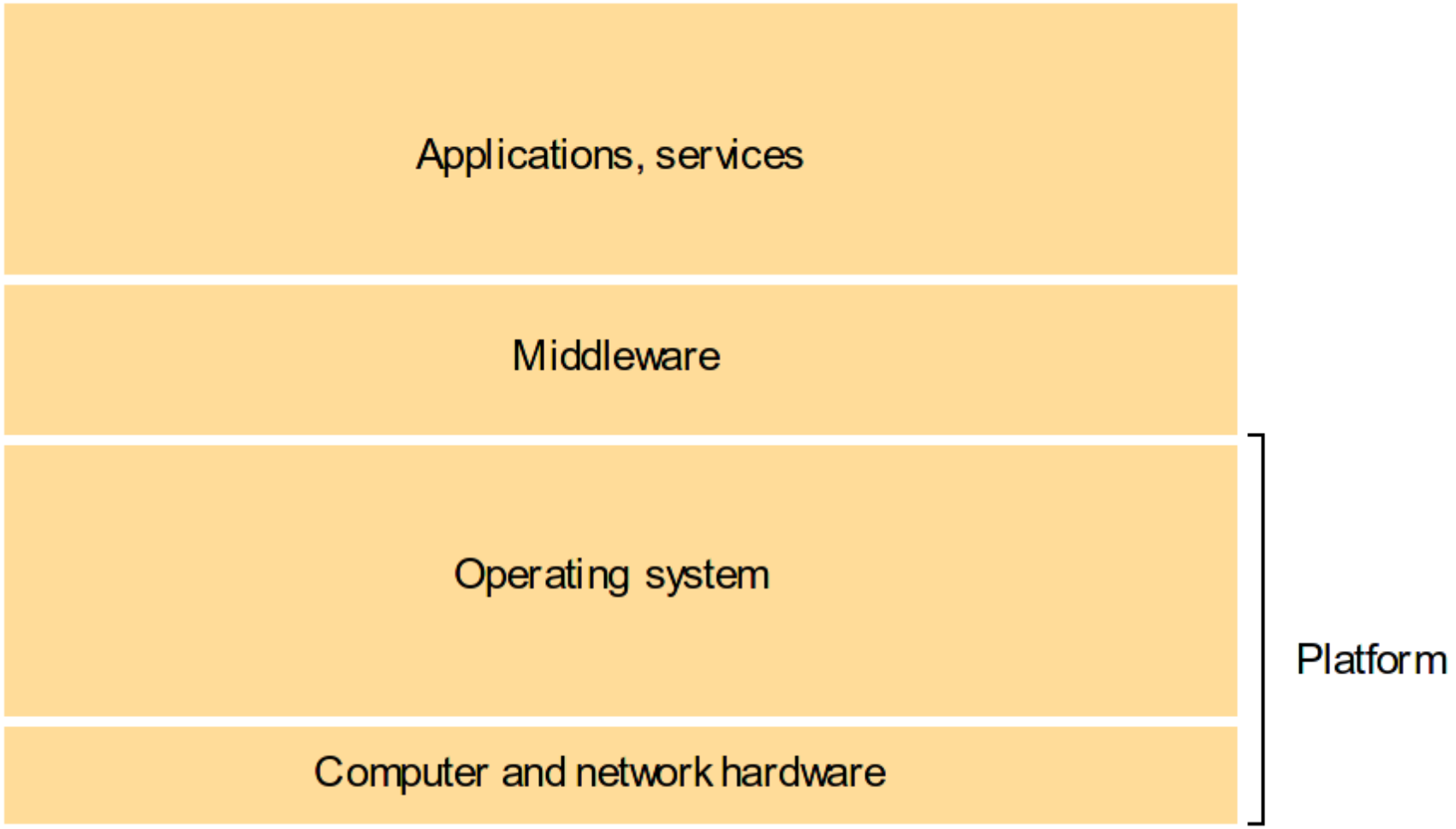
El término ***"arquitectura de software"*** se refería inicialmente a la estructuración del software como capas o módulos en un único computador y más recientemente en términos de los servicios ofrecidos y solicitados entre procesos localizados en el mismo o diferentes computadores.

Esta vista orientada a proceso y a servicio puede expresarse en términos de capas de servicio.

Un servidor es un proceso que acepta peticiones de otros procesos.

Un servicio distribuido puede proveerse desde uno o más procesos servidor, interactuando entre cada uno de ellos, y con los procesos clientes, para mantener una visión de los recursos del servicio consistente con el sistema.

Figura 2.1
Capas de servicio software y hardware en los sistemas distribuidos



Capas de servicio software y hardware en los sistemas distribuidos

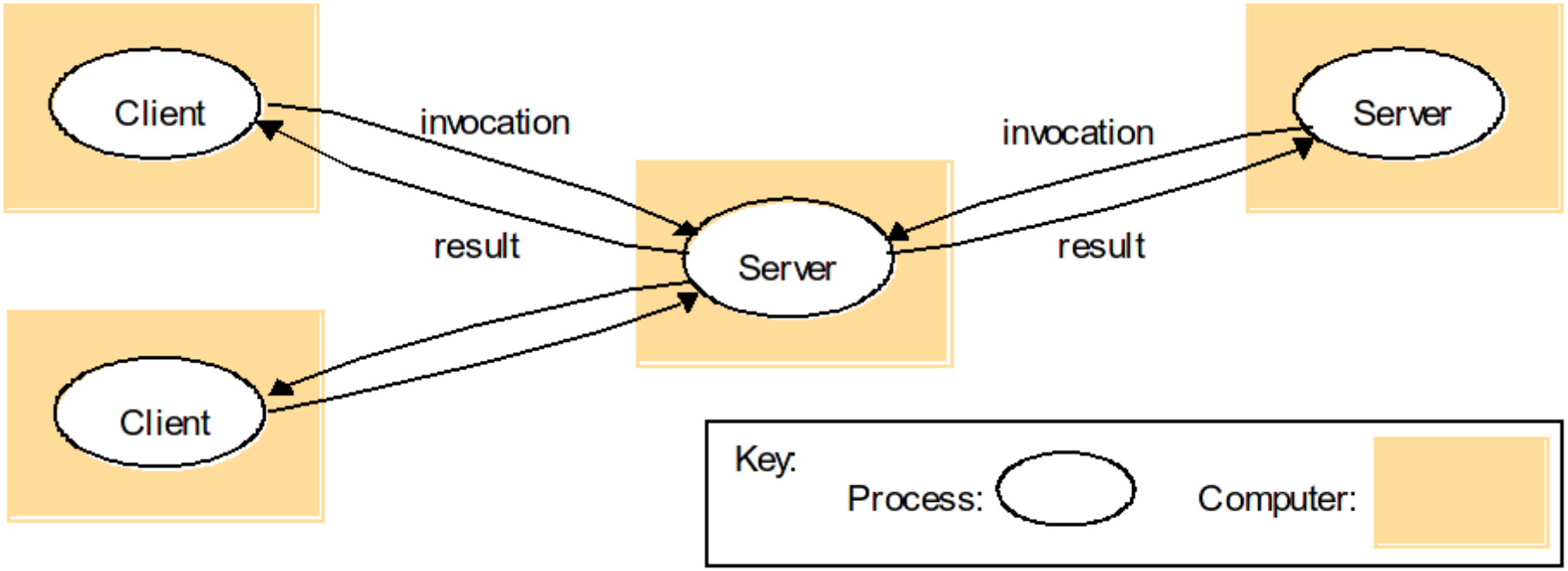
Plataforma. Al nivel de hardware y las capas más bajas de software se denominan “plataforma” para sistemas distribuidos y aplicaciones. Estas capas más bajas proporcionan servicios a las que están por encima de ellas, y que son implementadas independientemente en cada computador, proporcionando una interfaz de programación del sistema a un nivel que facilita la comunicación y coordinación entre procesos. Windows para lntel x86, Sun OS para Sun SP ARC, Solaris para lntel x86, Mac Os para PQwer PC, Linux para lntel x86 son los principales ejemplos.

Middleware. Es una capa de software cuyo propósito es enmascarar la heterogeneidad y proporcionar un modelo de programación conveniente para los programadores de aplicaciones. Se representa mediante procesos u objetos en un conjunto de computadores que interactúan entre sí para implementar mecanismos de comunicación y de recursos compartidos para aplicaciones distribuidas.

El middleware se ocupa de proporcionar bloques útiles para la construcción de componentes software que puedan trabajar con otros en un sistema distribuido.

Mejora el nivel de las actividades de comunicación de los programas de aplicación soportando abstracciones como: procedimiento de invocación remota, comunicación entre un grupo de procesos, notificación de eventos, replicación de datos compartidos y transmisión de datos multimedia en tiempo real.

Figura 2.2
Clientes que invocan a servidores individuales



Modelo cliente-servidor

Históricamente la arquitectura cliente- servidor es la más importante, y continúa siendo la más ampliamente utilizada.

La Figura 2.2 muestra la sencilla estructura sobre la que interaccionan los procesos clientes con los procesos servidores individuales, en computadores separados, con el fin de acceder a los recursos compartidos que ellos gestionan.

Los servidores pueden, a su vez, ser clientes de otros servidores, como se indica en la figura. Por ejemplo, un servidor web es, a veces, un cliente de un servidor de archivos local que gestiona los archivos en los que están almacenadas las páginas web.

Los servidores web y la mayoría del resto de los servicios de Internet son clientes del servicio DNS, que traduce Nombres de Dominio de Internet en direcciones de red.

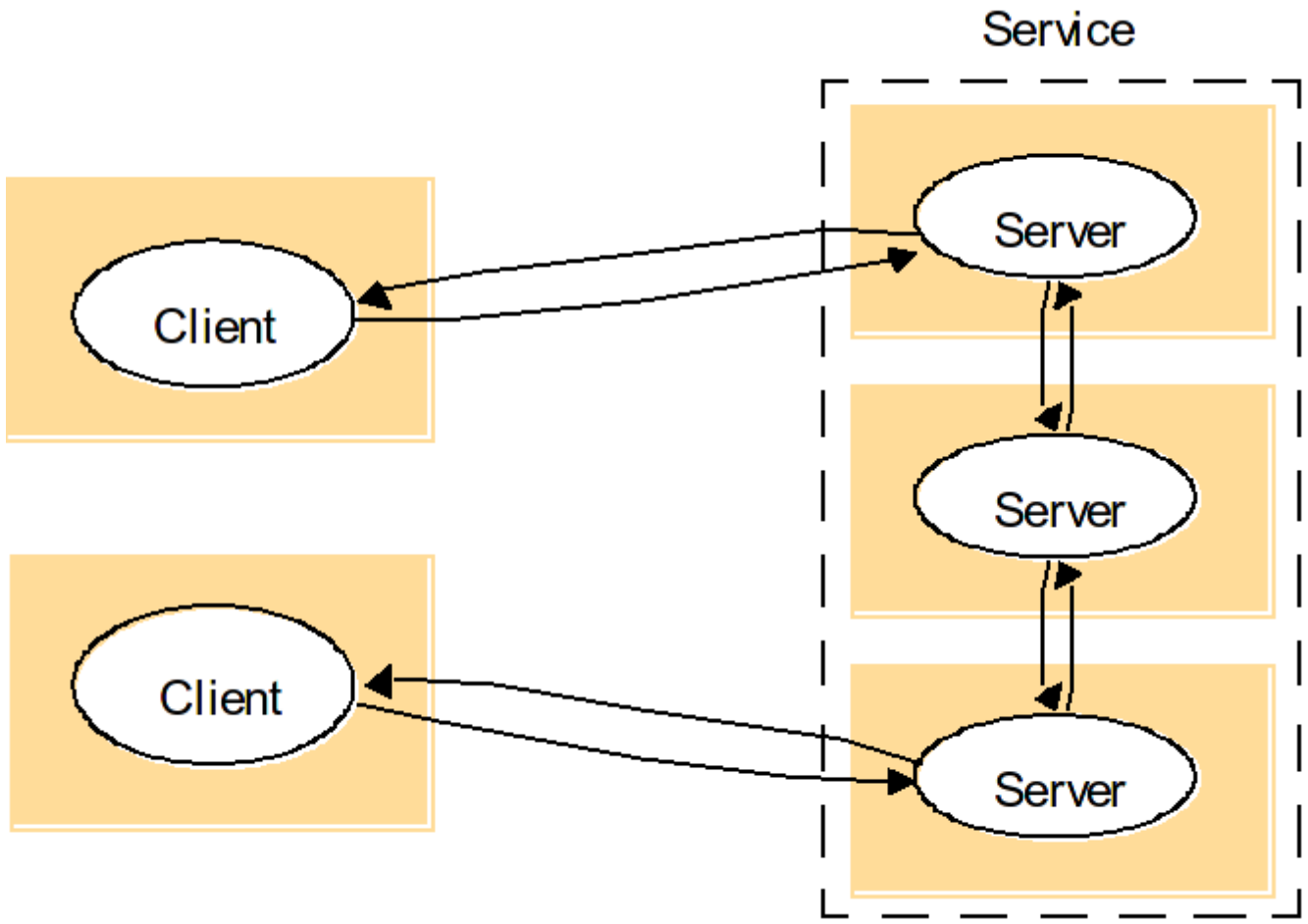
Modelo cliente-servidor

Otro ejemplo relacionado con web son los ***buscadores,*** que permiten a los usuarios buscar resúmenes de la información disponible en las páginas web de los sitios. Estos resúmenes están realizados por programas llamados *escaladores (crawlers) web,* que se ejecutan en segundo plano en el sitio del buscador utilizando peticiones HTTP para acceder a los servidores web a través de Internet.

Por tanto, una máquina de búsqueda es tanto un cliente como un servidor: responde a las consultas del navegador de los clientes y ejecuta web crawlers que actúan como clientes de otros servidores web.

En este ejemplo, las tareas del servidor (responder a las consultas de usuarios) y las tareas de crawler (hacer peticiones a otros servidores web) son totalmente independientes; hay muy poca necesidad de sincronizarlas y pueden ejecutarse concurrentemente. De hecho, un buscador típico debiera incluir muchos hilos *(threads)* de ejecución concurrente, algunos sirviendo a sus clientes y otros ejecutando crawlers web.

Figura 2.3
Un servicio proporcionado por múltiples servidores



Servicios proporcionados por múltiples servidores

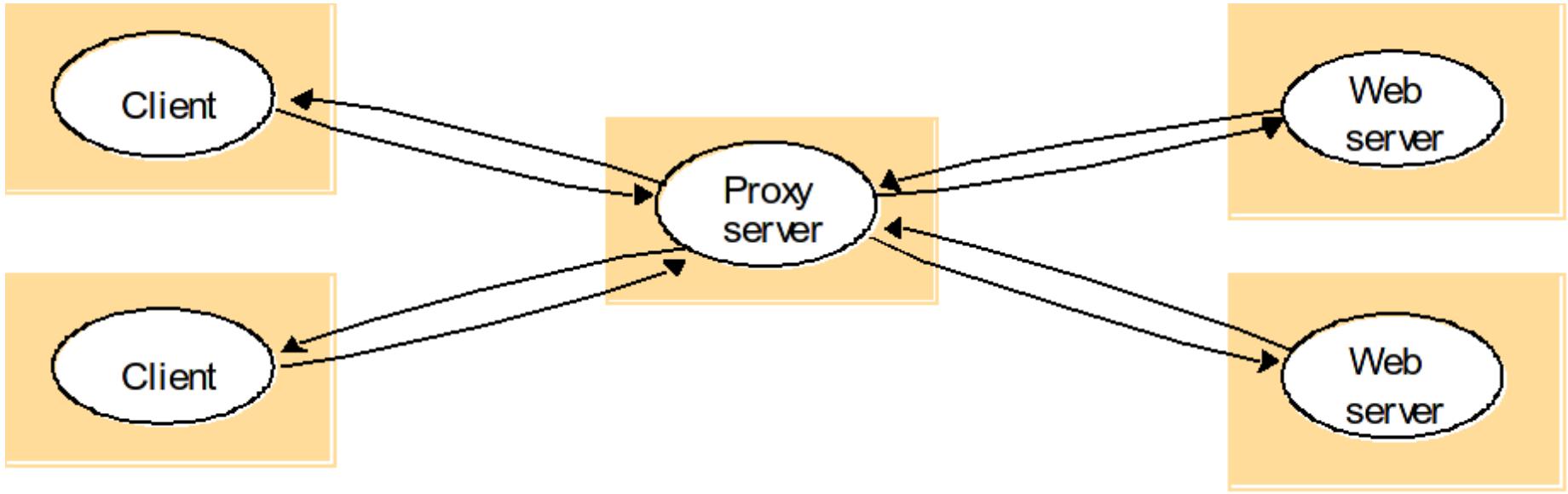
Los servicios pueden implementarse como distintos procesos de servidor en computadores separados interaccionando, cuando es necesario, para proporcionar un servicio a los procesos clientes

Los servidores pueden dividir el conjunto de objetos en los que está basado el servicio y distribuírselos entre ellos mismos, o pueden mantener copias replicadas de ellos en varias máquinas. Ej:

Web, ejemplo típico de **partición de datos**. Cada servidor web administra su propio conjunto de recursos. Un usuario puede emplear un navegador para acceder al recurso que está en cualquiera de los servidores.

La **replicación** se utiliza para aumentar las prestaciones y disponibilidad y para mejorar la tolerancia a fallos. Proporciona múltiples copias consistentes de datos en proceso que se ejecutan en diferentes computadores. Por ejemplo, el servicio web proporcionado por *altavista.digital.com* se encuentra redirigido hacia varios servidores con la base de datos replicada en memoria.

Figura 2.4
Servidor proxy de tipo web



Servidores proxy y cachés

Un caché es un almacén de objetos de datos utilizados recientemente, y que se encuentra más próximo que los objetos en sí. Al recibir un objeto nuevo en un computador se añade al almacén del caché, reemplazando, si fuera necesario, algunos objetos existentes.

Cuando se necesita un objeto en un proceso cliente, el servicio caché comprueba inicialmente la caché y le proporciona el objeto de una copia actualizada. Si no, se buscará una copia actualizada. Los cachés pueden estar ubicados en cada cliente o en un servidor proxy que puede compartirse desde varios clientes.

Los cachés se utilizan intensivamente en la práctica. Los navegadores web mantienen un caché de las páginas visitadas recientemente, y de otros recursos web, en el sistema local de ficheros del cliente; entonces utilizan una petición especial HTTP para comprobar si dichas páginas han sido actualizadas en el servidor antes de sacarlas en pantalla.

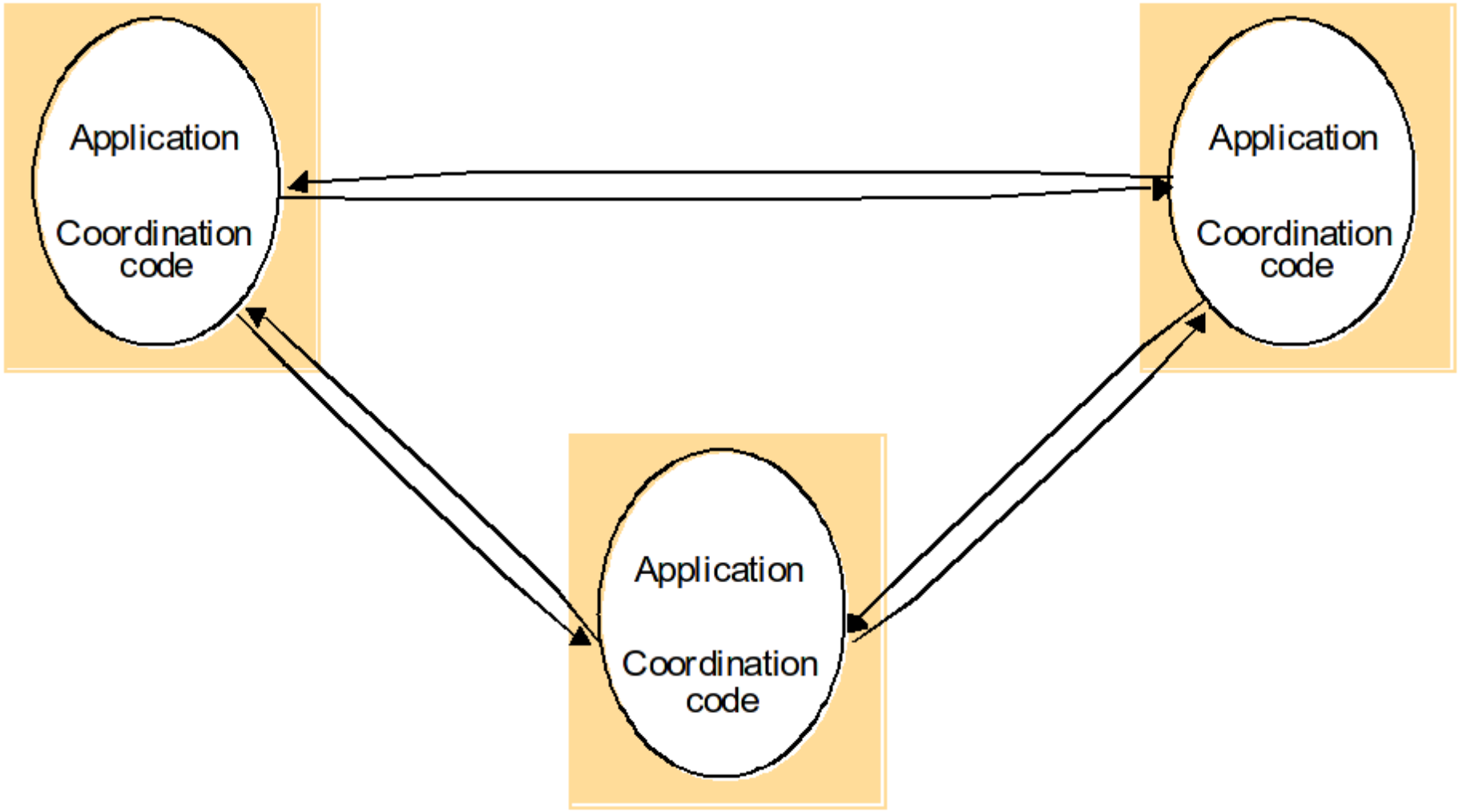
Ej: Un Servidor de Información que solicita a varios servidores diferentes tipos de datos.

Los servidores proxy para el web proporcionan un caché compartido de recursos web a las máquinas cliente de uno o más sitios.

El propósito de los servidores proxy es **incrementar la disponibilidad y prestaciones del servicio**, reduciendo la carga en redes de área amplia y en servidores web.

Los servidores proxy pueden desempeñar otros roles, por ejemplo, pueden ser utilizados para acceder a servidores web remotos a través de un cortafuegos.

Figura 2.5
Una aplicación distribuida basada en procesos de igual a igual



Procesos de igual a igual. *Peer to Peer*

En esta arquitectura todos los procesos desempeñan tareas semejantes, interactuando cooperativamente como iguales para realizar una actividad distribuida o cómputo sin distinción entre clientes y servidores.

En este modelo, el código en los procesos parejos o «iguales» mantiene la consistencia de los recursos y sincroniza las acciones a nivel de la aplicación cuando es necesario.

La Figura 2.5 muestra un ejemplo con tres instancias de esta situación; en general, *n* procesos parejos podrán interactuar entre ellos, dependiendo el patrón de comunicación de los requisitos de la aplicación.

La eliminación del proceso servidor reduce los retardos de comunicación entre procesos al acceder a objetos locales.

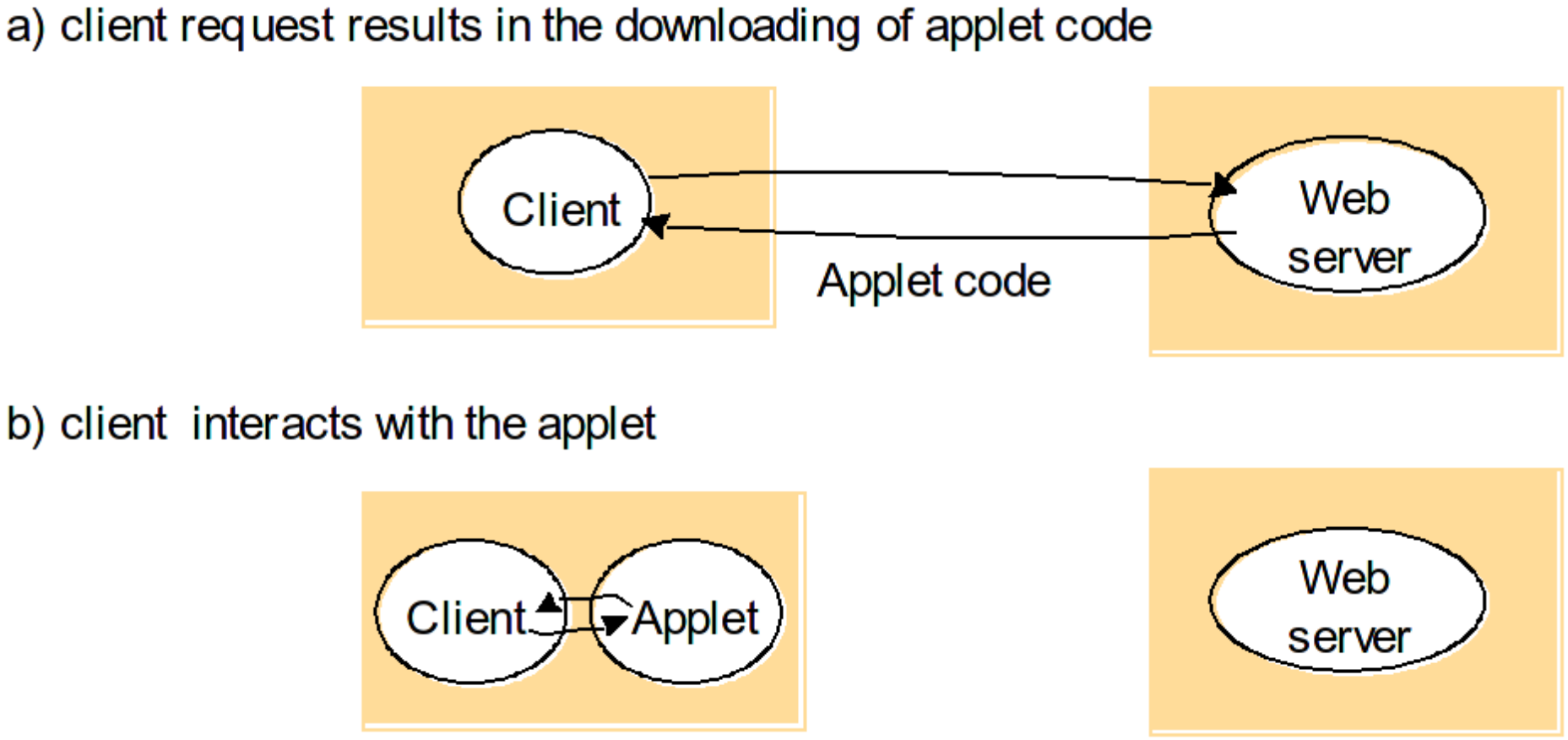
Procesos de igual a igual. *Peer to Peer*

Una aplicación de *pizarra* distribuida permite que usuarios en varios computadores vean y modifiquen interactivamente un dibujo que se comparte entre ellos.

Esto puede implementarse con un proceso de aplicación en cada sitio y que se basa en capas de middleware para **realizar notificación de eventos y comunicación a grupos** para indicar a todos los procesos de la aplicación de los cambios en el dibujo.

Esto proporciona, a los usuarios de objetos distribuidos compartidos, una respuesta interactiva mejor de la que se puede obtener con una arquitectura basada en servidor.

Figura 2.6
Applets de tipo Web



Variaciones en el modelo cliente-servidor

Podemos distinguir distintas variaciones del modelo cliente-servidor, dependiendo de la consideración de los factores siguientes:

- El uso de código móvil y agentes móviles.
- Las necesidades de los usuarios de computadores de coste bajo y con recursos hardware limitados, que son muy sencillos de manejar.
- El requisito de añadir o eliminar de una forma conveniente dispositivos móviles.

Variaciones en el modelo cliente-servidor

Código móvil. Los applets son el ejemplo más conocido y más ampliamente extendido de código móvil; un usuario que ejecute un navegador y que seleccione un enlace con un *applet,* cuyo código esté almacenado en un servidor web, descargará el código en el navegador y se ejecutará allí .

Una ventaja de ejecutar el código descargado localmente es que puede proporcionar una buena respuesta interactiva puesto que no sufre ni de los retardos ni de la variabilidad del ancho de banda asociados con la comunicación en red.

Acceder a los servicios significa ejecutar código que pueda invocar sus operaciones.

Consideremos una aplicación donde se necesite que los usuarios dispongan de información actualizada según se hagan cambios en el origen de los datos en el servidor. Esto no se puede conseguir mediante las interacciones normales con el servidor web, que siempre se inician por parte del cliente.

Variaciones en el modelo cliente-servidor

La solución es utilizar software adicional que trabaja de una forma a la que a veces se refiere como modelo ***push,*** donde el servidor, en lugar del cliente, es el que inicia las interacciones.

Por ejemplo, un corredor de bolsa puede proporcionar un servicio personalizado para notificar a los usuarios los cambios en las cotizaciones; para utilizar dicho servicio, cada usuario debiera haber descargado un *applet* especial, que recibe las actualizaciones del servidor del corredor, las presenta en pantalla al usuario y quizá realiza operaciones automáticas de compra y venta, desencadenadas por las condiciones prefijadas por el usuario y almacenados localmente en el computador del mismo.

http://pushkin.io/

Agentes móviles. es un programa en ejecución (lo que incluye tanto código como datos) que se traslada de un computador a otro en la red realizando una tarea para alguien; por ejemplo, recolectando información, y retornando eventualmente con los resultados.

Un agente móvil puede hacer muchas solicitudes a los recursos locales de los sitios que visita, por ejemplo, accediendo a anotaciones individuales en una base de datos. Si se compara esta arquitectura con un cliente estático que realiza solicitudes de algunos recursos, transfiriendo posiblemente grandes cantidades de datos, hay una reducción en el coste de la comunicación y en el tiempo con la sustitución de las solicitudes remotas por las locales.

Los agentes móviles (como el código móvil) son una amenaza potencial de seguridad para los recursos de los computadores que visitan.

El entorno debe decidir que recursos locales estará permitido el acceso.
La identidad de éste debe incluirse de una forma segura en el código y los datos del agente móvil.

Los agentes móviles pueden ser vulnerables, y pueden no ser capaces de finalizar su tarea si se les niega el acceso a la información que necesitan.

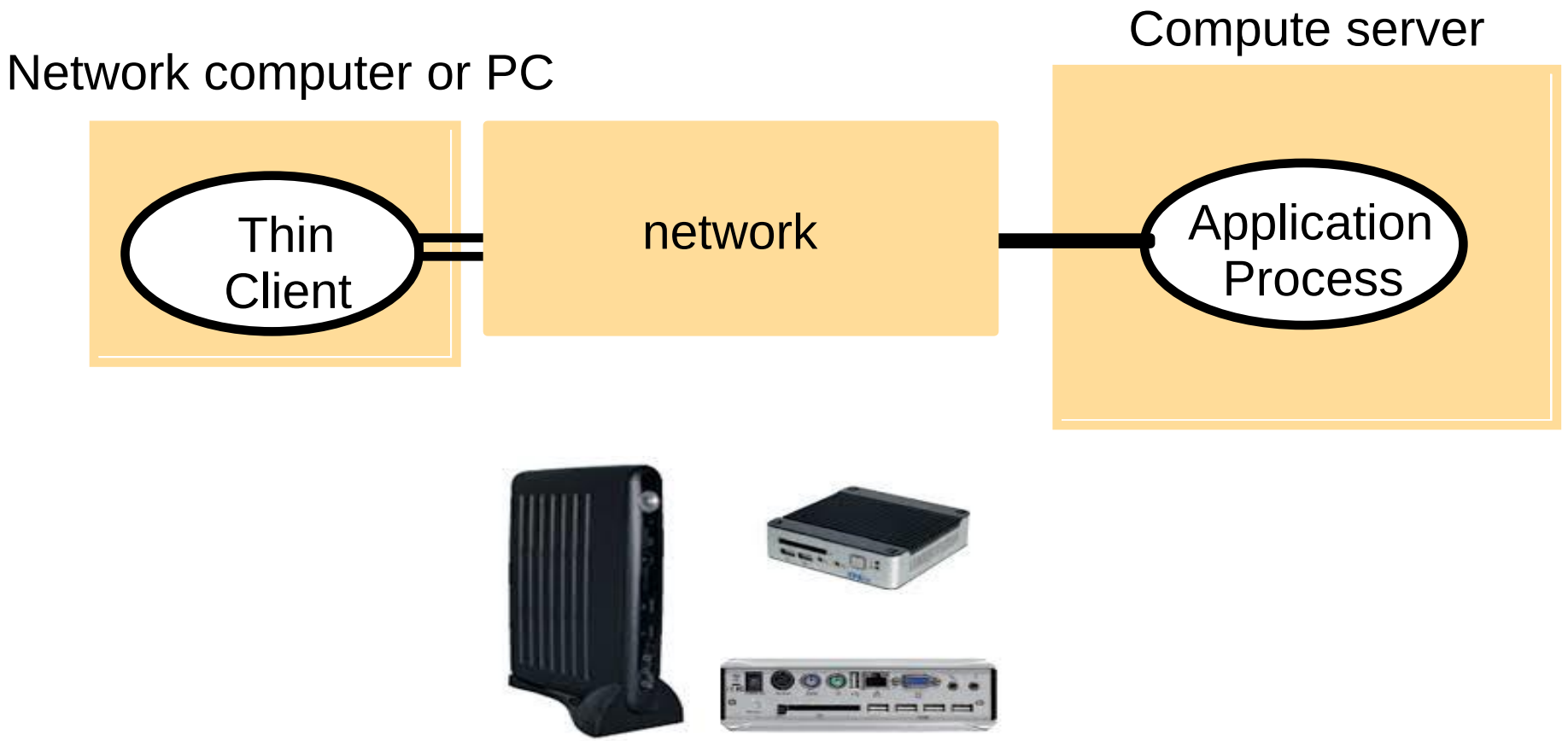
Las tareas realizadas por agentes móviles pueden realizarse por otros medios. Por ejemplo, los escaladores *(crawlers)* web que necesitan acceder a recursos en servidores web a través de Internet trabajan bastante bien realizando invocaciones remotas a los procesos del servidor.

Por estas razones, la aplicabilidad de agentes móviles puede ser limitada.

Un ejemplo de Agente Móvil, es el programa gusano *worm* desarrollado por XEROX PARC con el fin de utilizar computadores desocupados para realizar cálculos intensivos.

Se pueden utilizar agentes móviles para instalar y mantener software en los computadores de una organización, o comparar los precios de los productos de un número de vendedores visitando el sitio de cada vendedor y realizando una serie de operaciones sobre la base de datos

Figura 2.7
Clientes ligeros y servidores de cálculo



Variaciones en el modelo cliente-servidor

El término ***cliente ligero*** se refiere a una capa de aplicación que soporta una interfaz de usuario basada en ventanas sobre un computador local del usuario mientras se ejecutan programas de aplicación en un computador remoto.

Las aplicaciones se ejecutan en un *servidor de cómputo,* un potente computador que tiene capacidad para ejecutar gran número de aplicaciones simultáneamente. El servidor de cómputo (multiprocesador o un sistema de computadores fuertemente acoplado *(cluster)),* ejecuta una versión multiprocesador de un sistema operativo como UNIX o Windows NT.

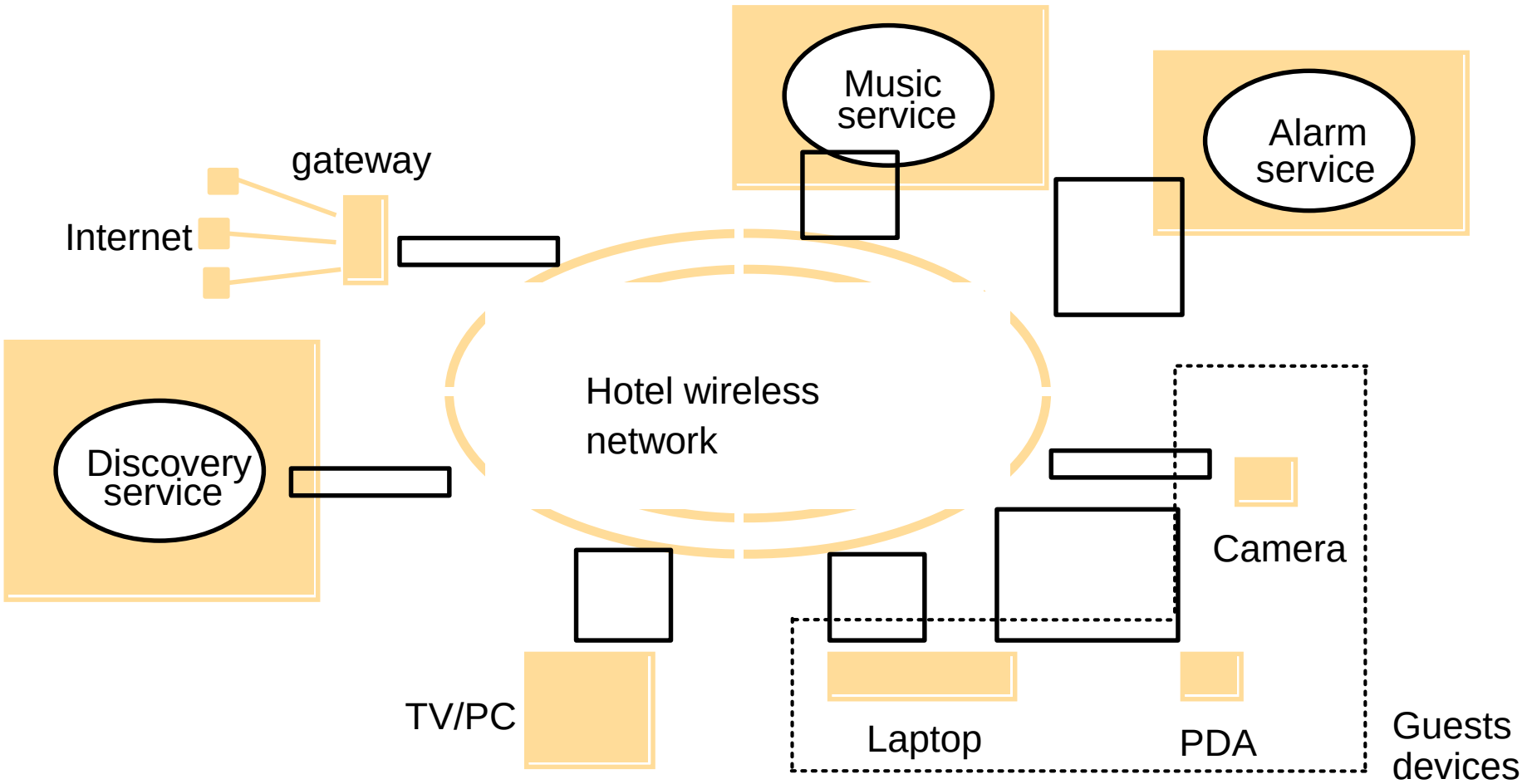
El principal inconveniente de una arquitectura de cliente ligero se encuentra en actividades gráficas fuertemente interactivas, como CAD o proceso de imagen, en las que los retrasos experimentados por los usuarios se ven aumentados por la necesidad de transferir imágenes e información vectorial entre el cliente ligero y los procesos de aplicación, padeciendo latencias tanto de red como de sistema operativo.

Implementaciones de clientes ligeros: Los sistemas de clientes ligeros son sencillos en concepto y sus implementaciones son inmediatas en algunos entornos. Por ejemplo, la mayoría de las variantes de UNIX incluyen el sistema de ventanas X-11 .

WinFrame de Citrix [www.citrix.com] es una implementación comercial del concepto de cliente ligero. Puede ejecutarse en una gran variedad de plataformas y soporta un entorno *(desktop)* que proporciona acceso interactivo a las aplicaciones corriendo en máquinas Windows NT.

Otras aproximaciones incluyen los sistemas de computadores de red virtual *(Teleporting* y *Virtual Network Computer,* VNC) desarrollados en los laboratorios AT&T de Cambridge, Inglaterra [Richardson y otros, 1998]. VNC establece la frontera en el nivel de operaciones de píxeles en pantalla y mantiene el contexto gráfico para los usuarios cuando éstos se mueven entre computadores.

Figura 2.8
Intercomunicación espontánea en un hotel



La figura muestra los dispositivos que el usuario ha llevado a la suite del hotel: un computador portátil, una cámara digital y una agenda portátil (PDA), cuyas posibilidades a través de infrarrojos le permiten actuar como un *controlador universal*.

Las características esenciales de la conexión a red espontánea son:

- *Conexión fácil a la red local:* los enlaces sin cable eliminan la necesidad de cableado preinstalado y eliminan el inconveniente y las cuestiones de fiabilidad que rodean a los conectores y enchufes. Sólo existen aquellos enlaces que pueden mantenerse aunque haya movimiento. Un dispositivo colocado en un nuevo entorno de red se reconfigura de modo transparente para conectarse allí; el usuario no tiene que introducir los nombres o direcciones de los servicios locales para obtenerlo.

- *Integración fácil con servicios locales:* los dispositivos que se encuentran insertados (y moviéndose) entre redes existentes de dispositivos descubren automáticamente qué servicios se proporcionan, sin acciones especiales de configuración por el usuario.

Para usuarios móviles se presentan otras cuestiones.

Conectividad limitada: los usuarios no siempre están conectados cuando se desplazan. Por ejemplo, al viajar en un tren por un túnel pueden estar desconectados eventualmente de su red inalámbrica. También pueden desconectarse durante largos períodos de tiempo en zonas sin cobertura o cuando el permanecer conectado sea demasiado costoso.

Seguridad y privacidad: en un escenario como el del huésped de un hotel se presentan muchas cuestiones de seguridad y privacidad personal. El hotel y sus huéspedes son vulnerables a los ataques de seguridad de los empleados y de otros huéspedes del hotel, que intenten conexiones inalámbricas de formas no supervisadas.

Prestaciones. Retos que surgen de la distribución de recursos aumentan la necesidad de gestión de actualizaciones concurrentes. Se presentan por las limitadas capacidades del proceso y comunicación se consideran bajo:

Capacidad de respuesta (Responsiveness): Los usuarios de aplicaciones interactivas necesitan rapidez y consistencia en las interfaces, pero, a menudo, los programas cliente necesitan acceder a recursos compartidos.

Cuando está implicado un servicio remoto, la velocidad a la que se genera la respuesta está determinada no sólo por la carga y prestaciones del servidor y la red sino también por los retardos de todos los componentes software implicados, la comunicación entre los sistemas operativos del cliente y del servidor y los servicios *middleware* (soporte para invocación remota, por ejemplo) así como el código del proceso que implementa el servicio.

En resumen, la transferencia de datos entre los procesos y la conmutación del control es relativamente lenta, incluso cuando los procesos residen en el mismo computador. Para obtener buenos tiempos de respuesta, los sistemas deben estar compuestos de relativamente **pocas capas de software** y la cantidad de datos transferidos entre el cliente y el servidor debe ser pequeña.

Estas cuestiones se pueden ver en las prestaciones de los visualizadores web clientes, donde se obtiene una respuesta más rápida cuando se accede localmente a páginas e imágenes de la caché, por lo que se encuentran almacenadas en la aplicación cliente.

El acceso a páginas de texto remotas se produce con una rapidez razonable porque son pequeñas, pero las imágenes gráficas se retrasan mucho más por el volumen de datos implicado

Productividad (Throughput): Una medida tradicional de prestaciones para computadores es la productividad, **la rapidez a la que se realiza el trabajo computacional**.

La capacidad de un sistema distribuido para realizar el trabajo para todos sus usuarios es importante. Depende de las velocidades de proceso de los clientes y los servidores y de las tasas de transferencia de datos.

Los datos que están localizados en un servidor remoto deben transferirse del proceso servidor al proceso cliente pasando a través de varias capas en ambos computadores.

La productividad de las capas de software que intervienen es bastante importante, así como la de la red.

Balance de cargas computacionales: uno de los propósitos de los sistemas distribuidos es permitir que las aplicaciones y los procesos de servicio evolucionen concurrentemente sin competir por los mismos recursos y explotando los recursos computacionales disponibles (procesador, memoria y capacidades de red).

Por ejemplo, la capacidad para ejecutar applets en los computadores cliente elimina carga del servidor web, permitiendo proporcionar un mejor servicio.

Un ejemplo más significativo está en el uso de varios computadores para alojar un único servicio. Esto se necesita, por ejemplo, en algunos servidores web excesivamente cargados (máquinas de búsqueda, grandes lugares comerciales, comercio electrónico y bolsa de valores).

En algunos casos, el balance de cargas puede implicar mover el trabajo parcialmente completado, como carga a un computador alternativo.

Calidad de servicio: las principales propiedades no funcionales de los sistemas, que afectan a la calidad del servicio experimentado por los clientes y usuarios, son: **fiabilidad, seguridad y prestaciones.**

La facilidad de adaptación *(adaptability)* para adecuar configuraciones variables de sistema y disponibilidad de recursos ha sido reconocida recientemente como otro aspecto importante de la calidad del servicio .

Aspectos de fiabilidad.

Tolerancia frente a fallos: Las aplicaciones estables deben continuar funcionando correctamente en presencia de fallos en el hardware, el software y las redes. La fiabilidad se consigue introduciendo **redundancia** y proporcionando múltiples recursos de modo que el software de sistema y aplicación pueda ser reconfigurado y seguir realizando sus tareas en presencia de fallos.

A nivel arquitectónico, la redundancia precisa el empleo de varios computadores donde lanzar cada proceso componente del sistema y múltiples caminos de comunicación sobre los que transmitir los mensajes. Los datos y los procesos pueden replicarse donde quiera que sea necesario para proporcionar el debido nivel de tolerancia a fallos.

Algunas aplicaciones críticas, tales como los sistemas de control de tráfico aéreo, precisan un alto nivel de garantías en cuanto a la tolerancia frente a fallos.

Seguridad: El impacto arquitectónico de los requisitos de seguridad afecta a la necesidad de ubicar los datos y otros recursos sensibles sólo en aquellos computadores equipados de un modo eficaz contra ataques.

Por ejemplo, una base de datos de un hospital alojará registros con un componente sensible sobre los pacientes y sólo ciertos empleados podrán acceder a ellos, mientras que otros componentes de los mismos registros pueden estar disponibles más ampliamente.

No sería apropiado construir un sistema en el que al acceder a los datos de un paciente se descargara el registro completo de éste en el computador de sobremesa del usuario, dado que el computador típico de sobremesa no constituye un entorno seguro, y los usuarios podrían lanzar programas para acceder o actualizar cualquier parte de los datos almacenados en su computador personal.

Interacción: el cómputo ocurre en los procesos; los procesos interaccionan por paso de mensajes, lo que deviene en comunicación y coordinación (sincronización y ordenamiento de actividades) entre procesos.

Fallo: la correcta operación de un sistema distribuido se ve amenazada donde aparezca un fallo en cualquier computador sobre el que se ejecuta (incluyendo fallos de software) o en la red que los conecta.

Seguridad: la naturaleza modular de los sistemas distribuidos y su extensibilidad los expone a ataques tanto de agentes externos como internos.
Cuanto mas extensibles son, más seguridad se necesita.

Algoritmo: secuencia de pasos que hay que seguir para realizar un cómputo deseado.

Algoritmo Distribuido: Una definición de los pasos que hay que llevar a cabo en cada uno de los procesos.

Prestaciones de los canales de comunicaciones: las características de prestaciones de la comunicación sobre una red de computadores incluyen la latencia, el ancho de banda y las fluctuaciones:

El **retardo** entre el envío de un mensaje por un proceso y su recepción por otro se denomina *latencia.* La latencia incluye:

- El tiempo que se toma en llegar a su destino el primero de una trama de bits transmitidos a través de una red.
- El retardo en acceder a la red, que es grande cuando la red está muy cargada. Por ejemplo, al transmitir sobre Ethernet la estación de emisión esperará a que la red esté libre de tráfico.
- El tiempo empleado por los servicios de comunicación del sistema operativo tanto en el proceso que envía como en el que recibe, y que varía según la carga actual de cada sistema operativo.

. El *ancho de banda* de una red de computadores es la cantidad total de información que puede transmitirse en un intervalo de tiempo dado. Cuando una red está siendo utilizada por un número grande de canales de comunicación, habrá que compartir el ancho de banda disponible.

. La *fluctuación* (*jitter)* es la variación en el tiempo invertido en completar la entrega de una serie de mensajes. La fluctuación es importante para los datos multimedia. Éste es el caso que si se reproducen muestras de audio consecutivas a un ritmo variable el sonido presentará graves distorsiones, se requiere un flujo constante.

Relojes de computadores y eventos de temporización: Cada computador de un sistema distribuido tiene su propio reloj interno; los procesos locales obtienen de él, el valor del tiempo actual.

Esta es la forma en que dos procesos en ejecución sobre dos computadores diferentes asocian marcas (o sellos) temporales a sus eventos. Sin embargo, incluso si dos procesos leen sus relojes a la vez, sus relojes locales proporcionarán valores de tiempo diferentes.

Esto ocurre porque los relojes presentan derivas con respecto al tiempo perfecto y, más importante, sus tasas de deriva difieren de una a otra.

Tasa de deriva del reloj: proporción en que el reloj de un computador difiere del reloj de referencia perfecto. Incluso, si los relojes de todos los computadores de un sistema distribuido se ponen en hora a la vez, a partir de un tiempo sus relojes variarán en una magnitud significativa a menos que se apliquen correcciones.

Hay varias aproximaciones a la corrección de los tiempos del reloj de los computadores.

Así, podría utilizarse receptores de radio para obtener lecturas de tiempo de un GPS con aproximaciones cercana al microsegundo. Pero los receptores GPS no operan dentro de los edificios, ni se justifica el costo para cada computador.

En su lugar, un computador dotado de una fuente de tiempo precisa, como un GPS, sí que podría enviar mensajes de temporización a otros computadores de la red. La concordancia final entre los tiempos de los relojes locales está, por supuesto, afectada por retardos variables en los mensajes

Dos variantes del modelo de interacción. En un sistema distribuido es difícil establecer cotas sobre el tiempo que debe tomar la ejecución de un proceso, el reparto de un mensaje o la deriva del reloj. Tenemos dos modelos simples que parten de posiciones extremas y opuestas: el primero tiene en cuenta una fuerte restricción sobre el tiempo; en el segundo no se hace ninguna presuposición.

Sistemas distribuidos síncronos: Hadzilacos y Toueg [1994] definen un sistema distribuido síncrono como aquel en el que se establecen los siguientes límites:

. El tiempo de ejecución de cada etapa de un proceso tiene ciertos límites inferior y superior conocidos.
. Cada mensaje transmitido sobre un canal se recibe en un tiempo limitado conocido.
. Cada proceso tiene un reloj local cuya tasa de deriva sobre el tiempo real tiene un límite conocido.

Sistemas distribuidos asíncronos: muchos sistemas distribuidos, ejemplo Internet, son de gran utilidad aun sin ser sistemas síncronos. En consecuencia necesitamos un modelo alternativo: un sistema distribuido asíncrono es aquel en que no existen limitaciones sobre:

. La velocidad de procesamiento (ejemplo, un paso de un proceso puede requerir tan sólo un pico segundo y otro un siglo; lo único que podemos decir es que cada paso se tomará un tiempo arbitrariamente largo).
. Los retardos de transmisión de mensaje (por ejemplo, al repartir un mensaje de un proceso A a otro B puede tardar un tiempo cero o bien varios años. En otras palabras, un mensaje puede recibirse tras un tiempo arbitrariamente largo).
. Las tasas de deriva de reloj (de nuevo, la tasa de deriva de reloj es arbitraria).

El modelo asíncrono no presupone nada sobre los intervalos de tiempo involucrados en cualquier ejecución. Éste es exactamente el modelo de Internet, en él no hay límite intrínseco en la carga del servidor o la red (en consecuencia no se sabe cuánto tomará, por ejemplo, transferir un archivo usando ftp). A veces un mensaje de correo puede tomarse varios días en llegar.

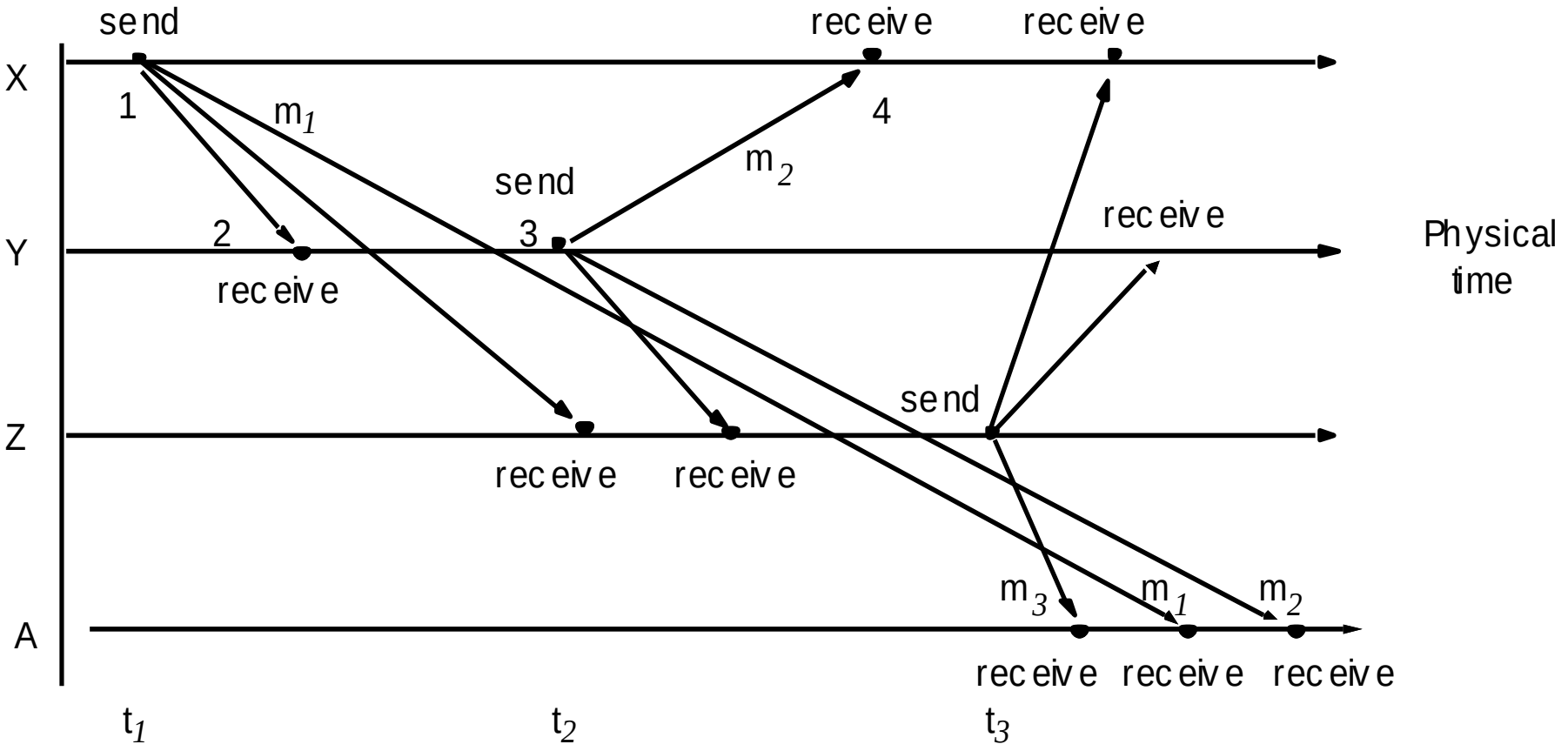
Ordenamiento de eventos. En muchos casos, nos interesa saber si un evento (enviar o recibir un mensaje) en un proceso ocurrió antes, después o concurrentemente con otro evento en algún otro proceso. La ejecución de un sistema puede describirse en términos de los eventos y su ordenación aun careciendo de relojes precisos.

Por ejemplo, considere el siguiente conjunto de intercambios entre un pequeño grupo de usuarios X, Y, Z y A de una lista de correo:

- El usuario X envía un mensaje con el tema *Reunión.*
- Los usuarios Y y Z responden con un mensaje con el tema *Re: Reunión.*

En tiempo real, se envía primero el mensaje de X, Y lo lee y responde; Z lee ambos mensajes, el de X y la respuesta de Y y entonces envía otra respuesta, que hace referencia a los dos anteriores. Pero debido a los retardos independientes en el reparto de los mensajes, los mensajes pudieran haber sido repartidos como se muestra en la Figura 2.9, y algunos usuarios podrían ver estos mensajes en el orden equivocado

Figura 2.9
Ordenamiento real en el tiempo de los eventos



Ordenamiento real en el tiempo de los eventos

Si los relojes de los computadores de X, Y y Z pudieran sincronizarse, podría utilizarse el tiempo asociado localmente a cada mensaje enviado. Por ejemplo, los mensajes *m1, m2* y *m3* llevarían la tiempos *t1, t2 y t3* donde *t1 < t2 < t3.* Los mensajes recibidos se mostrarían a los usuarios según este ordenamiento temporal.

Si los relojes están sincronizados aproximadamente la temporización ocurrirá en el orden correcto.

Como los relojes de un sistema distribuido no pueden sincronizarse de manera perfecta, Lamport [1987] propuso un modelo de *tiempo lógico* que pudiera utilizarse para ordenar los eventos de procesos que se ejecutan en computadores diferentes de un sistema distribuido.

El tiempo lógico permite inferir el orden en que se presentan los mensajes sin recurrir a relojes.

Ordenamiento real en el tiempo de los eventos

Es obvio que los mensajes se reciben después de su envío, así podemos admitir un ordenamiento lógico para los pares de sucesos que se muestran en la Figura 2.9; por ejemplo, considerando sólo los eventos relativos a X e Y:

También sabemos que las respuestas se reciben después de que se reciben los mensajes, de modo que tenemos el siguiente ordenamiento lógico para Y:

X envía *m1,* antes de que Y reciba *m1* ;
Y envía *m2* antes de que X reciba *m2.*
Y recibe *m1* antes de enviar *m2.*

El tiempo lógico lleva esta idea más lejos asignando un número a cada evento, que se corresponde con su ordenamiento lógico, de modo que los últimos eventos tendrán números mayores que los primeros. Como ejemplo, en la Figura 2.9 se consignan los números 1 a 4 en los eventos en X e Y.

"Define las formas en que puede ocurrir un Fallo."

Fallos por omisión. Las faltas clasificadas como *fallos por omisión* se refieren a casos en los que los procesos o los canales de comunicación no consiguen realizar acciones que se suponen que pueden hacer .

Fallos por omisión de procesos: El principal fallo por omisión de un proceso es el fracaso o ruptura accidentada del procesamiento *(crash).* Cuando decimos que un proceso se rompe queremos decir que ha parado y no ejecutará ningún paso de programa más.

Otros procesos pueden ser capaces de detectar tales fracasos por el hecho de que los procesos rotos fallan repetidamente en responder a los mensajes de invocación.

Desgraciadamente, este método de detección de rupturas descansa en el uso de *timeouts;* es decir un método en el cual un proceso otorga un período fijo de tiempo para que algo ocurra. En un sistema asíncrono un timeout puede indicar tan sólo que un proceso no responde; puede que se haya roto o sea lento, o que los mensajes no hayan llegado.

Modelo de fallo

La rotura de un proceso se denomina fallo-parada *(fail-stop)* si los otros procesos pueden detectar con certeza que el proceso ha fracasado.

El comportamiento fallo-parada puede producirse en un sistema síncrono si el proceso utiliza timeouts para detectar cuando los otros procesos fallan en la respuesta y está garantizado que los mensajes llegan.

Por ejemplo, si los procesos *p* y *q* están programados para que *q* responda a un mensaje de *p,* y el proceso *p* no ha recibido respuesta del proceso *q* en un tiempo máximo medido por el reloj local de *p,* entonces el proceso *p* puede concluir que el proceso *q* ha fallado.

Fallos por omisión de comunicaciones: Considere las primitivas de comunicación *envía* y *recibe.* Un proceso *p* realiza un *envío* insertando el mensaje *m* en su búfer de mensajes salientes. El canal de comunicación transporta *m* al búfer de mensajes entrantes de *q.* El proceso *q* realiza un *recibe* tomando *m* de su búfer de mensajes entrantes y repartiéndolo (véase la Figura 2.10). Los búferes entrante y saliente se proporcionan usualmente desde el sistema operativo.

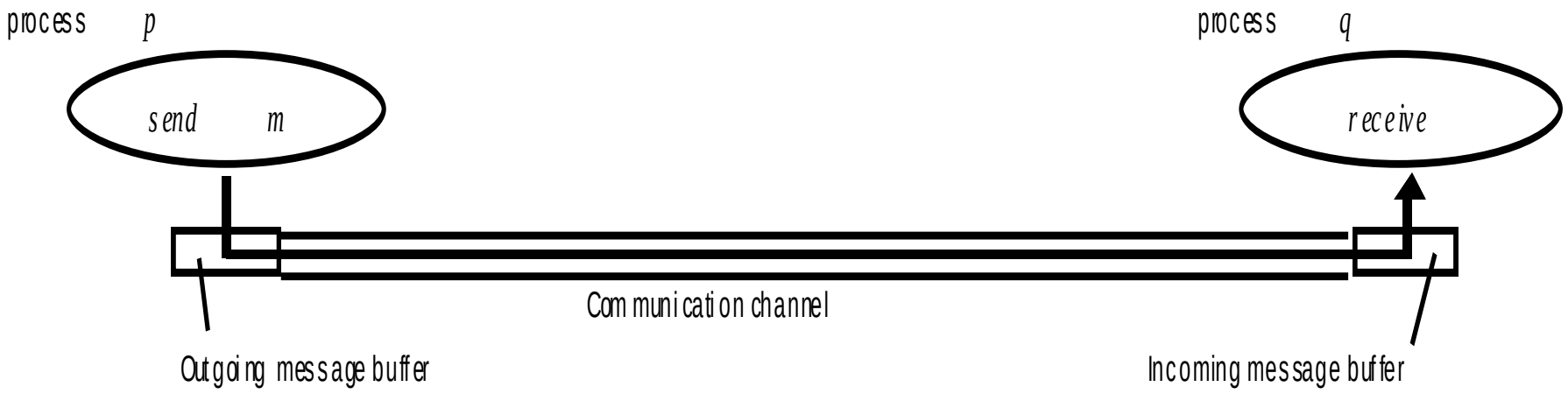


Figura 2.10
Procesos y canales

El canal de comunicación produce un fallo de omisión si no transporta un mensaje desde el búfer de mensajes salientes de *p* al búfer de mensajes entrantes de *q.*

A esto se denomina «perder mensajes» y su causa suele ser la falta de espacio en el búfer de recepción de alguna pasarela de entre medias, o por un error de red, detectable por una suma de chequeo de los datos del mensaje.

Hadzilacos y Toueg [1994] se refieren:

- *Fallos por omisión de envío:* pérdida de mensajes entre el proceso emisor y el búfer de mensajes de salida.
- *Fallos por omisión de recepción:* la pérdida de mensajes entre el búfer de mensajes de entrada y el proceso receptor como.
- *Fallos por omisión del canal:* la pérdida de mensajes entre los búferes.

Fallos arbitrarios. El término *arbitrario,* o fallo bizantino, se emplea para describir la peor semántica de fallo posible, en la que puede ocurrir cualquier tipo de error. Por ejemplo, un proceso podría cargar valores equivocados en su área de datos, o podría responder un valor incorrecto a una petición.

Un fallo arbitrario en un proceso es aquel en el que se omiten pasos deseables para el procesamiento o se realizan pasos no intencionados de procesamiento. En consecuencia los fallos arbitrarios en los procesos no pueden detectarse observando si el proceso responde a las invocaciones, dado que podría omitir arbitrariamente la respuesta.

Los canales de comunicación sufren fallos arbitrarios; por ejemplo: los contenidos de un mensaje pueden estar corruptos o se pueden repartir mensajes no existentes, o incluso algunos mensajes auténticos pueden repartirse más de una vez. Los fallos arbitrarios de los canales de comunicación son raros porque el software de comunicación es capaz de reconocerlos y rechazar los mensajes fallidos.

Fallos de temporización. Los fallos de temporización se aplican en los sistemas distribuidos síncronos donde se establecen límites en el tiempo de ejecución de un proceso, en el tiempo de reparto de un mensaje y en la tasa de deriva de reloj.

Fiabilidad y comunicación uno a uno.
El término *comunicación fiable* se define en términos de validez e integridad, definidos como sigue:

Validez: Cualquier mensaje en el búfer de mensajes salientes eventualmente llegará al búfer de mensajes entrantes;

Integridad: El mensaje recibido es idéntico al enviado, y no se entregan mensajes por duplicado.

Modelo de fallos

Las amenazas a la **“Integridad”** provienen de dos fuentes independientes:

- Cualquier protocolo que retransmita mensajes pero no rechace un mensaje que llegue dos veces. Los protocolos pueden adjuntar números de secuencia a los mensajes para detectar aquellos que se reparten por duplicado.

- Usuarios malintencionados que insertan mensajes espurios, repiten mensajes antiguos o modifican mensajes auténticos. Se pueden tomar medidas de seguridad para mantener la propiedad de integridad en caso de tales ataques.

Figura 2.11
Fallos por omisión y fallos arbitrarios

Clase de fallo	*Afecta a*	*Descripción*
Fallo-parada	Proceso	El proceso para y permanece parado. Otros procesos pueden detectar este estado.
Ruptura	Proceso	El proceso para y permanece parado. Otros procesos pueden no ser capaces de detectar este estado.
Omisión	Canal	Un mensaje insertado en el búfer de mensajes salientes nunca llega al búfer de mensajes entrantes del otro extremo.
Omisión de envío	Proceso	Un proceso completa *envía*, pero el mensaje no es colocado en su búfer de mensajes salientes.
Omisión de recepción	Proceso	El mensaje es colocado en la cola de mensajes del proceso, pero el proceso no lo recibe.
Arbitrario (Bizantino)	Proceso o canal	El proceso/canal presenta un comportamiento arbitrario: puede enviar/transmitir arbitrariamente mensajes en instantes arbitrarios, cometer omisiones; un proceso puede parar o realizar un paso incorrecto.

Figura 2.11. Fallos por omisión y fallos arbitrarios.

Figura 2.12
Fallo de temporización

Clase de fallo	*Afecta a*	*Descripción*
Reloj	Proceso	El reloj local del proceso excede el límite de su tasa de deriva sobre el tiempo real.
Prestaciones	Proceso	El proceso excede el límite sobre el intervalo entre dos pasos.
Prestaciones	Canal	La transmisión de un mensaje toma más tiempo que el límite permitido.

Figura 2.12. Fallo de temporización.

Modelo de seguridad

La seguridad de un sistema distribuido puede lograrse asegurando los procesos y los canales empleados para sus interacciones y protegiendo los objetos que encapsulan contra el acceso no autorizado.

Protección de objetos. Los usuarios pueden lanzar programas cliente que envían invocaciones al servidor para realizar operaciones sobre los objetos. El servidor realiza la operación indicada en cada invocación y envía el resultado al cliente.

Los objetos están construidos para ser usados de formas diferentes por usuarios diferentes. *Por* ejemplo, algunos objetos podrían incluir datos privados de un usuario, como un buzón de correo, y otros objetos pueden incluir datos compartidos como páginas web.

Para dar soporte a esto los ***derechos de acceso*** especifican a quién se permite realizar ciertas operaciones de un objeto, por ejemplo, a quién se permite leer o escribir su estado.

Así, debemos incluir a los usuarios en nuestro modelo como los beneficiarios de los derechos de acceso. Esto se realiza asociando a cada invocación y a cada resultado la autoridad con que se ordena. Tal autoridad se denomina "***principal***".

Un principal pudiera ser un usuario o un proceso.

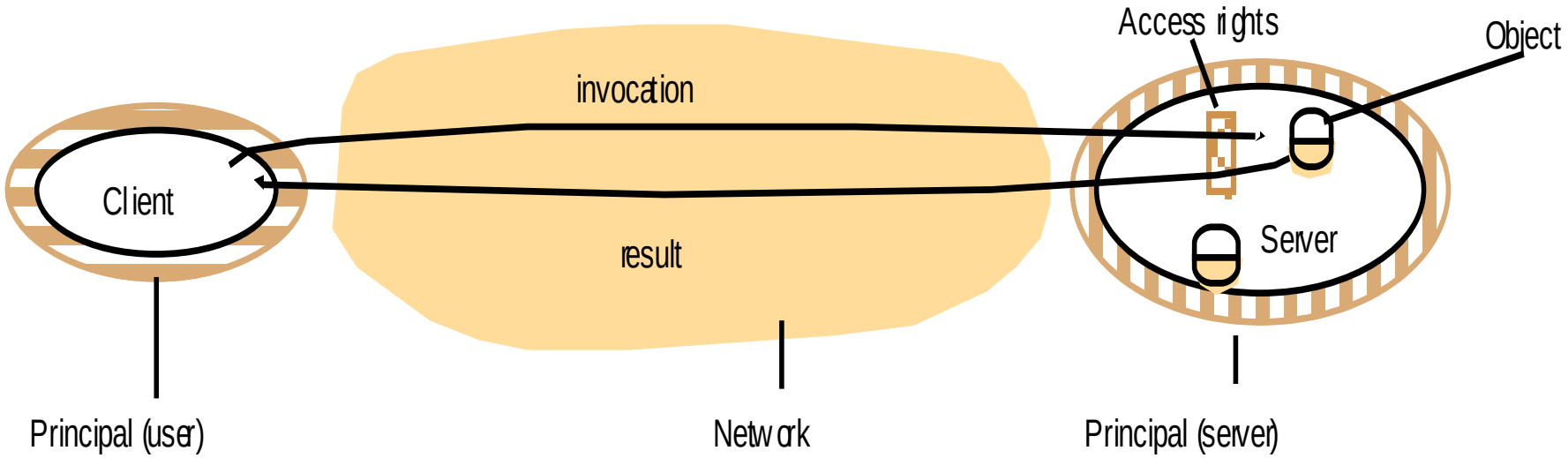
En nuestra ilustración, la invocación proviene de un usuario y el resultado de un servidor.

El servidor es responsable de verificar la identidad del principal tras cada invocación y comprobar que dispone de los suficientes derechos de acceso como para realizar la operación requerida sobre el objeto particular invocado, rechazando aquellas que no dispongan de ellos.

El cliente puede comprobar la identidad del principal tras el servidor para asegurar que el resultado proviene de servidor requerido.

Asegurar procesos y sus interacciones. Los procesos interaccionan enviando mensajes. Los mensajes están expuestos a un ataque dado que la red y el servicio de comunicación que usan es abierto, de modo que cualquier par de procesos puedan interactuar. Los servidores y procesos parejos exponen sus interfaces, permitiendo ser destino de los mensajes de otros procesos.

Figura 2.13
Objetos y principales

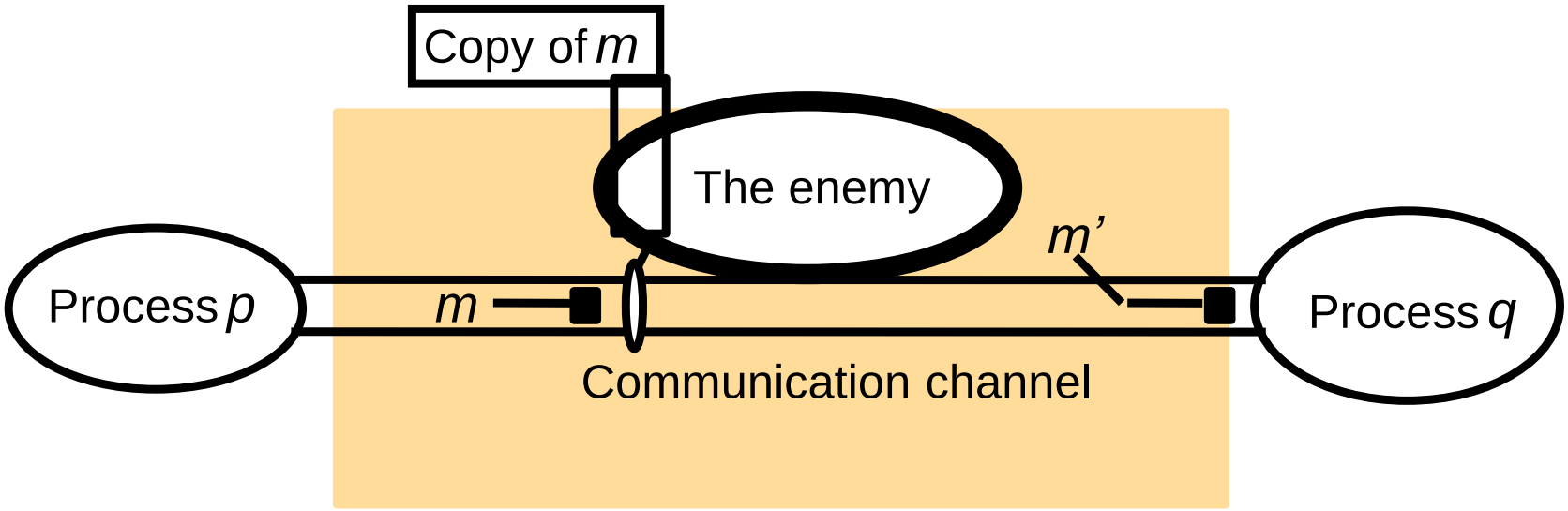


El enemigo. Para modelar amenazas de seguridad, postulamos que el enemigo (también conocido como adversario) es capaz de enviar cualquier mensaje a cualquier proceso y también de leer o copiar cualquier mensaje entre un par de procesos, como se muestra en la Figura 2.14.

Tales ataques pueden cometerse simplemente utilizando un computador conectado a una red para lanzar un programa que lea los mensajes de la red dirigidos a otros computadores de la misma red, o un programa que genere mensajes que realicen peticiones falsas a servicios dando a entender que provienen de usuarios autorizados. El ataque puede provenir de un computador legítimamente conectado a la red o también de alguna conectada de forma no autorizada.

Las amenazas de un enemigo potencial se discutirán según los siguientes apartados: *amenazas a procesos, amenazas a los canales de comunicación* y *denegación de servicio.*

Figura 2.14
El enemigo



Amenazas a procesos: Cualquier proceso diseñado para admitir peticiones puede recibir un mensaje de cualquier otro proceso del sistema distribuido, y bien pudiera no ser capaz de determinar la identidad del emisor. Los protocolos de comunicación como IP incluyen la dirección del computador origen de cada mensaje, pero no es problema para un enemigo el generar un mensaje con una dirección fuente falsa. Esta carencia de conocimiento fiable del origen de un mensaje es una amenaza al correcto funcionamiento, tanto de los servidores como de los clientes.

Servidores: dado que un servidor puede recibir invocaciones de muchos clientes diferentes, no necesariamente puede determinar la identidad del principal que se halla tras cualquier invocación particular.

Incluso si un servidor requiere la inclusión de la identidad del principal en cada invocación, un enemigo podría generar una invocación con una identidad falsa.

Sin un conocimiento fiable de la identidad del emisor, un servidor no puede decir si realizar la operación o rechazarla. Por ejemplo, un servidor de correo pudiera no saber si el usuario tras una invocación que recupera un correo de un buzón particular está autorizado para ello, o si era una petición de un enemigo.

Clientes: cuando un cliente recibe el resultado de una invocación de un servidor, no necesariamente puede decir si la fuente del mensaje resultado es del servidor que se pretendía o de un enemigo, quizás uno que esté suplantando *(spoofing)* el servidor de correo. Entonces el cliente pudiera recibir un resultado no relacionado con la invocación original, tal como un correo falso (uno que no esté en el buzón de correo del usuario).

Amenazas a los canales de comunicación: Un enemigo puede copiar, alterar o insertar mensajes según viajan a través de la red y sus pasarelas.

Esto es una amenaza a la privacidad y a la integridad de la información según ésta viaja a través de la red, y a la integridad del sistema. Por ejemplo, un mensaje resultado que contenga un correo de un usuario pudiera ser visto por otro usuario o podría ser convertido en algo bastante diferente.

Otra forma de ataque es la pretensión de hacer acopio de mensajes para volver a enviarlos más tarde, haciendo posible volver a usar el mismo mensaje una y otra vez. Por ejemplo, alguien podría beneficiarse de un mensaje en el que se invoca una petición de transferencia de dinero de una cuenta bancaria a otra.

Todas estas amenazas pueden vencerse empleando *canales seguros,* y que se basan en la criptografía y la autenticación.

Canales seguros: La encriptación y la autenticación se emplean para construir canales seguros en forma de capa de servicio sobre los servicios de comunicación existentes. Un canal seguro es un canal de comunicación que conecta un par de procesos, cada uno de los cuales actúa en representación de un principal, según se muestra en la Figura 2.15.

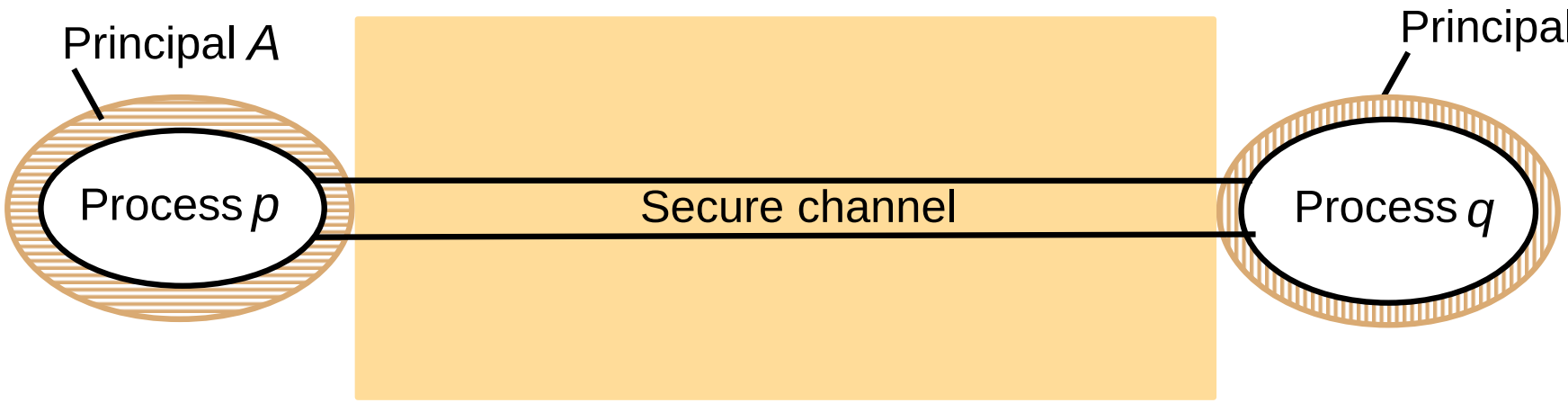


Figura 2.15
Canales seguros

Un canal seguro presenta las siguientes propiedades:

- Cada proceso conoce bien la identidad del principal en cuya representación se ejecuta otro proceso. Si un cliente y un servidor se comunican vía un canal seguro, el servidor conoce la identidad del principal tras las invocaciones y puede comprobar sus derechos de acceso antes de realizar una operación. Esto permite que el servidor proteja sus objetos correctamente y permite al cliente estar seguro de estar recibiendo resultados de un servidor que actúa de *buena fe.*

- Un canal seguro asegura la privacidad y la integridad (protección contra la manipulación) de los datos transmitidos por él.
- Cada mensaje incluye un sello de carácter temporal, de tipo físico o lógico, para prevenir el reenvío o la reordenación de los mensajes.

Otras posibilidades de amenaza de un enemigo.

Denegación de servicio: ésta es una forma de ataque en la que el enemigo interfiere con las actividades de los usuarios autorizados mediante un número excesivo de invocaciones sin sentido sobre servicios o la red, la que resulta en una sobrecarga de los recursos físicos (ancho de banda, capacidad de procesamiento del servidor).

Tales ataques suelen tener el fin de retrasar o impedir las acciones de los otros usuarios. Por ejemplo, podría desconectarse la operación de las cerraduras electrónicas de una edificación por medio de un ataque que sature el computador que las controla con peticiones inválidas.

Código móvil: el código móvil despierta nuevos e interesantes problemas para cualquier proceso que reciba y ejecute código de programas proveniente de algún otro sitio.

Tal código podría jugar fácilmente el papel de caballo de Troya, pretendiendo cumplir con un objetivo inocente pero, de hecho, incluyendo código que accede o modifica los recursos que están disponibles legítimamente para los procesos del host, pero no para el creador del código.

Los métodos mediante los cuales es posible llevar a cabo tales ataques son muchos y variados, en consecuencia el entomo del host debe construirse cuidadosamente para evitarlos. Muchas de estas cuestiones han sido tenidas en cuenta en Java y otros sistemas de código móvil, pero la historia reciente sobre el tema muestra la desnudez de algunas debilidades bastante embarazosas. Esta cuestión muestra la necesidad de realizar un análisis riguroso sobre el diseño de todos los sistemas seguros.

Aplicaciones de los modelos de seguridad. Pudiera pensarse que obtener sistemas distribuidos seguros debiera ser un asunto evidente que implica el control del acceso a los objetos según permisos de acceso predefinidos y el empleo de canales de comunicación seguros.

El empleo de técnicas de seguridad como la encriptación y el control de acceso conlleva un costo de procesamiento y administración sustancial. El modelo de seguridad esquematizado anteriormente da las bases del análisis y el diseño de sistemas seguros en los que se mantienen los costos al mínimo; pero las amenazas a un sistema distribuido se presentan desde muchos flancos, y se hace necesario un análisis cuidadoso de estas amenazas, que pudieran provenir de todas las fuentes posibles en el entorno de red del sistema, el entorno físico y el entorno humano.

Este análisis involucra la construcción de un *modelo de amenazas* que tabula todas las formas de ataque a las que se encuentra expuesto el sistema y una evaluación de los riesgos y consecuencias de cada uno.