2

MODELOS DEL SISTEMA

En este capítulo se proporciona una explicación de tres maneras importantes y complementarios en la que el diseño de los sistemas distribuidos de manera útil puede ser descrito y discutido:

Los modelos físicos considerar los tipos de equipos y dispositivos que constituyen un sistema y su interconectividad, sin detalles de tecnologías específicas.

Los modelos arquitectónicos describen un sistema en términos de las tareas de cálculo y de comunicación realizadas por sus elementos computacionales; los elementos computacionales ser ordenadores individuales o agregados de ellos apoyados por las interconexiones de red adecuados. *Servidor de cliente* y *de igual a igual* son dos de las formas más comunes de modelo de arquitectura para sistemas distribuidos.

modelos fundamentales tomar una perspectiva abstracta con el fin de describir las soluciones a los problemas individuales que se enfrentan los sistemas más distribuidos.

No hay tiempo global en un sistema distribuido, por lo que los relojes en equipos diferentes, no necesariamente dan al mismo tiempo que los otros. Todas las comunicaciones entre procesos se consigue por medio de mensajes. La comunicación de mensajes a través de una red informática puede verse afectada por los retrasos, pueden sufrir de una variedad de fallas y es vulnerable a ataques de seguridad. Estas cuestiones se abordan en tres modelos:

- El modelo de interacción ocupa de rendimiento y con la dificultad de establecer límites de tiempo en un sistema distribuido, por ejemplo, para la entrega de mensajes.
- El modelo de insuficiencia intenta dar una especificación precisa de los fallos que pueden ser exhibidos por los procesos y canales de comunicación. Define fiable comunicación y procesos correctos.
- El modelo de seguridad se analizan las posibles amenazas a los procesos y canales de comunicación. Se introduce el concepto de un canal seguro, que es segura contra esas amenazas.

2.1 Introducción

Los sistemas que están destinados para su uso en entornos del mundo real deben estar diseñadas para funcionar correctamente en la gama más amplia posible de las circunstancias y en la cara de muchas dificultades y amenazas posibles (para algunos ejemplos, véase el recuadro en la parte inferior de esta página).

La discusión y los ejemplos del capítulo 1 sugieren que los sistemas distribuidos de diferentes tipos comparten importantes propiedades subyacentes y dan lugar a problemas de diseño comunes. En este capítulo se muestra cómo las propiedades y los problemas de diseño de sistemas distribuidos pueden ser capturados y discutidos a través del uso de modelos descriptivos. Cada tipo de modelo está destinado a proporcionar una descripción abstracta, simplificada pero consistente de un aspecto relevante del diseño de sistema distribuido:

Los modelos físicos son la forma más explícita en la que para describir un sistema; capturan la composición de hardware de un sistema en términos de las computadoras (y otros dispositivos, como teléfonos móviles) y sus redes de interconexión.

Los modelos arquitectónicos describen un sistema en términos de las tareas de cálculo y de comunicación realizadas por sus elementos computacionales; los elementos computacionales ser ordenadores individuales o agregados de ellos apoyados por las interconexiones de red adecuados.

modelos fundamentales tomar una perspectiva abstracta con el fin de examinar los aspectos individuales de un sistema distribuido. En este capítulo se introducen los modelos fundamentales que examinan tres aspectos importantes de los sistemas distribuidos: *modelos de interacción,* que considerar la estructura y la secuencia de la comunicación entre los elementos del sistema; *modelos de insuficiencia,* que tengan en cuenta las formas en que un sistema puede dejar de funcionar correctamente y; *modelos de seguridad,* que tengan en cuenta cómo el sistema está protegido contra los intentos de interferir con su correcto funcionamiento o robar sus datos.

Las dificultades y las amenazas para los sistemas distribuidos • Éstos son algunos de los problemas que los diseñadores de sistemas distribuidos enfrentan.

Ampliamente variables modos de uso: Las partes componentes de sistemas están sujetos a amplias variaciones en la carga de trabajo - por ejemplo, algunas páginas web se accede a varios millones de veces al día. Algunas partes de un sistema pueden ser desconectados o mal conectados parte del tiempo - por ejemplo, cuando los equipos móviles se incluyen en un sistema. Algunas aplicaciones tienen requisitos especiales para la gran ancho de banda y baja latencia de comunicación - por ejemplo, aplicaciones multimedia.

Amplia gama de entornos de sistema: Un sistema distribuido debe acomodar el hardware heterogéneo, sistemas operativos y redes. Las redes pueden diferir ampliamente en su funcionamiento - redes inalámbricas operan a una fracción de la velocidad de las redes locales. Sistemas de escalas muy diferentes, que van desde decenas de ordenadores a millones de ordenadores, deben ser compatibles.

Los problemas internos: relojes no sincronizados, los conflictos de actualizaciones de datos y muchos modos de hardware y software fracaso la participación de los componentes individuales del sistema.

Las amenazas externas: Los ataques a la integridad de los datos y el secreto, ataques de denegación de servicio.

2.2 Los modelos físicos

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

modelo físico de línea de base: Un sistema distribuido se definió en el capítulo 1 como uno en el que los componentes de hardware o software ubicados en ordenadores en red se comunican y coordinan sus acciones sólo mediante el paso de mensajes. Esto conduce a un modelo físico mínimo de un sistema distribuido como un conjunto extensible de nodos de ordenadores interconectados por una red de ordenadores para el paso requerido de mensajes.

Más allá de este modelo de línea de base, podemos identificar de manera útil a tres generaciones de sistemas distribuidos.

Los primeros sistemas distribuidos: Tales sistemas surgieron a finales de 1970 y principios de 1980 en respuesta a la aparición de la tecnología de redes de área local, por lo general Ethernet (ver sección 3.5). Estos sistemas normalmente consisten en entre 10 y 100 nodos interconectados por una red de área local, con conectividad a Internet es limitado y apoyaron una pequeña gama de servicios tales como impresoras locales compartidos y servidores de archivos, así como el correo electrónico y transferencia de archivos a través de Internet. Los sistemas individuales fueron en gran medida homogénea y la apertura no era una preocupación primaria. Proporcionar calidad de servicio era todavía muy en su infancia y fue un punto focal para gran parte de la investigación en torno a este tipo de sistemas tempranos.

sistemas distribuidos a escala de Internet: Partiendo de esta base, los sistemas distribuidos a gran escala comenzaron a surgir en la década de 1990 en respuesta al espectacular crecimiento de Internet durante este tiempo (por ejemplo, el motor de búsqueda de Google se puso en marcha por primera vez en 1996). En tales sistemas, la infraestructura física subyacente consta de un modelo físico como se ilustra en el Capítulo 1, la figura 1.3; es decir, un conjunto extensible de nodos interconectados por una *la red de redes (* La Internet). Tales sistemas se aprovechan de la infraestructura ofrecida por la Internet para llegar a ser verdaderamente global. Incorporan un gran número de nodos y proporcionar servicios de sistemas distribuidos para las organizaciones globales ya través de fronteras organizacionales. El nivel de heterogeneidad en este tipo de sistemas es significativa en términos de redes, arquitectura de computadores, sistemas operativos, lenguajes empleados y los equipos de desarrollo involucradas. Esto ha conducido a un creciente énfasis en estándares abiertos y tecnologías de middleware asociados como CORBA y, más recientemente, los servicios web. Se emplearon los servicios adicionales para proporcionar una calidad de extremo a extremo de las propiedades del servicio en tales sistemas globales.

sistemas distribuidos modernas: En los sistemas anteriores, los nodos eran computadoras típicamente de escritorio y por lo tanto relativamente estático (es decir, que queda en una ubicación física durante períodos prolongados), discretos (no incrustado dentro de otras entidades físicas) y autónomos (en gran medida independiente de otras computadoras en términos de su infraestructura física). Las tendencias clave identificados en la Sección 1.3 se han traducido en avances significativos en otros modelos físicos:

- La aparición de la informática móvil ha dado lugar a modelos físicos donde los nodos tales como ordenadores portátiles o teléfonos inteligentes pueden trasladarse de un lugar a otro en un sistema distribuido, lo que lleva a la necesidad de capacidades adicionales, tales como el descubrimiento de servicios y apoyo para la interoperación espontánea.

- La aparición de la computación ubicua ha dado lugar a un movimiento de los nodos discretos para arquitecturas donde los ordenadores están incrustados en objetos de uso cotidiano y en el medio ambiente circundante (por ejemplo, en las lavadoras o en casas inteligentes, más en general).

- La aparición de la computación en nube y, en particular, las arquitecturas de racimo ha dado lugar a un movimiento de nodos autónomos que realizan una función determinada a grupos de nodos que en conjunto proporcionan un servicio determinado (por ejemplo, un servicio de búsqueda que ofrece Google).

El resultado final es una arquitectura física con un aumento significativo en el nivel de abrazar heterogeneidad, por ejemplo, los dispositivos más pequeño embebidos utilizados en la computación ubicua a través de elementos computacionales complejos que se encuentran en la computación Grid. Estos sistemas se despliegan un conjunto cada vez más variada de tecnologías de red y ofrecen una amplia variedad de aplicaciones y servicios. Dichos sistemas involucran potencialmente a cientos de miles de nodos.

Los sistemas distribuidos de sistemas • Un informe reciente describe la aparición de sistemas ultralarge escala (ULS) distribuidos [www.sei.cmu.edu]. El informe refleja la complejidad de los sistemas distribuidos modernos, haciendo referencia a las arquitecturas tales como (físicas) *sistemas de sistemas (* que refleja el punto de vista de la Internet como una red de redes). Un sistema de sistemas puede definirse como un sistema complejo que consiste en una serie de subsistemas que son sistemas en su propio derecho y que se unen para realizar una tarea o tareas particulares.

Como un ejemplo de un sistema de sistemas, considere un sistema de gestión ambiental para la predicción de inundaciones. En tal escenario, habrá redes de sensores desplegados para vigilar el estado de los diversos parámetros medioambientales relacionados con los ríos, llanuras de inundación, los efectos de marea y así sucesivamente. Esto puede ir acompañado de sistemas que son responsables de la predicción de la probabilidad de inundaciones, mediante la ejecución (a menudo complejas) simulaciones sobre, por ejemplo, los equipos del clúster (como se discutió en el capítulo 1). Otros sistemas pueden ser establecidos para mantener y analizar datos históricos o para proporcionar sistemas de alerta temprana a las principales partes interesadas a través de teléfonos móviles.

Resumen • El desarrollo histórico general capturado en esta sección se resume en la figura 2.1, con la tabla de relieve los retos significativos asociados con los sistemas distribuidos contemporáneos en cuanto a la gestión de los niveles de heterogeneidad y la disponibilidad propiedades clave tales como la apertura y la calidad del servicio.

2.3 Los modelos arquitectónicos

La arquitectura de un sistema es su estructura en términos de componentes especificados por separado y sus interrelaciones. El objetivo general es asegurar que la estructura se reunirá presentes y futuros probables demandas en él. Las principales preocupaciones son que el sistema sea fiable, manejable, adaptable y rentable. El diseño arquitectónico de un edificio tiene aspectos similares - se determina no sólo su apariencia, sino también su estructura general y el estilo arquitectónico (gótico, neoclásico y moderno) y proporciona un marco coherente de referencia para el diseño.

Figura 2.1 Las generaciones de sistemas distribuidos

Los sistemas distribuidos: Temprano	*A escala de Internet*	*Contemporáneo*
Escala	Pequeña	Grande	Ultra-grande
Heterogeneidad	Limited (típicamente configuraciones relativamente homogéneos)	Significativo en términos de plataformas, lenguajes y middleware	Añadido dimensiones introducidas incluyendo radicalmente diferentes estilos de la arquitectura
Franqueza	No es una prioridad	prioridad significativa con la serie de normas introducido	desafío importante de investigación con las normas existentes aún no capaz de abrazar los sistemas complejos
Calidad de servicio	En su infancia	prioridad significativa con la gama de servicios introducidos	desafío importante de investigación con los servicios existentes aún no capaz de abrazar los sistemas complejos

En esta sección se describen los principales modelos arquitectónicos empleados en sistemas distribuidos - los estilos arquitectónicos de los sistemas distribuidos. En particular, se sientan las bases para una comprensión profunda de los enfoques tales como los modelos cliente-servidor, enfoques peer-to-peer, objetos distribuidos, componentes distribuidos, sistemas distribuidos y eventbased las diferencias clave entre estos estilos.

La sección adopta un enfoque de tres etapas:

- mirando el núcleo subyacente elementos arquitectónicos que sustentan los sistemas distribuidos modernos, destacando la diversidad de enfoques que existen en la actualidad;
- examinar los patrones arquitectónicos compuestos que pueden ser utilizados de manera aislada o, más comúnmente, en combinación, en el desarrollo de soluciones de sistemas distribuidos más sofisticados;
- y, por último, considerando plataformas middleware que están disponibles para apoyar a los diversos estilos de programación que emergen de los estilos arquitectónicos anteriores. Tenga en cuenta que hay muchas ventajas y desventajas asociadas con las opciones identificadas en este capítulo en términos de los elementos arquitectónicos empleados, los patrones adoptados y (en su caso) el middleware utilizarse, por ejemplo afectando el rendimiento y la eficacia del sistema resultante. La comprensión de este tipo de compensaciones es posiblemente la habilidad clave en el diseño de sistemas distribuidos.

2.3.1 Los elementos arquitectónicos

Para entender los bloques de construcción fundamentales de un sistema distribuido, es necesario tener en cuenta cuatro preguntas clave:

- ¿Cuáles son las entidades que se comunican en el sistema distribuido?

- ¿Cómo se comunican, o, más específicamente, lo *paradigma de la comunicación* se utiliza?
- Lo que (potencialmente) cambiantes roles y responsabilidades tienen en la arquitectura global?
- ¿Cómo se asignan a la infraestructura distribuida física (lo que es su *colocación)?*

• entidades comunicantes Las primeras dos preguntas anteriores son absolutamente central para la comprensión de los sistemas distribuidos; lo que se comunica y cómo esas entidades se comunican juntos definen un espacio de diseño rico para el desarrollador de sistemas distribuidos a considerar. Es útil para hacer frente a la primera pregunta de un sistema orientado y una perspectiva orientada a los problemas.

Desde el punto de vista del sistema, la respuesta es normalmente muy claro en que las entidades que se comunican en un sistema distribuido son típicamente *procesos,* que conduce a la opinión predominante de un sistema distribuido como procesos acoplados con los paradigmas de comunicación entre procesos apropiados (como se discute, por ejemplo, en el capítulo 4), con dos salvedades:

- En algunos entornos primitivos, como redes de sensores, los sistemas operativos subyacentes pueden no soportar abstracciones de proceso (o de hecho cualquier forma de aislamiento), y por lo tanto de las entidades que se comunican en tales sistemas se *linfáticos.*
- En entornos de sistemas más distribuidos, los procesos se complementan con *trapos,* por lo que, en sentido estricto, es hilos que son los puntos finales de la comunicación. En un nivel, esto es suficiente para modelar un sistema distribuido y de hecho los modelos fundamentales considerados en la sección 2.4 adoptan este punto de vista. Desde una perspectiva de programación, sin embargo, esto no es suficiente, y se han propuesto más abstracciones orientados hacia los problemas:

Objetos: Los objetos se han introducido para permitir y fomentar el uso de enfoques orientados a objetos en sistemas distribuidos (incluyendo tanto diseño orientado a objetos y lenguajes de programación orientado a objetos). En distribuido de objetos basado en se acerca, un cálculo consiste en un número de interactuar objetos que representan unidades naturales de descomposición para el dominio del problema dado. Los objetos se accede a través de interfaces, con un lenguaje de definición de interfaz asociado (o IDL) proporcionar una especificación de los métodos definidos en un objeto. objetos distribuidos se han convertido en un área importante de estudio en los sistemas distribuidos, y se le da mayor consideración a este tema en los capítulos 5 y 8.

componentes: Desde su introducción una serie de problemas significativos han sido identificados con objetos distribuidos, y el uso de la tecnología de componentes ha surgido como una respuesta directa a estas debilidades. Componentes se asemejan a objetos ya que ofrecen abstracciones orientadas a problemas para la construcción de sistemas distribuidos y también se puede acceder a través de interfaces. La diferencia clave es que los componentes especifican no sólo sus interfaces (incluidos), sino también los supuestos que hacen en términos de otros componentes / interfaces que deben estar presentes para que un componente cumple su función - en otras palabras, por lo que todas las dependencias explícita y proporcionar una contrato más completo para la construcción del sistema. Este enfoque más contractual y alienta

permite el desarrollo de otros fabricantes de componentes y también promueve un enfoque composicional más pura a la construcción de sistemas distribuidos mediante la eliminación de dependencias ocultas. middleware basado en componentes a menudo proporciona apoyo adicional para áreas clave como la implementación y el soporte para la programación del lado del servidor [Heineman y Councill 2001]. Más detalles de los enfoques basados en componentes se pueden encontrar en el capítulo 8.

Servicios web: servicios web representan el tercer paradigma importante para el desarrollo de sistemas distribuidos [Alonso *et al.* 2004]. Los servicios Web están estrechamente relacionados con los objetos y componentes, otra vez tomando un enfoque basado en la encapsulación de la conducta y el acceso a través de interfaces. Por el contrario, sin embargo, los servicios web están integrados intrínsecamente en la World Wide Web, el uso de estándares web para representar y descubrir los servicios. El consorcio World Wide Web (W3C) define un servicio web como:

> … una aplicación de software identificado por un URI, cuyas interfaces y los enlaces son susceptibles de ser definido, descrito y descubierto como artefactos XML. Un servicio Web es compatible con interacciones directas con otros agentes de software que utilizan los intercambios de mensajes basados en XML a través de protocolos basados en Internet.

En otras palabras, los servicios web están parcialmente definidos por las tecnologías basadas en la Web que adoptan. Una distinción importante aún deriva del estilo de uso de la tecnología. Mientras que los objetos y componentes se utilizan a menudo dentro de una organización para desarrollar aplicaciones fuertemente acoplados, servicios web en general se consideran como servicios completos en sí mismos que se pueden combinar para lograr servicios de valor añadido, a menudo cruzan límites de la organización y por lo tanto el logro de negocio para la integración de negocios . Los servicios Web pueden implementarse por diferentes proveedores y el uso de diferentes tecnologías subyacentes. Los servicios Web se consideran en el capítulo 9.

paradigmas de la comunicación • Ahora dirigimos nuestra atención a cómo las entidades se comunican en un sistema distribuido, y consideramos tres tipos de paradigma de la comunicación:

- comunicación entre procesos;
- invocación remota;
- la comunicación indirecta.

la comunicación entre procesos se refiere al soporte de relativamente bajo nivel para la comunicación entre procesos en sistemas distribuidos, incluyendo primitivas de paso de mensajes, acceso directo a la API ofrecida por los protocolos de Internet (programación socket) y el apoyo para la comunicación de multidifusión. Dichos servicios se discuten en detalle en el capítulo 4.

La invocación remota representa el paradigma de comunicación más comunes en sistemas distribuidos, cubriendo una gama de técnicas basadas en un intercambio bidireccional entre entidades comunicantes en un sistema distribuido y que resulta en la llamada de una operación remota, procedimiento o método, según se define más adelante (y se considera detalle en el capítulo 5):

protocolos de petición-respuesta: protocolos de petición-respuesta son efectivamente un patrón impuesta a un servicio de paso de mensajes subyacente para apoyar la computación cliente-servidor. En

en particular, tal protocolos implican típicamente un intercambio por pares de mensajes desde el cliente al servidor y luego desde el servidor de nuevo a cliente, con el primer mensaje que contiene una codificación de la operación a ser ejecutado en el servidor y también una matriz de bytes que sostiene argumentos asociados y la segundo mensaje que contiene ningún resultado de la operación, de nuevo codificado como una matriz de bytes. Este paradigma es bastante primitivo y sólo muy utilizado en sistemas embebidos que el rendimiento es de suma importancia. El enfoque también se utiliza en el protocolo HTTP describe en la Sección 5.2. La mayoría de los sistemas distribuidos elegirán utilizar llamadas a procedimientos remotos o invocación remota de métodos, como veremos más adelante, pero tenga en cuenta que ambos enfoques son compatibles con los intercambios requestreply subyacentes.

llamadas a procedimiento remoto: El concepto de una llamada a procedimiento remoto (RPC), inicialmente atribuido a Birrell y Nelson [1984], representa un gran avance intelectual importante en computación distribuida. En RPC, procedimientos en los procesos en equipos remotos puedan ser llamados como si son procedimientos en el espacio de direcciones local. El sistema RPC subyacente entonces oculta aspectos importantes de la distribución, incluyendo la codificación y decodificación de parámetros y resultados, el paso de mensajes y la conservación de la semántica necesarios para la llamada de procedimiento. Este enfoque directamente y elegantemente soporta la computación cliente-servidor con los servidores que ofrecen un conjunto de operaciones a través de una interfaz de servicio y los clientes llamar a estas operaciones directamente como si estuvieran disponibles localmente. Por lo tanto, los sistemas RPC ofrecen (como mínimo) el acceso y la transparencia de ubicación.

invocación de método remoto: invocación de método remoto (RMI) se parece mucho a llamadas de procedimiento remoto, pero en un mundo de objetos distribuidos. Con este enfoque, un objeto que llama puede invocar un método en un objeto remoto. Al igual que con RPC, los datos subyacentes son generalmente ocultas para el usuario. implementaciones RMI pueden, sin embargo, ir más allá mediante el apoyo a la identidad del objeto y la capacidad asociada a pasar identificadores de objeto como parámetros en llamadas remotas. Ellos también se benefician más generalmente de una integración más estrecha en lenguajes orientados a objetos como se discute en el capítulo 5. El conjunto anterior de técnicas de todos tienen una cosa en común: la comunicación representa una relación bidireccional entre un emisor y un receptor con remitentes dirigir explícitamente mensajes / invocaciones a los receptores asociados. Los receptores también suelen ser conscientes de la identidad de los remitentes, y en la mayoría de los casos ambas partes deben existir al mismo tiempo. En contraste, un número de técnicas han surgido mediante el cual la comunicación es indirecta, a través de una tercera entidad, lo que permite un alto grado de desacoplamiento entre emisores y receptores. En particular:

- Remitentes no necesitan saber quiénes están enviando a (*desacoplamiento espacio).*
- Emisores y receptores no tienen que existir al mismo tiempo (*desacoplar el tiempo).*

La comunicación indirecta se analiza con más detalle en el capítulo 6.

técnicas clave para la comunicación indirecta incluyen:

la comunicación de grupo: La comunicación en grupo se refiere a la entrega de mensajes a un conjunto de destinatarios y por lo tanto es un paradigma de comunicación multipartidista apoyo uno-a-muchos comunicación. La comunicación en grupo se basa en la abstracción de un grupo que está representado en el sistema por un identificador de grupo.

Los beneficiarios eligen para recibir los mensajes enviados a un grupo al unirse al grupo. Remitentes a continuación, enviar mensajes al grupo a través del identificador de grupo, y por lo tanto no necesitan saber los destinatarios del mensaje. Grupos normalmente también mantienen la pertenencia al grupo e incluyen mecanismos para hacer frente a la insuficiencia de los miembros del grupo.

Publicación-suscripción sistemas: Muchos sistemas, como el ejemplo de las operaciones financieras en el Capítulo 1, se pueden clasificar como sistemas de difusión de información en el que un gran número de productores (o editores) distribuir los elementos de información de interés (eventos) a un número igualmente grande de los consumidores (o suscriptores) . Sería complicado e ineficaz de emplear cualquiera de los paradigmas de la comunicación núcleo discutidos anteriormente para este fin y, por tanto, los sistemas de publicación-suscripción (a veces llamados también sistemas basados en eventos distribuidos) han surgido para satisfacer esta importante necesidad [Muhl *et al.* 2006]. Publicación-suscripción sistemas comparten la característica crucial de proporcionar un servicio de intermediación que garantiza eficazmente la información generada por los productores se dirige a los consumidores que desean esta información.

Las colas de mensajes: Mientras publicación-suscripción sistemas ofrecen un estilo de uno-a-muchos de comunicación, colas de mensajes ofrecen un servicio punto a punto mediante el cual los procesos de producción pueden enviar mensajes a una cola especificada y los procesos de consumo pueden recibir mensajes de la cola o de ser notificado de la llegada de nuevos mensajes en la cola. Por lo tanto, las colas ofrecen una indirección entre los procesos de producción y de consumo.

espacios de tuplas: espacios de tuplas ofrecen un servicio de comunicación más indirecta mediante el apoyo a un modelo en el que los procesos pueden colocar los elementos arbitrarios de datos estructurados, llamado tuplas, en un espacio de tuplas persistente y otros procesos pueden leer o eliminar tales tuplas del espacio de tuplas especificando patrones de interés. Dado que el espacio de tuplas es persistente, los lectores y los escritores no tienen que existir al mismo tiempo. Este estilo de programación, también conocida como comunicación generativo, fue introducido por Gelernter [1985] como paradigma de la programación en paralelo. Un número de implementaciones distribuidas También se han desarrollado, ya sea adoptando una aplicación al estilo de cliente servidor o un enfoque más descentralizado peer-to-peer.

la memoria compartida distribuida: sistemas de memoria compartida distribuida (DSM) proporcionan una abstracción para el intercambio de datos entre los procesos que no comparten memoria física. Los programadores son, sin embargo presentan con una abstracción familiar de lectura o escritura de estructuras (compartido) de datos como si estuvieran en sus propios espacios de direcciones locales, presentando así un alto nivel de transparencia de distribución. La infraestructura subyacente debe garantizar una copia se proporciona de una manera oportuna y también se ocupan de cuestiones relacionadas con la sincronización y la coherencia de los datos. Una visión general de la memoria compartida distribuida se puede encontrar en el capítulo 6.

Las elecciones arquitectónicas discutidos hasta ahora se resumen en la Figura 2.2.

Funciones y responsabilidades • En unos procesos distribuidos de sistema - o incluso objetos, componentes o servicios, incluyendo servicios web (pero en aras de la simplicidad que utilizamos el término proceso a lo largo de esta sección) - interactuar entre sí para llevar a cabo una actividad útil, por ejemplo, para soportar una sesión de chat. De este modo, los procesos asumen funciones dadas, y estas funciones son fundamentales para establecer la arquitectura general para ser

Figura 2.2 Entidades comunicantes y paradigmas de comunicación

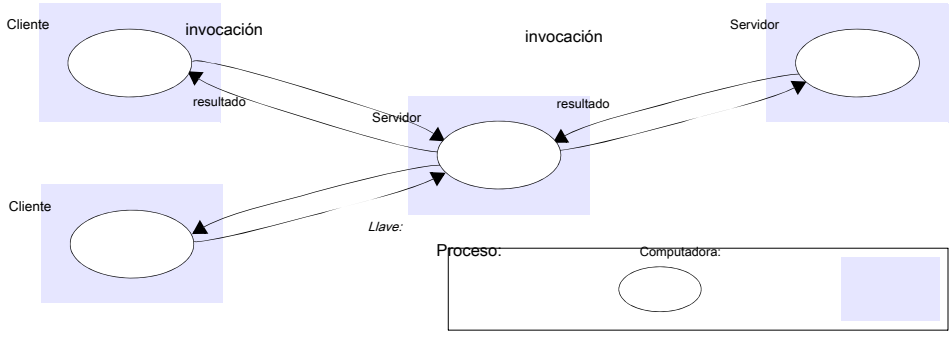
Entidades comunicantes (lo que se está comunicando)		*paradigmas de la comunicación (como se comunican)*		
Sistema orientado entidades	*entidades Problemoriented*	*la comunicación entre procesos*	*La invocación remota*	*La comunicación indirecta*
nodos	Objetos servicios	Paso de mensajes de sockets Multicast	Requestreply RPC RMI	la comunicación de grupo publicación-suscripción Colas de mensajes espacios de tuplas DSM
Procesos	Web Components			

adoptado. En esta sección, examinamos dos estilos arquitectónicos derivados del papel de los procesos individuales: cliente-servidor y peer-to-peer.

Servidor de cliente: Esta es la arquitectura que se cita a menudo más cuando se discuten los sistemas distribuidos. Es históricamente la más importante y sigue siendo el más ampliamente utilizado. Figura 2.3 ilustra la estructura sencilla en la que procesos tienen sobre las funciones de ser clientes o servidores. En particular, los procesos cliente interactúan con los procesos del servidor individuales en equipos host potencialmente separados con el fin de acceder a los recursos compartidos que administran.

Los servidores pueden a su vez ser clientes de otros servidores, como indica la figura. Por ejemplo, un servidor web es a menudo un cliente de un servidor de archivos local que gestiona los archivos en los que se almacenan las páginas web. la mayoría de los servidores web y otros servicios de Internet son clientes del servicio DNS, que traduce los nombres de dominio de Internet a direcciones de red. Otra preocupación ejemplo relacionados con la web *los motores de búsqueda,* que permite al usuario consultar los resúmenes de la información disponible en las páginas web en los sitios en todo el Internet. Estos resúmenes son hechas por los programas llamados *rastreadores web,* que corren en segundo plano en un sitio motor de búsqueda usando las peticiones HTTP para acceder a los servidores web a través de Internet. Así, un motor de búsqueda es a la vez un servidor y un cliente: responde a las consultas de los clientes de navegador y se ejecuta rastreadores web que actúan como clientes de otros servidores web. En este ejemplo, las tareas de servidor (para responder a consultas de los usuarios) y las tareas del rastreador (que realizan solicitudes a otros servidores web) son totalmente independientes; hay poca necesidad de sincronizar ellos y que puede ejecutar simultáneamente. De hecho, un motor de búsqueda típica normalmente incluye muchos hilos de ejecución concurrentes, algunos sirven a sus clientes y otras personas que ejecutan los rastreadores web. En el ejercicio 2.5, se invita al lector a considerar el único problema de sincronización que se plantea para un motor de búsqueda simultánea del tipo descrito aquí.

Figura 2.3 Clientes invocan servidores individuales



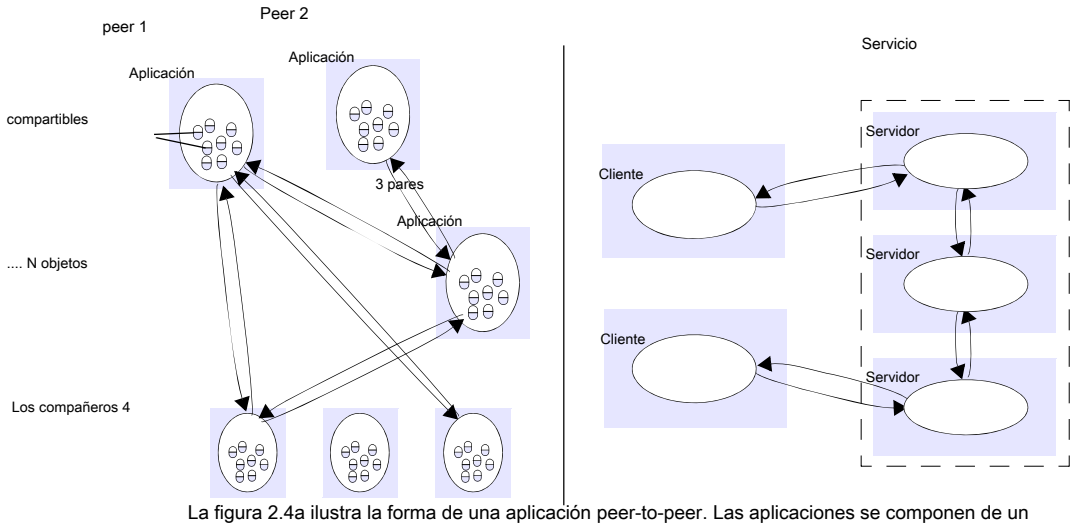
De igual a igual: En esta arquitectura de todos los procesos implicados en una tarea o actividad desempeñan papeles similares, interactuando cooperativamente como *compañeros* sin ninguna distinción entre procesos de cliente y servidor o los equipos en los que se ejecutan. En términos prácticos, todos los procesos que participan ejecutar el mismo programa y ofrecen el mismo conjunto de interfaces entre sí. Mientras que el modelo cliente-servidor ofrece un enfoque directo y relativamente simple para el intercambio de datos y otros recursos, que se redimensiona. La centralización de la provisión y gestión de servicios que implica la colocación de un servicio en una sola dirección no escala mucho más allá de la capacidad del equipo que aloja el servicio y el ancho de banda de sus conexiones de red.

Una serie de estrategias de colocación han evolucionado en respuesta a este problema (véase la discusión de la colocación más adelante), pero ninguno de ellos aborda la cuestión fundamental

- la necesidad de distribuir los recursos compartidos mucho más ampliamente con el fin de compartir las cargas informáticas y de comunicaciones incurridos en el acceso a ellos, entre un número mucho más grande de los ordenadores y los enlaces de red. La idea clave que llevó al desarrollo de sistemas peer-to-peer es que los recursos de red y computación propiedad de los usuarios de un servicio también podrían ser objeto de un uso para apoyar ese servicio. Esto tiene como consecuencia útil que los recursos disponibles para ejecutar el servicio crece con el número de usuarios.

La funcionalidad de la capacidad del hardware y el sistema operativo de los ordenadores de sobremesa de hoy en día es superior a la de los servidores de ayer, y la mayoría están equipados con permanente de banda ancha las conexiones de red. El objetivo de la arquitectura peer-to-peer es explotar los recursos (tanto de datos como de hardware) en un gran número de equipos participantes para la realización de una tarea o actividad determinada. aplicaciones y sistemas peer-to-peer se han construido con éxito que permiten a decenas o cientos de miles de ordenadores para proporcionar acceso a datos y otros recursos que colectivamente almacenan y gestionan. Uno de los primeros ejemplos fue la aplicación Napster para compartir archivos de música digital. Aunque Napster no era una pura arquitectura peer-to-peer (y también ganado notoriedad por razones más allá de su arquitectura), la demostración de la viabilidad se ha traducido en el desarrollo del modelo arquitectónico en muchas direcciones valiosas. Un ejemplo más reciente y ampliamente utilizado es el sistema de intercambio de archivos BitTorrent (discutido en más profundidad en la Sección 20.6.2).

Figura 2.4a Peer-to-peer arquitectura **la figura 2.4b** Un servicio ofrecido por varios servidores



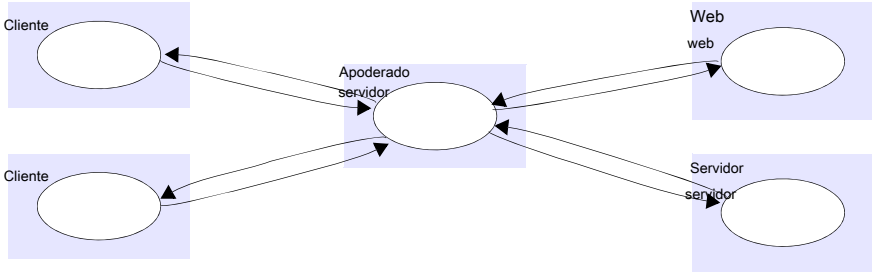
La figura 2.4a ilustra la forma de una aplicación peer-to-peer. Las aplicaciones se componen de un gran número de procesos de pares que se ejecutan en equipos independientes y el patrón de comunicación entre ellos depende enteramente de los requisitos de aplicación. Un gran número de objetos de datos son compartidos, un equipo individual tiene sólo una pequeña parte de la base de datos de aplicaciones y el almacenamiento, procesamiento y cargas de comunicación para el acceso a los objetos están distribuidos en varios equipos y enlaces de red. Cada objeto se replica en varios ordenadores para distribuir aún más la carga y para proporcionar la capacidad de recuperación en caso de desconexión de equipos individuales (como es inevitable en las grandes redes, heterogéneos a la que se dirigen los sistemas peer-to-peer).

El desarrollo de aplicaciones y middleware peer-to-peer para apoyarlos se describe en profundidad en el capítulo 10.

Colocación • La última cuestión a considerar es cómo las entidades tales como objetos o servicios de mapa en la infraestructura distribuida física subyacente que consistirá en un número potencialmente grande de las máquinas interconectadas por una red de complejidad arbitraria. La colocación es crucial en cuanto a la determinación de las propiedades del sistema distribuido, más evidente en relación con el rendimiento, sino también a otros aspectos, como la fiabilidad y la seguridad.

La cuestión de dónde colocar un cliente o servidor dada en términos de máquinas y procesos dentro de las máquinas es una cuestión de diseño cuidadoso. La colocación debe tener en cuenta los patrones de comunicación entre entidades, la fiabilidad de las máquinas de propuestas y de su carga actual, la calidad de la comunicación entre las diferentes máquinas y así sucesivamente. La colocación debe ser determinado con conocimiento de la aplicación fuerte, y hay algunas pautas universales para la obtención de una solución óptima. Por lo tanto, nos centramos principalmente en las siguientes estrategias de colocación, que pueden alterar significativamente las características de un diseño dado (aunque volvemos a la cuestión clave de la cartografía de la infraestructura física en la Sección 2.3.2, cuando nos fijamos en la arquitectura en capas):

Figura 2.5 servidor proxy Web



- mapeo de servicios a varios servidores;
- almacenamiento en caché;
- código móvil;
- agentes móviles.

Mapeo de los servicios a varios servidores: Los servicios pueden ser implementados como varios procesos de servidor en equipos host separados que interactúan como sea necesario para proporcionar un servicio a los procesos del cliente (Figura 2.4b). Los servidores pueden particionar el conjunto de objetos sobre los que se basa el servicio y distribuir esos objetos entre sí, o pueden mantener copias replicadas de ellos en varias anfitriones. Estas dos opciones se ilustran mediante los siguientes ejemplos.

La web ofrece un ejemplo común de datos divididos en el que cada servidor web gestiona su propio conjunto de recursos. Un usuario puede utilizar un navegador para acceder a un recurso en cualquiera de los servidores.

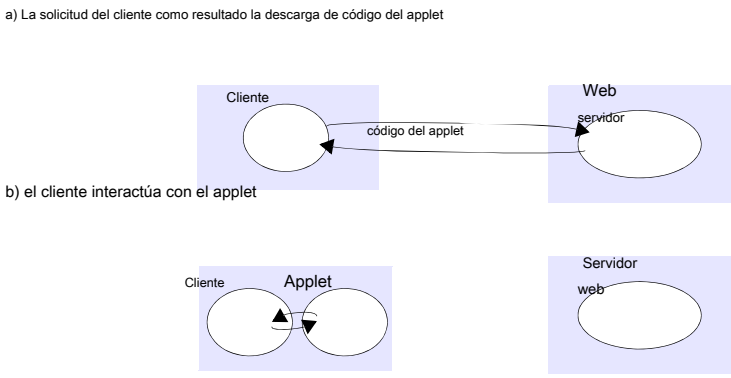
Un ejemplo de un servicio basado en los datos replicada es la Red de Servicio de Información Sun (NIS), que se utiliza para permitir que todos los ordenadores de una LAN para acceder a los mismos datos de autenticación de usuario cuando los usuarios inician sesión. Cada servidor NIS tiene su propia réplica de un archivo de contraseña común que contiene una lista de nombres de usuario de los usuarios y contraseñas cifradas. Capítulo 18 discute técnicas para la replicación en detalle.

Un tipo más estrechamente acoplado de la arquitectura de varios servidores es la agrupación, como se introdujo en el Capítulo 1. Un racimo se construye de hasta miles de placas de procesamiento de los productos básicos, y de procesamiento de servicio puede dividirse o replica entre ellos.

Almacenamiento en caché: UN *cache* es una tienda de objetos de datos utilizados recientemente que está más cerca de un cliente o de un grupo particular de clientes que los propios objetos. Cuando se recibe un nuevo objeto a partir de un servidor que se agrega al almacén de caché local, la sustitución de algunos objetos existentes si es necesario. Cuando un objeto se necesita por un proceso de cliente, el servicio de almacenamiento en caché comprueba primero la caché y suministra el objeto a partir de ahí, si una copia actualizada está disponible. Si no es así, una copia actualizada es descabellada. Cachés pueden ser co-localizados con cada cliente o pueden estar situados en un servidor proxy que puede ser compartida por varios clientes.

Cachés se utilizan ampliamente en la práctica. navegadores web mantienen una caché de las páginas web visitadas recientemente y otros recursos web en el sistema de archivos local del cliente, mediante una petición HTTP especial comprobar con el servidor original que almacenan en caché páginas son actualizada antes de mostrarlos. servidores proxy Web (Figura 2.5) proporcionan una memoria caché compartida de

Figura 2.6 applets web



Recursos Web para las máquinas cliente en un sitio o a través de varios sitios. El propósito de servidores proxy es aumentar la disponibilidad y el rendimiento del servicio mediante la reducción de la carga en los servidores de red de ancho área y web. Los servidores proxy pueden asumir otras funciones; por ejemplo, se pueden utilizar para acceder a servidores web remotos a través de un cortafuegos.

código móvil: Capítulo 1 introduce el código móvil. Los applets son un ejemplo bien conocido y ampliamente utilizado de código móvil - el usuario que ejecuta un navegador selecciona un vínculo a un applet cuyo código se almacena en un servidor web; el código se descarga en el navegador y se ejecuta allí, como se muestra en la Figura 2.6. Una ventaja de ejecutar el código descargado localmente es que puede dar una buena respuesta interactiva ya que no sufre de los retrasos o la variabilidad del ancho de banda asociado con la comunicación de red.

Acceso a los servicios de medios de código de ejecución que puede invocar sus operaciones. Algunos servicios es probable que sean normalizadas de tal forma que podamos acceder a ellos con una aplicación existente y conocida - la Web es el ejemplo más común de esto, pero incluso allí, algunos sitios web utilizan la funcionalidad no se encuentra en los navegadores estándar y requieren la descarga de código adicional. El código adicional puede, por ejemplo, comunicarse con el servidor. Considere una aplicación que requiere que los usuarios pueden mantenerse al día con los cambios que se producen en una fuente de información en el servidor. Esto no puede lograrse mediante interacciones normales con el servidor web, que siempre son iniciadas por el cliente. La solución es utilizar software adicional que funciona de una manera a menudo referido como una *empujar*

modelo - una en la que el servidor en lugar del cliente inicia interacciones. Por ejemplo, un corredor de bolsa podría proporcionar un servicio personalizado para notificar a los clientes de los cambios en los precios de las acciones; para utilizar el servicio, cada cliente tendría que descargar un subprograma especial que recibe actualizaciones desde el servidor del corredor, los muestra al usuario y tal vez realiza la compra automática y vender las operaciones desencadenadas por las condiciones establecidas por el cliente y se almacenan localmente en el ordenador del cliente .

código móvil es una amenaza para la seguridad de los recursos locales en el equipo de destino. Por lo tanto los navegadores dan applets acceso limitado a los recursos locales, utilizando un esquema discutido en la Sección 11.1.1.

Los agentes móviles: Un agente móvil es un programa en ejecución (incluyendo tanto el código y datos) que viaja desde un ordenador a otro en una red de llevar a cabo una tarea en nombre de una persona, tales como la recopilación de información, y el tiempo de regresar con los resultados. Un agente móvil puede hacer muchas invocaciones a los recursos locales en cada sitio que visita - para

ejemplo, el acceso a las entradas de base de datos individuales. Si comparamos esta arquitectura con un cliente estática haciendo invocaciones remotas a algunos recursos, posiblemente, la transferencia de grandes cantidades de datos, hay una reducción en el costo y tiempo de comunicación a través de la sustitución de las invocaciones remotas con las locales.

Los agentes móviles podrían utilizarse para instalar y mantener el software en los ordenadores dentro de una organización o de comparar los precios de los productos de un número de vendedores visitando el sitio de cada proveedor y la realización de una serie de operaciones de bases de datos. Un ejemplo temprano de una idea similar es el llamado programa de gusano desarrollado en el Xerox PARC [Shoch y Hupp 1982], que fue diseñado para hacer uso de los ordenadores de inactividad con el fin de llevar a cabo los cálculos intensivos.

Los agentes móviles (como código móvil) son una amenaza para la seguridad de los recursos en los ordenadores que visitan. El entorno de la recepción de un agente móvil debe decidir cuál de los recursos locales se debe dejar de usar, basado en la identidad del usuario en cuyo nombre el agente está actuando - Se debe incluir su identidad de una manera segura con el código y los datos de el agente móvil. Además, los agentes móviles pueden ser vulnerables a sí mismos - que pueden no ser capaces de completar su tarea si se les niega el acceso a la información que necesitan. Las tareas realizadas por los agentes móviles pueden llevarse a cabo por otros medios. Por ejemplo, los rastreadores web que necesitan acceder a recursos en servidores web en todo el trabajo de Internet con bastante éxito al hacer invocaciones remotas a los procesos del servidor. Por estas razones, la aplicabilidad de los agentes móviles puede ser limitada.

2.3.2 Los patrones arquitectónicos

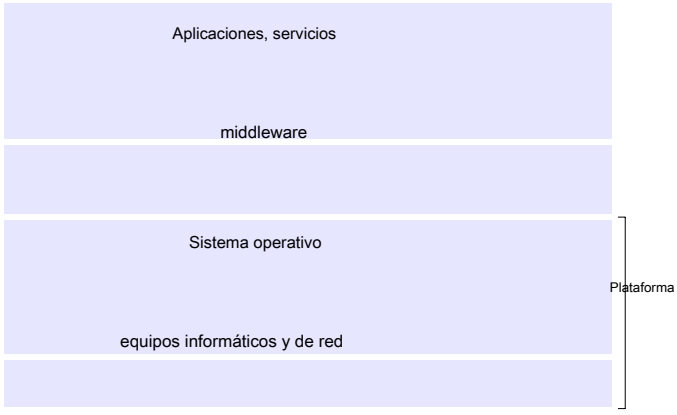
Los patrones arquitectónicos se basan en los elementos arquitectónicos más primitivas discutidos anteriormente y proporcionan estructuras recurrentes compuestos que han demostrado que funcionan bien en circunstancias dadas. No son ellos mismos necesariamente soluciones completas sino que ofrecen perspectivas parciales que, cuando se combina con otros patrones, el diseñador conducen a una solución para un problema dado dominio.

Este es un tema muy amplio, y muchos patrones arquitectónicos han sido identificados para sistemas distribuidos. En esta sección, se presentan varios patrones arquitectónicos claves en sistemas distribuidos, incluyendo capas y arquitecturas con gradas y el concepto relacionado de clientes ligeros (incluyendo el mecanismo específico de la informática de red virtual). También examinamos los servicios web como un patrón arquitectónico y dar consejos a otros que pueden ser aplicables en sistemas distribuidos.

• estratificación El concepto de estratificación es familiar y está estrechamente relacionado con la abstracción. En un enfoque por capas, un sistema complejo se divide en un número de capas, con un determinado uso toma de capa de los servicios ofrecidos por la capa de abajo. Por tanto, una capa dada ofrece una abstracción de software, con las capas superiores no estar al tanto de los detalles de implementación, o de hecho de cualquier otra capa debajo de ellos.

En términos de sistemas distribuidos, esto equivale a una organización vertical de los servicios en las capas de servicios. Un servicio distribuido puede ser proporcionado por uno o más procesos de servidor, que interactúan entre sí y con los procesos de cliente con el fin de mantener una visión de todo el sistema coherente de los recursos del servicio. Por ejemplo, un servicio de hora de red se realiza a través de Internet basado en el protocolo de tiempo de red (NTP) por procesos de servidor que se ejecutan en los servidores a través de Internet que suministran la hora actual a cualquier cliente que lo solicite y ajustar su versión de la hora actual como Un resultado de

Figura 2.7 capas de software y servicios de hardware en sistemas distribuidos



interacciones con otros. Dada la complejidad de los sistemas distribuidos, a menudo es útil para organizar este tipo de servicios en capas. Presentamos una visión común de una arquitectura en capas en la Figura 2.7 y desarrollamos este punto de vista en el aumento de detalle en los capítulos 3 a 6.

Figura 2.7 introduce los términos importantes *plataforma* y *middleware,* que definimos como sigue:

- Una plataforma para sistemas y aplicaciones distribuidas se compone de las capas de hardware y software de nivel más bajo. Estas capas de bajo nivel proporcionan servicios a las capas por encima de ellos, que se implementan de forma independiente en cada equipo, con lo que la interfaz de programación del sistema hasta un nivel que facilite la comunicación y coordinación entre los procesos.

  Intel x86 / Windows, Intel x86 / Solaris, Intel x86 / Mac OS X, Intel x86 / Linux y ARM / Symbian son los principales ejemplos.

- Middleware se define en la sección 1.5.1 como una capa de software cuyo objetivo es enmascarar la heterogeneidad y para proporcionar un modelo de programación conveniente para los programadores de aplicaciones. Middleware está representado por los procesos u objetos en un conjunto de equipos que interactúan entre sí para implementar el soporte de comunicación y el intercambio de recursos para las aplicaciones distribuidas. Tiene que ver con la provisión de materiales de construcción útiles para la construcción de componentes de software que pueden trabajar con otros en un sistema distribuido. En particular, se eleva el nivel de las actividades de comunicación de programas de aplicaciones a través del apoyo de abstracciones tales como la invocación remota de métodos; la comunicación entre un grupo de procesos; notificación de eventos; la partición, la colocación y recuperación de objetos de datos compartidos entre ordenadores cooperantes; la replicación de objetos de datos compartidos; y la transmisión de datos multimedia en tiempo real. Volvemos a este importante tema en la Sección 2.3.3 a continuación.

arquitectura por niveles • arquitecturas gradas son complementarias a capas. Mientras que la estratificación se ocupa de la organización vertical de los servicios en las capas de abstracción, por niveles es una técnica para organizar la funcionalidad de una determinada capa y colocar esta funcionalidad en

servidores apropiados y, como una consideración secundaria, en que los nodos físicos. Esta técnica es más comúnmente asociado con la organización de aplicaciones y servicios como en la Figura 2.7 anteriormente, pero también se aplica a todas las capas de una arquitectura de sistemas distribuidos.

Examinemos primero los conceptos de la arquitectura de dos y de tres niveles. Para ilustrar esto, considere la descomposición funcional de una aplicación dada, de la siguiente manera:

- la lógica de presentación, que se ocupa de la manipulación la interacción del usuario y actualizar la vista de la aplicación tal como se presenta al usuario;
- la lógica de aplicación, que se ocupa de la transformación detallada específica de la aplicación asociada a la aplicación (también referido como la lógica de negocio, aunque el concepto no se limita sólo a aplicaciones de negocios);
- la lógica de datos, que se ocupa con el almacenamiento persistente de la aplicación, típicamente en un sistema de gestión de base de datos.

Ahora, vamos a considerar la implementación de una aplicación utilizando la tecnología cliente-servidor. Las soluciones de dos niveles y de tres niveles asociados se presentan juntos para la comparación en la Figura 2.8 (a) y (b), respectivamente.

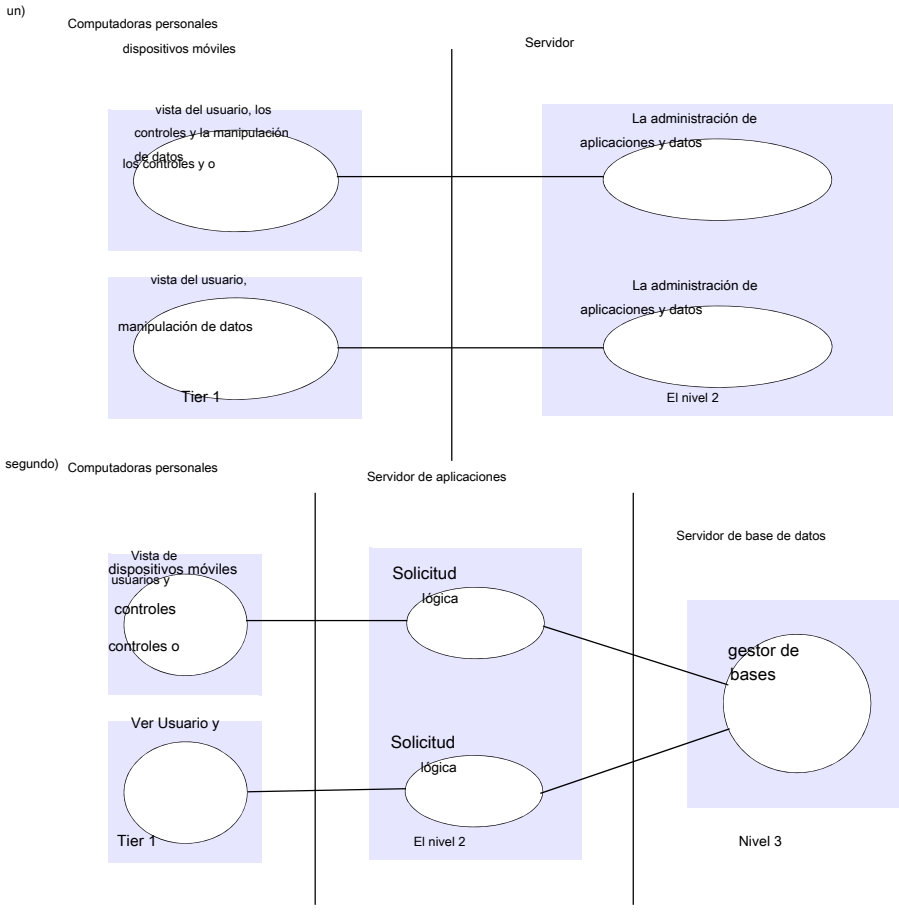
En la solución de dos niveles, los tres aspectos antes mencionados deben dividirse en dos procesos, el cliente y el servidor. Esto se realiza con mayor frecuencia por la división de la lógica de la aplicación, con un poco residen en el cliente y el resto en el servidor (aunque otras soluciones también son posibles). La ventaja de este esquema es baja latencia en términos de interacción, con sólo un intercambio de mensajes para invocar una operación. La desventaja es la división de la lógica de aplicación a través de un límite de proceso, con la consiguiente restricción sobre qué partes de la lógica pueden ser invocados directamente desde la cual otra parte.

En la solución de tres niveles, hay un mapeo uno a uno de los elementos lógicos a servidores físicos y por lo tanto, por ejemplo, la lógica de la aplicación se lleva a cabo en un lugar, que a su vez puede mejorar la mantenibilidad del software. Cada nivel también tiene un papel bien definido; por ejemplo, el tercer nivel es simplemente una base de datos que ofrece una interfaz de servicio (potencialmente estandarizada) relacional. El primer nivel también puede ser una interfaz de usuario simple que permite apoyo intrínseco a los clientes ligeros (como se discute a continuación). Las desventajas son la complejidad añadida de la gestión de tres servidores y también el tráfico de red añadido y la latencia asociada a cada operación.

Tenga en cuenta que este enfoque generaliza a las soluciones de n niveles (o de múltiples niveles) en la que un dominio de aplicación dada se divide en elementos n lógicos, cada uno asignado a un elemento servidor dado. A modo de ejemplo, Wikipedia, la enciclopedia editable públicamente en Internet, adopta una arquitectura de varios niveles para hacer frente a la gran cantidad de peticiones web (hasta 60.000 solicitudes de páginas por segundo).

El papel de AJAX: En la Sección 1.6 hemos introducido AJAX (Asynchronous JavaScript y XML) como una extensión al estilo cliente-servidor estándar de interacción utilizado en la World Wide Web. AJAX se encuentra con la necesidad de comunicación de grano fino entre un programa JavaScript que se ejecuta front-end en un navegador web y un servidor de base de back-end programa de celebración de los datos que describen el estado de la aplicación. Para recapitular, en el estilo web estándar de la interacción de un navegador envía una petición HTTP a un servidor para una página, imagen u otro recurso con una determinada URL. El servidor responde enviando una página entera que se lee ya sea desde un archivo en el servidor o generados por un programa, dependiendo de qué tipo de

Figura 2.8 De dos niveles y tres niveles arquitecturas



de recurso es identificado en la URL. Cuando el contenido resultante se recibe en el cliente, el navegador presenta de acuerdo con el método de visualización relevante por su tipo MIME (*text / html, imagen / jpg,* etc.). A pesar de que una página web puede estar compuesto por varios elementos de contenido de diferentes tipos, toda la página está compuesta y presentada por el navegador de la manera especificada en su definición página HTML.

Este estilo estándar de la interacción limita el desarrollo de aplicaciones web en varios aspectos importantes:

- Una vez que el navegador se ha emitido una petición HTTP de una nueva página web, el usuario no puede interactuar con la página hasta que el nuevo contenido HTML es recibida y presentada por el navegador. Este intervalo de tiempo es indeterminado, porque está sujeto a retrasos de redes y servidores.

- Con el fin de actualizar incluso una pequeña parte de la página actual con datos adicionales del servidor, toda una nueva página debe ser solicitada y se muestra. Esto se traduce en un retraso en la respuesta al usuario, el procesamiento adicional en el cliente y el servidor y el tráfico de red redundante.

Figura 2.9 AJAX ejemplo: actualizaciones de los marcadores de fútbol

```
nueva Ajax.Request ( 'scores.php juego = Arsenal: Liverpool,
        {OnSuccess: updateScore});

función updateScore (request) {
.....
        ( solicitud contiene el estado de la petición Ajax incluyendo el resultado devuelto.
                El resultado se analiza para obtener un texto que da la puntuación, que se utiliza para actualizar la
                parte correspondiente de la página actual.)
.....
  }
```

- El contenido de una página que se muestra a un cliente no se pueden actualizar en respuesta a los cambios en los datos de las aplicaciones realizadas en el servidor.

La introducción de JavaScript, un lenguaje de programación multiplataforma y multi-navegador que se descarga y ejecuta en el navegador, constituye un primer paso hacia la eliminación de esas limitaciones. Javascript es un lenguaje de propósito general que permita tanto la interfaz de usuario y la lógica de la aplicación para ser programados y ejecutados en el marco de una ventana del navegador.

AJAX es el segundo paso innovador que se necesitaba para que los grandes aplicaciones web interactivo para ser desarrollado y desplegado. Permite a los programas de front-end JavaScript para solicitar nuevos datos directamente de los programas de servidor. Cualquier artículo de datos pueden ser solicitados y la página actual actualizan de forma selectiva para mostrar los nuevos valores. De hecho, el extremo delantero puede reaccionar a los nuevos datos en cualquier forma que sea útil para la aplicación.

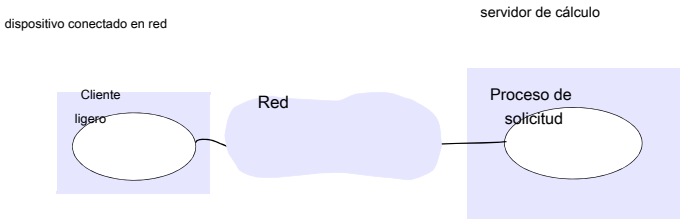
Muchas aplicaciones web permiten a los usuarios acceder y actualizar bases de datos compartidas sustanciales que pueden estar sujetas a cambios en respuesta a la entrada de otros clientes o datos Feeds recibidos por un servidor. Requieren un componente front-end de respuesta que se ejecuta en cada navegador del cliente para llevar a cabo acciones de la interfaz de usuario, tales como la selección del menú, pero también requieren acceso a un conjunto de datos que se deben a cabo en el servidor para permitir el intercambio. Tales conjuntos de datos son generalmente demasiado grandes y demasiado dinámico para permitir el uso de cualquier arquitectura basada en la descarga de una copia de todo el estado de la aplicación al cliente en el inicio de sesión de un usuario para la manipulación por parte del cliente.

AJAX es el 'pegamento' que apoya la construcción de tales aplicaciones; se proporciona un mecanismo de comunicación que permita componentes front-end que se ejecutan en un navegador para emitir solicitudes y recibir los resultados de los componentes de back-end que se ejecutan en un servidor. Clientes emitir solicitudes a través del Javascript *XmlHttpRequest* objeto, que gestiona un intercambio HTTP (véase la sección 1.6) con un proceso de servidor. Porque *XmlHttpRequest* tiene un API compleja que es también algo depende del explorador, por lo general se accede a través de una de las muchas librerías Javascript que están disponibles para apoyar el desarrollo de aplicaciones web.

En la figura 2.9 ilustramos su uso en el *prototype.js* biblioteca de Javascript [www.prototypejs.org].

El ejemplo es un extracto de una aplicación web que muestra una página de listado de puntuaciones upto actualizada para los partidos de fútbol. Los usuarios pueden solicitar actualizaciones de las puntuaciones de los juegos individuales haciendo clic en la línea correspondiente de la página, que ejecuta la primera línea de la

Figura 2.10 clientes y servidores informáticos delgada



ejemplo. los *Ajax.Request* objeto envía una solicitud HTTP a una *scores.php* programa ubicado en el mismo servidor que la página web. los *Ajax.Request* objeto se devuelve el control, lo que permite que el navegador para que siga respondiendo a otras acciones del usuario en la misma ventana o de otras ventanas. Cuando el *scores.php* programa ha obtenido el puntaje más reciente que lo devuelve en una respuesta HTTP. los *Ajax.Request* a continuación, se reactiva objeto; invoca la *updateScore* función (porque es el *onSuccess* acción), que analiza el resultado y lo inserta en el marcador en la posición relevante en la página actual. El resto de la página no se ve afectado y no se vuelve a cargar.

Esto ilustra el tipo de comunicación utilizado entre los niveles 1 y 2 componentes. A pesar de que *Ajax.Request (* y el subyacente *XmlHttpRequest* objeto) ofrece tanto la comunicación síncrona y asíncrona, la versión asíncrona se utiliza casi siempre debido a que el efecto sobre la interfaz de usuario de las respuestas del servidor retraso es inaceptable.

Nuestro simple ejemplo ilustra el uso de AJAX en una aplicación de dos niveles. En una aplicación de tres niveles del componente de servidor (*scores.php* en nuestro ejemplo) sería enviar una petición a un componente gestor de datos (típicamente una consulta SQL en un servidor de base de datos) para los datos requeridos. Dicha solicitud sería síncrona, ya que no hay ninguna razón para devolver el control al componente de servidor hasta que la solicitud sea satisfecha.

El mecanismo AJAX constituye una técnica eficaz para la construcción de aplicaciones web sensibles en el contexto de la latencia indeterminado de Internet, y ha sido ampliamente desplegado. La aplicación de Google Maps [www.google.com II] Es un ejemplo sobresaliente. Los mapas se muestran como una serie de contiguos 256 x 256 píxeles de las imágenes (llamado *azulejos).* Cuando el mapa se mueve las fichas visibles son reposicionados por el código Javascript en el navegador y se solicita a los azulejos adicionales necesarios para rellenar el área visible con una llamada AJAX a un servidor de Google. Ellos se muestran tan pronto como se reciben, pero el navegador sigue respondiendo a la interacción del usuario mientras se está a la espera.

Clientes delgados • La tendencia en computación distribuida es hacia la complejidad en movimiento lejos del dispositivo del usuario final hacia los servicios en Internet. Esto es más evidente en el movimiento hacia la computación en nube (discutido en el capítulo 1), pero también se puede ver en las arquitecturas escalonadas, como se discutió anteriormente. Esta tendencia ha dado lugar a un interés en el concepto de una *cliente ligero,* permitiendo el acceso a los servicios de red sofisticados, proporcionado por ejemplo por una solución de nube, con pocas suposiciones o demandas en el dispositivo cliente. Más específicamente, el cliente ligero término se refiere a una capa de software que soporta una interfaz de usuario basado en ventanas que es local para el usuario durante la ejecución de programas de aplicación o, más en general, para acceder a servicios en un ordenador remoto. Por ejemplo, la figura 2.10 ilustra un cliente ligero para acceder a un servidor de computación a través de Internet. La ventaja de este enfoque es que los dispositivos locales potencialmente simples (incluyendo, por ejemplo, teléfonos inteligentes

y otros dispositivos con recursos limitados) se pueden mejorar de forma significativa con una gran cantidad de servicios de red y capacidades. El principal inconveniente de la arquitectura de cliente ligero es en actividades gráficas altamente interactivos tales como CAD y procesamiento de imágenes, donde los retardos experimentados por los usuarios se incrementan hasta niveles inaceptables por la necesidad de transferir la imagen y la información de vector entre el cliente delgado y el proceso de solicitud, debido tanto a sistema operativo de red y latencias.

Este concepto ha dado lugar a la aparición de *informática de red virtual (* VNC). Esta tecnología fue introducida por primera vez por los investigadores en el Laboratorio de Investigación de Olivetti y Oracle [Richardson *et al.* 1998]; el concepto inicial ha evolucionado hasta convertirse en implementaciones como RealVNC [www.realvnc.com], Que es una solución de software, y Adventiq [www.adventiq.com], Que es una solución basada en hardware de soporte la transmisión de eventos de teclado, vídeo y ratón sobre IP (KVM-over-IP). Otros implementationss VNC incluyen Apple Remote Desktop, TightVNC y Aqua Connect.

El concepto es sencillo, proporcionar acceso remoto a las interfaces gráficas de usuario. En esta solución, un cliente VNC (o espectador) interactúa con un servidor VNC a través de un protocolo VNC. El protocolo funciona a un nivel primitivo en términos de apoyo gráficos, basado en framebuffers y con una sola operación: la colocación de un rectángulo de los datos de píxeles en una posición dada en la pantalla (algunas soluciones, tales como XenApp de Citrix funcione a un nivel más alto en términos de operaciones de la ventana [www.citrix.com]). Este enfoque garantiza el protocolo de bajo nivel funciona con cualquier sistema operativo o aplicación. Aunque es sencillo, la implicación es que los usuarios puedan acceder a sus instalaciones informáticas desde cualquier lugar en una amplia gama de dispositivos, lo que representa un importante paso adelante en la computación móvil.

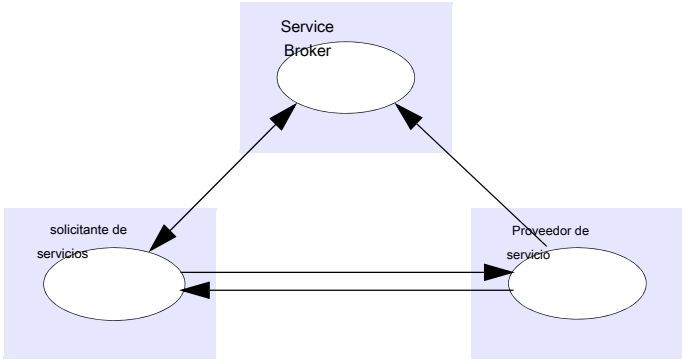
Virtual Network Computing ha reemplazado equipos de la red, un intento anterior para darse cuenta de las soluciones de cliente ligero a través de dispositivos de hardware simples y de bajo costo que son completamente dependientes de los servicios en red, la descarga de su sistema operativo y el software de aplicaciones que necesita el usuario de un servidor de archivos remoto. Dado que todos los datos de la aplicación y el código se almacena en un servidor de archivos, los usuarios pueden migrar de un ordenador de la red a otra. En la práctica, la informática de red virtual ha demostrado ser una solución más flexible y ahora domina el mercado.

Otros patrones que ocurren comúnmente • Como se mencionó anteriormente, ahora se han identificado y documentado un gran número de patrones arquitectónicos. Aquí hay algunos ejemplos clave:

- los *apoderado* patrón es un patrón comúnmente recurrente en sistemas distribuidos diseñados especialmente para apoyar transparencia de ubicación en llamadas a procedimiento remoto o invocación de método remoto. Con este enfoque, un proxy se crea en el espacio de direcciones local para representar el objeto remoto. Este proxy ofrece exactamente la misma interfaz que el objeto remoto, y el programador hace llamadas en este objeto proxy y por lo tanto no tiene que ser consciente de la naturaleza distribuida de la interacción. El papel de la representación en el apoyo a esa transparencia ubicación en RPC y RMI se discute más adelante en el capítulo 5. Nota que los proxies también se pueden utilizar para encapsular otras funcionalidades, tales como las políticas de colocación de replicación o el almacenamiento en caché.

- El uso de *corretaje* en los servicios web de manera útil puede ser visto como un patrón arquitectónico apoyar la interoperabilidad en infraestructuras distribuidas potencialmente complejas. En particular, este patrón está formado por el trío de proveedor de servicios,

Figura 2.11 El servicio web patrón arquitectónico



solicitante de servicios y corredor de servicio (un servicio que coincide con los servicios proporcionados a las solicitadas), como se muestra en la Figura 2.11 . Este patrón de corretaje se replica en muchas áreas de los sistemas distribuidos, por ejemplo, con el registro de RMI de Java y el servicio de nombres en CORBA (como se discute en los capítulos 5 y 8, respectivamente).

- *Reflexión* es un patrón que se utiliza cada vez en sistemas distribuidos como un medio de apoyar tanto la introspección (el descubrimiento dinámico de propiedades del sistema) y la intercesión (la capacidad de modificar dinámicamente estructura o comportamiento). Por ejemplo, las capacidades de introspección de Java se utilizan eficazmente en la implementación de RMI para proporcionar reenvío genérico (como se discute en la Sección 5.4.2). En un sistema reflexivo, interfaces de servicio estándar están disponibles en el nivel de base, pero una interfaz de meta-nivel también está disponible proporcionar acceso a los componentes y sus parámetros involucrados en la realización de los servicios. Una variedad de técnicas están generalmente disponibles en el meta-nivel, incluyendo la capacidad de interceptar los mensajes entrantes o invocaciones, para descubrir de forma dinámica la interfaz ofrecido por un objeto dado y para descubrir y adaptar la arquitectura subyacente del sistema. Reflexión se ha aplicado en una variedad de áreas en sistemas distribuidos, en particular dentro del campo de middleware reflectante, por ejemplo para apoyar arquitecturas de middleware más configurables y reconfigurables [Kon *et al.* 2002].

Otros ejemplos de patrones arquitectónicos relacionados con los sistemas distribuidos se pueden encontrar en Bushmann *et al. [*2007].

2.3.3 soluciones de software intermedio asociado

Middleware ya ha sido introducido en el capítulo 1 y revisado en la discusión de capas en la Sección 2.3.2 anterior. La tarea de middleware es proporcionar una abstracción de programación de alto nivel para el desarrollo de sistemas distribuidos y, a través de capas, a lo abstracto sobre la heterogeneidad en la infraestructura subyacente para promover la interoperabilidad y portabilidad. soluciones de middleware se basan en los modelos arquitectónicos introducidos en la Sección 2.3.1, y también un apoyo más complejo arquitectónico

Figura 2.12 Categorías de middleware

Categorías mayores:	*Subcategoría*	*sistemas de ejemplo*
objetos distribuidos (capítulos 5, 8)	Estándar	RM-ODP
	Plataforma	CORBA
	Plataforma	Java RMI
componentes distribuidos (capítulo 8)	Componentes ligeros componentes de peso ligero del	
	fractal OpenCom Los servidores de aplicaciones	
		SUN EJB
	Los servidores de aplicaciones	CORBA Modelo de Componentes
	Los servidores de aplicaciones	JBoss
Publicación-suscripción sistemas (capítulo 6)	-	Servicio de eventos CORBA
	-	Escriba
	-	JMS
Las colas de mensajes (capítulo 6)	-	Websphere MQ
	-	JMS
Los servicios Web (capítulo 9)	servicios web	Apache Axis
	servicios grid	El kit de herramientas Globus
Peer-to-peer (Capítulo 10)	superposiciones de enrutamiento	Pastelería
	superposiciones de enrutamiento	Tapiz
	Específica de la aplicación	Ardilla
	Específica de la aplicación	Oceanstore
	Específica de la aplicación	Hiedra
	Específica de la aplicación	Gnutella

patrones. En esta sección, se revisan brevemente las principales clases de middleware que existen hoy en día y preparar el terreno para un mayor estudio de estas soluciones en el resto del libro.

Categorías de middleware • paquetes procedimiento de llamada a distancia, como Sun RPC (Capítulo 5) y los sistemas de comunicación de grupo, tales como ISIS (capítulos 6 y 18) estaban entre los primeros ejemplos de middleware. Desde entonces una amplia gama de estilos de middleware han surgido, basado en gran medida en los modelos de arquitectura introducidas anteriormente. Presentamos una taxonomía de tales plataformas middleware en la Figura 2.12 , Incluyendo cruzada referencias a otros capítulos que cubren las distintas categorías en más detalle. Hay que destacar que las categorizaciones no son exactos y que las plataformas de middleware modernas tienden a ofrecer soluciones híbridas. Por ejemplo, muchas plataformas de objetos distribuidos tienen una distribución servicios de eventos para complementar el apoyo más tradicional para la invocación remota de métodos. Del mismo modo, muchas plataformas basadas en componentes (y de hecho otras categorías de plataforma) también soportan interfaces de servicios Web y estándares, por razones de interoperabilidad. También hay que destacar que esta taxonomía no pretende ser completa en términos del conjunto de normas y tecnologías de middleware disponibles en la actualidad,

sino que más bien se pretende que sea indicativo de las principales clases de middleware. Otras soluciones (no mostrados) tienden a ser más específico, por ejemplo oferta particular paradigmas de comunicación tales como el paso de mensajes, llamadas a procedimientos remotos, la memoria compartida distribuida, espacios de tuplas o la comunicación de grupo.

La categorización de nivel superior de middleware en la Figura 2.12 es impulsado por la elección de entidades comunicantes y paradigmas de comunicación asociados, y sigue a cinco de los principales modelos de arquitectura: objetos distribuidos, componentes distribuidos, sistemas publishsubscribe, colas de mensajes y servicios web. Estos se complementan con sistemas peerto-peer, una rama en lugar separado de middleware basado en el enfoque cooperativo discute en la Sección 2.3.1. La subcategoría de componentes distribuidos mostrados como servidores de aplicaciones también proporciona apoyo directo para arquitecturas de tres niveles. En particular, los servidores de aplicaciones proporcionan una estructura para soportar una separación entre la lógica de aplicación y almacenamiento de datos, junto con soporte para otras propiedades tales como la seguridad y la fiabilidad. Más detalle se aplaza hasta el capítulo 8.

Además de las abstracciones de programación, middleware también puede proporcionar servicios de infraestructura de sistemas distribuidos para su uso por los programas de aplicación u otros servicios. Estos servicios de infraestructura están estrechamente vinculados al modelo de programación distribuida que proporciona el middleware. Por ejemplo, CORBA (capítulo 8) proporciona a las aplicaciones una gama de servicios CORBA, incluyendo la capacidad para crear aplicaciones seguras y fiables. Como se mencionó anteriormente y discutido en el capítulo

8, los servidores de aplicaciones también proporcionan apoyo intrínseco a tales servicios.

• Limitaciones de middleware Muchas aplicaciones distribuidas dependen por completo de los servicios prestados por el middleware para apoyar sus necesidades de comunicación e intercambio de datos. Por ejemplo, una aplicación que se adapta al modelo cliente-servidor como una base de datos de nombres y direcciones, puede confiar en middleware que proporciona método sólo remoto invocación.

Se ha avanzado mucho en la simplificación de la programación de sistemas distribuidos a través del desarrollo del apoyo de middleware, pero algunos aspectos de la fiabilidad de los sistemas de requerir apoyo en el nivel de aplicación.

Considere la transferencia de grandes mensajes de correo electrónico desde el servidor de correo del remitente a la del destinatario. A primera vista esto una sencilla aplicación del protocolo de transmisión de datos TCP (discutido en el capítulo 3). Pero tenga en cuenta el problema de un usuario que intenta transferir un archivo muy grande sobre una red potencialmente poco fiables. TCP proporciona alguna detección y corrección de errores, pero no puede recuperarse de grandes interrupciones en la red. Por lo tanto, el servicio de transferencia de correo añade otro nivel de tolerancia a fallos, el mantenimiento de un registro del progreso y la reanudación de la transmisión mediante una nueva conexión TCP si el original se rompe.

Un trabajo clásico de Saltzer, Reed y Clarke [Saltzer *et al.* 1984] hace un punto similar y valiosa sobre el diseño de sistemas distribuidos, que llaman la 'el argumento de extremo a extremo'. Parafraseando a su declaración:

> Algunas funciones relacionadas con la comunicación pueden ser completamente implementadas y fiable sólo con el conocimiento y la ayuda de la aplicación de pie en los puntos extremos del sistema de comunicación. Por lo tanto, siempre que la función como una característica del propio sistema de comunicación no siempre es sensible. (Aunque una versión incompleta de la función proporcionada por el sistema de comunicación puede a veces ser útil como una mejora de rendimiento).

Se puede observar que su argumento va en contra de la opinión de que todas las actividades de comunicación se pueden abstraer lejos de la programación de aplicaciones mediante la introducción de capas de middleware apropiadas.

El meollo de su argumento es que el comportamiento correcto en programas distribuidos depende de cheques, los mecanismos de corrección de errores y medidas de seguridad en muchos niveles, algunos de los cuales requieren el acceso a los datos dentro del espacio de direcciones de la aplicación. Cualquier intento de realizar los controles en el sistema de comunicación por sí solo garantizar sólo una parte de la exactitud requerida. El mismo trabajo es, por tanto, susceptibles de ser duplicado en los programas de aplicación, perdiendo el esfuerzo de programación y, más importante, añadiendo una complejidad innecesaria y cálculos redundantes.

No hay espacio para detallar sus argumentos aquí con más detalle, pero la lectura del documento citado es muy recomendable - que está repleta de ejemplos que iluminan. Uno de los autores originales ha señalado recientemente que los beneficios sustanciales que el uso del argumento trajo al diseño de la Internet se colocan en riesgo por los recientes avances hacia la especialización de los servicios de red para satisfacer las necesidades actuales de la aplicación [www.reed.com].

Este argumento plantea un verdadero dilema para los diseñadores de middleware, y de hecho las dificultades están aumentando dada la amplia gama de aplicaciones (y las condiciones ambientales asociados) en sistemas distribuidos contemporáneos (véase el capítulo 1). En esencia, el comportamiento de middleware subyacente derecha es una función de los requisitos de una aplicación determinada o un conjunto de aplicaciones y el contexto ambiental asociado, tales como el estado y el estilo de la red subyacente. Esta percepción está impulsando interés en las soluciones sensibles al contexto y de adaptación para Middleware, como se discute en Kon *et al [* 2002].

2.4 modelos fundamentales

Todos los, bastante diferentes, modelos de sistemas comparten algunas propiedades fundamentales anteriores. En particular, todos ellos se componen de los procesos que se comunican entre sí mediante el envío de mensajes a través de una red informática. Todos los modelos comparten los requisitos de diseño para lograr las características de rendimiento y fiabilidad de los procesos y de las redes y garantizar la seguridad de los recursos en el sistema. En esta sección, se presentan los modelos basados en las propiedades fundamentales que nos permitan ser más específico acerca de sus características y de los fracasos y los riesgos de seguridad que podrían exhibir.

En general, un modelo tan fundamental debería contener sólo los ingredientes esenciales que debemos tener en cuenta a fin de comprender y razonar acerca de algunos aspectos del comportamiento de un sistema. El propósito de un modelo de este tipo es:

- Para hacer explícitos todos los supuestos relevantes acerca de los sistemas que se está modelando.
- Para hacer generalizaciones respecto a lo que es posible o imposible, teniendo en cuenta esos supuestos. Las generalizaciones pueden tomar la forma de algoritmos de propósito general o propiedades deseables que están garantizados. Las garantías son dependientes de análisis lógico y, en su caso, la prueba matemática. Hay mucho que ganar al saber lo que nuestros diseños hacen, y no lo hacen, dependen. Que nos permite decidir si un diseño funcionará si tratamos de ponerlo en práctica en un sistema en particular: sólo tenemos que preguntar si nuestras suposiciones son en ese sistema. Además, al hacer

nuestras suposiciones claras y explícitas, podemos esperar probar las propiedades del sistema utilizando técnicas matemáticas. Estas propiedades se mantenga para cualquier sistema de cumplimiento de nuestras suposiciones. Por último, mediante la abstracción solamente las entidades del sistema y las características esenciales de distancia de detalles tales como hardware, podemos aclarar nuestra comprensión de nuestros sistemas.

Los aspectos de los sistemas distribuidos que se desean capturar en nuestros modelos fundamentales están destinados a ayudarnos a discutir y razonar acerca de:

Interacción: Computation se produce dentro de los procesos; los procesos interactúan por paso de mensajes, lo que resulta en la comunicación (flujo de información) y coordinación (sincronización y ordenación de las actividades) entre procesos. En el análisis y diseño de sistemas distribuidos nos preocupa especialmente con estas interacciones. El modelo de interacción debe reflejar el hecho de que la comunicación se lleva a cabo con retrasos que a menudo son de duración considerable, y que la precisión con la que los procesos independientes pueden coordinarse está limitada por estos retrasos y por la dificultad de mantener la misma noción de tiempo a través de toda la equipos de un sistema distribuido.

Fracaso: El correcto funcionamiento de un sistema distribuido se ve amenazada cada vez que se produce un fallo en cualquiera de los equipos en los que se ejecuta (incluyendo fallos de software) o en la red que los conecta. Nuestro modelo define y clasifica los fallos. Esto proporciona una base para el análisis de sus efectos potenciales y para el diseño de sistemas que son capaces de tolerar fallos de cada tipo sin dejar de funcionar correctamente.

Seguridad: La naturaleza modular de los sistemas distribuidos y su apertura expone que ataquen por ambos agentes externos e internos. Nuestro modelo de seguridad define y clasifica las formas que este tipo de ataques pueden tomar, proporcionando una base para el análisis de las amenazas a un sistema y para el diseño de sistemas que son capaces de resistirse a ellas. Como ayudas para la discusión y el razonamiento, los modelos presentados en este capítulo se simplifican necesariamente, omitiendo muchos de los detalles de los sistemas del mundo real. Su relación con los sistemas del mundo real, y la solución en ese contexto de los problemas que los modelos ayudan a llevar a cabo, es el tema principal de este libro.

modelo 2.4.1 Interacción

La discusión de las arquitecturas de sistemas en la Sección 2.3 indica que los sistemas distribuidos fundamentalmente están compuestos de muchos procesos que interactúan de manera compleja. Por ejemplo:

- procesos de servidor de múltiples pueden cooperar entre sí para proporcionar un servicio; los ejemplos mencionados anteriormente fueron el Sistema de Nombres de Dominio, que replica las particiones y sus datos a los servidores a través de Internet, y el Servicio de Información de Red de Sun, que sigue replicado copias de los archivos de contraseñas en varios servidores en una red de área local.

- Un conjunto de procesos de pares puede cooperar entre sí para lograr un objetivo común: por ejemplo, un sistema de conferencia de voz que distribuye los flujos de datos de audio de una manera similar, pero con limitaciones estrictas de tiempo real. La mayoría de los programadores estarán familiarizados con el concepto de una *algoritmo* - una secuencia de pasos que deben tomarse con el fin de realizar un cálculo deseado. Los programas simples son

controlado por algoritmos en el que los pasos son estrictamente secuencial. El comportamiento del programa y el estado de las variables del programa se determina por ellos. Tal programa se ejecuta como un proceso único. Los sistemas distribuidos compuestos por múltiples procesos tales como los descritos anteriormente son más complejas. Su comportamiento y el estado pueden ser descritos por una *algoritmo distribuido* - una definición de los pasos para ser tomada por cada uno de los procesos de los que el sistema está compuesto, *incluyendo la transmisión de mensajes entre ellos*. Los mensajes se transmiten entre los procesos de transferir información entre ellos y para coordinar su actividad.

La velocidad a la que cada proceso continúa y la temporización de la transmisión de mensajes entre ellos no pueden, en general, pueden predecir. También es difícil de describir todos los estados de un algoritmo distribuido, ya que debe hacer frente a los fallos de uno o más de los procesos que intervienen o el fracaso de las transmisiones de mensajes.

procesos interactivos realizan toda la actividad en un sistema distribuido. Cada proceso tiene su propio estado, que consiste en el conjunto de datos que se puede acceder y actualizar, incluyendo las variables en su programa. El estado que pertenece a cada proceso es completamente privada - es decir, que no se puede acceder o actualizar cualquier otro proceso.

En esta sección, se discuten dos factores importantes que afectan a los procesos de interacción en un sistema distribuido:

- Rendimiento de comunicación es a menudo una característica limitante.
- Es imposible mantener una única idea global del tiempo.

El rendimiento de los canales de comunicación • Los canales de comunicación en nuestro modelo se realizan en una variedad de formas en sistemas distribuidos - por ejemplo, por una aplicación de corrientes o por simple mensaje que pasa a través de una red informática. La comunicación a través de una red informática tiene las siguientes características de rendimiento relacionados con la latencia, ancho de banda y el jitter:

- El retraso entre el inicio de la transmisión de un mensaje de un proceso y el comienzo de su recepción por otra se conoce como *estado latente*. La latencia incluye:
  - El tiempo necesario para el primero de una cadena de bits transmitidos a través de una red para llegar a su destino. Por ejemplo, la latencia para la transmisión de un mensaje a través de un enlace por satélite es el momento para una señal de radio en viajar al satélite y la espalda.
  - El retraso en el acceso a la red, lo que aumenta significativamente cuando la red está muy cargada. Por ejemplo, para la transmisión de Ethernet de la estación emisor espera que la red esté libre de tráfico.
  - El tiempo tomado por los servicios de comunicación del sistema operativo en tanto los procesos de envío y el de recepción, que varía de acuerdo con la carga de corriente en los sistemas operativos.
- los *ancho de banda* de una red de ordenadores es la cantidad total de información que se puede transmitir sobre ella en un momento dado. Cuando un gran número de canales de comunicación están utilizando la misma red, tienen que compartir el ancho de banda disponible.
- *Estar nervioso* es la variación en el tiempo necesario para entregar una serie de mensajes. Jitter es relevante para datos multimedia. Por ejemplo, si las muestras consecutivas de los datos de audio se reproducen con diferentes intervalos de tiempo, el sonido será muy distorsionado.

relojes de los equipos y eventos de sincronización • Cada ordenador en un sistema distribuido tiene su propio reloj interno, que puede ser utilizado por los procesos locales para obtener el valor de la hora actual. Por tanto, dos procesos que se ejecutan en equipos diferentes marcas de tiempo asociadas pueden tener cada uno con sus eventos. Sin embargo, incluso si los dos procesos leen sus relojes al mismo tiempo, sus relojes locales pueden suministrar diferentes valores de tiempo. Esto se debe a la deriva de los relojes del ordenador momento perfecto y, más importante, sus tasas de deriva son diferentes entre sí. El termino *velocidad de sincronización del reloj* se refiere a la velocidad a la que un reloj de ordenador se desvía de un reloj de referencia perfecta. Incluso si los relojes de todos los equipos de un sistema distribuido se establecen en el mismo tiempo al principio, sus relojes con el tiempo pueden variar de forma significativa menos que se apliquen correcciones.

Hay varios enfoques para la corrección de los tiempos de los relojes del ordenador. Por ejemplo, los ordenadores pueden utilizar receptores de radio para obtener lecturas de tiempo desde el sistema de posicionamiento global con una precisión de aproximadamente 1 microsegundo. Sin embargo, los receptores GPS no funcionan dentro de los edificios, ni pueden justificarse el costo para cada equipo. En lugar de ello, un equipo que tiene una fuente de tiempo exacta como el GPS puede enviar mensajes de sincronización a otras computadoras en su red. El acuerdo resultante entre los tiempos de los relojes locales es, por supuesto, afectada por los retrasos de mensajes variables. Para una discusión más detallada de la sincronización del reloj y la sincronización del reloj, consulte el Capítulo 14.

Dos variantes del modelo de interacción • En un sistema distribuido, es difícil establecer límites en el tiempo que se pueden tomar para la ejecución del proceso, la entrega de mensajes o la deriva del reloj. Dos posiciones extremas opuestas proporcionan un par de modelos simples - el primero tiene una fuerte presunción de tiempo y el segundo no hace suposiciones sobre el tiempo:

sistemas distribuidos síncronos: Hadzilacos y Toueg [1994] definir un sistema distribuido síncrono que ser uno en el que se definen los siguientes límites:

- El tiempo para ejecutar cada paso de un proceso ha conocido límites inferior y superior.
- Cada mensaje transmitido por un canal se recibe dentro de un tiempo acotado conocido.
- Cada proceso tiene un reloj local cuya tasa de deriva de tiempo real tiene un conocido atado.

Es posible sugerir posibles límites superior e inferior para el tiempo de ejecución del proceso, retardo del mensaje y las tasas de sincronización del reloj en un sistema distribuido, pero es difícil llegar a valores realistas y proporcionar garantías de los valores elegidos. A menos que los valores de los límites se pueden garantizar, cualquier diseño basado en los valores elegidos no será fiable. Sin embargo, el modelado de un algoritmo como un sistema síncrono puede ser útil para dar una idea de cómo se comportará en un sistema real distribuido. En un sistema síncrono es posible utilizar tiempos de espera de, por ejemplo, para detectar el fallo de un proceso, como se muestra en la Sección 2.4.2 abajo.

sistemas distribuidos síncronos pueden ser construidos. Lo que se requiere es que los procesos para realizar tareas con las necesidades de recursos conocidos para los que se pueden garantizar suficientes ciclos de procesador y capacidad de la red, y para los procesos que se suministran con los relojes con tasas de desplazamiento limitado.

sistemas distribuidos asíncronos: Muchos sistemas distribuidos, tales como Internet, son muy útiles sin ser capaz de calificar como sistemas síncronos. Por lo tanto necesitamos un modelo alternativo. Un sistema distribuido asíncrono es una en la que no hay límites en:

- velocidades de ejecución de proceso - por ejemplo, un paso de proceso pueden tomar sólo un picosegundo y otro de un siglo; todo lo que se puede decir es que cada paso puede tardar un tiempo arbitrariamente largo.

- retrasos en la transmisión de mensajes - por ejemplo, un mensaje del proceso de A a B proceso pueden ser entregados en el tiempo insignificante y otro pueden tardar varios años. En otras palabras, un mensaje puede ser recibido después de un tiempo arbitrariamente largo.

- Las tasas de desplazamiento de reloj - una vez más, la tasa de deriva de un reloj es arbitraria. El modelo asíncrono permite suposiciones sobre los intervalos de tiempo que participan en cualquier ejecución. Esto es exactamente modelos de Internet, en la que no hay con destino en el servidor o la carga de la red y, por lo tanto intrínseca de cuánto tiempo se tarda, por ejemplo, para transferir un archivo a través de FTP. A veces un mensaje de correo electrónico puede tardar días en llegar. El cuadro en esta página ilustra la dificultad de llegar a un acuerdo en un sistema distribuido asíncrona.

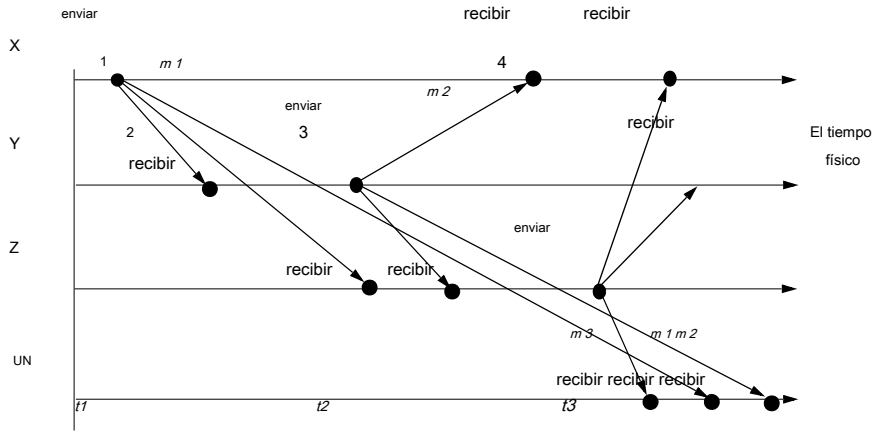
Sin embargo, algunos problemas de diseño pueden resolverse incluso con estas suposiciones. Por ejemplo, aunque la web no siempre puede proporcionar una respuesta en particular dentro de un plazo razonable, los navegadores se han diseñado para permitir a los usuarios hacer otras cosas mientras que están esperando. Cualquier solución que es válido para un sistema distribuido asíncrona también es válido para un uno síncrona.

sistemas distribuidos reales son muy a menudo asíncrona debido a la necesidad de procesos para compartir los procesadores y para los canales de comunicación para compartir el

Acuerdo en Pepperland • Dos divisiones del ejército Pepperland, 'Apple' y 'Orange', se asentaron en la parte superior de dos colinas cercanas. Además a lo largo del valle son los invasores Blue Meanies. Las divisiones Pepperland son seguros siempre y cuando se mantengan en sus campamentos, y pueden enviar mensajeros de forma fiable a través del valle de comunicarse. Las divisiones Pepperland tienen que ponerse de acuerdo sobre cuál de ellos llevará la carga contra los Blue Meanies y cuando la carga se llevará a cabo. Incluso en una Pepperland asíncrona, es posible llegar a un acuerdo sobre quién va a llevar la carga. Por ejemplo, cada división puede enviar el número de sus miembros restantes, y el de mayor dará lugar (si es un empate, la división Apple gana más de naranja). Pero cuándo deben cobrar? Por desgracia, en Pepperland asíncrono, los mensajeros son muy variables en su velocidad. Si, por ejemplo, Manzana envía un mensajero con el mensaje 'A la carga!', Orange podría no recibir el mensaje de, digamos, tres horas; o puede tener, por ejemplo, cinco minutos en llegar. En una Pepperland sincrónica, todavía hay un problema de coordinación, pero las divisiones conocen algunas limitaciones útiles: cada mensaje lleva por lo menos

min minutos y como máximo *máx* minutos en llegar. Si la división que llevará la carga envía un mensaje 'A la carga!', Espera *min* minutos; a continuación, se carga. La otra división espera durante 1 minuto después de la recepción del mensaje, entonces cobra. Su carga se garantiza que sea después de la división del líder, pero no más de (*máx* - *min* + 1) minuto después de ella.

Figura 2.13 pedidos en tiempo real de los eventos



red. Por ejemplo, si hay demasiados procesos de carácter desconocido están compartiendo un procesador, entonces el rendimiento resultante de cualquiera de ellos no se puede garantizar. Sin embargo, hay muchos problemas de diseño que no pueden ser resueltos por un sistema asíncrono que puede ser resuelto cuando se utilizan algunos aspectos del tiempo. La necesidad de que cada elemento de una secuencia de datos multimedia para ser entregado antes de un plazo es un problema. En caso de problemas tales como éstos, se requiere un modelo síncrono.

• Evento de pedidos En muchos casos, estamos interesados en saber si un evento (enviar o recibir un mensaje) en un proceso ocurrió antes, después o simultáneamente con otro evento en otro proceso. La ejecución de un sistema se puede describir en términos de eventos y ordenar que a pesar de la falta de relojes de precisión.

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

1. El usuario X envía un mensaje con el asunto *Reunión.*
2. Los usuarios Y y Z respuesta mediante el envío de un mensaje con el asunto *Re: Reunión.*

En tiempo real, el mensaje de X se envía en primer lugar, y Y lo lee y respuestas; Z lee entonces tanto el mensaje de X y la respuesta de Y y envía otra respuesta, que hace referencia a ambos de X y mensajes de Y. Pero debido a los retrasos independientes en la entrega de mensajes, los mensajes pueden ser entregados como se muestra en la Figura 2.13 , y algunos usuarios pueden ver estos dos mensajes de la orden incorrecto. Por ejemplo, el usuario A puede ver

Reunión:
Bandeja de entrada:

ít.	*De*	*Tema*
		Reunión Re:
23	ZXY	Re: Reunión
24		
25		

Si los relojes de X, de Y y de las computadoras de Z puede ser sincronizado, entonces cada mensaje podría llevar a la hora del reloj del equipo local cuando fue enviado. Por ejemplo, los mensajes *metro 1, metro 2* y *metro 3* llevaría veces *t 1, t 2* y *t 3* dónde *t 1 < t 2 < t 3*. Los mensajes recibidos se muestran a los usuarios en función de su tiempo pidiendo. Si los relojes están sincronizados o menos, entonces estas marcas de tiempo a menudo estarán en el orden correcto.

Desde relojes no se pueden sincronizar perfectamente a través de un sistema distribuido, Lamport [1978] propuso un modelo de *tiempo lógico* que se puede utilizar para proporcionar un orden entre los eventos en los procesos que se ejecutan en diferentes equipos en un sistema distribuido. tiempo lógico permite que el orden en que se presentan los mensajes que se infiere, sin recurrir a los relojes. Se presenta en detalle en el capítulo 14, pero sugerimos aquí cómo algunos aspectos de la ordenación lógica se puede aplicar a nuestro problema de ordenación de correo electrónico.

Lógicamente, sabemos que se recibe un mensaje después de que se envió. Por lo tanto podemos afirmar una ordenación lógica para pares de eventos mostrados en la figura 2.13, por ejemplo, teniendo en cuenta sólo los eventos relativos X y Y:

X envía *metro 1* antes Y recibe *metro 1;* Y envía *metro 2* antes de X recibe *metro 2*.

También sabemos que las respuestas se envían después de recibir los mensajes, por lo que tenemos el siguiente orden lógico para Y:

Y recibe *metro 1* antes de enviar *metro 2*.

tiempo lógico lleva esta idea aún más mediante la asignación de un número a cada evento correspondiente a su orden lógico, de forma que sucesos posteriores tienen números más altos que los anteriores. Por ejemplo, la Figura 2.13 muestra los números 1 a 4 en los eventos en X e Y.

2.4.2 El fracaso del modelo

yo na sistema tanto los procesos como los canales de comunicación pueden fallar distribuye - es decir, que pueden apartarse de lo que se considera ser el comportamiento correcto o deseable. El modelo de insuficiencia define las formas en las que puede ocurrir fallo con el fin de proporcionar una comprensión de los efectos de los fallos. Hadzilacos y Toueg [1994] proporcionan una taxonomía que distingue entre los fallos de los procesos y canales de comunicación. Estos se presentan en los epígrafes omisión fracasos, fallos arbitrarios y fracasos de temporización.

El modelo de fallos se utiliza en todo el libro. Por ejemplo:

- En el capítulo 4, se presentan las interfaces Java a datagrama y la comunicación corriente, que proporcionan diferentes grados de fiabilidad.
- El capítulo 5 presenta el protocolo de petición-respuesta, que soporta RMI. Sus características de fallo dependen de las características de fallo de ambos procesos y canales de comunicación. El protocolo puede ser construido de cualquiera de datagrama o comunicación corriente. La elección puede ser decidido de acuerdo a la consideración de la simplicidad de implementación, rendimiento y fiabilidad.
- Capítulo 17 presenta la confirmación de dos fases de protocolo para las transacciones. Está diseñado para completar en la cara de fracasos bien definidas de los procesos y los canales de comunicación.

• **fallos de omisión** Los defectos clasificados como *fracasos de omisión* se refieren a los casos en que un proceso o canal de comunicación falla para llevar a cabo acciones que se supone que debe hacer.

fallas en los procesos de omisión: El fracaso omisión jefe de un proceso es estrellarse. Cuando decimos que un proceso se ha estrellado nos referimos a que se ha detenido y no se ejecutará ningún otro paso de su programa nunca. El diseño de los servicios que puede sobrevivir en la presencia de fallos puede simplificarse si se puede suponer que los servicios de los que dependen accidente limpia - es decir, sus procesos, ya sea la función correctamente o si no se detienen. Otros procesos pueden ser capaces de detectar un accidente de tal por el hecho de que el proceso falla varias veces para responder a los mensajes de invocación. Sin embargo, este método de detección de accidentes se basa en el uso de

los tiempos de espera - es decir, un método en el que un proceso permite un período fijo de tiempo para que algo ocurra. En un sistema asíncrono un tiempo de espera puede indicar solamente que un proceso no está respondiendo - que puede haber estropeado o puede ser lento, o los mensajes puede no haber llegado.

Un choque proceso se llama *a prueba de detener* si otros procesos pueden detectar sin duda que el proceso se ha estrellado. A prueba de detener el comportamiento se puede producir en un sistema síncrono si los procesos utilizan los tiempos de espera para detectar cuando otros procesos no responden y los mensajes están garantizados para ser entregado. Por ejemplo, si los procesos *pag* y *q* están programados para *q* para responder a un mensaje de *pag,* y si el proceso de *pag* ha recibido ninguna respuesta del proceso *q* en un tiempo máximo medido en *pag* 'S reloj local, entonces el proceso *pag* puede concluir que el proceso

q ha fallado. La caja opuesta ilustra la dificultad de detectar fallos en un sistema asíncrono o de llegar a un acuerdo en la presencia de fallos.

fracasos de omisión de comunicación: Considere las primitivas de comunicación *enviar* y *recibir*. Un proceso *pag* lleva a cabo una *enviar* insertando el mensaje *metro* en su memoria intermedia de mensaje de salida. Los transportes de canal de comunicación *metro* a *q* Es por buffer de mensaje entrante. Proceso *q* lleva a cabo una *recibir* tomando *metro* de su búfer mensaje entrante y la entrega (véase la figura 2.14). Los almacenamientos intermedios de mensajes entrantes y salientes se proporcionan típicamente por el sistema operativo.

El canal de comunicación produce un error de omisión si no transporta un mensaje de *pag* Es por tampón mensaje de salida a *q* Es por buffer de mensaje entrante. Esto se conoce como 'mensajes que caen' y generalmente es causada por la falta de espacio de memoria intermedia en el receptor o en una puerta de enlace intermedio, o por un error de transmisión de la red, detectado por una suma de comprobación realizada con los datos del mensaje. Hadzilacos y Toueg [1994] se refieren a la pérdida de mensajes entre el proceso de envío y el búfer de mensaje saliente *fracasos, sendomission* a la pérdida de mensajes entre el búfer de mensajes entrantes y el proceso de recepción como *recibir-omisión fracasos,* y a la pérdida de mensajes en el medio como

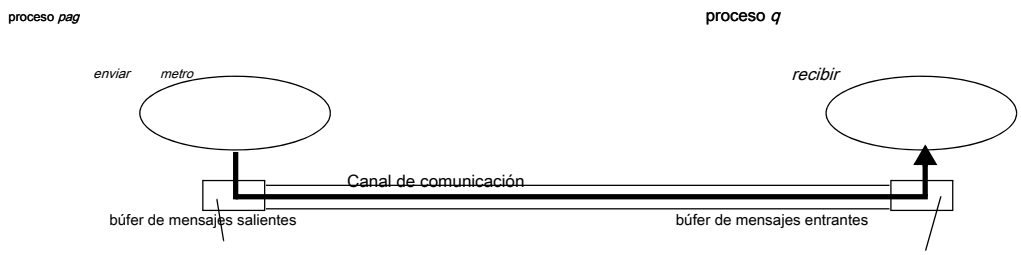
fracasos canal de omisión. Los fracasos de omisión se clasifican junto con fallos arbitrarios en la Figura 2.15.

Las fallas pueden ser clasificados de acuerdo con su gravedad. Todos los fallos que hemos descrito hasta ahora son *benigno* fracasos. La mayoría de las fallas en los sistemas distribuidos son benignos. fracasos benignos incluyen fallos de omisión, así como fallos de temporización y fracasos de rendimiento.

• **fallos arbitrarios** El termino *arbitrario* o *bizantino* fracaso se utiliza para describir los peores semántica de fallo posibles, en los que se puede producir cualquier tipo de error. Por ejemplo, un proceso puede establecer valores erróneos en sus elementos de datos, o puede devolver un valor incorrecto en respuesta a una invocación.

Un fallo arbitraria de un proceso es uno en el cual arbitrariamente omite pasos de procesamiento previsto o toma medidas de procesamiento no deseados. fallas en los procesos arbitrarios

Figura 2.14 Procesos y canales



no puede ser detectado por ver si el proceso responde a las invocaciones, porque podría omitir de forma arbitraria para responder.

Los canales de comunicación pueden sufrir de fallos arbitrarias; por ejemplo, el contenido del mensaje puede estar dañado, los mensajes pueden ser entregados inexistentes o mensajes reales se pueden entregar más de una vez. fracasos arbitrario de canales de comunicación son raros

• Detección de fallos En el caso de las divisiones Pepperland asentaron en las cimas de las colinas (véase la página 65), suponen que los Blue Meanies son, después de todo suficiente fuerza para atacar y derrotar a una u otra división, mientras acampado - es decir, que o bien puede fallar. Supongamos además que, aunque invicto, las divisiones regularmente envían mensajeros para informar de su estado. En un sistema asíncrono, ni división puede distinguir si el otro ha sido derrotado o el tiempo que se está llevando a los mensajeros para cruzar el valle intervenir es muy largo. En una Pepperland sincrónica, una división puede decir a ciencia cierta si el otro ha sido derrotado por la ausencia de un mensajero regular. Sin embargo, la otra división puede haber sido derrotado justo después de que envió el último mensajero.

Imposibilidad de llegar a un acuerdo puntual en presencia de fallas en la comunicación • Hemos supuesto que los mensajeros Pepperland siempre se las arreglan para cruzar el valle con el tiempo; pero ahora supongamos que los Blue Meanies puede capturar cualquier mensajero y evitar que lleguen. (Supondremos que es imposible para los Blue Meanies lavar el cerebro de los mensajeros para dar un mensaje equivocado - los Meanies no son conscientes de sus precursores bizantinos traidores.) ¿Pueden las divisiones manzana y naranja enviar mensajes a fin de que tanto constantemente decidir cobrar al los Meanies o ambos deciden rendirse? Por desgracia, como el teórico Pepperland Ringo la Gran demostró, en estas circunstancias, las divisiones no pueden garantizar que decidir constantemente qué hacer. Para ver esto, supongamos que al contrario que las divisiones ejecutan un protocolo de Pepperland que logra acuerdo. Cada propone 'la carga!' o 'Surrender!', y los resultados de protocolo en los dos ponerse de acuerdo sobre uno u otro curso de acción. Consideremos ahora el último mensaje enviado en cualquier ejecución del protocolo. El mensajero que lleva a que podría ser capturado por los Blue Meanies, por lo que el resultado final debe ser el mismo si el mensaje llega o no. Podemos prescindir de ella. Ahora podemos aplicar el mismo argumento para el mensaje final que queda. Pero este argumento se aplica de nuevo a ese mensaje y seguirá siendo de aplicación, por lo que deberá terminar sin mensajes enviados a todos! Esto demuestra que puede existir ningún protocolo que garantice el acuerdo entre las divisiones Pepperland si mensajeros pueden ser capturados. por lo que el resultado final debe ser el mismo si el mensaje llega o no. Podemos prescindir de ella. Ahora podemos aplicar el mismo argumento para el mensaje final que queda. Pero este argumento se aplica de nuevo a ese mensaje y seguirá siendo de aplicación, por lo que deberá terminar sin mensajes enviados a todos! Esto demuestra que puede existir ningún protocolo que garantice el acuerdo entre las divisiones Pepperland si mensajeros pueden ser capturados. por lo que el resultado final debe ser el mismo si el mensaje llega o no. Podemos prescindir de ella. Ahora podemos aplicar el mismo argumento para el mensaje final que queda. Pero este argumento se aplica de nuevo a ese mensaje y seguirá siendo de aplicación, por lo que deberá terminar sin mensajes enviados a todos! Esto demuestra que puede existir ningún protocolo que garantice el acuerdo entre las divisiones Pepperland si mensajeros pueden ser capturados.

Figura 2.15 fracasos omisión y arbitrarias

Clase de fallo	*Descripción afecta*	
A prueba de detener	Proceso proceso se detiene y permanece detenido. Otros procesos pueden detectar este estado.	
Choque	Proceso proceso se detiene y permanece detenido. Otros procesos pueden No ser capaz de detectar este estado.	
Omisión	Canalizar un mensaje insertado en un búfer de mensajes salientes Nunca llega a búfer de mensajes entrantes del otro extremo.	
Enviar-omisión	Procesar un proceso de completa una *enviar* operación, pero el mensaje No se pone en su memoria intermedia de mensaje de salida.	
Receiveomission	Un mensaje de proceso se pone en el mensaje entrante de un proceso búfer, pero ese proceso no lo recibe.	
Arbitraria (bizantino)	Proceso o canal	Proceso / canal exhibe un comportamiento arbitrario: puede enviar / transmitir mensajes arbitrarios en momentos arbitrarios o cometer omisiones; un proceso puede detener o tomar una medida incorrecta.

debido a que el software de comunicación es capaz de reconocerlos y rechazar los mensajes defectuosos. Por ejemplo, las sumas de comprobación se utilizan para detectar mensajes dañados, y números de secuencia de mensaje se pueden utilizar para detectar mensajes no existentes y duplicados.

• Sincronización fracasos fallos de temporización son aplicables en sistemas distribuidos síncronos donde los límites de la hora se ajustan en tiempo de ejecución del proceso, el tiempo de entrega de mensajes y la velocidad de deriva del reloj. fallos de temporización se enumeran en la Figura 2.16. Cualquiera de estas fallas puede resultar en respuestas no esté disponible a los clientes dentro de un intervalo de tiempo especificado.

En un sistema distribuido asíncrono, un servidor sobrecargado puede responder muy lentamente, pero no podemos decir que tiene un fallo de sincronización ya no es garantía ha sido ofrecido.

sistemas operativos en tiempo real están diseñados con el fin de proporcionar garantías de sincronización, pero son más difíciles de diseñar y pueden requerir hardware redundante. La mayoría de los sistemas operativos de propósito general, tales como UNIX no tienen que cumplir con restricciones de tiempo real.

El tiempo es particularmente relevante para ordenadores multimedia con canales de audio y vídeo. La información de vídeo puede requerir una gran cantidad de datos a transferir. La entrega de dicha información sin fallos de sincronización puede hacer exigencias muy especiales, tanto en el sistema operativo y el sistema de comunicación.

Enmascaramiento de fallos • Cada componente en un sistema distribuido se construye generalmente a partir de una colección de otros componentes. Es posible construir servicios fiables a partir de componentes que exhiben fracasos. Por ejemplo, varios servidores que contienen réplicas de datos pueden seguir prestando un servicio cuando uno de ellos se estrella. El conocimiento de las características de fallo de un componente puede permitir un nuevo servicio que ser diseñados para enmascarar el

Figura 2.16 fallos de temporización

Clase de fallo	*afecta*	*Descripción*
Reloj	Proceso	reloj local del proceso excede los límites de su velocidad de desplazamiento de tiempo real.
Actuación	Proceso	Proceso supera los límites en el intervalo entre dos pasos.
Actuación	Canal	La transmisión de un mensaje tarda más de lo establecido para realizar el atado.

fallo de los componentes de los que depende. Un servicio *máscaras* una falla, ya sea por ocultar por completo o mediante su conversión en un tipo más aceptable de fracaso. Para un ejemplo de este último, las sumas de comprobación se utilizan para enmascarar mensajes dañados, convertir eficazmente un fallo arbitrario en un fallo de omisión. Veremos en los capítulos 3 y 4 que la omisión fracasos se pueden ocultar mediante el uso de un protocolo que retransmite mensajes que no llegan a su destino. El capítulo 18 presenta el enmascaramiento mediante la replicación. Incluso bloquea el proceso pueden estar enmascarados, reemplazando el proceso y restaurar su memoria a partir de información almacenada en el disco por su predecesor.

La fiabilidad de uno-a-uno de comunicación • Aunque un canal de comunicación básico puede exhibir las fallas de omisión descritos anteriormente, es posible utilizarlo para construir un servicio de comunicación que enmascara algunos de esos fracasos.

El termino *una comunicación fiable* se define en términos de validez y la integridad de la siguiente manera:

Validez: Cualquier mensaje en el buffer de mensaje de salida finalmente se entrega al buffer de mensaje entrante.

Integridad: El mensaje recibido es idéntico a uno enviado, y no hay mensajes se entregan dos veces.

Las amenazas a la integridad provienen de dos fuentes independientes:

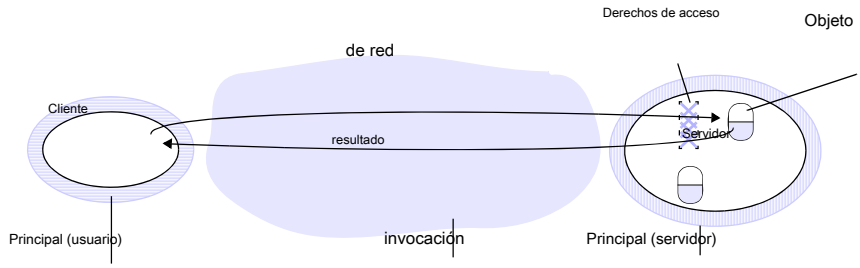
- Cualquier protocolo que retransmite los mensajes, pero no rechaza un mensaje que llega dos veces. Los protocolos pueden adjuntar los números de secuencia a los mensajes con el fin de detectar aquellos que se entregan dos veces.

- Los usuarios malintencionados que pueden inyectar mensajes espurios, reproducir mensajes antiguos o manipular mensajes. Las medidas de seguridad se pueden tomar para mantener la integridad de la propiedad en la cara de este tipo de ataques.

2.4.3 Modelo de seguridad

En el capítulo 1 identificamos la distribución de los recursos como un factor de motivación para sistemas distribuidos, y en la Sección 2.3 describimos su arquitectura en términos de procesos, potencialmente encapsular abstracciones de nivel superior, tales como objetos, componentes o

Figura 2.17 Objetos y directores



servicios, y el acceso a ellos a través de las interacciones con otros procesos. Ese modelo de arquitectura proporciona la base de nuestro modelo de seguridad:

> la seguridad de un sistema distribuido puede lograrse asegurando los procesos y los canales usados por sus interacciones y mediante la protección de los objetos que encapsulan contra el acceso no autorizado.

La protección se describe en términos de objetos, aunque los conceptos se aplican igualmente bien a los recursos de todo tipo.

• la protección de objetos La Figura 2.17 muestra un servidor que gestiona una colección de objetos en nombre de algunos usuarios. Los usuarios pueden ejecutar programas cliente que envían invocaciones al servidor para realizar operaciones en los objetos. El servidor lleva a cabo la operación especificada en cada invocación y envía el resultado al cliente.

Los objetos están destinados a ser utilizados de diferentes maneras por diferentes usuarios. Por ejemplo, algunos objetos pueden contener datos privados del usuario, tales como su buzón de correo, y otros objetos pueden contener datos como páginas web compartido. Para apoyar esto, *derechos de acceso* especificar quién está autorizado para realizar las operaciones de un objeto - por ejemplo, que se le permite leer o escribir su estado.

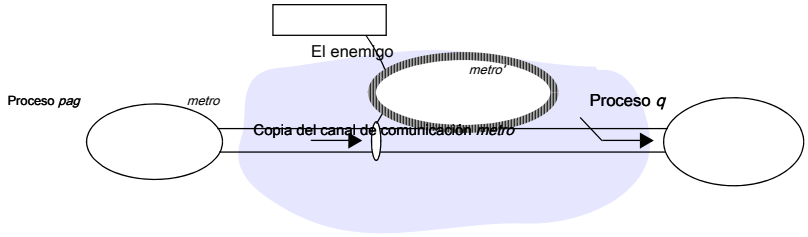
Por lo tanto hay que incluir a los usuarios en nuestro modelo como beneficiarios de derechos de acceso. Lo hacemos mediante la asociación con cada llamada y cada resultado de la autoridad de que se hayan emitido. Tal autoridad se llama una *director de escuela*. Un director puede ser un usuario o un proceso. En nuestro ejemplo, la invocación proviene de un usuario y el resultado de un servidor.

El servidor es responsable de verificar la identidad del principal detrás de cada invocación y la comprobación de que tienen derechos de acceso suficientes para realizar la operación solicitada en el objeto particular invocado, rechazando aquellas que no lo hacen. El cliente puede comprobar la identidad del principal detrás del servidor para asegurarse de que el resultado viene del servidor necesario.

procesos y sus interacciones • Asegurar Procesos interactúan mediante el envío de mensajes. Los mensajes se presentan a atacar porque la red y el servicio de comunicación que utilizan están abiertas, para permitir que cualquier par de procesos para interactuar. Los servidores y los procesos de pares exponen sus interfaces, lo que permite invocaciones a ser enviado a ellos por cualquier otro proceso.

Los sistemas distribuidos son a menudo instalación y utilización de tareas que pueden ser objeto de ataques externos de los usuarios hostiles. Esto es especialmente cierto para las aplicaciones que

Figura 2.18 El enemigo



manejar las transacciones financieras, la información confidencial o reservada o cualquier otra información cuya confidencialidad o integridad es crucial. La integridad está amenazada por violaciónes de seguridad, así como fallas en la comunicación. Por lo que sabemos que no es probable que sean las amenazas a los procesos que componen este tipo de aplicaciones y a los mensajes que viajan entre los procesos. Pero ¿cómo podemos analizar estas amenazas con el fin de identificar y derrotar a ellos? La siguiente discusión se presenta un modelo para el análisis de las amenazas de seguridad.

El enemigo • Para modelar amenazas a la seguridad, postulamos un enemigo (a veces también conocido como el adversario) que es capaz de enviar cualquier mensaje a cualquier proceso y la lectura o la copia de cualquier mensaje enviado entre un par de procesos, como se muestra en la Figura 2.18. Este tipo de ataques se pueden hacer simplemente mediante el uso de un ordenador conectado a una red para ejecutar un programa que lee los mensajes de la red dirigidos a otros equipos de la red, o un programa que genera los mensajes que hacen falsas peticiones a los servicios, que pretende provenir de los usuarios autorizados. El ataque puede venir de un ordenador que está conectado legítimamente a la red o de una que está conectado de forma no autorizada.

Las amenazas de un enemigo potencial incluyen *amenazas a los procesos* y *amenazas a los canales de comunicación.*

Las amenazas a los procesos: Un proceso que está diseñado para manejar las peticiones entrantes pueden recibir un mensaje de cualquier otro proceso en el sistema distribuido, y no pueden necesariamente determinar la identidad del remitente. Los protocolos de comunicación tales como IP sí incluyen la dirección del equipo de origen de cada mensaje, pero no es difícil para un enemigo para generar un mensaje con una dirección de origen forjado. Esta falta de conocimiento fiable de la fuente de un mensaje es una amenaza para el correcto funcionamiento de los servidores y clientes, como se explica a continuación:

servidores: Puesto que un servidor puede recibir invocaciones de muchos clientes diferentes, no necesariamente se puede determinar la identidad del principal detrás de cualquier invocación particular. Incluso si un servidor requiere la inclusión de la identidad del representado en cada invocación, un enemigo puede generar una invocación con una identidad falsa. Sin el conocimiento fiable de la identidad del remitente, el servidor no puede decir si se debe realizar la operación o para rechazarla. Por ejemplo, un servidor de correo no sabría si el usuario está autorizado detrás de una invocación que solicita un elemento de correo de un buzón especial para hacerlo, o si se trataba de una solicitud de un enemigo.

Clientela: Cuando un cliente recibe el resultado de una invocación de un servidor, no necesariamente puede decir si la fuente del mensaje de resultado es desde el servidor deseado

o de un enemigo, tal vez 'spoofing' el servidor de correo. Así, el cliente puede recibir un resultado que no estaba relacionado con la invocación original, como un elemento de correo falsa (uno que no está en el buzón del usuario).

Las amenazas a los canales de comunicación: Un enemigo puede copiar, alterar o inyectar mensajes a medida que viajan a través de la red y sus pasarelas intermedias. Estos ataques representan una amenaza para la privacidad y la integridad de la información a medida que viaja por la red y de la integridad del sistema. Por ejemplo, un mensaje de resultados que contiene elemento de correo de un usuario puede ser revelado a otro usuario o podría ser alterado para decir algo bastante diferente.

Otra forma de ataque es el intento de guardar copias de los mensajes y para reproducirlas en un momento posterior, por lo que es posible reutilizar el mismo mensaje una y otra vez. Por ejemplo, alguien podría beneficiarse volviendo a enviar un mensaje de invocación que solicita una transferencia de una suma de dinero de una cuenta bancaria a otra.

Todas estas amenazas pueden ser derrotados por el uso de *canales seguros,* que se describen a continuación y se basan en la criptografía y autenticación.

La derrota de las amenazas de seguridad • Aquí presentamos las principales técnicas en que se basan los sistemas de seguridad. Capítulo 11 discute el diseño e implementación de sistemas distribuidos seguras con mucho más detalle.

Criptografía y compartidos secretos: Supongamos que un par de procesos (por ejemplo, un cliente en particular y un servidor particular) comparten un secreto; es decir, que ambos conocen el secreto, pero ningún otro proceso en el sistema distribuido lo sabe. Entonces, si un mensaje intercambiado entre ese par de procesos incluye información que demuestra el conocimiento del remitente del secreto compartido, el destinatario sabe con certeza que el remitente era el otro proceso en el par. Por supuesto, se debe tener cuidado para asegurar que el secreto compartido no se revela a un enemigo.

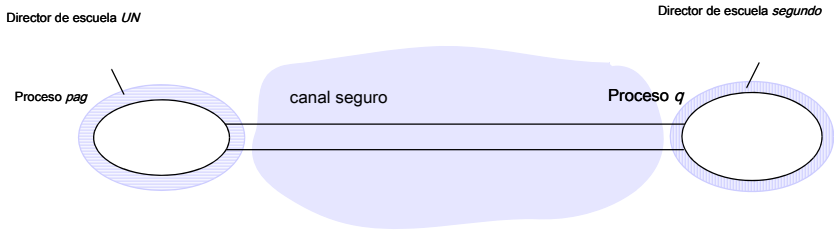
Criptografía es la ciencia de mantener mensajes seguros, y *encriptación* es el proceso de codificar un mensaje de tal manera como para ocultar su contenido. La criptografía moderna se basa en algoritmos de encriptación que utilizan claves secretas - grandes números que son difíciles de adivinar - para transformar los datos de una manera que sólo puede ser revertida con el conocimiento de la clave de descifrado correspondiente.

Autenticación: El uso de secretos compartidos y cifrado proporciona la base para la *autenticación* de mensajes - probar la identidad suministrados por sus remitentes. La técnica de autenticación básica es incluir en un mensaje cifrado que una porción contiene suficiente de los contenidos del mensaje para garantizar su autenticidad. La porción de autenticación de una petición a un servidor de archivos para leer parte de un archivo, por ejemplo, podría incluir una representación de la identidad del representado solicitante, la identidad del archivo y la fecha y hora de la solicitud, todos los cifrados con una clave secreta compartida entre el servidor de archivos y el proceso solicitante. El servidor podría descifrar esto y asegurarse de que corresponde a los datos no cifrados especificados en la solicitud.

Los canales seguros: El cifrado y la autenticación se utilizan para construir canales seguros como una capa de servicio en la parte superior de 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 nombre de un principal, como se muestra en la Figura 2.19. Un canal seguro tiene las siguientes propiedades:

- Cada uno de los procesos conoce de forma fiable la identidad del principal en cuyo nombre el otro proceso está ejecutando. Por lo tanto, si un cliente y el servidor se comunican a través de un canal seguro, el servidor conoce la identidad del principal detrás de la

Figura 2.19 Los canales seguros



invocaciones y pueden comprobar sus derechos de acceso antes de realizar una operación. Esto permite al servidor para proteger sus objetos de forma correcta y permite al cliente para asegurarse de que está recibiendo los resultados de una *De buena fe* servidor.

- Un canal seguro asegura la privacidad y la integridad (protección contra la manipulación) de los datos transmitidos a través de ella.
- Cada mensaje incluye una marca de tiempo físico o lógico para impedir que los mensajes se reproducido o reordenado.

La construcción de canales seguros se discute en detalle en el capítulo 11. Los canales seguros han convertido en una importante herramienta práctica para asegurar el comercio electrónico y la protección de la comunicación. redes privadas virtuales (VPNs, discutido en el capítulo 3) y el protocolo Secure Sockets Layer (SSL) (discutido en el capítulo 11) son instancias.

Otros posibles amenazas de un enemigo • Sección 1.5.3 introdujo brevemente dos amenazas de seguridad adicionales - ataques de denegación de servicio y el despliegue de código móvil. Reiteramos estos como posibles oportunidades para el enemigo para interrumpir las actividades de los procesos:

Negación de servicio: Esta es una forma de ataque en el que el enemigo interfiere con las actividades de los usuarios autorizados, haciendo invocaciones excesivas e inútiles sobre servicios o transmisiones de mensajes en una red, lo que resulta en una sobrecarga de los recursos físicos (ancho de banda, capacidad de procesamiento del servidor). Este tipo de ataques se hacen generalmente con la intención de retrasar o prevenir las acciones de otros usuarios. Por ejemplo, el funcionamiento de cerraduras electrónicas en un edificio podría ser desactivado por un ataque que satura el ordenador que controla las cerraduras electrónicas con las solicitudes no válidas.

código móvil: código móvil plantea nuevos e interesantes problemas de seguridad para cualquier proceso que recibe y ejecuta el código del programa de otros lugares, tales como el adjunto de correo electrónico se ha mencionado en la Sección 1.5.3. Tal código puede fácilmente jugar un papel de caballo de Troya, que pretende cumplir un objetivo inocente, pero, de hecho, incluyendo el código que accede o modifica los recursos que son legítimamente a disposición del proceso de host, pero no al autor del código. Los métodos por los que este tipo de ataques pueden ser realizadas en son muchos y variados, y el entorno de acogida deben ser construidos con mucho cuidado a fin de evitar ellos. Muchas de estas cuestiones se han abordado en Java y otros sistemas de código móvil, pero la historia reciente de este tema ha incluido la exposición de

algunas debilidades embarazosas. Esto ilustra bien la necesidad de un análisis riguroso en el diseño de todos los sistemas seguros.

Los usos de los modelos de seguridad • Podría pensarse que el logro de la seguridad en los sistemas distribuidos sería un asunto sencillo que implica el control de acceso a los objetos de acuerdo con los derechos de acceso predefinidos y el uso de canales seguros de comunicación. Por desgracia, esto no es generalmente el caso. El uso de técnicas de seguridad como encriptación y control de acceso incurre en costos de tratamiento y gestión sustanciales. El modelo de seguridad descrito anteriormente, proporciona la base para el análisis y diseño de sistemas de seguridad en el que estos costes se reducen al mínimo, pero las amenazas a un sistema distribuido surgen en muchos puntos, y un cuidadoso análisis de las amenazas que podría surgir de todas las posibles fuentes en el entorno de red del sistema, se necesita medio físico y medio ambiente humano. Este análisis implica la construcción de una *modelo de amenazas* una lista de todas las formas de ataque a los que está expuesto el sistema y una evaluación de los riesgos y consecuencias de cada uno. La eficacia y el coste de las técnicas de seguridad necesarias se pueden equilibrarse con las amenazas.

2.5 Resumen

Como se ilustra en la Sección 2.2, los sistemas distribuidos son cada vez más complejo en términos de sus características físicas subyacentes; por ejemplo, en términos de la escala de los sistemas, el nivel de heterogeneidad inherente a dichos sistemas y las demandas reales para proporcionar soluciones de extremo a extremo en términos de propiedades tales como la seguridad. Esto lugares cada vez más importancia en ser capaz de entender y razonar acerca de los sistemas distribuidos en términos de modelos. En este capítulo se siguió el examen de los modelos físicos subyacentes con un examen en profundidad de los modelos de arquitectura y fundamentales que sostienen sistemas distribuidos.

En este capítulo se ha presentado una aproximación a la descripción de los sistemas distribuidos en términos de un modelo arquitectónico global que da sentido a este espacio de diseño de examinar las cuestiones esenciales de lo que se comunica y cómo se comunican estas entidades, complementado por la consideración de las funciones que cada elemento puede jugar junto con las estrategias de colocación apropiada dada la infraestructura física distribuida. El capítulo también introdujo el papel clave de los patrones arquitectónicos en que permite diseños más complejos para ser construidos a partir de los elementos básicos subyacentes, tales como el modelo cliente-servidor se indica arriba, y pone de relieve las principales estilos de soluciones de middleware de apoyo, incluyendo soluciones basadas en objetos distribuidos, componentes, servicios web y eventos distribuidos.

En cuanto a los modelos arquitectónicos, el enfoque cliente-servidor es prevalente - la web y otros servicios de Internet como FTP, noticias y enviarlo por correo, así como los servicios web y los DNS se basan en este modelo, como son la presentación y otros servicios locales. Los servicios como el DNS que tiene un gran número de usuarios y gestionar una gran cantidad de información se basan en múltiples servidores y partición de datos y el uso de replicación para mejorar la disponibilidad y tolerancia a fallos. El almacenamiento en caché por los clientes y servidores proxy se utiliza ampliamente para mejorar el desempeño de un servicio. Sin embargo, ahora hay una gran variedad de enfoques para el modelado de sistemas distribuidos incluyendo filosofías alternativas tales como peer-to-peer

la informática y soporte para más abstracciones orientados a problemas tales como objetos, componentes o servicios.

El modelo de arquitectura se complementa con modelos fundamentales, que ayudan en el razonamiento acerca de las propiedades del sistema distribuido en términos de, por ejemplo, el rendimiento, la fiabilidad y la seguridad. En particular, presentamos los modelos de interacción, el fracaso y la seguridad. Se identifican las características comunes de los componentes básicos de los cuales se construyen los sistemas distribuidos. El modelo de interacción se refiere a la realización de los procesos y los canales de comunicación y la ausencia de un reloj global. Se identifica un sistema síncrono como uno en el que los límites conocidos pueden ser colocados en el tiempo de ejecución del proceso, el tiempo de entrega de mensaje y deriva del reloj. Se identifica un sistema asíncrono como uno en el que no tiene límites pueden ser colocados en el tiempo de ejecución del proceso, el tiempo de entrega de mensaje y deriva del reloj - que es una descripción del comportamiento de la Internet.

El modelo de insuficiencia clasifica los fallos de los procesos y canales de comunicación básicas en un sistema distribuido. El enmascaramiento es una técnica por la cual un servicio más fiable se construye desde una menos fiable al enmascarar algunas de las fallas que exhibe. En particular, un servicio de comunicación fiable puede ser construido a partir de un canal de comunicación básico enmascarando sus fracasos. Por ejemplo, los fallos de omisión pueden enmascararse mediante la retransmisión de mensajes perdidos. La integridad es una propiedad de una comunicación fiable - que requiere que un mensaje recibido sea idéntico al que fue enviado y que ningún mensaje se enviará dos veces. La validez es otra propiedad - que requiere que cualquier mensaje puesto en el buffer de salida será entregado finalmente a la memoria intermedia de mensaje entrante.

El modelo de seguridad identifica las posibles amenazas a los procesos y los canales de comunicación en un sistema distribuido abierto. Algunas de estas amenazas están relacionadas con la integridad: los usuarios maliciosos pueden manipular los mensajes o reproducirlas. Otros amenazan su privacidad. Otro problema de seguridad es la autenticación del principal (usuario o servidor) en cuyo nombre se envió un mensaje. Los canales seguros utilizan técnicas criptográficas para asegurar la integridad y privacidad de mensajes y para autenticar pares de directores que se comunican.

CEREMONIAS

2.1 Proporcionar tres ejemplos específicos y contrastantes de los crecientes niveles de heterogeneidad con experiencia en sistemas distribuidos contemporáneos como se define en la Sección 2.2. *página 39*

2.2 ¿Qué problemas a prever en el acoplamiento directo entre entidades comunicantes que está implícita en los enfoques de la invocación remota? En consecuencia, ¿qué ventajas puede anticipar desde un nivel de desacoplamiento según lo ofrecido por el espacio y el tiempo de desacoplamiento? Nota: es posible que desee volver a examinar esta respuesta después de leer los capítulos 5 y 6. *página 43*

2.3 Describir e ilustrar la arquitectura cliente-servidor de uno o más importante de Internet aplicaciones (por ejemplo, la Web, correo electrónico o Netnews). *página 46*

2.4 Para las aplicaciones discutidas en el Ejercicio 2.1, lo que se emplean estrategias de entrada en la implementación de los servicios asociados? *página 48*

2.5 Un motor de búsqueda es un servidor web que responde a las peticiones de los clientes para buscar en su almacenado índices y (al mismo tiempo) se ejecuta varias tareas rastreador web para construir y actualizar los índices. ¿Cuáles son los requisitos para la sincronización entre estas actividades concurrentes?

página 46

2.6 Los equipos host utilizados en los sistemas peer-to-peer son a menudo simplemente ordenadores de escritorio en oficinas u hogares de los usuarios. ¿Cuáles son las implicaciones de esto para la disponibilidad y la seguridad de los datos de los objetos compartidos que tienen y en qué medida pueden las debilidades superarse mediante el uso de la replicación?

páginas 47, 48

2.7 Enumerar los tipos de recursos locales que son vulnerables a un ataque de un programa que no se confía que se descarga desde un sitio remoto y ejecutar en un equipo local. *página 50*

2.8 Dar ejemplos de aplicaciones en las que el uso del código móvil es beneficioso. *página 50*

2.9 Considere una hipotética empresa de alquiler de coches y esbozar una solución de tres niveles para la prestación de su servicio de alquiler de coches distribuido subyacente. Usar esto para ilustrar las ventajas e inconvenientes de una solución de tres niveles teniendo en cuenta cuestiones tales como el rendimiento, la escalabilidad, que trata de fracaso y también mantener el software en el tiempo. *página 52*

2.10 Dar un ejemplo concreto del dilema que ofrece de Saltzer argumento de extremo a extremo en el contexto de la prestación de apoyo de middleware para aplicaciones distribuidas (es posible que desee centrarse en un aspecto de proporcionar sistemas distribuidos fiables, por ejemplo relacionados con tolerancia a fallos o la seguridad).

página 60

2.11 Considere un servidor simple que lleva a cabo las solicitudes del cliente sin tener acceso a otros servidores. Explicar por qué por lo general no es posible establecer un límite en el tiempo que tarda un servidor, para responder a una petición de cliente. ¿Qué tendría que hacer para que el servidor capaz de ejecutar las solicitudes dentro de un tiempo limitado? ¿Es esta una opción práctica? *página 62*

2.12 Para cada uno de los factores que contribuyen al tiempo empleado para transmitir un mensaje entre dos procesos de más de un canal de comunicación, estado de lo que serían necesarias medidas para establecer un límite en su contribución al tiempo total. ¿Por qué estas medidas no previstas en los sistemas distribuidos de propósito general actuales? *página 63*

2.13 El servicio Network Time Protocol se puede utilizar para sincronizar los relojes de los equipos. Explicar por qué, incluso con este servicio, no se garantiza con destino está determinado por la diferencia entre dos relojes.

página 64

2.14 Considere dos servicios de comunicación para su uso en sistemas distribuidos asíncronos. En Un servicio, los mensajes se pueden perder, duplica o se retrasa y las sumas de comprobación sólo se aplican a las cabeceras. En el servicio B, los mensajes se pueden perder, retrasan o se entregan demasiado rápido para que el destinatario para manejarlos, pero aquellos que se entregan llegan con todos los componentes.

Describir las clases de fracaso exhibida por cada servicio. Clasificar sus fallos de acuerdo con sus efectos sobre las propiedades de validez e integridad. ¿Puede el servicio B puede describir como un servicio de comunicación fiable? *página 67, página 71*

2.15 Considere un par de procesos X e Y que utilizan el servicio de comunicación B a partir de Ejercicio 2.14 para comunicarse entre sí. Supongamos que X es un cliente y un servidor de Y y que una *invocación* consta de un mensaje de solicitud de X a Y, seguido por Y llevar a cabo la petición, seguido por un mensaje de respuesta de Y a X. Describe las clases de fracaso que pueden ser exhibidos por una invocación.

página 67

2.16 Supongamos que una lectura básica de disco a veces puede leer valores que son diferentes de las escrito. Estado del tipo de fallo exhibido por una lectura básica de disco. Sugiere cómo este fracaso puede enmascararse con el fin de producir una forma benigna diferente de fracaso. Ahora sugerir cómo enmascarar el fracaso benigna.

página 70

2.17 Definir la propiedad integridad de una comunicación fiable y enumerar todas las posibles amenazas a la integridad de los usuarios y de los componentes del sistema. ¿Qué medidas se pueden tomar para garantizar la integridad de la propiedad en la cara de cada una de estas fuentes de amenazas.

páginas 71, 74

2.18 Describir posibles ocurrencias de cada uno de los principales tipos de amenaza a la seguridad (amenazas a la los procesos, las amenazas a los canales de comunicación, denegación de servicio) que podrían ocurrir en Internet.

páginas 74, 75

Esta página se ha dejado intencionadamente en blanco