Computación Distribuida

Tema 1: Paradigmas de la computación distribuida

Paradigmas para aplicaciones distribuidas

- El concepto paradigma significa “un patrón, ejemplo o modelo”. En el estudio de cualquier materia que tenga una complejidad elevada resulta útil identificar patrones o modelos básicos y clasificar los elementos de esa materia de acuerdo a esos modelos. Aquí vamos a presentar una clasificación de los distintos paradigmas para la computación distribuida.

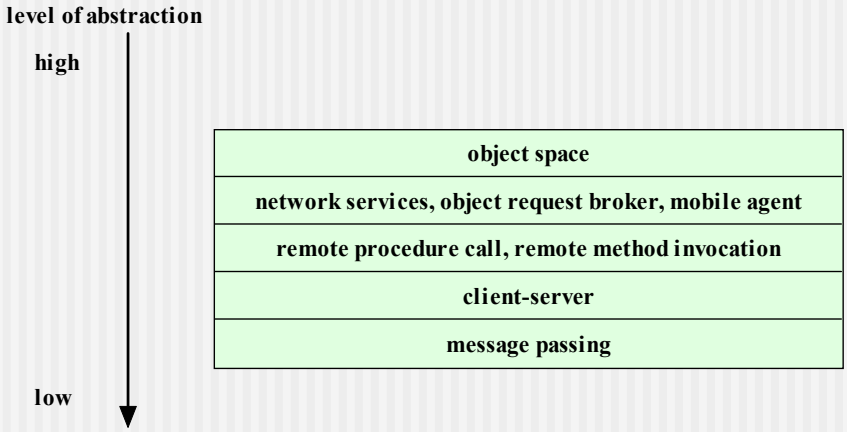
Paradigmas para aplicaciones distribuidas

- Las características que distinguen a una aplicación distribuida de una aplicación convencional que se ejecuta en una máquina son las siguientes:
  - ***Comunicación entre procesos*: Una aplicación distribuida necesita la participación de dos o más entidades independientes (procesos). Los procesos deben de tener la capacidad de intercambiar datos entre ellos.**
  - ***Sincronización de eventos*: En una aplicación distribuida, el envío y recepción de datos entre los distintos participantes debe estar sincronizado.**

Abstracciones

- Abstracción es uno de los conceptos más fundamentales de las ciencias de la computación y tiene que ver con el proceso de *esconder los detalles.*
- **En ingeniería del software, la abstracción se consigue mediante herramientas que permitan construir software al desarrollador sin necesidad de que éste conozca todos los entresijos de la complejidad del sistema.**

Paradigmas en computación distribuida

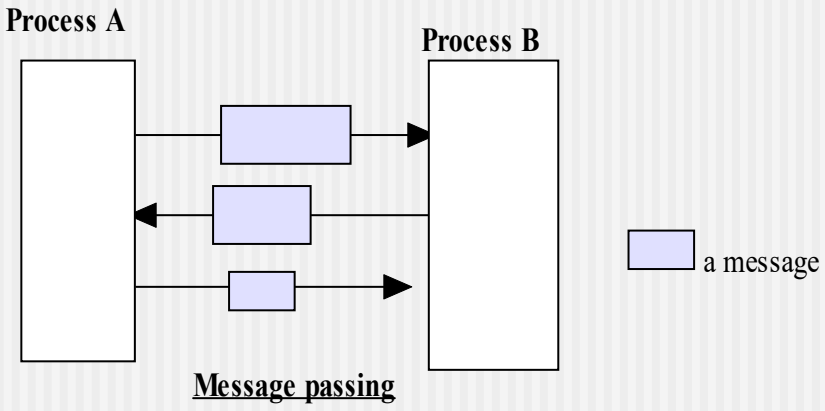


El paradigma del envío de mensajes

El envío de mensajes es el paradigma más importante para la construcción de aplicaciones distribuidas.

- **Un proceso envía un mensaje que representa una petición.**
- **El mensaje es entregado al receptor que procesa la petición y envía un mensaje de respuesta.**
- **Como consecuencia, la respuesta puede disparar nuevas peticiones que pueden llevar a nuevas respuestas, …**

El paradigma del envío de mensajes - 2



El paradigma del envío de mensajes - 3

- **Las operaciones básicas necesarias para dar soporte al paradigma del envío de mensajes son *envía* y *recibe*.**
- **Para una comunicación orientada a conexión también son necesarias las operaciones *conecta* y *desconecta*.**

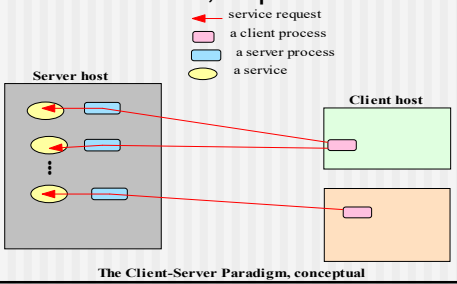
El paradigma del envío de mensajes - 4

- **Con este modelo de abstracción, los procesos realizan operaciones de entrada/salida entre ellos de forma similar a como se realizaría con un archivo. Las operaciones de entrada/salida encapsulan los detalles de la comunicación de red a nivel del sistema operativo.**
- **El API de programación de sockets está basado en este paradigma**

El paradigma Cliente-Servidor

Quizás sea el paradigma mejor conocido para el desarrollo de aplicaciones distribuidas. Este paradigma asigna roles asimétricos a los dos procesos que colaboran.

Un proceso, el servidor, juega el papel de proveedor del servicio que espera de forma pasiva a la llegada de peticiones. El otro proceso, el cliente, envía peticiones específicas al servidor, esperando su respuesta.



The Client-Server Paradigm, conceptual

El paradigma Cliente-Servidor - 2

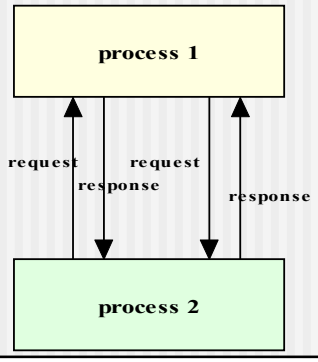
- El modelo cliente-servidor proporciona una abstracción eficiente para la provisión de servicios de red.
- Las operaciones requeridas son, en el caso del servidor, aquellas que permitan esperar y aceptar peticiones y, en el caso del cliente, aquellas que emitan peticiones y acepten respuestas.
- Al asignar roles asimétricos, la sincronización de eventos se simplifica: el proceso servidor espera peticiones y el cliente espera respuestas.
- Muchos servicios de Internet son aplicaciones cliente-servidor. Estos servicios son a menudo conocidos por el protocolo implementado por la aplicación. Ejemplos de servicios de Internet conocidos son HTTP, FTP, DNS, finger, gopher, etc.

La arquitectura de igual a igual (peer to peer)

- La arquitectura peer-to-peer es una arquitectura donde se intercambian de forma directa recursos y servicios entre ordenadores.
- Entre estos servicios y recursos compartidos se incluye el intercambio de información, ciclos de procesamiento, almacenamiento, etc.
- En este tipo de arquitectura, los ordenadores que tradicionalmente se utilizaban como clientes se comunican de forma directa entre ellos y pueden actuar al mismo tiempo como clientes y servidores en función de cuál es el rol más eficiente para la red.

El paradigma de computación distribuida peer-to-peer

En el paradigma peer-to-peer, los procesos participantes juegan roles iguales, con capacidades y responsabilidades equivalentes (de ahí el término “peer” que significa igual). Cada participante puede enviar una petición a otro participante y recibir una respuesta.



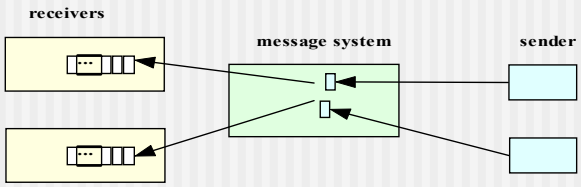
La computación distribuida peer-to-peer

El paradigma cliente-servidor es ideal para un servicio de red centralizado mientras que el modelo peer-to-peer es más apropiado para aplicaciones menos centralizadas tales como la mensajería instantánea, la transferencia de ficheros de igual a igual, la videoconferencia y el trabajo colaborativo. Es posible que una aplicación use ambos modelos de computación.

Un ejemplo bien conocido de computación peer-to-peer es la transferencia de archivos ofrecida por *Napster.com* o empresas similares. Esta aplicación hace uso de un servidor que hace la función de directorio y de la computación peer-to-peer.

El paradigma del sistema de mensajes

- El Sistema de Mensajes o Message-Oriented Middleware (MOM) es una sofistificación del paradigma de paso de mensajes.
- En este paradigma, un servidor (el sistema de mensajes) actúa como intermediario entre dos procesos independientes.
- El sistema de mensajes actúa como un conmutador de mensajes a través del cual los procesos intercambian mensajes de forma asíncrona de una manera desacoplada.
- Un emisor deposita un mensaje en el sistema de mensajes, que a su vez deposita en una cola de mensajes asociada a cada receptor.



Dos subtipos de modelos de sistemas de mensajes

El modelo Punto-a-Punto

- En este modelo, el sistema de mensajes reenvía el mensaje del emisor a la cola de mensajes del receptor. Frente al modelo de paso de mensajes, el middleware proporciona un almacén de mensajes que permite al emisor y receptor estar desacoplados. A través del middleware el emisor deposita el mensaje en la cola de mensajes del proceso receptor. El proceso receptor extrae los mensajes de su cola de mensajes y los gestiona de forma apropiada.
- Este modelo proporciona un mayor nivel de abstracción para operaciones asíncronas que el proporcionado por el modelo de paso de mensajes. Para conseguir el mismo resultado, en el paradigma de paso de mensajes el programador tendría que hacer uso de threads o procesos hijo.

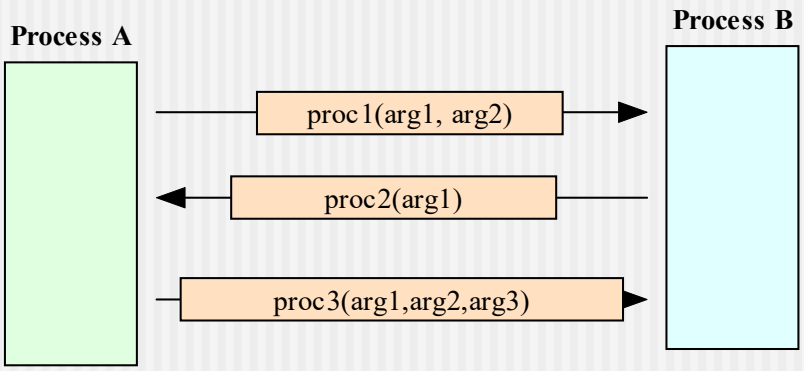
El modelo Publica/Suscribe

- **En este modelo, cada mensaje tiene asociado un evento específico. Las aplicaciones interesadas en la ocurrencia de un evento pueden suscribirse a mensajes para dicho evento. Cuando ocurre el evento esperado, el proceso publica un mensaje anunciando dicho evento. El middleware distribuye el mensaje a todos los suscriptores.**
- **Este modelo proporciona una abstracción muy potente para comunicaciones multicast o en grupo. La operación *publish* permite a un proceso enviar un mensaje a un grupo de procesos y la operación *subscribe* permite a un proceso recibir esos mensajes.**

Remote Procedure Call

- Conforme las aplicaciones aumentan de complejidad, es deseable tener paradigmas que permitan programar software distribuido de una forma similar a las aplicaciones que se ejecutan en un único procesador.
- El modelo Remote Procedure Call (RPC) proporciona dicha abstracción. Usando este modelo, la comunicación se establece mediante llamadas a procedimientos o funciones que son familiares a los programadores.

Remote Procedure Call - 2



Remote Procedure Call - 3

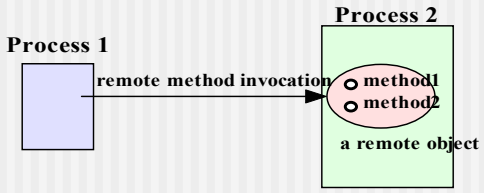
- RPC permite a los programadores construir aplicaciones de red usando construcciones similares a las utilizadas en la invocación local de métodos, proporcionando una abstracción de la comunicación entre procesos y su sincronización.
- Desde su introducción a comienzos de los años 80, el modelo RPC ha sido muy utilizado para construir aplicaciones de red.
- Hay fundamentalmente dos APIs para su uso en RPC.
  - The *Open Network Computing Remote Procedure Call*, ha evolucionado del API RPC de Sun Microsystems.
  - The *Open Group Distributed Computing Environment* (DCE) RPC.
- Ambas APIs proporcionan una herramienta, *rpcgen*, para transformar invocaciones remotas a llamadas a procedimientos locales pertenecientes al denominado stub.

El paradigma de Objetos Distribuidos

- La idea de aplicar una orientación a objetos a las aplicaciones distribuidas es una extensión natural del desarrollo de software orientado a objetos.
- Las aplicaciones acceden a objetos distribuidos a través de una red.
- Los objetos proporcionan métodos a través de los cuales una aplicación obtiene acceso a servicios.
- El paradigma de objetos distribuidos incluye:
  - Remote method invocation (RMI)
  - Network services
  - Object request broker
  - Object spaces

Remote Method Invocation (RMI)

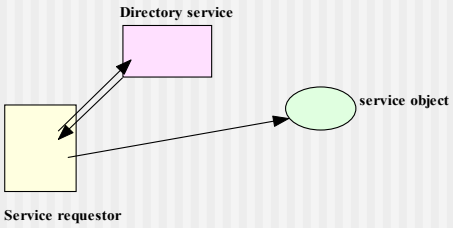
- Remote method invocation es el equivalente orientado a objetos de la invocación remota de métodos.
- En este modelo, un proceso invoca los métodos de un objeto que puede residir en otra máquina.
- Al igual que con RPC, se pueden pasar argumentos en la invocación.



The Remote Method Call Paradigm

El paradigma de los Servicios de Red

- En este paradigma, los proveedores de servicios se registran en servidores de directorio en una red. Un proceso que desea un servicio en particular contacta con el servidor de directorio en tiempo real y, si el servicio está disponible, obtendrá una referencia a dicho servicio. Usando dicha referencia el proceso interactúa con el servicio.
- Este paradigma es esencialmente una extensión del paradigma de la invocación remota de métodos. La diferencia es que los objetos que proporcionan el servicio están registrados en un servicio global de directorio, permitiendo que sean localizados y accedidos por otros procesos en una red.
- La tecnología Jini es un ejemplo de este paradigma.



El paradigma del Object Request Broker

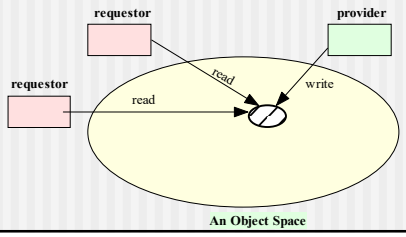
- En este paradigma una aplicación envía una petición a un *object request broker* (ORB), que redirige dicha petición al objeto apropiado que proporciona el servicio deseado.
- Este paradigma se parece al modelo de invocación remota de métodos en su soporte al acceso de objetos remotos. La diferencia es que el ORB en este paradigma funciona como un middleware que permite a una aplicación acceder a múltiples objetos remotos (o locales).
- El ORB puede funcionar como mediador entre objetos heterogéneos, permitiendo la interacción entre objetos implementados utilizando diferentes APIs y/o ejecutándose en diferentes plataformas.

El paradigma del Object Request Broker - 2

- Este paradigma es la base de la arquitectura CORBA del Object Management Group (OMG).
  http://www.corba.org/
- Ejemplos de herramientas que utilizan esta arquitectura son:
  - MicroFocus Visibroker
    https://www.microfocus.com/es-es/products/corba/visibroker/#
  - Java's Interface Development Language (Java IDL)
    http://docs.oracle.com/javase/7/docs/technotes/guides/idl/
  - MicroFocus Orbix
    https://www.microfocus.com/es-es/products/corba/orbix/

El paradigma del Espacio de Objetos

- **Se trata posiblemente del paradigma orientado a objetos más abstracto. Este paradigma asume la existencia de entidades lógicas denominadas *espacios de objetos*.**
- **El participante de una aplicación converge en un espacio de objetos común.**
- **Un proveedor de servicios coloca objetos a modo de entradas en un espacio de objetos y el proceso que usa dichos servicios se suscribe a dicho espacio y accede a las entradas.**



El paradigma del Espacio de Objetos - 2

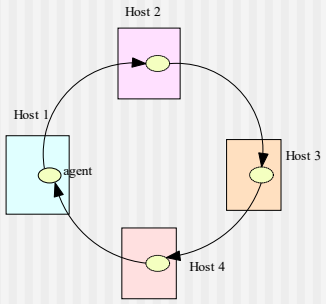
- En adición a las abstracciones proporcionadas por otros paradigmas, el paradigma del espacio de objetos proporciona un espacio virtual entre proveedores y clientes de recursos de red. Esta abstracción esconde los detalles de localización de objetos necesarios en paradigmas tales como la invocación remota de métodos, el ORB o los servicios de red.
- **Un ejemplo de este paradigma es JavaSpaces**

Tecnologías basadas en Componentes

- Las tecnologías basadas en componentes, tales como COM y DCOM de Microsoft, Java Bean y Enterprise Java Bean, también están basadas en paradigmas de computación distribuida pues, los componentes son esencialmente paquetes de objetos especializados y diseñados para interactuar unos con otros a través de interfaces estandarizados.
- Adicionalmente, los *servidores de aplicaciones,* muy populares en aplicaciones empresariales, no son otra cosa que middleware para proporcionar acceso a objetos y componentes.

El paradigma de Agentes Móviles

- Un agente móvil es un objeto o programa transportable.
- En este modelo se lanza un agente desde un ordenador en particular.
- El agente viaja de máquina en máquina de acuerdo a un itinerario.
- En cada parada el agente accede a los recursos y servicios necesarios y realiza las tareas necesarias para cumplir su misión.

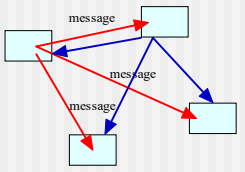


El paradigma de Agentes Móviles - 2

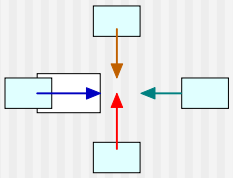
- Este paradigma ofrece la abstracción para poder tener objetos o programas transportables.
- En lugar de intercambiar mensajes, los datos son transportados por el objeto/programa a medida que viaja entre máquinas.
- Ejemplos de paquetes que dan soporte a este paradigma los tenemos en:
  - Mitsubishi Electric ITA's Concordia system
    https://www.cis.upenn.edu/~bcpierce/courses/629/papers/Concordia-MobileAgentConf.html
  - IBM's Aglet system.
    http://web.media.mit.edu/~stefanm/ibm/AgletsHomePage/index_new4.html

El paradigma de las aplicaciones colaborativas (Groupware)

- En este modelo, los procesos participan en una sesión colaborativa como grupo. Cada proceso participante puede contribuir a la entrada de parte o la totalidad del grupo.
- Los procesos pueden hacer esto usando:
  - multicasting para enviar datos a todos o parte del grupo
  - *Pizarras* virtuales que permiten a cada participante leer y escribir datos en una pantalla compartida.



Message-based groupware paradigm



Whiteboard-based groupware paradigm

Resumen - 1

- Hemos visto un amplio espectro de paradigmas de computación distribuida.
- Los paradigmas vistos son:
  - Paso de mensajes
  - Cliente-servidor
  - Sistemas de mensajes: Punto-a-punto; Publicación/Suscripción
  - Objetos distribuidos:
    - Remote method invocation
    - Object request broker
    - Espacio de objetos
  - Agentes móviles
  - Servicios de red
  - Aplicaciones colaborativas

Resumen - 2

- Estos paradigmas proporcionan la abstracción necesaria para aislar a los desarrolladores de los detalles de la comunicación entre procesos y la sincronización de eventos, permitiendo al programador el concentrarse en la propia aplicación.
- La elección de un paradigma tiene pros y contras entre los que se encuentran el coste computacional, la escalabilidad, la portabilidad, etc.