Sistemas Distribuidos

Tema 1: INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

E. U. Informática en Segovia
Departamento de Informática
Universidad de Valladolid

Sistemas Distribuidos

Introducción a la Computación Distribuida

- Sistema distribuido: conjunto de computadores independientes, interconectados a través de una red y con capacidad de colaborar con el fin de realizar una tarea
- Computación distribuida: computación que se lleva a cabo en un sistema distribuido
  - Servicio de red. Servicio proporcionado por un tipo de programa especial denominado servidor en una red. Ej: servidor web, servidor correo electrónico, servidor ftp
  - Aplicación de red. Aplicación para usuarios finales, que se ejecuta en computadores conectados a través de una red. Ej: tiendas virtuales, subastas electrónicas, juegos en red

  La diferencia entre servicios y aplicaciones de red no es siempre nítida

Ventajas de la Computación Distribuida

- Los computadores y el acceso a la red son económicos. Los computadores personales actuales tienen una potencia superior a los primeros *mainframes*, además de tener mucho menos tamaño y precio
- Compartición de recursos. La arquitectura de la computación distribuida refleja la arquitectura de computación de las organizaciones modernas. Cada organización mantiene de forma independiente los computadores y recursos locales, mientras permite compartir recursos a través de la red
- Escalabilidad. En la computación monolítica, los recursos disponibles están limitados por la capacidad de un computador. Por el contrario, la computación distribuida proporciona escalabilidad, debido a que permite incrementar el número de recursos compartidos según la demanda
- Tolerancia a fallos. Al contrario que la computación monolítica, la computación distribuida permite que un recurso pueda ser replicado con el fin de dotar al sistema de tolerancia a fallos, de tal forma que proporcione disponibilidad de dicho recurso en presencia de fallos

Desventajas de la Computación Distribuida

- Múltiples puntos de fallo. Hay más puntos de fallo en la computación distribuida. Debido a que múltiples computadores están implicados en la computación distribuida, y todos son dependientes de la red para su comunicación, el fallo de uno o más computadores, o uno o más enlaces de red, puede suponer problemas para un sistema de computación distribuida
- Aspectos de seguridad. En un sistema distribuido hay más posibilidad de ocurrencia de ataques. Mientras que en un sistema centralizado los recursos están bajo el control de una administración única, en un sistema distribuido la gestión es descentralizada y frecuentemente implica a un gran número de organizaciones independientes

Paradigmas de Computación Distribuida: cliente-servidor (1)

- Los orígenes del modelo cliente-servidor se basan en los sistemas de **paso de mensajes**
  - Los datos, representados en forma de mensajes, se intercambian entre dos procesos, un emisor y un receptor
  - Un proceso envía un mensaje que representa una petición. El mensaje se entrega a un receptor, que procesa la petición y envía un mensaje como respuesta. En secuencia, la réplica puede disparar posteriores peticiones, que llevan a sucesivas respuestas, y así, sucesivamente
  - Las operaciones básicas necesarias para dar soporte al paradigma de paso de mensajes son *enviar* y *recibir*. Para comunicaciones orientadas a conexión, también se necesitan las operaciones *conectar* y *desconectar*

Paradigmas de Computación Distribuida: cliente-servidor (2)

- El modelo cliente-servidor (c-s) asigna roles diferentes a los dos procesos que colaboran.
  - El servidor interpreta el papel de proveedor de servicio, esperando de forma pasiva la llegada de peticiones
  - El cliente invoca determinadas peticiones al servidor y aguarda sus respuestas
- De una concepción simple, el modelo cliente-servidor proporciona una abstracción eficiente para facilitar servicios de red. Muchos servicios de Internet dan soporte a aplicaciones cliente-servidor, por ejemplo: HTTP, DNS, FTP, etc.
- Ejemplo: Sistemas de subastas on-line
  - Intencionadamente se ignoran detalles relacionadas con la interfaz de usuario y almacenamiento de datos
  - El sistema se simplifica de modo que:
    - En cada sesión de subasta, sólo se puja por un objeto
    - Durante la sesión, el objeto está abierto a pujas emitidas por los participantes en la subasta
    - Al finalizar la sesión, el subastador anuncia el resultado

Paradigmas de Computación Distribuida: cliente-servidor (3)

- Ejemplo: Sistemas de subastas on-line
  - Se diferencian dos tipos de entidades: el subastador y los participantes
  - Cada entidad asume a la vez el papel de cliente y servidor de pendiendo de los diferentes escenarios:
  - En relación con el control de la sesión:
    - El participante actúa como servidor, en el sentido de esperar escuchar el anuncio por parte del subastador de:
      (1) cuándo comienza la sesión
      (2) cuándo se produce un cambio en la puja máxima
      (3) cuándo termina la sesión
    - El subastador actúa como cliente, enviando una petición que anuncia los tres tipos de eventos antes comentados
  - En relación con la aceptación de las pujas:
    - El subastador actúa como servidor, en el sencido de aceptar nuevas pujas y actualizar la puja máxima
    - Cada participante actúa como un cliente, enviando un a nueva puja al subastador

Paradigmas de Computación Distribuida: peer-to-peer

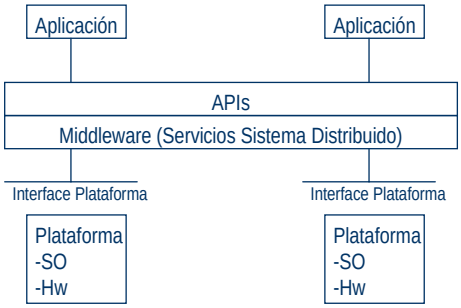
- En el paradigma *peer-to-peer* (p2p) los procesos participantes interpretan los mismos papeles, con idénticas capacidades y responsabilidades (lo que sugiere interacciones directas entre las partes). Ej: Napster.com
  - Cada participante puede solicitar una petición a cualquier otro participante y recibir una respuesta
- Mientras que el paradigma cliente-servidor es un modelo ideal para servicios centralizados de red, el paradigma p2p resulta más apropiado para aplicaciones como mensajería instantánea, transferencia de ficheros, video-conferencia y trabajo colaborativo
- También es posible que un sistema se base en ambos modelos: cliente-servidor y p2p
- El paradigma p2p se puede implementar por medio de bibliotecas de paso de mensajes en aplicaciones sencillas o utilizando tecnologías específicas en el caso del desarrollo de aplicaciones más complejas:
  - JXTA [www.jxta.org]
    Conjunto de protocolos abiertos para permitir a cualquier dispositivo conectado a la red comunicarse y colaborar con otros dispositivos siguiendo el modelo p2p
  - Jabber [www.jabber.org]
    Conjunto de tecnologías y protocolos de *streaming* basados en XML que permiten a dos entidades conectadas a Internet intercambiar mensajes, presencia u otra información estructurada, prácticamente, en tiempo real

Middleware (1)

- El middleware (Mw) es el software de conectividad que está compuesto por un conjunto de servicios que permiten a varios procesos (que se ejecutan en una o varias máquinas) interactuar a través de la red
- El Mw es fundamental para:
  - migrar las aplicaciones monolíticas basadas en *mainframes* a aplicaciones cliente-servidor
  - soportar la comunicación entre procesos a través de plataformas heterogéneas
- Esta tecnología ha evolucionado durante la década de los 90 y los entornos Mw más conocidas son:
  - DCE (*Distributed Computing Environment*) desarrollado por *Open Software Foundation* [http://www.opengroup.org/dce/]
  - CORBA (*Common Object Request Broker Architecture*) desarrollado por *Object Management Group* [http://www.corba.org/]
  - COM/DCOM (*Component Object Model/Distributed COM*) desarrollado por Microsoft [www.microsoft.com/com/]

Middleware (2)

- Como muestra la figura los servicios del Mw son una capa de software distribuida, que se localiza entre la aplicación y la plataforma concreta sobre la que se implementa la aplicación (SO + Red)



- Los servicios del Mw proporcionan un conjunto de APIs (*Application Programming Interfaces*) más funcional que el sistema operativo y los servicios de red para permitir a una aplicación:
  - Localización transparente a través de la red, proporcionando interacción con otra aplicación o servicio
  - Ser independiente de los servicios de red
  - Ser fiable y disponible
  - Ser escalable, en el sentido de poder aumentar su capacidad sin pérdida de funcionalidad

Middleware (3)

- El Middleware puede tomar una de las siguientes formas:
  - Monitores de procesamiento de transacciones o teleproceso (TP, *Transaction Processing monitors*), que proporcionan herramientas y un entorno para el desarrollo y explotación de aplicaciones distribuidas
  - Llamadas a procedimientos Remotos (RPC, *Remote Procedure Call*), que permiten que la lógica de una aplicación esté distribuida a través de una red. La lógica del programa en el sistema remoto puede ejecutarse tan simplemente como se realiza una invocación a una rutina local
  - Sistemas de Mensajes (MOM, *Message-Oriented Middleware*), que proporciona intercambio de datos aplicación a aplicación, posibilitando la creación de aplicaciones distribuidas. Los sistemas de colas son análogos a los sistemas de correo electrónico en el sentido de ser asíncronos y requerir que los receptores de los mensajes interpreten su significado y tomen las acciones apropiadas
  - Agentes de solicitud de objetos (ORBs, *Object Request Brokers*), que permite que los objetos que componen una aplicación sean distribuidos y compartidos a través de redes heterogéneas

Middleware (4)

- El propósito principal de los servicios del Mw es ayudar a resolver muchos de los problemas de conectividad e interoperabilidad. Sin embargo, estos servicios no son la panacea:
  - Existe un gran salto conceptual entre los principios y la práctica. Muchos servicios Mw utilizan implementaciones propietarias (haciendo a las aplicaciones dependientes de un único producto de un vendedor)
  - El propio número de servicios del Mw es una barrera para utilizarlos. Para mantener un entorno de computación manejable y sencillo, los desarrolladores deben seleccionar un pequeño número de servicios que satisfagan sus necesidades en cuanto funcionalidad y cobertura de plataformas
  - Mientras los servicios del Mw incrementan el nivel de abstracción de las técnicas de programación de aplicaciones distribuidas, éstas dejan aún al desarrollador la responsabilidad de tomar decisiones de diseño complicadas. Por ejemplo, el desarrollador debe decidir qué funcionalidad poner en el lado del cliente y del servidor de la aplicación distribuida

Middleware (5)

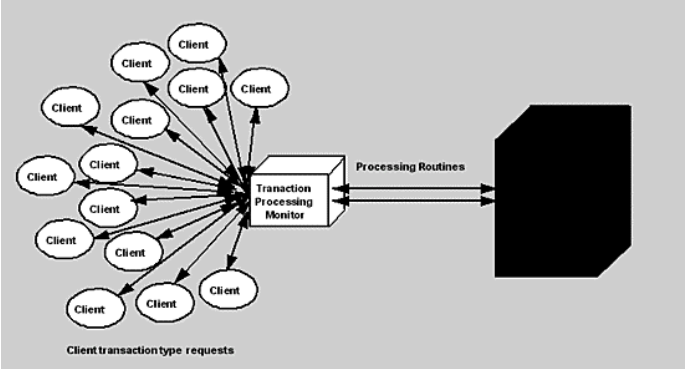
- La clave para superar las deficiencias anteriores se basa en entender completamente tanto el dominio de la aplicación como el valor de los servicios del Mw que puede permitir la aplicación distribuida. Para determinar el tipo de Mw necesario, el desarrollador debe identificar las funciones requeridas, que caen en una de las tres clases:
  - Servicios propios de los sistemas distribuidos, entre otros, comunicaciones componente a componente y servicios de manejo de datos. Este tipo de servicio incluye RPC, MOMs y ORBs
  - Mw como software que habilita servicios, los cuales dan a las aplicaciones, acceso a los servicios distribuidos y la red subyacente. Este tipo de servicios incluyen a monitores de teleproceso y servicios de bases de datos (por ejemplo, el lenguaje SQL)
  - Servicios de gestión del Mw, que posibilitan monitorizar continuamente a aplicaciones y funciones del sistema para garantizar un rendimiento óptimo del entorno distribuido

Monitores de Procesamiento de Transacciones (1)

- La tecnología de Monitores de Procesamiento de Transacciones (TP, *Transaction Processing*) proporciona al entorno cliente-servidor distribuido, la capacidad de desarrollar, ejecutar y gestionar aplicaciones de transacciones de forma eficiente y fiable
- Un monitor TP controla las aplicaciones basadas en transacciones, soportando la lógica de negocio y las actualizaciones de la base de datos
- Esta tecnología se ha venido utilizando desde hace 25 años en aplicaciones de gestión que requieren un soporte *online* para compartir servicios de aplicación y recursos de información con un entorno de *mainframes* (basado en sistemas operativos de tiempo compartido y de procesamiento por lotes)
- Tradicionalmente, los monitores de teleproceso (CICS, IMS/DC, IDMS/DC) son subsistemas que agrupan sentencias de actualización de bases de datos relacionadas y que las presentan todas juntas (como un lote) al gestor de base de datos. Gracias a ello, el gestor de base de datos no necesita preocuparse de gestionar la consistencia y corrección de la base de datos. El monitor de teleproceso se asegura de que los grupos de actualizaciones se hacen en su totalidad o no se hace ninguna. Esta característica redunda en la robustez y productividad del sistema

Monitores de Procesamiento de Transacciones (2)

- La tecnología de los monitores de teleproceso se basa en multiplexar las peticiones de transacción de los clientes (de acuerdo a su tipo) sobre un conjunto reducido y controlado de rutinas de procesamiento que soportan los servicios específicos



- Los clientes son ligados, servidos y liberados utilizando servidores sin estado, con el fin de minimizar la sobrecarga. El gestor de la base de datos sólo como clientes al conjunto reducido de rutinas

Monitores de Procesamiento de Transacciones (3)

- Por lo tanto, la tecnología de los monitores de teleproceso:
  - "Mapea" las peticiones de los clientes con las rutinas de servicio para mejorar el rendimiento del sistema
  - Permite controlar parte de la lógica de la aplicación en la parte cliente, con lo que se reduce las actualizaciones requeridas en las plataformas de los clientes
  - Incluyen numerosas características de gestión, como el reinicio de procesos fallidos, el balance dinámico de la carga, y hace respetar la consistencia de la base de datos
  - Además es fácilmente escalable mediante la incorporación de nuevos servidores de teleproceso para adecuarse al número de clientes
  - Es independiente de la arquitectura de la base de datos
  - Da soporte para un modelado de la aplicación flexible y robusto, y fuerza a la definición de procedimientos modulares y reutilizables
  - Los monitores de teleproceso permiten incorporar APIs específicas para soportar diferentes bibliotecas de clientes y gestores de recursos y bases de datos
  - La tecnología de los monitores de teleproceso se ha venido utilizando ampliamente en los últimos 25 años por parte de las empresas, para el desarrollo de aplicaciones de gestión

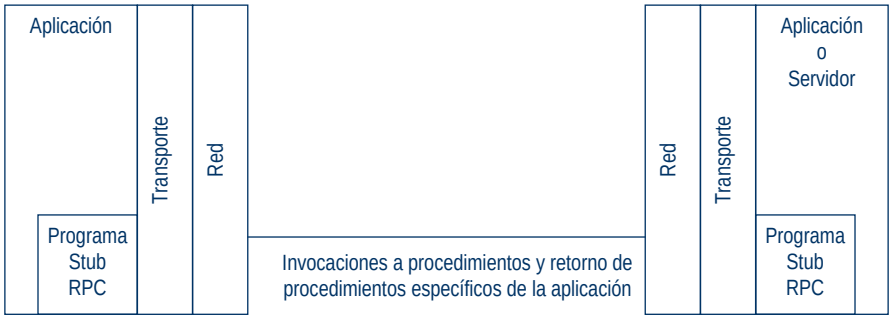
Llamada a Procedimientos Remotos (1)

- Las Llamadas a Procedimientos Remotos (RPC, *Remote Procedure Call*) constituyen una infraestructura del modelo cliente-servidor que incrementa la:
  - Interoperabilidad (capacidad de dos o más sistemas o componentes Sw para intercambiar información y utilizar la información que has sido intercambiada)
  - Portabilidad (facilidad con la que un sistema o componente Sw puede transferirse de un entorno Hw o Sw a otro)
  - Flexibilidad (facilidad con la que un sistema o componente Sw puede modificarse de modo que pueda utilizarse en otras aplicaciones o entornos para los que fue expresamente diseñado(

  de una aplicación, permitiendo que ésta se distribuya sobre múltiples plataformas heterogéneas
- Reduce la complejidad del desarrollo de aplicaciones que comprenden múltiples sistemas operativos y protocolos de red mediante el aislamiento del desarrollador de los detalles de la plataforma subyacente (cuando se utiliza RPC, las invocaciones a funciones son las interfaces del programador)

Llamada a Procedimientos Remotos (2)

- Para acceder a la porción de código del servidor remoto de una aplicación, las RPC o invocaciones especiales a funciones, son insertadas dentro de la porción de código del cliente
- Cuando se compila la aplicación, el compilador genera un programa "*stub*" local para el lado cliente y un "*stub*" remoto para la parte servidora de la aplicación.
- Son a estos *stubs* a los que se les invoca cuando la aplicación requiere una función remota y, habitualmente, soportan llamadas síncronas entre el cliente y el servidor (análogas a una invocación en local)



Llamada a Procedimientos Remotos (3)

- Mediante el uso de RPC, se reduce la complejidad asociada al desarrollo de un procesamiento distribuido, gracias a dotar a una invocación remota de la misma semántica que una invocación en local
  - Sin embargo, RPC incrementa el grado de participación del desarrollador de la aplicación, como consecuencia de la naturaleza maestro-esclavo del mecanismo
- RPC incrementa la flexibilidad de una arquitectura, permitiendo a un cliente emplear una invocación a una función para acceder a un servidor en un sistema remoto
- RPC permite que el componente remoto sea accesible sin necesidad de conocer su dirección de red o cualquier otra información de bajo nivel
- La mayoría de las implementaciones de RPC utilizan un protocolo solicitud-respuesta síncrono (esquema call/wait) que implica el bloqueo del cliente hasta que el servidor satisface su solicitud
- Las implementaciones de RPC están disponibles mediante herramientas propias de los sistemas operativos más habituales (Windows, Linux/Unix, NetWare) y en los entornos Mw: DCE (*Distributed Computing Environment*) y ONC (*Open Network Computing*)

Paradigmas de Computación Distribuida: Sistema de Mensajes (1)

- Los Sistemas de Mensajes o Middleware Orientado a Mensajes (MOM, *Message-Oriented Middleware*) proporcionan un método de comunicación entre aplicaciones o componentes software
  - Es una facilidad para el desarrollo de aplicaciones p2p: un cliente del sistema puede enviar y recibir mensajes de cualquier otro cliente.
    - Cada cliente se conecta a un agente del sistema de mensajes que proporciona facilidades para crear, enviar, recibir y leer mensajes
  - Los sistemas de mensajes habilitan comunicaciones entre procesos distribuidos con un bajo acoplamiento
    - Un componente envía un mensaje a un destino y el receptor puede recuperar un mensaje de un destino
      - Sin embargo, el emisor y el receptor no tienen porqué estar disponibles al mismo tiempo para poder comunicarse
      - De hecho, el emisor no necesita conocer nada sobre el receptor, ni el receptor conocer nada sobre el emisor
      - El emisor y receptor sólo necesitan conocer que formato de mensaje y qué destino utilizar
      - Esta es una diferencia frente a modelos de comunicación distribuidos fuertemente acoplados, como por ejemplo la Invocación Remota de Métodos, que requieren que la aplicación conozca los métodos remotos

Paradigmas de Computación Distribuida: Sistema de Mensajes (2)

- Otras propiedades deseables de un sistema de mensajes son:
  - Soportar el *reparto asíncrono* de mensajes, es decir, repartir los mensajes cuando lleguen los clientes, sin necesidad de que tengan que solicitar los mensajes para recibirlos
  - Ser un *sistema fiable*, es decir, tener la posibilidad de garantizar que un mensaje sólo se reparte una, y sólo una, vez
- Aunque parecido en la filosofía al sistema de correo electrónico, su uso está orientado a comunicar aplicaciones o componentes software en lugar de interconectar personas o aplicaciones con personas
- Aplicación de los Sistemas de Mensajes. Este paradigma es adecuado cuando se dan las siguientes circunstancias
  - Se requiere que los componentes no dependan de la información acerca de otras interfaces de componentes, de forma que los componentes puedan reemplazarse fácilmente
  - Se requiere que la aplicación se ejecute independientemente de si todos los componentes están activos y ejecutándose simultáneamente
  - El modelo de negocio de la aplicación permite que un componente envíe información a otro y pueda seguir operando sin recibir una respuesta inmediata

Paradigmas de Computación Distribuida: Sistema de Mensajes (3)

Por ejemplo, en el mundo del automóvil …

- Un fabricante de automóviles utiliza un componente software para gestionar su stock de automóviles fabricados. Cuando el número de unidades de un determinado modelo baja de una determinada cantidad, el componente stock puede enviar un mensaje al componente fábrica, solicitando la fabricación de más coches
- El componente fábrica puede enviar mensajes a los componentes suministradores de piezas para obtener las piezas necesarias en la fabricación de nuevos coches
- Los componentes asociados a los suministradores de piezas pueden enviar mensajes a sus propios stocks y departamentos de pedidos para actualizar su stock, y si llega el caso, solicitar pedidos a sus proveedores
- Tanto la fábrica como los suministradores de piezas pueden enviar mensajes al componente de contabilidad para actualizar sus cuentas
- La empresa puede comunicar su catálogo de productos actualizados al departamento de ventas

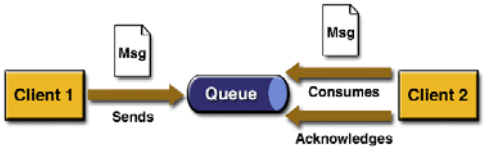
Paradigmas de Computación Distribuida: Sistema de Mensajes (4)

Existen dos modalidades de sistemas de mensajes:

Modelo de Mensajes punto a punto

Modelo de Mensajes publicación/suscripción

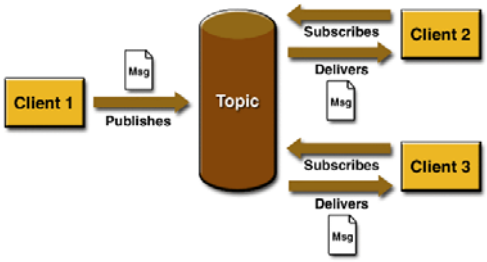
- **Modelo de Mensajes punto a punto**



  - Se basa en el concepto de colas de mensajes, emisores y receptores. Cada mensaje es dirigido a una cola específica y los receptores extraen mensajes de la(s) cola(s) que tienen establecidas para mantener sus mensajes
    - Las colas mantienen todos los mensajes enviados hasta que se consumen o caducan (si están definidos plazos de caducidad)
    - Cada mensaje tiene asociado un único consumidor
    - El emisor y receptor de un mensaje no presentan dependencias de tiempo o sincronización (el receptor puede obtener el mensaje independientemente de si se estaba ejecutando o no, cuando el cliente envió el mensaje)
    - El receptor puede devolver un mensaje reconociendo el procesamiento con éxito del mensaje

Paradigmas de Computación Distribuida: Sistema de Mensajes (5)

- **Modelo de Mensajes publicación/suscripción**



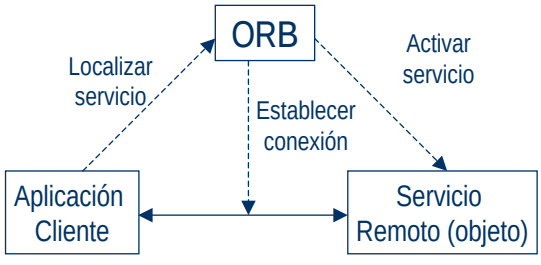
  - En este modelo, los clientes dirigen sus mensajes a un tema (*topic*). Los publicadores y suscriptores con generalmente anónimos y pueden publicar o suscribirse dinámicamente a cualquier tema de la jerarquía de contenidos
  - El sistema de mensajes se encarga de distribuir los mensajes que llegan de varios publicadores de un mismo tema a los posibles suscriptores. Los temas retienen los mensajes sólo el tiempo necesario para distribuir éstos a los suscriptores actuales
  - El sistema de publicación/suscripción posee las siguientes características:
    - Cada mensaje puede tener múltiples consumidores
    - Los publicadores y suscriptores tienen una dependencia temporal. Un cliente que se suscribe a un tema sólo puede consumir los mensajes publicados después de haberse creado su suscripción y debe mantenerse activo para poder continuar consumiendo mensajes
- APIs disponibles para implementar Sistemas de Mensajes
  - JMS (Java Message Service) [http://java.sun.com/products/jms/]
  - MSMQ (Microsoft Message Queuing) [http://www.microsoft.com/windows2000/technologies/communications/msmq/]

Agentes de Solicitud de Objetos (1)

- Un Agente de Solicitud de Objetos (ORB, *Object Request Broker*) es una tecnología Mw que gestiona la comunicación e intercambio de datos entre objetos
- Los ORBs promocionan la interoperabilidad de los sistemas de objetos distribuidos ya que permiten a los programadores construir sistemas mediante la estructuración de objetos (de diferentes vendedores) que se comunican entre ellos vía el agente OR
- Los detalles de implementación del ORB no son generalmente importantes para el desarrollador cuando se construye el sistema distribuido. Éstos sólo tratan con los detalles del interface del objeto.
  - Esta forma de ocultar información mejora el mantenimiento del sistema ya que los detalles de comunicación entre objetos se ocultan y aíslan en el ORB

Agentes de Solicitud de Objetos (2)

- La tecnología de ORBs fomenta el objetivo de la comunicación entre objetos a través de los barreras impuestas por las máquinas, el software y los vendedores
- Las funciones relevantes de la tecnología ORB son:
  - Definición de la interface
  - Localización y posible activación de los objetos remotos
  - Comunicación entre clientes y el objeto
- Un ORB actúa como una operadora telefónica. Proporciona un directorio de servicios y ayuda a establecer conexiones entre clientes y estos servicios



Agentes de Solicitud de Objetos (3)

- El ORB debe soportar muchas funciones para operar de forma efectiva y consistente, pero la mayor parte de estas funciones están ocultas al usuario del ORB
- Es responsabilidad del ORB proporcionar la ilusión de localidad, es decir, dar la apariencia de que el objeto es local al cliente, cuando éste, en realidad, puede residir en una máquina o proceso diferente. Por lo tanto, el ORB proporciona un marco de trabajo para la comunicación entre objetos a través de sistemas
- Además de la comunicación a través de los diferentes sistemas, debe lograrse la comunicación a través de las diferentes plataformas. Un ORB permite a los objetos ocultar sus detalles de implementación a los clientes, detalles que incluyen el lenguaje de programación, el sistema operativo, el Hw del servidor y la localización del objeto
- Existen múltiples formas de implementar la idea de los ORB. Por ejemplo, las funciones del ORB pueden ser compiladas junto con los clientes, pueden ser procesos independientes o pueden formar parte del núcleo del sistema operativo

Agentes de Solicitud de Objetos (4)

- Entre los entornos que soportan la tecnología ORB están:
  - La especificación CORBA (*Common Object Request Broker Architecture*) de Object Management Group
  - El modelo COM/DCOM de Microsoft
- Otra posibilidad es la utilización de la Invocación Remota de Métodos (RMI, *Remote Method Invocation*) especificado como parte del lenguaje Java y que proporciona funcionalidades de un ORB

ORB	Disponibilidad para plataformas	Aplicable a	Mecanismo	Implementación
DCOM	Originalmente PCs	Arquitectura de Sistemas Distribuidos centrada en PCs	APIs propietarias para los sistemas	Única (para PCs)
CORBA	Independiente de la plataforma y con interoperabilidad entre plataformas	Arquitectura General de Sistemas Distribuidos	Especificación de Tecnología de Objetos Distribuidos	Muchas (ORBIX, NEO, VisiBroker, PowerBroker, SmallTalkBroker, ...)
JavaRMI	Cualquiera que soporte una JVM	Arquitectura General de Sistemas Distribuidos e Intranets basadas en Web	Implementación de Tecnología de Objetos Distribuidos	Varias (tantas como implementaciones de JVMs)