Software Intermediario

Desarrollo de Software Basado en Componentes y Servicios

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es

TOADADTR

Máster Universitario en Desarrollo de Software

Índice

1 Conceptos fundamentales

2 Modelos de Software Intermediario

3 Orientación a Componentes

4 Conductores de Software

Índice

1 Conceptos fundamentales

2 Modelos de Software Intermediario

3 Orientación a Componentes

4 Conductores de Software

Índice

1. Conceptos fundamentales
2. Modelos de Software Intermediario
3. Orientación a Componentes
4. Conductores de Software

Índice

1. Conceptos fundamentales
2. Modelos de Software Intermediario
3. Orientación a Componentes
4. Conductores de Software

Software Intermediario

Middleware o Software Intermediario

"Software que reside entre la aplicación y los sistemas operativos, protocolos y hardware subyacente y cuyo objetivo es permitir que componentes heterogéneos y distribuidos se interconecten y colaboren entre si".

Misión del software intermediario

Respaldar el desarrollo, despliegue, ejecución y mantenimiento de aplicaciones, salvando la distancia entre la capa de aplicación y las capas subyacentes: sistema operativo, pila de red y hardware.

Objetivos Generales

- Ofrecer un modelo de programación distribuida de alto nivel
  - Elimina tareas tediosas y propensas a errores
- Soportar la heterogeneidad
  - independencia del lenguaje, sistema operativo, protocolos de comunicación, etc.
- Permitir reutilizar código previamente desarrollado
- Permitir utilizar Componentes de Terceras Partes (Commercial-off-the-shelf Components- COTS)
- Permitir definir *marcos de trabajo* con distribución de componentes que faciliten la reutilización del software

Problemática del Desarrollo de Sistemas Basados en Componentes Distribuidos

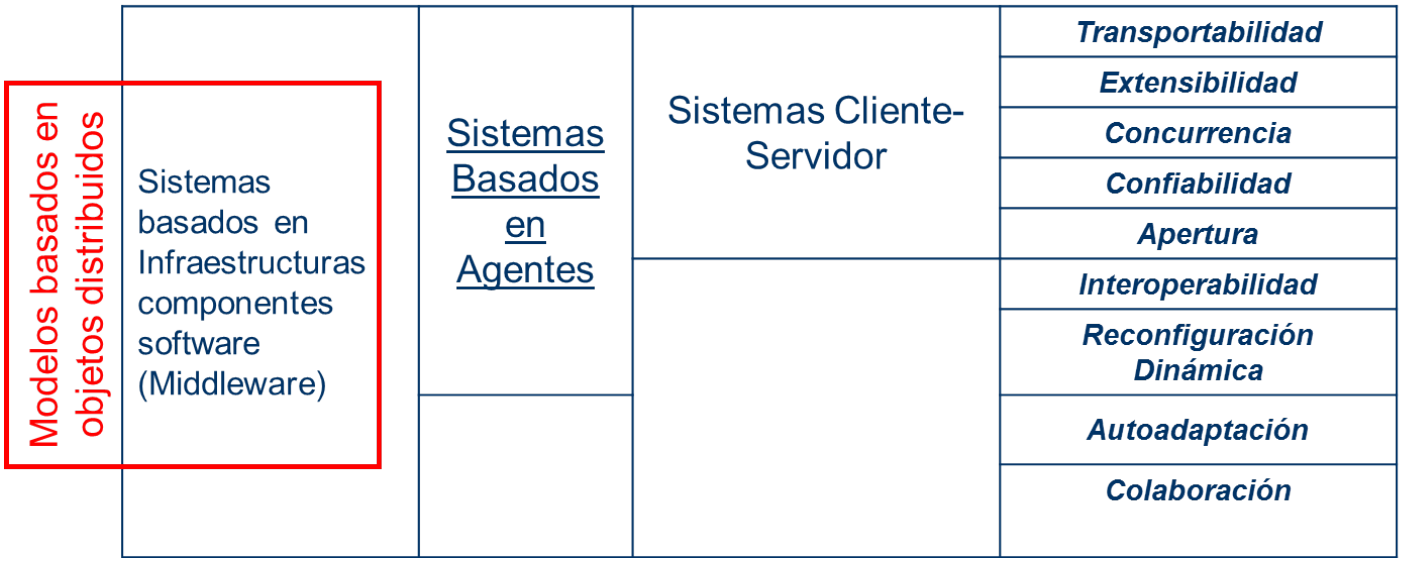


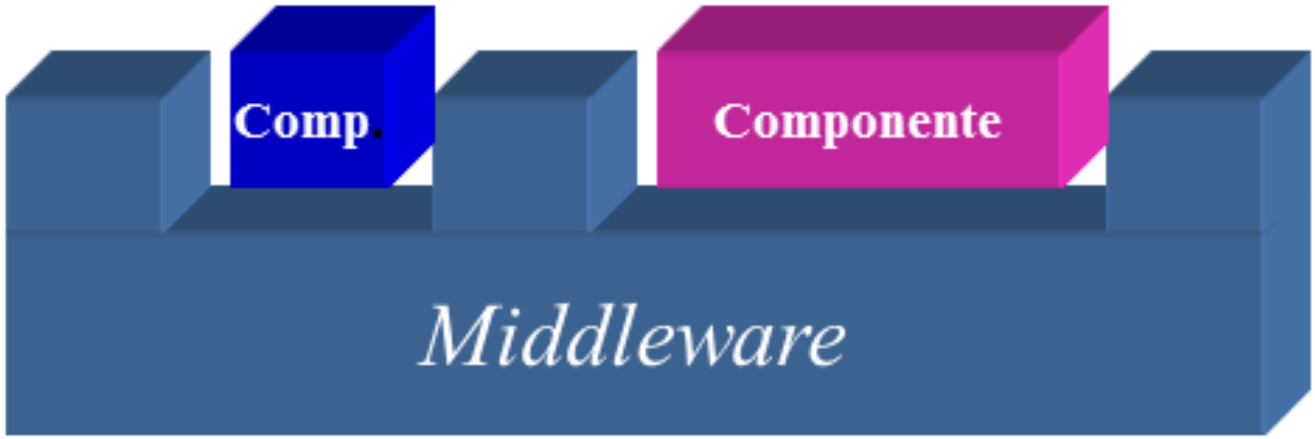
Figura: Taxonomía del software respecto del cumplimento de propiedades.

Objetivos específicos

Soporte para requisitos no funcionales

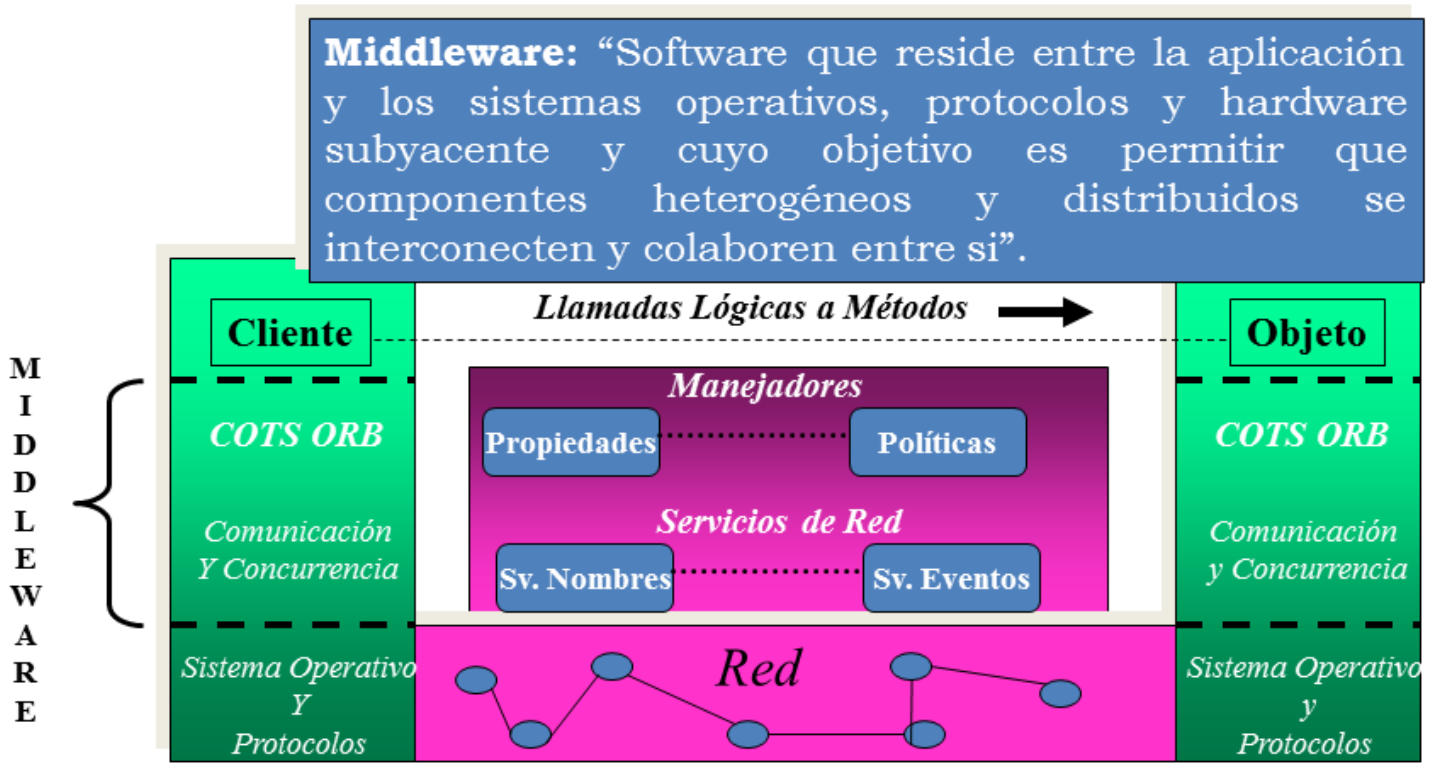
- Compartidos con modelos de componentes más antiguos:
  - *escalabilidad*, *extensibilidad*,
  - *confiabilidad*, *disponibilidad*,
  - *manejabilidad*, *usabilidad*, *estabilidad*
- Compartidos con modelos basados en agentes:
  - *interoperabilidad*,
  - *reusabilidad*, *equilibrado de carga*,
  - *adaptabilidad*,
- Propios del software intermediario y SOAs
  - *auto-organización*,
  - *publicación de servicios*, *descubrimiento de servicios*,
  - *Calidad de Servicio (QoS)*, *eficiencia*, *autonomía* y *seguridad*

Plataformas Distribuidas de Componentes



- Una plataforma de componentes es un entorno de ejecución para componentes software
- El middleware en sistemas distribuidos puede ser visto como una plataforma distribuida para componentes

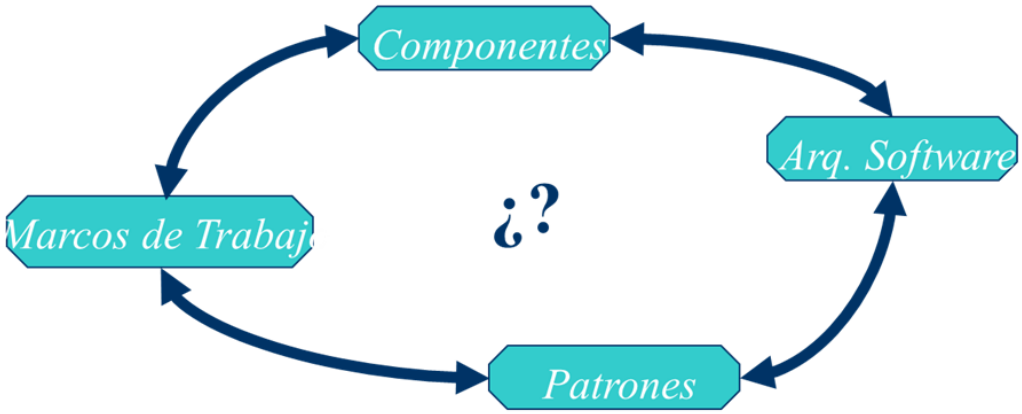
Arquitecturas de Software y Software Intermediario



Marcos de Trabajo y Software Intermediario

- Puede ser también considerado como un marco de trabajo para la construcción de sistemas distribuidos
- Las plataformas de componentes distribuidos hacen referencia al mismo concepto, pero hacen más énfasis en la facilidad de composición
- Los marcos de trabajo pueden ser:
  - Horizontales: infraestructuras de comunicación, interfaces de usuario, entornos de trabajo
  - Verticales o dependientes del dominio: telecomunicaciones (TINA), sistemas en tiempo real

Ontología Simple del Software Intermediario



El Middleware es la pieza clave en la aplicación de conceptos de software para “reutilización” en sistemas abiertos distribuidos”

Estructura del Software Intermediario

Categorías

- Infraestructura:
  - Encapsula los mecanismos de bajo nivel del sistema operativo para concurrencia y comunicaciones
  - Proporciona un modelo de objetos distribuidos de alto nivel que automatiza tareas comunes como: empaquetado y paso de parámetros, multiplexado y demultiplexado de llamadas, manejo de errores
- Servicios comunes:
  - Proporciona servicios de más alto nivel
  - Constituyen un marco de trabajo para desarrollo de aplicaciones

Modelos principales

Clasificación por nivel de abstracción

- Software Intermediario *lógico*
- Software Intermediario *físico*

Modelo lógico y configuración de las partes

- *Punto-a-punto*
- *Muchos-a-muchos*
- *Uno-a-muchos*

Punto a punto

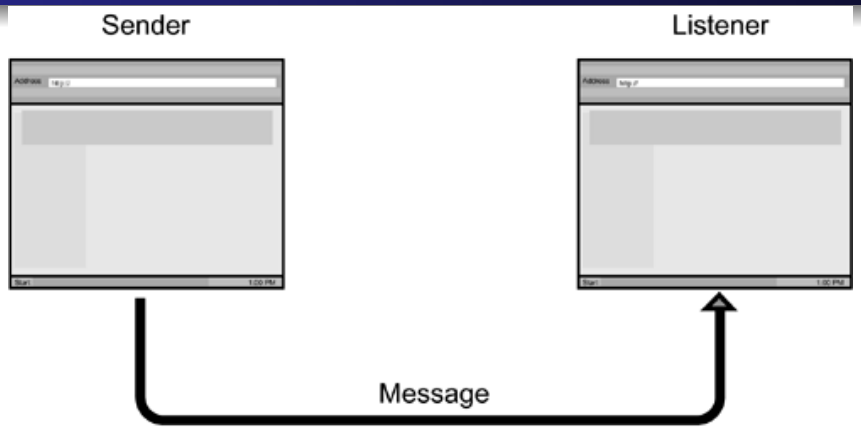


Figura: Software intermediario punto a punto: configuración.

Características

- Utilización de *encauzamiento simple* (*pipes and filters*) como arquitectura base
- Se adapta mejor al modelo RPC que al de objetos distribuidos
- ejemplos de implementación: MQSeries de IBM, DCE de un consorcio (Digital, HP, IBM,etc.)

Punto a punto II

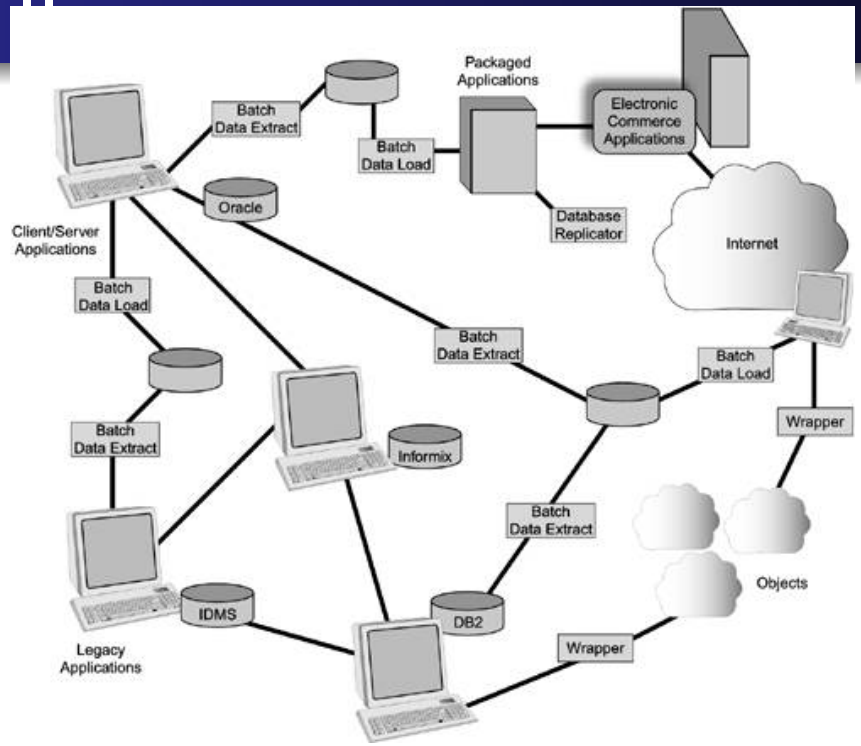


Figura: No funciona bien con algunas aplicaciones.

Desventajas

- Limitación en el desempeño del software intermediario
- No es una solución efectiva para integración de aplicaciones, termina en configuraciones alta complejidad

Muchos a muchos

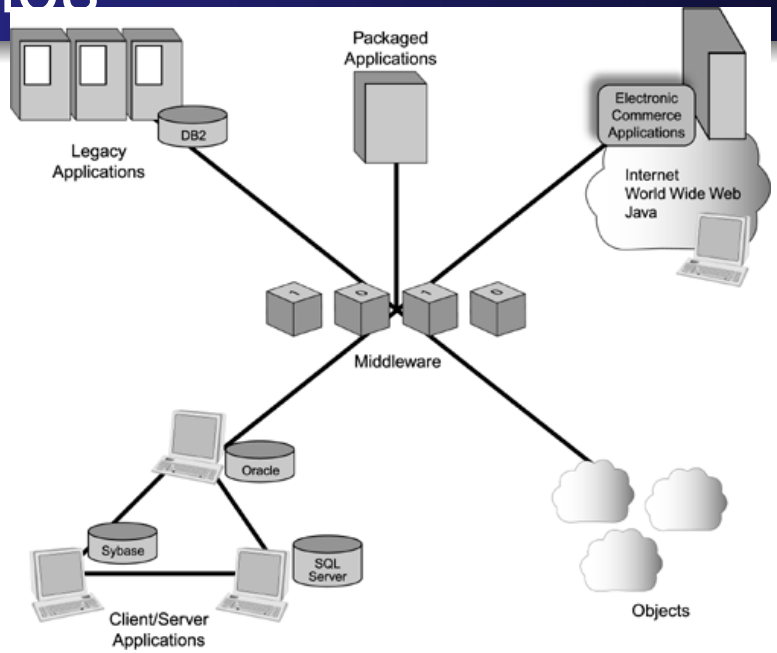


Figura: Software intermediario muchos-a-muchos: configuración.

Características

- Mejor para integración de aplicaciones dispares y remotas
- Modelo más potente: flexibilidad e interoperabilidad
- Se adapta bien al modelo de *objetos distribuidos*

Sincronización del *software intermediario*

Mecanismos de sincronización de mensajes

- Asíncrono
  - Mueve información entre 1 o varias aplicaciones, desvinculándose del origen y del destino
  - No se bloquea a las aplicaciones en ningún caso
  - Una aplicación no dependerá del estado de las otras para seguir ejecutándose
- Síncrono
  - *Acoplamiento fuerte* a las aplicaciones
  - Mayor fiabilidad en el proceso de llamadas a funciones
  - Se podría definir un estado cuyo valor depende de un cálculo remoto
  - Mayor robustez del software y facilidad de verificación
  - Caída del rendimiento de la aplicación global

Modelo físico de software intermediario

Característica

Describe tanto el método que se sigue para mover la información como la tecnología empleada para hacerlo

Tipologías del modelo físico

- Orientado a conexión
  - Las 2 partes se conectan, intercambian y se desconectan
  - Puede ser síncrono o asíncrono
- Sin orientación a conexión
  - La aplicación receptora actúa como consecuencia de una solicitud de la emisora, pero ambas no tienen por qué establecer conexión

Comunicación directa

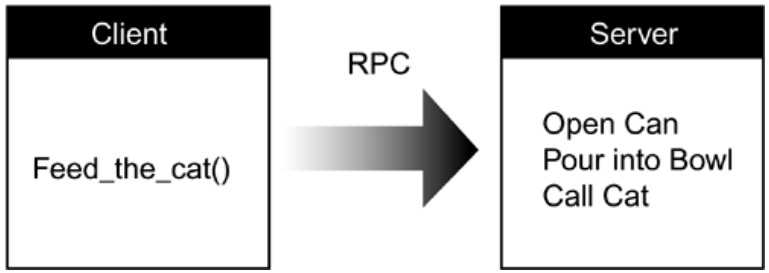


Figura: Llamadas a funciones ejecutadas remotamente con el mecanismo RPC.

Características

- El software intermediario se encarga de pasar el mensaje aceptado a la aplicación remota
- El modo de comunicación suele ser síncrono
- Adoptado por la mayoría del software intermediario basado en RPCs

Comunicación encolada

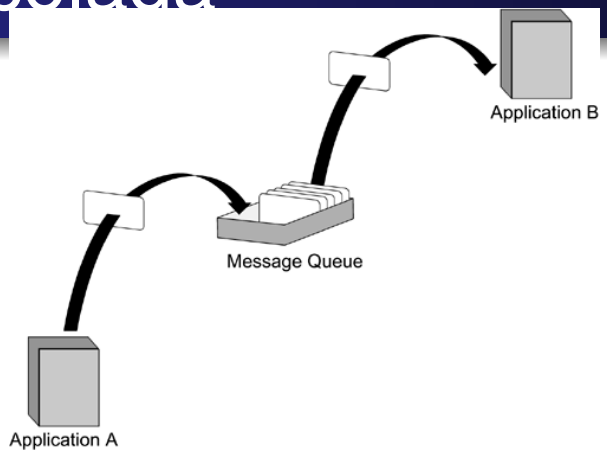


Figura: Comunicación de aplicaciones a través de una cola.

Características

- El software intermediario no bloquea a ninguna aplicación (solicitante u objetivo)
- El mensaje de la emisora se recibirá en un futuro
- La aplicación remota no necesitará estar activa
- Adoptado por la mayoría del denominado “Message Oriented Middleware”

Publicar y suscribir

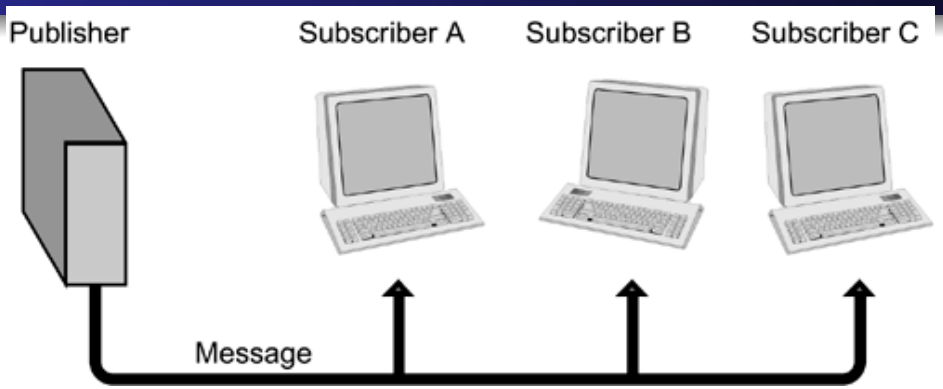


Figura: El modelo de software intermediario “publicar y suscribir”.

Características

- Cada mensaje está asociado con un tópico específico
  - Sigue un modelo basado en mensajes
  - Las aplicaciones interesadas se pueden suscribir a los mensajes de ese evento.
  - Cuando el evento esperado ocurre, se publica un mensaje
  - El sistema de mensajes del software intermediario distribuye el mensaje a los suscriptores.

Publicar y suscribir II

Características

- Ofrece una abstracción potente para multicasting o comunicación de grupo:
  - La operación *publicar* permite a un proceso hacer multidifusión a un grupo de procesos
  - La operación *subscribir* permite a los procesos escuchar tal multicast

Petición y respuesta

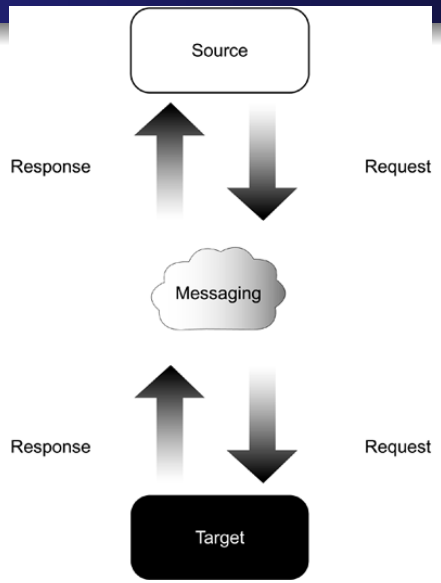


Figura: El modelo de middleware petición y respuesta.

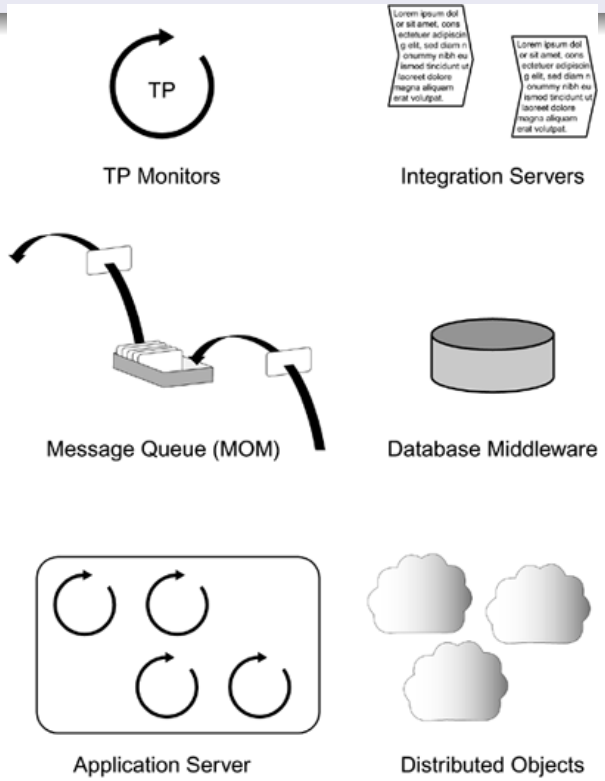
Características

- Software intermediario que se encarga de dar respuesta a peticiones asíncronas entre aplicaciones
- Ejemplos de este software: integración de servidores o implementación de servidores de aplicaciones

El problema de la clasificación del software intermediario

Idea fundamental

Existen varios tipos de software intermediario, cada uno de ellos resuelve su propio conjunto de problemas



El problema de la clasificación del software intermediario II

Clasificación atendiendo a la integración de aplicaciones

1. RPCs y MOM
2. Objetos Distribuidos
3. Orientación a Bases de Datos
4. Transaccional: monitores TP y servidores de aplicaciones

Modelo RPC

Características

- Este modelo da lugar al tipo más antiguo de software intermediario
- Se conoce como “middleware de bloqueo”: utiliza un mecanismo de comunicación síncrono entre procesos
- El bajo desempeño puede ser un problema del software construido con este modelo:
- Complementado con Message Oriented Model (MOM) se puede mejorar el desempeño del software que sigue este modelo

Modelo RPC II

- Excesiva sobrecarga a la red motivada por el uso de comunicaciones síncronas
- No escalabilidad del servicio de directorio
- Interoperabilidad limitada de las aplicaciones que utilicen este software
- Para que funcione bien se necesita una potencia de cálculo muy elevada:
  - En DCE de OSF, 1 RPC normalmente necesita 24 pasos para completar la solicitud
  - Varias llamadas a la red
  - Genera de 10,000 -15,000 instrucciones para procesar una solicitud remota
  - Llamadas a servicios de nombres y de traducción de direcciones IP

Modelo Orientado a Mensajes (MOM)

Idea fundamental

Intento de corregir las deficiencias del modelo RPC en cuenta a desempeño mediante el uso de una infraestructura de paso de mensajes.

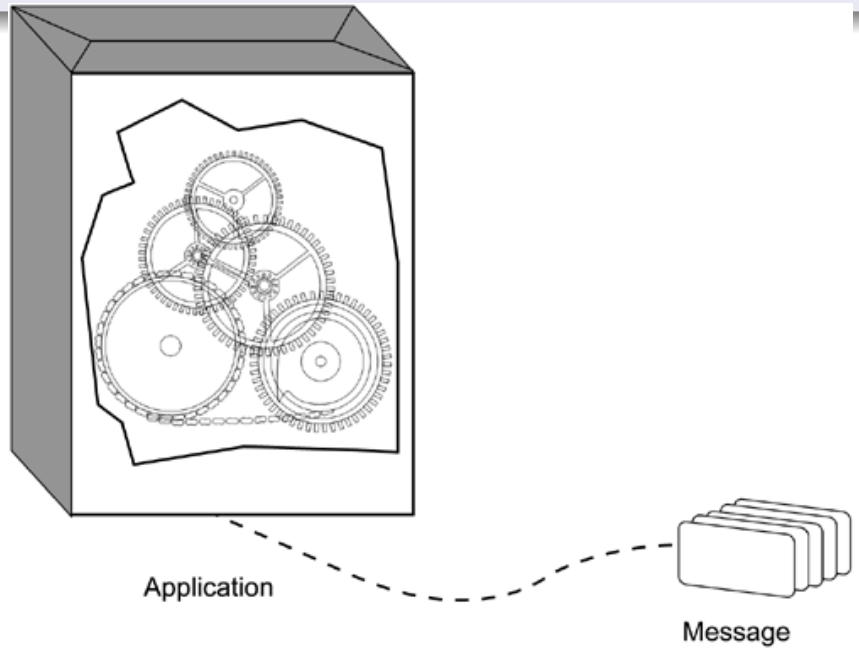


Figura: El uso de un MOM no hace parar el proceso de las aplicaciones.

Modelo Orientado a Mensajes (MOM)II

Facilidades

- MOM ha tenido una larga historia dentro de las aplicaciones distribuidas.
- Message Queue Services (MQS) han estado en uso desde los 80.
- IBM MQ*Series es un ejemplo de esto: http://www4.ibm.com/software/ts/mqseries/
- Otro soporte existente para este paradigma es Microsoft's Message Queue (MSQ): http://msdn.microsoft.com/library/psdk/msmq/msmq_overview_4ilh.htm
- Así como: Java's Message Service http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/

Modelo de Objetos Distribuidos

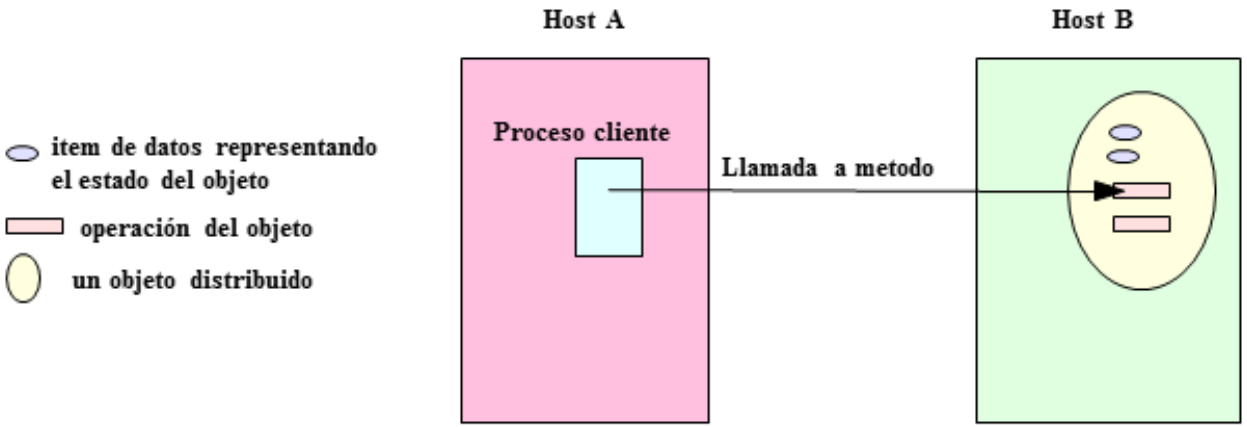


Figura: El modelo de software intermedio basado en objetos distribuidos.

"Los recursos de red se representan mediante objetos distribuidos. Para pedir un servicio desde un recurso, un proceso invoca una de sus operaciones o métodos, pasándole datos como parámetros del método, que se ejecuta en el *host* remoto, y se devuelve un valor como respuesta".

Modelo de Objetos Distribuidos II

Idea fundamental

- Aplicar orientación a objetos a las aplicaciones distribuidas como una extensión natural del desarrollo de software orientado a objetos.
- “Applications access objects” se encuentran distribuidos en los nodos de una red.
- Los objetos proporcionan métodos a través de cuya invocación desde las aplicaciones se obtiene servicio
- Submodelos actuales:
  - Remote Method Invocation (RMI) de Java
  - Object Request Broker (ORB) de CORBA
  - Servicios Web (parcialmente)
  - Object Spaces

Modelo de Objetos Distribuidos III

Características

- Orientación a *acciones* (objetos) vs. orientación a *datos* (paso de mensajes)
- Aunque es menos intuitivo para los humanos, el desarrollo de software con objetos distribuidos es un modelo más natural para realizar dicha actividad
- Un proceso que haga uso de un objeto distribuido se dice que es un proceso *cliente* del objeto, y los métodos del objeto se llaman *métodos remotos*
- Los objetos distribuidos aunque constituyen un enfoque elegante de desarrollo de software desde un punto de vista arquitectónico, generalmente no escalan ni proporcionan un software con buen desempeño

Remote Method Invocation (RMI)

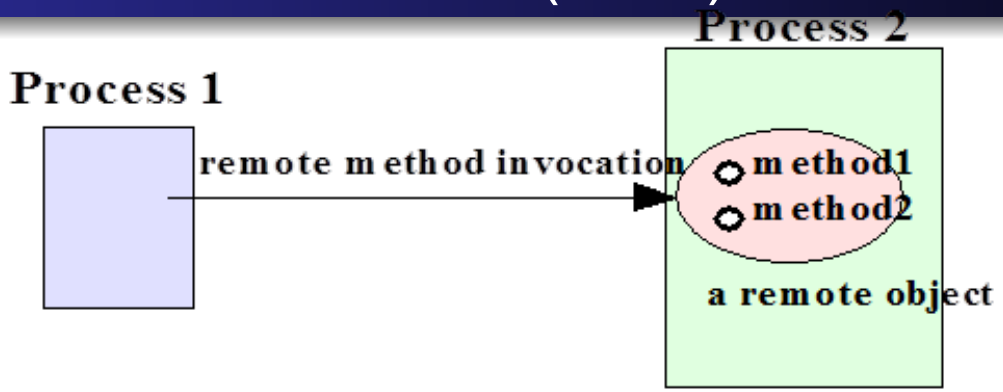


Figura: El modelo RMI.

Características

- Remote method invocation (RMI) es el equivalente "orientado a objetos" del modelo de programación basado en llamadas remotas (RPCs).
- En este modelo, un proceso invoca los métodos de un objeto, que puede residir en una máquina remota.
- Como ocurre con el protocolo RPC, los argumentos pueden pasarse con la invocación a un método.

Object Request Broker

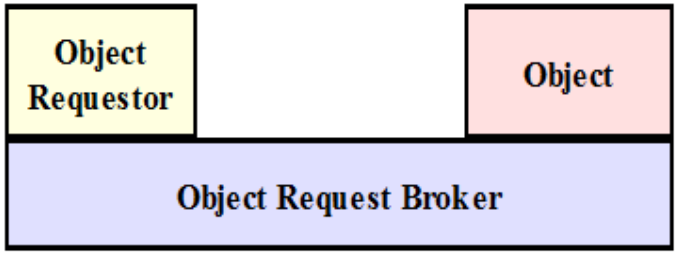


Figura: El modelo Object Request Broker.

Características

- Las aplicaciones envían peticiones a un intermediador de peticiones (ORB) que dirige la petición al objeto apropiado
- A diferencia de RMI, un ORB funciona como un software intermediario que permite a una aplicación cliente poder acceder a múltiples objetos locales o remotos.
- Un ORB funciona como un mediador entre objetos heterogéneos, permitiendo interacciones entre objetos implementados usando diferentes APIs y/o utilizando diferentes plataformas de ejecución.

El modelo Object Request Broker

Facilidades

- Este paradigma es la base del modelo que Object Management Group propugnó para la arquitectura CORBA (Common Object Request Broker Architecture): http://www.corba.org/
- Algunas cajas de herramientas basadas en esta arquitectura son las siguientes:
  - Inprise's Visibroker http://www.inprise.com/visibroker/
  - Java's Interface Development Language (Java IDL) http://java.sun.com/products/jdk/idl/
  - Orbix's IONA, y TAO del Object Computing, Inc. http://www.corba.org/vendors/pages/iona.html

Servicios en red

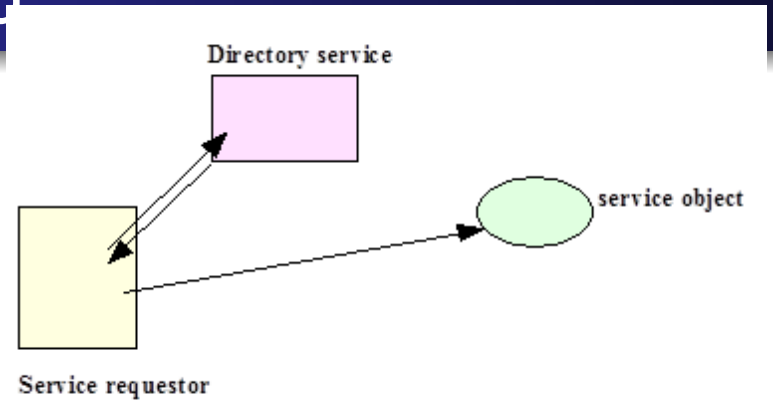


Figura: Modelo de software intermedio basado en Servicios.

Características

- Una aplicación que requiera un servicio contactará con el directorio , se le proporcionará una referencia
- El proceso interaccionará con el servicio, en lo sucesivo, utilizando la referencia.
- A diferencia de RMI, los objetos son buscados y accedidos por peticionarios del servicio en una red federada.
- La tecnología Jini de Java sigue este paradigma

Modelo “Object Spaces”

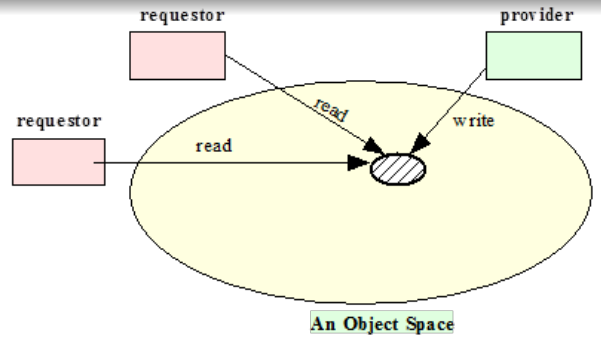


Figura: El modelo de software intermediario “object spaces”.

Características

- Se supone la existencia de entidades lógicas llamadas “object spaces”.
- Los participantes en un aplicación convergen en un espacio de objetos común.
- Un proveedor coloca objetos como si fueran entradas (entries) en un espacio de objetos, y las aplicaciones clientes que se suscriben y acceden a dichas entradas.

Modelo “Object Spaces” II

Facilidades

- Además se suele proporcionar un espacio virtual o *sala de reunión* entre proveedores y suministradores de recursos de red u objetos.
- Oculta el detalle derivado de las búsquedas de recursos u objetos, que se necesita en paradigmas tales como RMI, ORB, o Servicios en Red.
- Las facilidades actuales basadas en este paradigma incluyen el producto JavaSpaces de Sun: `http://java.sun.com/products/javaspaces/`.

Software Intermediario orientado a Bases de Datos

Definición

Se trata de cualquier software de intermediación que facilite la comunicación con una base de datos, bien desde una aplicación o desde otra base de datos. Utilizado para extraer información tanto de bases de datos locales como remotas.

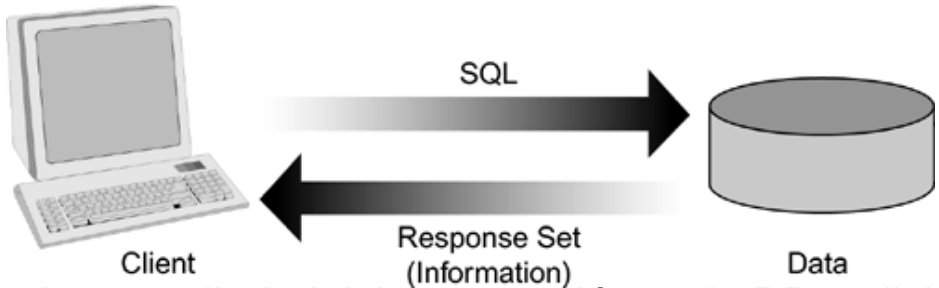


Figura: El software intermediario inicia una sesión en la BD, solicita información y la procesa .

Software Intermediario orientado a Bases de Datos II

Tipos de interacción del software

- Interfaces a Nivel de Llamada (CLI)
  - API común que abarca varios tipos de bases de datos
  - Ejemplos: ODBC de Microsoft (arquitectura y motor de drivers)
  - JDBC de Java, que funciona desde cualquier entidad de código: applet, servlet, JSP, EJB, etc.
- Software nativo para una BD dada

Software intermediario orientado a transacciones

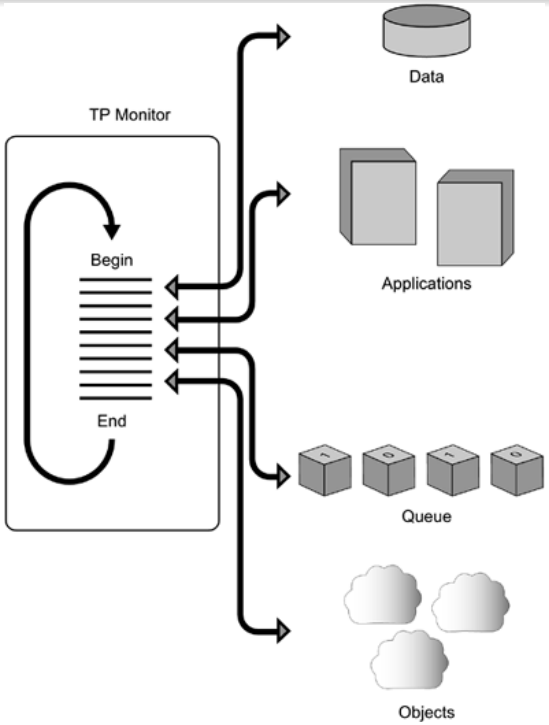


Figura: Esquema de un monitor TP.

Concepto

El software intermediario transaccional coordina los movimientos de información y la compartición de métodos

Software intermediario orientado a transacciones

Características

- Tiende a crear una solución basada en integración fuertemente acoplada de aplicaciones
- Los modelos basados en intercambio de mensajes propician soluciones más cohesivas

Tipos

- Monitores TP
- Servidores de aplicaciones

Tipos de sistemas para TP

Monitores TP

- Mecanismos para facilitar la comunicación entre 2 o más aplicaciones,
- Ubicación de la lógica de la aplicación,
- Consiguen incrementar el rendimiento de las aplicaciones: equilibrado de carga, planificación prioritaria de mensajes y creación automática de hebras en el servidor
- Ejemplos: Tuxedo de BEA Systems, MTS de Microsoft y CICS de IBM

Conductores de software

Idea fundamental

Una capa de abstracción-software que oculta a las aplicaciones los detalles de dispositivos hardware o de otro software de más bajo nivel. Está inspirado en el concepto de software intermediario pero no se le puede considerar esto propiamente.

A diferencia de un software intermediario, los conductores de software no definen un auténtico modelo de componentes.

Conductores de Software

Ejemplos actuales

- La distribución de software *Mer* (http://en.wikipedia.org/wiki/Mer_(software_distribution))
- El sistema operativo Android proporciona una capa de software intermediario que incluye bibliotecas que proporcionan servicios tales como almacenamiento de datos, pantallas, multimedia y exploradores Web. Los servicios se consiguen ejecutar muy rápido porque que las bibliotecas de software intermediario se compilan contra un lenguaje máquina. La capa de software intermediario de Android también contiene a la máquina virtual Dalvik y sus bibliotecas de aplicación Java nucleares.

Conductores de Software II

Ejemplos actuales

- Software de *motor de juegos*, tales como *Gamebryo* y *Renderware*
- Los desarrolladores de redes de sensores inalámbricas (WSNs) pueden utilizar *software de intermediación* para abordar los desafíos que presentan las WSNs y sus tecnologías.
- El sistema oprativo QNX ofrece *software de intermediación* para proporcionar servicios multimedia de uso en automóviles, aviones y otras plataformas de software empotrado.

Conductores de Software III

Ejemplos actuales

- El “Sistema de Sonido Miles” (MSS) proporcionó un conductor de *software de intermediación* que permite una amplia variedad de tarjetas de sonido distintas sin que fuera necesario preocuparse de los detalles
- ILAND es un *software de intermediación* basado en servicios dedicado a las aplicaciones de tiempo real. Ofrece respaldo para la reconfiguración determinística en un plazo de tiempo imitado.
- Televisión, media y software de Sintonizador de TDT: “middleware OpenTV 5", es una distribución HTML5 y Linux para sintonizadores de TDT, que incluye una guía de navegación para programas de televisión y proporciona APIs abiertas

Bibliografía Fundamental

Erl, T. (2005).

Service-Oriented Architecture: Concepts, Technology, and Design.

Prentice-Hall.

Szyperski, C. (1998).

Component Software. Beyond Object-Oriented Programming.

Addison–Wesley, **Básica**.

Verissimo, P. and Rodrigues, L. (2004).

Distributed Systems for System Architects.

Kluwer Academic.