# Referencias

Cabello, A. L. (2014). *UF1846: Desarrollo de Aplicaciones Web Distribuidas.* Antequera: ic Editorial.

Muñoz Escoí, F. D., Argente Villaplana, E., Espinosa Miguet, A. R., Galdámez Saiz, P., García Fornes, A., de Juan Marín, R., & Renda Roig, J. S. (2012). *CONCURRENCIA Y SISTEMAS DISTRIBUIDOS.* Valencia: UNIVERSITAT POLITECNICA DE VALENCIA.

**Qué son**

Se entiende por sistema distribuido a esa colección de ordenadores que están separados físicamente pero conectados entre sí por una red de comunicaciones distribuida. Cada ordenador que está conectado a esta red distribuida posee sus propios componentes hardware y software . Desarrollo de Aplicaciones Web Distribuidas. Pág. 7.

Normalmente el usuario desconoce que está accediendo a un sistema distribuido, ya que esto no es importante para él. Accederá al RPC (recursos remotos de las máquinas) de la misma forma que si lo hiciera a un recurso local de su máquina. Desarrollo de Aplicaciones Web Distribuidas. Pág. 7.

**Tipos y configuraciones**

**Centralización**

<https://es.wikipedia.org/wiki/Computaci%C3%B3n_centralizada#Estado_actual>

**Características básicas de un directorio de servicios**

Los directorios van a permitir localizar la información. Obviamente esta información tiene que estar almacenada de un modo específico (organización de la información) para que no se pierda mucho tiempo en poder acceder a ella. Un directorio se puede ver como una base de datos especializada y distribuida, pero obviamente diferirá de esta en:

**Distribución de los datos.**

Normalmente las bases de datos no suelen estar distribuidas, aunque pudiera ser el caso. Una base de datos distribuida va a constar de una tabla en un servidor y otra en otro servidor (se conoce como fragmentación vertical) y unos protocolos o estándares complejos para poder distribuir la información (este último punto es lo que complica el uso de base de datos distribuidas). Sin embargo, los directorios de servicios permitirán que los datos relativos a una entidad sean almacenados en un servidor controlado por dicha entidad (fragmentación horizontal frente a la vertical de las bases de datos distribuidas). Mediante esto se obtiene una mayor potencia y optimización en las simplificaciones, búsquedas y consultas.

**Replicación de los datos.**

Si se está en una base de datos distribuida se puede preparar para usar una réplica de los datos, eso sí, en un número reducido de servidores dado que se debe asegurar la consistencia y el sincronismo de los datos (hacerlo todo al mismo tiempo en todos los servidores). En el caso del directorio de servicios se tendrán que aceptar ciertas inconsistencias temporales, haciendo uso en esos casos del servidor replicado. Además se puede mejorar el rendimiento haciendo réplicas cercanas a los usuarios que acceden (se optimiza el acceso al directorio). Se tiene que anotar que si un directorio de servicio es usado por varias aplicaciones para autentificar, controlar, o gestionar implica una menor fiabilidad del mismo.

Distribuidas. Pág. 112.

**Utilidad**

Objetivo

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora y mainframe, mientras se mantiene la flexibilidad de trabajar en múltiples problemas más pequeños. Por lo tanto, la computación en malla (grid) es naturalmente un entorno multiusuario; por ello, las técnicas de autorización segura son esenciales antes de permitir que los recursos informáticos sean controlados por usuarios remotos.

<https://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida#Objetivo>

Estructura y Arquitectura

CORBA [Obj11a] es un est´ andar multilenguaje y multiplataforma (facilita interoperabilidad), pero su complejidad limita su difusi´ on.

RMI [Ora12f, Ora12d] ´ unicamente soporta Java (no es multilenguaje).

Concurrencia y sistemas distribuidos. Pág. 179.

Desde los inicios de la informática las empresas han ido desarrollando sus actividades y sus servidores en base a las necesidades de negocio, bien para ser usados de forma aislada, o bien para ser empleados de forma muy ligera en la red. En el momento que comienza la expansión a nivel mundial de una empresa se hace necesario aumentar las capacidades de sus sistemas informáticos, pero no solo es preciso aumentar, sino integrar todo en uno para que se realicen de forma correcta las actividades. Intentar integrar algo que no ha sido diseñado para ese fin es una tarea bastante complicada, por eso surge la Arquitectura Orientada a Servicios (SOA), la cual consta de actividades o procesos que están diseñados para ofertar servicios. Un servicio es un componente software bien definido e independiente de su implementación (transportable).

Un servicio web es una definición en un lenguaje estándar de un programa objeto, base de datos, etc. Un servicio web se rige por los siguientes estándares:

■ El lenguaje en que los servicios son descritos.

■ El protocolo de comunicación de mensajes escritos (SOAP).

Desarrollo de Aplicaciones Web Distribuidas. Pág. 13.

**5. Implementación de arquitecturas orientadas a servicios mediante tecnologías web**

Hace algunos años, la gran mayoría de las organizaciones tomó parte en la estrategia de desarrollar aplicaciones distribuidas usando distintas tecnologías que no tenían nada que ver entre sí, con el inconveniente de que debían interactuar entre ellas. Algunos ejemplos de estas tecnologías son:

■ COM (Component Objetct Model).

■ CORBA (Common Object Request Broker Architecture).

■ EJB (Entrepise Java Beans).

■ Etc.

Sin embargo, actualmente, la tecnología SOAP (Simple Object Access Protocol) permite resolver los problemas de falta de interoperabilidad entre las opciones anteriores, todo esto gracias a que tiene una base en HTML y XML.

**CORBA**

3.10 corba

El paradigma orientado a objetos juega un importante rol en el desarrollo de software y cuenta con gran popularidad desde su introducción. La origen usa solo una instancia de cada clase. CORBA también hace uso de una capa de software conocida como ORB (Object Request Broker), que permite a los objetos realizar llamadas a métodos situados en máquinas remotas a través de una red. Además, ORB maneja la transferencia de estructuras de datos de manera que sean compatibles entre los dos objetos. ORB es un componente

fundamental de la arquitectura CORBA y su misión es facilitar la comunicación entre objetos. ORB también debe de facilitar diferentes transparencias, tales como de localización, implementación, comunicación y ejecución.

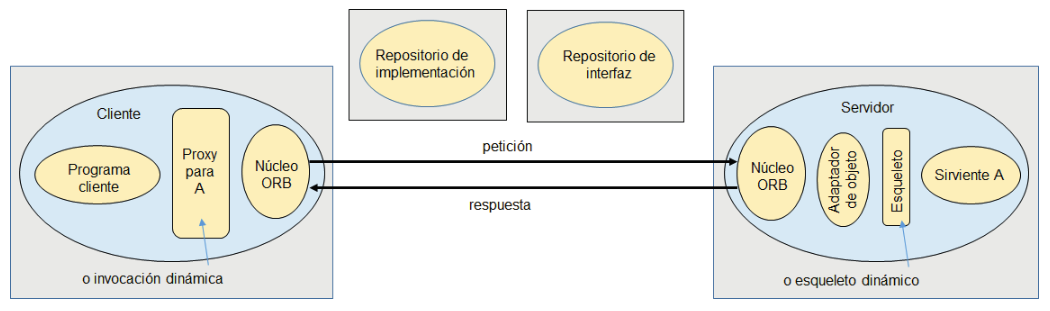
En CORBA [Coulouris et al., 2001], el servidor crea objetos remotos, hace referencias accesibles a objetos remotos y espera a que los clientes invoquen a estos objetos remotos o a sus métodos.

Por su parte, el cliente obtiene una referencia de uno o más objetos remotos en el servidor e

Invoca a sus métodos.

La arquitectura CORBA (ver figura 3.13) está diseñada para dar soporte al rol de un intermediario de peticiones de objetos que permita que los clientes invoquen métodos en objetos remotos, donde tanto los clientes como los servidores pueden implementarse en diversos lenguajes de programación.

Figura 3.13. Principales componentes de la arquitectura CORBA [Coulouris et al., 2001]



La función de los principales componentes se describen a continuación [Coulouris et al., 2001]. El núcleo de ORB es un módulo de comunicación el cual permite una interfaz que incluye las operaciones de arranque y paro, las operaciones para la conversión entre referencias a objetos remotos y cadenas de texto, así como las operaciones para obtener listas de argumentos para llamadas que emplean invocación dinámica. El adaptador de objeto sirve como puente entre los objetos con interfaces IDL y las interfaces IDL, además de las interfaces del lenguaje de programación de las correspondientes clases sirvientes. Las clases esqueleto se generan en el lenguaje del servidor a través de un compilador de IDL. El esqueleto sirve para despachar las invocaciones a los métodos remotos, asimismo desempaqueta los argumentos desde los mensajes de petición y empaqueta las excepciones y los resultados en mensajes de respuesta.

Los resguardo/proxy son los encargados de empaquetar los argumentos de los mensajes de invocación y desempaqueta las excepciones y los resultados de las respuestas.

Cada repositorio de implementación activa los servidores registrados bajo demanda y localiza los servidores que están en ejecución en cada momento usando el nombre del adaptador de objeto. El repositorio de interfaz proporciona información sobre las interfaces IDL registradas a los clientes y a los servidores que lo requieran. La interfaz de invocación dinámica permite que los clientes puedan realizar invocaciones dinámicas sobre objetos remotos CORBA cuando no es práctico el uso de proxies.

La interfaz de esqueleto dinámica permite que un objeto CORBA acepte invocaciones sobre una interfaz para la que no hay un esqueleto.

El código delegado permite convertir a objetos CORBA aquellos fragmentos de código existente que fue diseñado sin prever los objetos distribuidos.

2013 Sistemas Distribuidos. Pág. 50

Desarrollo de Aplicaciones Web Distribuidas. Pág. 57.

**5.1. Especificaciones de servicios web de uso común: SOAP, REST, etc. A continuación, se muestra un listado con los estándares más comunes en los servicios web:**

■ WWW. Ofrece un conjunto de servicios y protocolos de web.

■ XML. Formato estándar de intercambio de datos en la web.

■ SOAP. Protocolo sobre el que se realiza ese intercambio de datos en la web.

■ XML-RPC. Protocolo sobre el que se realiza el intercambio de datos en la web.

■ HTTP. Protocolo para navegar por la web.

■ FTP. Protocolo para intercambiar datos.

■ SMTP. Protocolo para intercambiar mensajes (email).

■ WSDL. Lenguaje de interfaz pública para los servicios web (al fin y al cabo es un documento codificado bajo XML).

■ UDDI. Protocolo para publicar la información de los servicios web (se puede saber mediante este protocolo qué servicios se tienen disponibles y cuáles no).

■ WS-Security. Protocolo de seguridad estándar para la web. Garantiza la autentificación y la confidencialidad en los mensajes que se realicen.

Desarrollo de Aplicaciones Web Distribuidas. Pág. 58.

**SOAP**

SOAP (Simple Object Access Protocol) es un protocolo de intercambio de información estructurada en un entorno que está distribuido. Se basa en el uso de XML para describir el formato de los mensajes que se intercambian y se usa conjuntamente en los servicios web. SOAP se apoya en dos principios fundamentales que son:

■ Simplicidad.

■ Extensibilidad. La especificación de SOAP para poder intercambiar información consta de:

■ El modelo de procesos (necesario para el procesamiento de los mensajes).

■ El modelo de extensión (necesario para las características de los mensajes).

■ El protocolo de enlace (necesario para establecer las reglas de intercambio de mensajes).

Desarrollo de Aplicaciones Web Distribuidas. Pág. 59.

**8.4.5 RMI(RemoteMethodInvocation)** Los entornos middleware de comunicación norientada a objetos tradicionales (CORBA, DCOM, ICE, ...) soportan aplicaciones distribuidas cuyos componentes estén implantados en distintos lenguajes de programación. Una alternativa m´ as simple es proporcionar una solución sobre un único lenguaje orientado a objetos que garantice portabilidad. Java RMI [Ora12f] implementa el mecanismo de invocación de objetos remotos para el lenguaje Java. Extiende el lenguaje con la posibilidad de invocar métodos de un objeto Java en otra máquina virtual Java ( JVM :“ Java Virtual Machine ”), y pasar objetos Java como argumentos al invocar dichos métodos. Como el lenguaje Java incluye el concepto de interfaz (interface ), no se necesita un IDL adicional. Los proxy y esqueleto se generan a partir de la especificaci´ondesuinterfazremota.Enlasprimerasversiones(antesdelaedici´ on J2SE v5.0) se deb´ıa utilizar un generador de stubs ( rmic ).Ahorayanoresulta necesario.

Las reglas para programar utilizando objetos remotos son:

La interfaz del objeto remoto se define como una interfaz cualquiera en Java, pero debe extender la interfaz java.rmi.Remote (es un marcador para identificar interfaces remotas). Gracias a ello el compilador de Java generar´ alos proxies y esqueletos para gestionar las invocaciones remotas.

Lainvocaci´ondeunm´ etodo sobre un objeto remoto debe indicar que se puedegenerarlaexcepci´onpredefinida java.rmi.RemoteException .

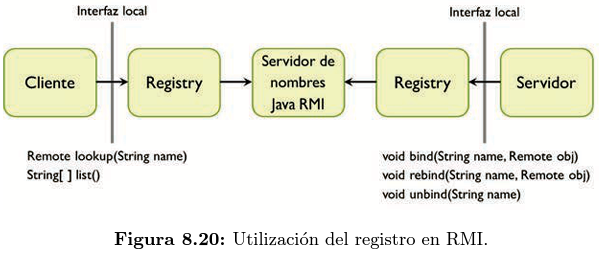
Adem´as de implantar la interfaz Remote la clase que mantenga el c´ odigo servidor debe extender la clase java.rmi.server.UnicastRemoteObject . Esto resulta necesario para registrar los objetos en el gestor de invocaciones (ORB) de Java. Para ello se utiliza, de manera impl´ıcita, el m´ etodo exportObject() de esa clase java.rmi.server.UnicastRemoteObject .

Alinvocarunm´etodopodemospasarobjetoscomoargumentos.Losobjetoslocales (los que residen en el nodo del invocador) se pasan por valor y por lo tanto deben ser serializables. Por su parte, los no locales se pasan por referencia, transmitiendo un proxy del objeto.

Elmodelodeprogramaci´onRMInopersiguetransparenciatotal:parapasaruna aplicaci´oncentralizadaadistribuidasonnecesariasciertasmodificaciones:

Un modelo de objetos remotos se basa en un servidor de nombres. RMI utiliza un servidor de nombres en el que se registran objetos remotos bajo nombres simb´ olicos (junto a cada nombre se registra la referencia del objeto correspondiente). El servidor de nombres puede residir en cualquier nodo, y resulta accesible tanto a cliente como a servidor bajo una interfaz local denominada Registry . La figura 8.20 muestra las operaciones disponibles en esa interfaz.

En el caso de las distribuciones Java desarrolladas por Oracle, el servidor de nombres se lanza mediante la orden rmiregistry .



El servidor puede utilizar los m´ etodos bind(name, obj) , rebind(name, obj) y unbind(name) . Las operaciones bind() y rebind() permiten registrar objetos, mientras que unbind() elimina ese registro.

El cliente puede utilizar los m´ etodos lookup(name) y list() .Laoperaci´ on lookup() permite obtener la referencia asociada a un nombre, y list() retorna un vector de cadenas que contiene los nombres de todos los objetos registrados.

Concurrencia y sistemas distribuidos. Pág. 190.

**Características**

Los sistemas distribuidos tienen que ser bastante fiables, dado que si un ordenador de esa red no está disponible hay que tomar la decisión de que se reemplace por otro que haga su trabajo (tolerancia a fallos). El tamaño de estos sistemas distribuidos es bastante variable, por ello se habla de escalabilidad (puede ir de una decena a millones de ordenadores resolviendo un problema). Desarrollo de Aplicaciones Web Distribuidas. Pág. 7.

2. Características generales de las arquitecturas de servicios distribuidos Un sistema distribuido deberá cumplir una serie de características o propiedades para garantizar un funcionamiento correcto y óptimo.

Estas características son:

■ Transparencia.

■ Escalabilidad.

■ Fiabilidad y tolerancia a fallos.

■ Consistencia.

Desarrollo de Aplicaciones Web Distribuidas. Pág. 7.

2.1. Transparencia

Cuando se habla de transparencia se hace referencia al hecho de que ni el usuario ni las aplicaciones que este usa saben que están frente a un sistema distribuido, es decir, más bien parece que está siendo todo gestionado por una sola máquina. Asimismo, la distribución física de los recursos que componen el sistema distribuido debe ser transparente para el usuario. La propiedad de transparencia dentro de un sistema distribuido se aplica en varias partes, tales como las que se describen a continuación.

Identificación

Los recursos de los cuales consta un sistema distribuido han de ser totalmente independientes de la topología de red y de la distribución de estos mismos. Una aplicación no tiene por qué saber en qué lugar se encuentra el recurso, simplemente lo llama y lo utiliza para obtener los resultados esperados.

Ubicación física de recursos

Esto implica que ni el usuario ni la aplicación tienen que conocer el recurso, ni en qué nodo del sistema distribuido se encuentra este. Gracias a ello, si un nodo que tiene un recurso falla, se puede sustituir sin que el usuario o la aplicación lo noten.

Desarrollo de Aplicaciones Web Distribuidas. Pág. 8.

**Ventajas y Desventajas de los Sistemas Distribuidos**

1.2 Ventajas y desVentajas de los sIstemas dIstrIbuIdos

1.2.1 Ventajas de los sistemas distribuidos con respecto a los sistemas centralizados

Entre las principales ventajas de los sistemas distribuidos con respecto a las computadoras centralizadas se encuentran:

• Economía: Los microprocesadores ofrecen una mejor relación precio/ rendimiento que las computadoras centrales.

• Velocidad: Un sistema distribuido puede tener mayor poder de cómputo que una computadora centralizada individual.

• Distribución inherente: Implica que un sistema distribuido puede emplear aplicaciones instaladas en computadoras remotas.

• Confiabilidad: El sistema es consistente, aun si una computadora del sistema deja de funcionar.

• Crecimiento proporcional: Cada vez que se requiera mayor poder de cómputo en el sistema, solo se pueden adicionar los incrementos de cómputo requeridos.

1.2.2 Ventajas de los sistemas distribuidos con respecto a las computadoras aisladas

Con respecto a las computadoras aisladas, es decir, aquellas que no se encuentran conectadas a una red, los sistemas distribuidos tienen las siguientes ventajas [Coulouris, Dollimore & Kinderberg, 2001]:

• Datos compartidos: Permite que distintos usuarios tengan acceso a una base de datos o archivo común.

• Dispositivos compartidos: Permite compartir un recurso costoso entre distintos usuarios, como plotters o impresoras láser.

• Comunicación: Brinda la posibilidad de comunicación de usuario a usuario (telnet, correo electrónico, etc.).

También mirar el resumen del capítulo 1 item 8 en Desarrollo de Aplicaciones Web Distribuidas. Pág.

2013 Sistemas Distribuidos. Pág. 17.

**Relación de los sistemas distribuidos con el modelo cliente servidor y sus ventajas.**

**Estructura y arquitectura.**

**2.1. Arquitectura cliente/servidor**

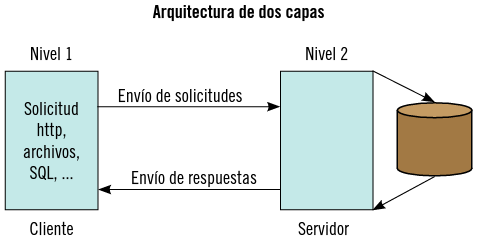
Se tiene que hacer mención a la arquitectura cliente/servidor (sistema donde el cliente es una máquina que solicita un determinado servicio al servidor que es la máquina que lo proporciona). Un servicio puede ser un determinado programa o el acceso a determinados datos y a sus diferentes arquitecturas. Arquitectura cliente/servidor de dos capas Solo aparecen los componentes o capas de presentación y lógica de la aplicación. Normalmente se usa esta arquitectura cuando se produzcan los siguientes escenarios:

■ Poco requerimiento de datos.

■ Base de datos centralizada en un solo servidor.

■ Base de datos estática.

■ Mínimo mantenimiento del sistema.



Desarrollo de Aplicaciones Web Distribuidas. Pág. 132.

Arquitectura cliente/servidor de tres capas Nivel 2 Servidor Dentro de esta arquitectura se encuentra la capa de presentación, la capa de lógica de la aplicación y otra capa de base de datos. Normalmente se implementa en alguno de los siguientes escenarios:

■ Mucho procesamiento de datos en la aplicación.

■ En aplicaciones donde la funcionalidad esté cambiando constantemente.

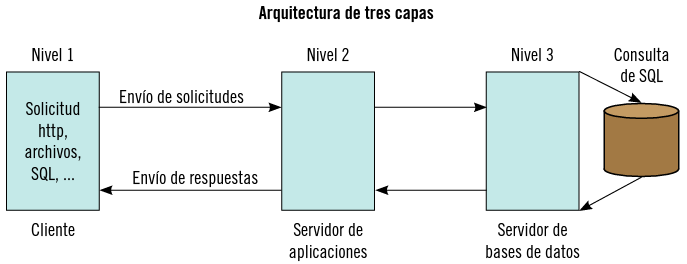
Desarrollo de Aplicaciones Web Distribuidas. Pág. 132.

■ Cuando los procesos no estén fuertemente relacionados con los datos.

■ Cuando se necesite aislar la tecnología de la base de datos para que su mantenimiento (tanto de base de datos como de tecnología) sea fácil de manipular.

■ Cuando se necesite separar código del cliente para facilitar su mantenimiento.

■ Cuando se use POO (Programación Orientada a Objetos).



Desarrollo de Aplicaciones Web Distribuidas. Pág. 133.

**Aplicación práctica**

Usted pertenece al departamento de programación de una empresa y tiene que montar un videoclub virtual. Justifique con qué modelo de capas llevaría a cabo el proyecto.

SOLUCIÓN

El videoclub virtual tendrá usuarios que se registrarán (previo pago o no) para poder alquilar películas (que estarán disponibles). Dado que el volumen de usuarios puede ser elevado, el nivel de archivos de películas también será alto, pero además se debe conservar información de los alquileres. Todo esto obliga a crear, como mínimo, una base de datos y lo ideal sería apostar por el modelo de dos capas, sobre todo si se busca seguridad en los datos (que es el caso).

Desarrollo de Aplicaciones Web Distribuidas. Pág. 133.

**Sockets**

Sistemas\_Distribuidos\_Sockets NIVELES DE COMUNICACIÓN.pdf

**Tendencia de los sistemas distribuidos**

Los sistemas distribuidos están experimentando un período de cambios significativos y esto se remonta a una serie de tendencias influyentes:

• el surgimiento de una tecnología de red generalizada;

• el surgimiento de la computación ubicua junto con el deseo de apoyar la movilidad del usuario en sistemas distribuidos;

• la creciente demanda de servicios multimedia;

• la vista de los sistemas distribuidos como una utilidad.

<http://www.padakuu.com/article/214-trends-in-distributed-systems>

**Qué es middleware.**

3. Programación de diferentes tipos de acceso a servicios

Se sabe que el concepto de Middlware hace referencia a un software que permite interactuar con otras aplicaciones informáticas. Gracias a este concepto se reduce la tarea del programador en el diseño de sistemas distribuidos facilitando una mejora de la calidad del servicio, de la seguridad, del envío de mensajes, etc.

En concreto, se tienen los siguientes accesos a servicios:

■ Publicación/Suscripción.

■ Repositorios.

■ Agentes de usuario.

■ Proveedores y consumidores.

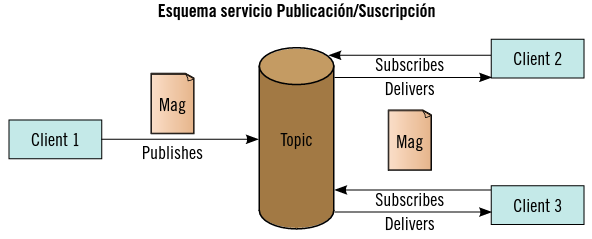
3.1. Servicios basados en publicación/suscripción

Gracias al servicio de Publicación/Suscripción se consigue que las aplicaciones no estén íntimamente unidas a una determinada tecnología. Este servicio se va a encargar únicamente del manejo de los datos como si fueran mensajes y libera del proveedor de dichos datos.

Desarrollo de Aplicaciones Web Distribuidas. Pág. 149.

Al proveedor de la información se le denomina publicador (proporcionan información sobre un determinado asunto o tema) mientras que el consumidor de un servicio se suele identificar con el término de suscriptor (consumen de forma anónima información sobre un determinado asunto o tema).

En este servicio se asocia cada mensaje con un determinado asunto o tema para que las aplicaciones que estén interesadas en dicho asunto o tema puedan suscribirse a los mensajes del asunto. Cuando en el asunto se produce algún tipo de publicación, inmediatamente un proceso publicará un mensaje anunciando a los suscriptores de la existencia del mismo. El encargado de distribuir los mensajes cuando se produce una publicación en el asunto a los suscriptores es el Middleware.



Recuerde

Middleware es un software que auxilia a otra aplicación para interactuar o comunicarse con otras aplicaciones, redes, software, hardware, sistemas operativos, etc.

Desarrollo de Aplicaciones Web Distribuidas. Pág. 150.

Las características principales que va a ofrecer este servicio son:

■ Un solo mensaje puede tener muchos y variados consumidores.

■ Los publicadores y suscriptores tienen una dependencia limitada en el tiempo.

Un cliente el cual se suscribe a un determinado tema solo podrá consumir los mensajes publicados después de haber obtenido su suscripción y además debe mantenerse suscrito para poder continuar consumiendo mensajes de dicho publicador.

Las API disponibles para poder implementar un servicio de Publicación/ Suscripción son las siguientes:

■ JMS. Java Message Service disponible en la web: http://www.oracle. com/technetwork/java/jms/index.html.

■ MSMQ. Microsoft Message Queuing disponible en la web: http://technet. microsoft.com/es-ES/windowsserver.

Desarrollo de Aplicaciones Web Distribuidas. Pág. 151.

El principal componente de un middleware orientado a objetos es el ORB (“ Object Request Broker ”), que se encarga de:

Identificar y localizar objetos. Para ello los objetos deben ser registrados en el ORB. En ese momento se les asigna un determinado identificador con el que se crea su primera referencia. Las referencias son empleadas posteriormente para realizarlas invocaciones, permitiendolocalizar en qu´e nodo y proceso reside el objeto asociado a la referencia.

Realizar invocaciones remotas sobre objetos. Se emplea el mecanismo ROI que es similar a una RPC. La diferencia principal reside en el componente que reemplaza al stub servidor, llamado esqueleto pues ahora debe gestionar una interfaz formada por m´ ultiples operaciones, por lo que debe ser capaz de servir las invocaciones recibidas sobre todo ese conjunto de operaciones, en lugar de una sola, como ocurr´ ıa en las RPC. Por su parte el stub cliente pasa a llamarse proxy [Sha86] y ofrece la misma interfaz que el objeto al querepresenta.Elestadodeunproxyest´aformadoporunareferenciaal objeto remoto. En los mensajes de petici´ on se debe incluir como informaci´ on adicional el identificador del m´ etodoqueseest´einvocando.Esonoresultaba necesario en una RPC.

Gestionar el ciclo de vida de los objetos (creaci´ on, registro, activaci´ on y eliminaci´ on de los objetos).

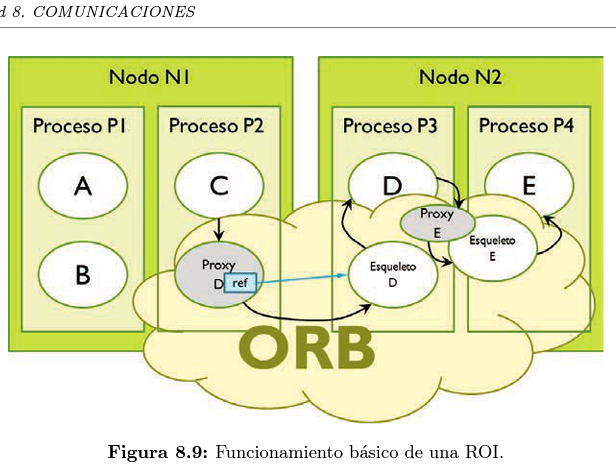
Concurrencia y sistemas distribuidos. Pág. 179.

Elesquemab´asicodefuncionamientoparaunainvocaci´ on remota es similar al descrito para RPC, y se muestra en la figura 8.9. En ella se ilustra el camino seguido por los mensajes de petici´ on en aquellas invocaciones que ya aparecieron anteriormente en la figura 8.8 y que deban ser gestionadas por un ORB. No se muestra el camino seguido por los mensajes de respuesta.

En el caso de la interacci´ on entre los objetos C y D, los pasos utilizados son:

El proceso cliente invoca el m´ etodo sobre el proxy local del objeto remoto (act´ ua como representante local del objeto D y posee la misma interfaz que D y una referencia a su esqueleto).

Concurrencia y sistemas distribuidos. Pág. 179.



El proxy empaqueta (marshall) los argumentos y, utilizando la referencia, llama al ORB para que ´ este gestione la invocaci´ on. ElORBconsiguequeelmensajedepetici´onlleguealesqueleto.

El esqueleto desempaqueta argumentos y reenv´ ıa la solicitud al m´ etodo invocado. Cuando dicho m´ etodo termina, el esqueleto empaqueta los resultados y devuelve el control al ORB.

El ORB hace llegar el mensaje de respuesta al proxy. El proxy desempaqueta los resultados y los devuelve al proceso cliente, completando as´ı lainvocaci´Conelloretornaelcontrolalc´odigodelobjetoC. on.

La invocaci´ on entre procesos en el mismo nodo (D,E) podr´ıa realizarse como una invocaci´ on remota, pero puede optimizarse usando memoria compartida. Se crea un par proxy-esqueleto especializado que act´ ua como puente entre los espacios de memoria de cliente y servidor, como se observa en la figura 8.9.

Concurrencia y sistemas distribuidos. Pág. 180.

# Referencias

Cabello, A. L. (2014). *UF1846: Desarrollo de Aplicaciones Web Distribuidas.* Antequera: ic Editorial.

Muñoz Escoí, F. D., Argente Villaplana, E., Espinosa Miguet, A. R., Galdámez Saiz, P., García Fornes, A., de Juan Marín, R., & Renda Roig, J. S. (2012). *CONCURRENCIA Y SISTEMAS DISTRIBUIDOS.* Valencia: UNIVERSITAT POLITECNICA DE VALENCIA.