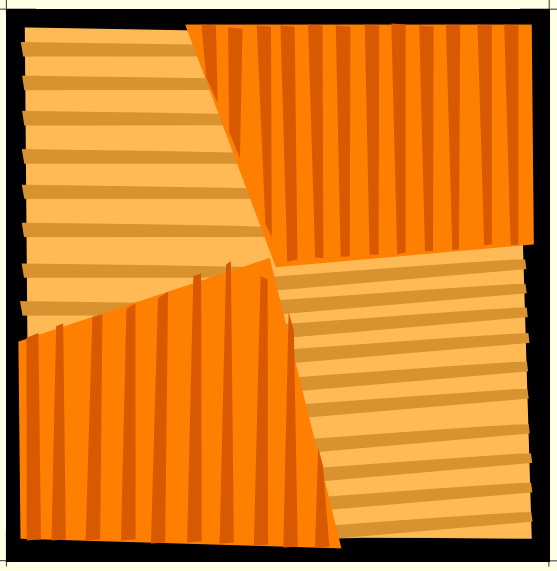
UNIVERSIDAD DE ALICANTE

Fundamentos de computación distribuida

LECCIÓN MAGISTRAL



VIRGILIO GILART IGLESIAS

SISTEMAS OPERATIVOS EN RED

ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES
TECNOLOGÍA INFORMÁTICA Y COMPUTACIÓN

Lección Magistral

FUNDAMENTOS DE COMPUTACIÓN DISTRIBUIDA

VIRGILIO GILART IGLESIAS

OCTUBRE DE 2007

UNIVERSIDAD DE ALICANTE

LECCIÓN MAGISTRAL

FUNDAMENTOS DE COMPUTACIÓN DISTRIBUIDA

PRESENTADA POR
VIRGILIO GILART IGLESIAS

ÁREA DE CONOCIMIENTO
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES
DEPARTAMENTO DE TECNOLOGÍA INFORMÁTICA Y COMPUTACIÓN

Contenido

Sección 1

Contexto

Introducción

La asignatura Sistemas Operativo en Red se divide en nueve temas. La unidad seleccionada para la lección magistral se ubica dentro del primer tema de la asignatura Fundamentos y Modelos.

programación	T
1. Fundamentos y modelos	6
2. Tecnologías Web y middleware	6
3. Sistemas de nombrado y localización	4,5
4. Tiempo y estados globales	4,5
5. Seguridad	6
6. Interacción y cooperación distribuida	6
7. Sistemas de archivos distribuidos	4,5
8. Memoria compartida distribuida	4,5
9. Tendencias	3

TABLA 1.1: PROGRAMA DE SOR.

Exactamente, es la segunda unidad y en ella se estudian los fundamentos de la computación distribuida. Se trata de una unidad introductoria y de carácter conceptual cuyos aspectos se tratan con más detalle en los temas posteriores.

programación
Unidad 1. Modelos de sistemas
Unidad 2. Fundamentos de computación distribuida
Unidad 3. Propiedades de los SOR
Unidad 4. Infraestructuras para los SOR

TABLA 1.2: CONTENIDOD DEL TEMA 1 SOR.

Objetivos

El objetivo principal de esta lección es familiarizar al alumno con los fundamentos de la computación distribuida que permita introducir a los conocimientos que se verán con mayor profundidad en temas posteriores.

A continuación se concretan los objetivos de esta lección atendiendo a los aspectos de adquisición de conocimientos y habilidades, y adopción de actitudes en el contexto de la asignatura. Aunque se han desglosado por cuestiones de claridad expositiva, estos objetivos tienen sentido en un contexto de consecución global.

Objetivos cognitivos

- Conocer y Comprender los fundamentos de la computación distribuida y su evolución.
- Comprender los modelos organizativos de los sistemas de computación distribuida.
- Conocer las propiedades y características de los diferentes modelos de comunicación distribuida.

Objetivos procedimentales

- Saber aplicar los diferentes paradigmas de computación distribuida
- Elegir los modelos estructurales más adecuados para el diseño de sistemas de computación distribuida
- Saber utilizar los principales mecanismos de comunicación distribuida para conformar sistemas distribuidos

Objetivos actitudinales

- Esquemas mentales para asimilar los avances de las TIC
- Adquirir capacidad para abstraer y generalizar los mecanismos de computación distribuida
- Desarrollar un espíritu crítico hacia el diseño de sistemas distribuidos

Contenido

Los contenidos que se ven en esta unidad son:

- Paradigmas de computación distribuida
  - Evolución de los modelos de computación distribuida
- Modelos arquitectónicos de sistemas distribuidos
  - C/S, P2P, MOM, SOA, agentes, colaborativas
- Mecanismos de comunicación distribuida
  - Recursos: IPC, sockets, RPC, RMI, ORB, Servicios Web

Bibliografía

- M.L. Liu, 2004. ***Computación Distribuida. Fundamentos y Aplicaciones.*** Person Education.

  Este libro se centra, principalmente, en los conceptos fundamentales de la intercomunicación entre procesos presentando la evolución de los diferentes modelos de computación distribuida y los diferentes niveles de abstracción que presenta cada uno. A lo largo del libro, se van presentando ejemplos reales que se desarrollan con los diferentes paradigmas de computación distribuida y se describen las ventajas y desventajas de cada uno de ellos.

El contenido de esta unidad se centra en los temas 2, 3, 4, 5, 7 y 12.

- G. Coulouris, J. Dollimore, T. Kindberg, 2001. ***Sistemas Distribuidos. Conceptos y Diseño***, Addison Wesley, 3ª edición.

  Los contenidos de este libro abarcan gran parte del temario, incorpora material actualizado sobre los sistemas en red y distribuidos, profundizando especialmente en lo que tiene que ver con los modelos de objetos distribuidos, la seguridad en red integrada con otras tecnologías, paradigmas de diseño orientados al modelo objetual y los agentes móviles, etc. Incluye capítulos específicos sobre Corba y Mach configurándose como la principal referencia para seguir adecuadamente los contenidos de la asignatura.

En la última edición se ha incorporado, sobre todo, material de tipo tecnológico y aplicado. Así, se ha realizado una profunda revisión en la que se han incorporado aspectos referentes a objetos distribuidos, integración de aplicaciones en Internet y Web, computación móvil y sistemas multimedia. Por el contrario, ha dejado de asumirse UNIX como único entorno posible para estos sistemas.

El contenido de esta unidad se centra en los temas 4 y 5.

Sección 2

Paradigmas de computación distribuida

En esta sección se describe, de manera conceptual, la evolución que ha seguido la computación, desde los modelos basados en sistema monolíticos hasta los sistemas distribuidos actuales, centrándose en la resolución de la problemática asociada a la gestión de la comunicación de procesos distribuidos a través de la red de comunicaciones.

Además, se presenta los diferentes enfoques de sistemas distribuidos en función de la ubicación de los mecanismos comunicación.

Evolución

Antes de comenzar con los modelos distribuidos se va a realizar un breve recorrido de la evolución de los sistemas de computación hasta llegar al estado actual.

Inicialmente, las aplicaciones se centraban en un diseño monolítico donde toda la lógica de la aplicación formaba parte de un único módulo binario. Este tipo de aplicaciones conlleva una serie de problemas. A medida que una aplicación de tipo monolítico crece, la complejidad de gestión y mantenimiento aumenta.

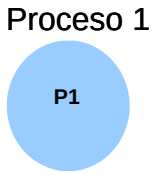


FIGURA 2.1: APLICACIÓN MONOLÍTICA

Debido a esta problemática, surge la necesidad de dividir este tipo de aplicaciones en aplicaciones más pequeñas y modulares, que encapsulen parte de la lógica que inicialmente se ubicaba en una única aplicación. Este enfoque, además de facilitar el mantenimiento de las aplicaciones, facilita la modificación de pequeños módulos de la aplicación sin que para ello sea necesario modificar la aplicación original.

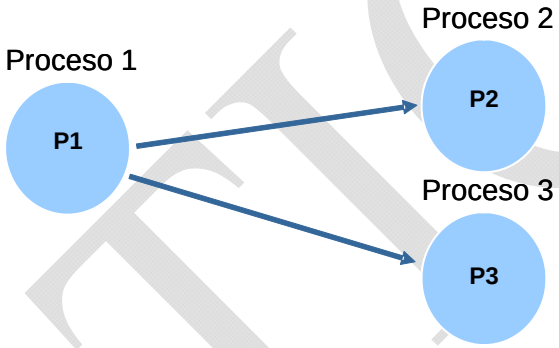


FIGURA 2.2: APLICACIÓN MODULAR

La división de las aplicaciones en módulos implica la posibilidad de que esos módulos puedan ser utilizados por diferentes aplicaciones mejorando la productividad en el desarrollo de aplicaciones gracias a la reusabilidad.

Hasta este momento, la evolución descrita se centra en procesos que se encuentran ubicados en el mismo procesador.

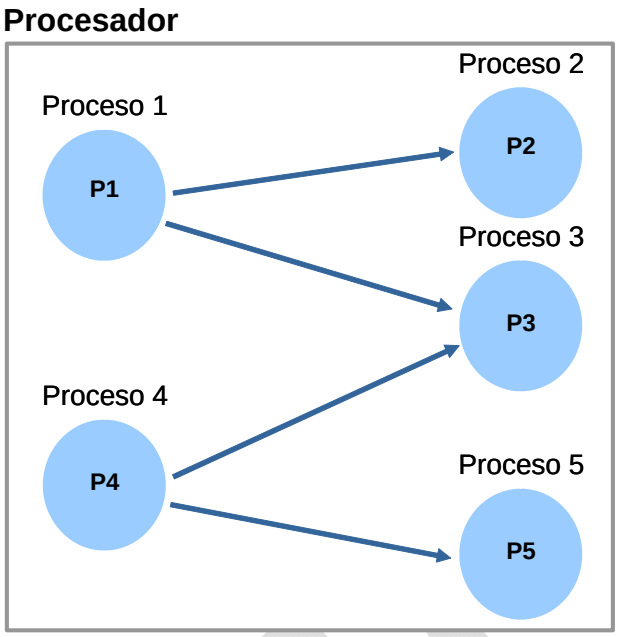


FIGURA 2.3: REUTILIZACIÓN DE CÓDIGO EN UN ÚNICO PROCESADOR

Con la aparición de los computadores personales y las arquitecturas multiprocesadores surge la posibilidad de ubicar los diferentes módulos en distintos procesadores. De esta forma se incrementan las ventajas presentadas anteriormente, se obtiene una mejora de rendimiento, posibilidad de tolerancia a fallos y continuidad en el negocio y sobre todo escalabilidad. Las arquitecturas multiprocesadores han sido tratadas en la asignatura Arquitectura e Ingeniería del Computador. La asignatura se centra en la gestión de los mecanismos de computación distribuida a través de las redes de comunicaciones, tanto de área amplia como locales. La comunicación entre los diferentes procesos distribuidos no es sencilla debido a la aparición de este nuevo recurso que se debe gestionar, la red de computadores. Por ese motivo se estudiarán los modelos, técnicas y arquitecturas que hagan posible la interoperación de los procesos en un entorno distribuido de manera transparente.

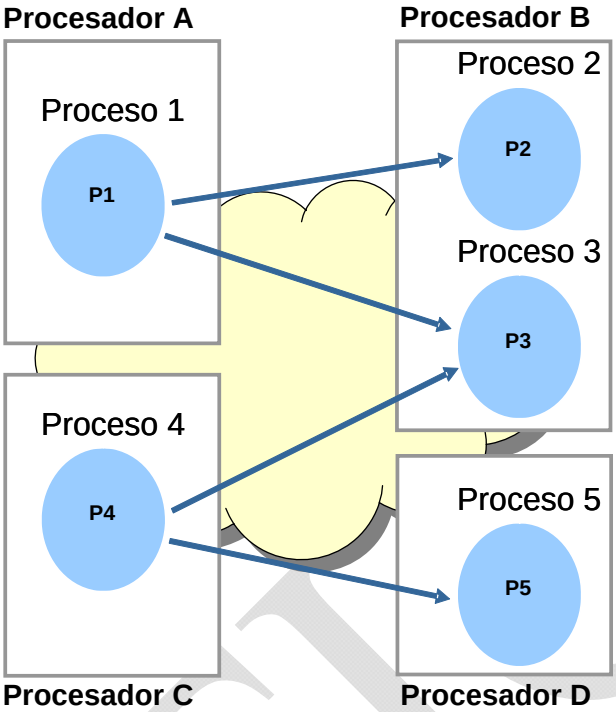


FIGURA 2.4: ESCENARIO DISTRIBUIDO

En un principio, toda la lógica necesaria para establecer la comunicación en un entorno distribuido se incluía en la propia aplicación, lo que suponía un gran esfuerzo por parte de los programadores.

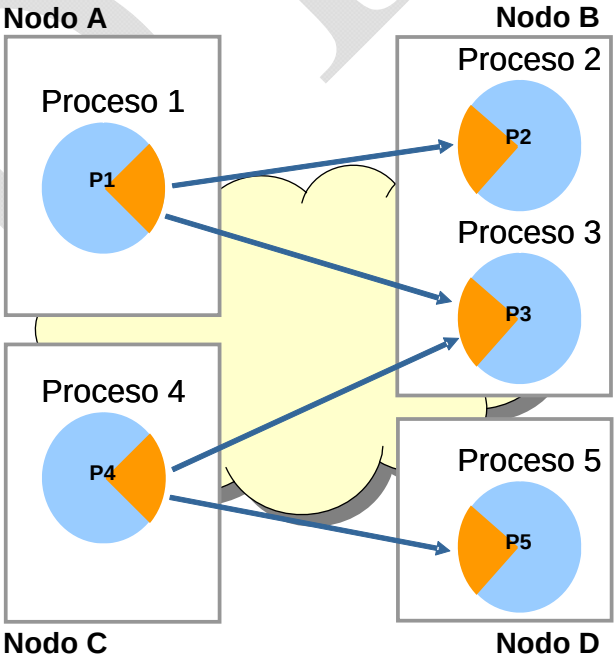


FIGURA 2.5: MODELO DE APLICACIÓN CENTRADO EN LA COMUNICACIÓN

Poco a poco esta lógica se fue extrayendo y ubicando en módulos comunes a las aplicaciones ubicadas en el mismo procesador.

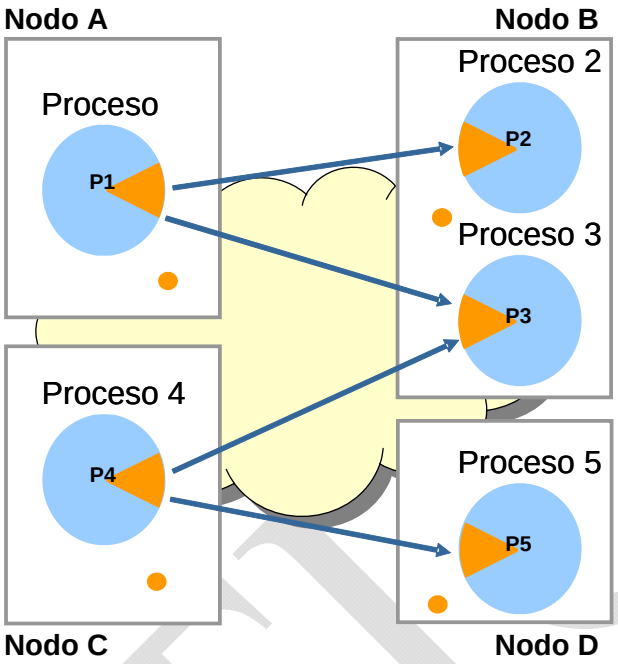


FIGURA 2.6: EVOLUCIÓN DEL PROCESO.

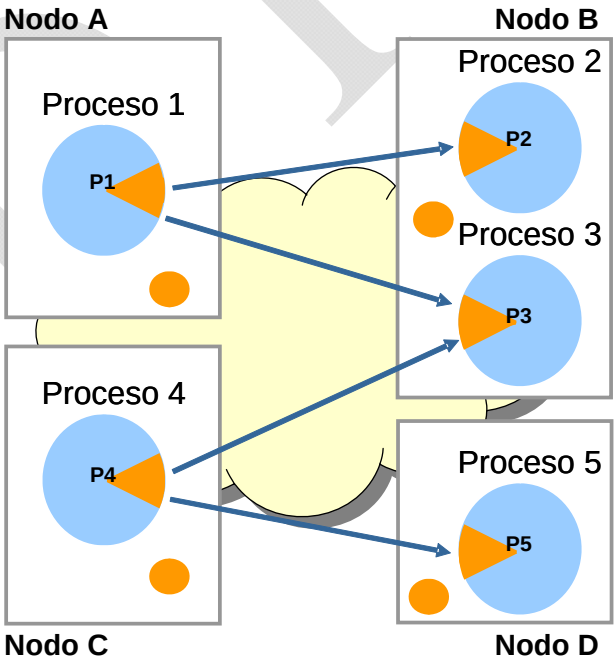


FIGURA 2.7 : EVOLUCIÓN DEL PROCESO II.

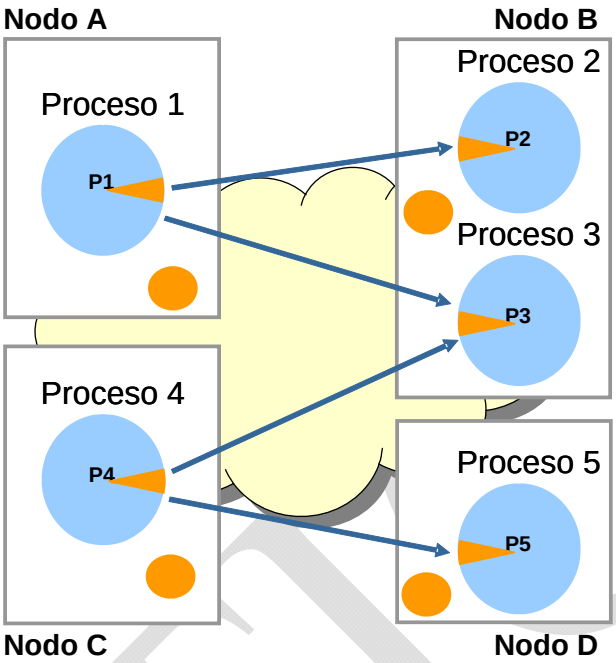


FIGURA 2.8: EVOLUCIÓN DEL PROCESO III

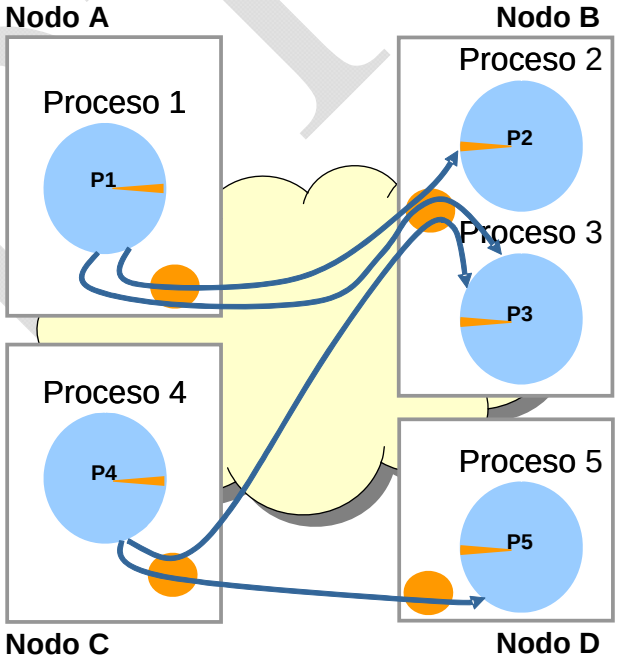


FIGURA 2.9 :EXTRACCIÓN FINAL DE LA LÓGICA DE COMUNICACIÓN.

Finalmente, los mecanismos que permiten la comunicación entre aplicaciones distribuidas es ubicada en un módulo o capa común a los elementos conectados a la red de comunicaciones. Este módulo facilita el desarrollo de este tipo de aplicaciones situando el escenario al mismo nivel que en el caso de un único procesador.

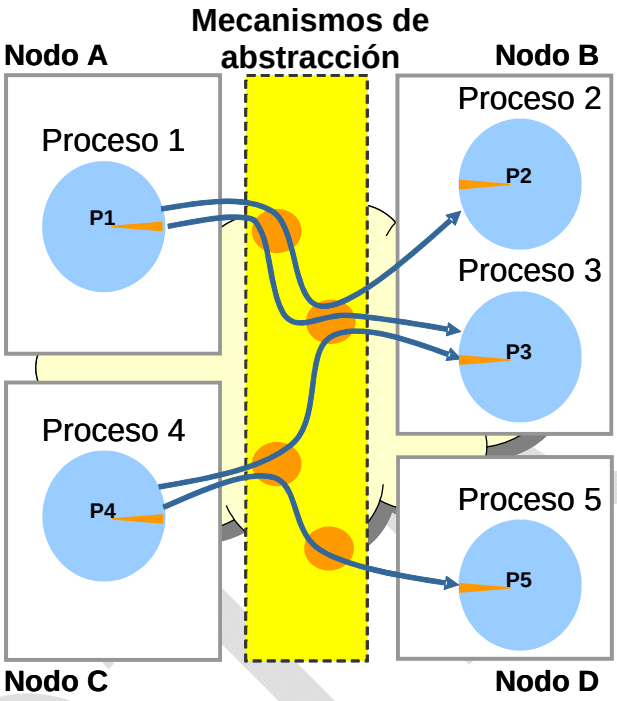


FIGURA 2.10: MÓDULO COMÚN DE MECANISMOS DE COMPUTACIÓN DISTRIBIDA.

Enfoques de los sistemas operativos para los sistemas distribuidos

En esta sección se plantea las consecuencias y la repercusión que conlleva la ubicación de los mecanismos de abstracción en los diferentes enfoques de sistemas distribuidos descritos en la unidad anterior.

Sistemas Operativos en Red

En los sistemas operativos en red los mecanismos de abstracción de comunicaciones se ubican generalmente en el propio sistema operativo, aunque en ocasiones se pueden presentar como librerías de usuario sobre el sistema operativo, pero dependientes del mismo.

Debido a que los sistemas operativos en red se caracterizan por su elevada heterogeneidad, tanto a nivel de hardware como software, dichos mecanismos se implementarán de forma diferente en cada sistema operativo.

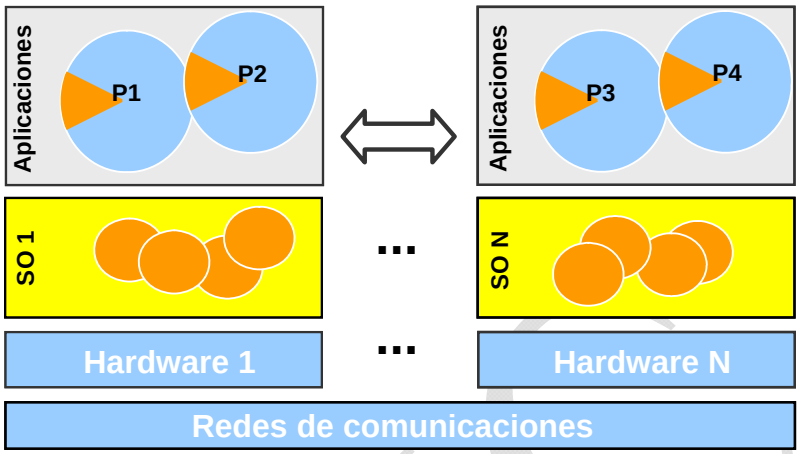


FIGURA 2.11: UBICACIÓN EN LOS SOR.

Como ejemplos de este tipo de sistemas se pueden mencionar: Linux, Windows o Novell NetWare.

A continuación se detallan algunas ventajas de los sistemas operativos en red con respecto a dichos mecanismos:

- Son más flexibles puesto que no existe una fuerte dependencia a nivel de software entre los nodos que conforman la red.
- Son adecuados para entornos donde la gestión de la red se encuentra dispersa y es incontrolable (por ejemplo Internet).
- Al tratarse de sistemas operativos convencionales, las técnicas usadas para su diseño poseen un alto grado de madurez.

Como desventajas de los sistemas operativos en red, se podría destacar:

- No existen transparencia en la gestión de los recursos distribuidos. El usuario ve un conjunto de máquinas independientes.
- Exige un mayor esfuerzo por parte de los desarrolladores en la implementación de aplicaciones distribuidas que trabajen conjuntamente (protocolos comunes).

Sistemas Operativos Distribuidos

En este tipo de sistemas distribuidos los mecanismos de abstracción de comunicaciones se ubican en el sistema operativo, al igual que en el enfoque anterior,

con la diferencia de que estos mecanismos son comunes a todos los nodos que conforman el sistema puesto que comparten un único sistema operativo.

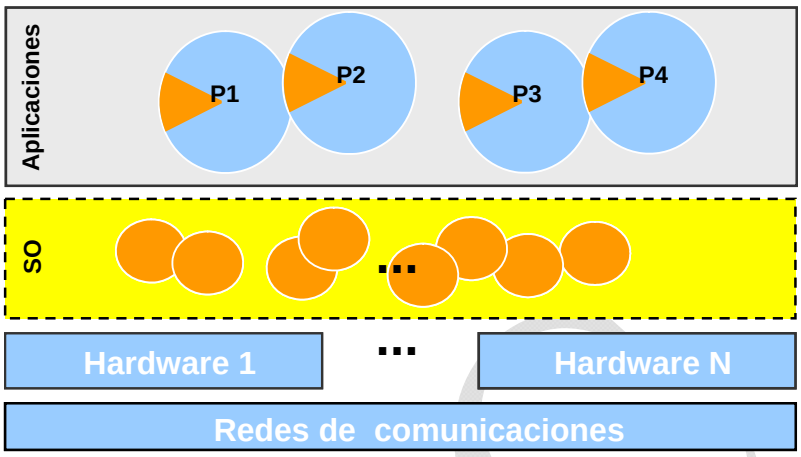


FIGURA 2.12: UBICACIÓN EN LOS SOD

Ejemplos de este tipo de sistemas son: Amoeba, Mach o Chorus.

Las principales ventajas de estos sistemas son:

- Se comporta como un único sistema ofreciendo una visión única a la hora de compartir y gestionar recursos.
- La integración entre los nodos que conforman el sistema es transparente.
- La escalabilidad del sistema se realiza de manera transparente.

Como desventajas se puede señalar entre otras:

- Las técnicas de diseño de sistemas operativos distribuidos son más complejas que las de los sistemas operativos convencionales.
- El sistema operativo distribuido debe ser común a todos los nodos que conforman la red, lo que resulta difícil de aplicar en entorno de red de gran tamaño y con gestión distribuida.
- Necesidad de un entorno de red de alta velocidad.

Middleware

El enfoque del middleware, sitúa los mecanismos de abstracción de comunicaciones en una capa situada por encima del sistema operativo convencional pero dicha capa es común a los elementos que conforman la red.

Podríamos decir, que el enfoque propuesto por los sistemas operativos distribuidos, conceptualmente, es la mejor propuesta para la gestión de recursos a través de la red.

Aunque debido a la heterogeneidad de los entornos de red existentes junto con los intereses económicos asociados a las empresas del sector hacen que esta propuesta hoy en día sea poco viable. Por lo tanto, es necesario el planteamiento de enfoques que se acerquen a los objetivos de los sistemas operativos distribuidos pero sobre enfoques más adecuados a los entorno de redes actuales, como son los sistemas operativos en red.

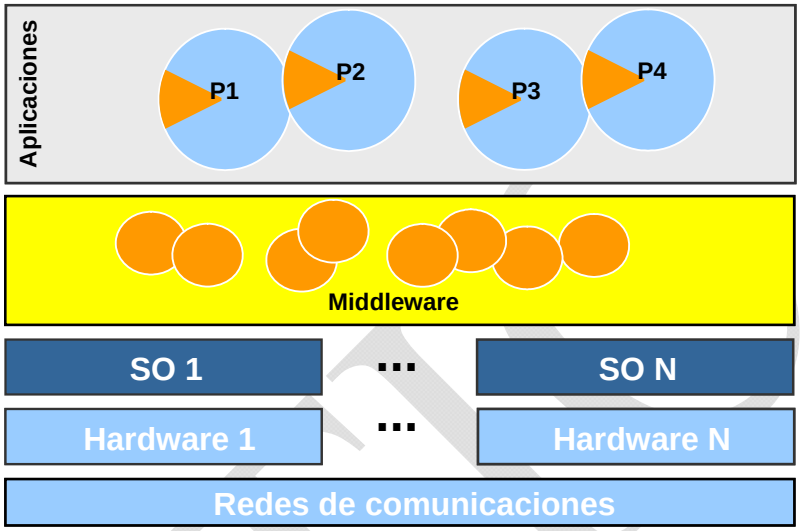


FIGURA 2.13: UBICACIÓN EN EL MIDDLEWARE

Como ejemplos de este enfoque podemos destacar: CORBA, J2EE o .NET Framework.

Este enfoque pretende unir las ventajas de los sistemas operativos en red y de los sistemas operativos distribuidos:

- Flexibilidad
- Transparencia
- Integración
- Madurez
- Escalabilidad

No obstante, existen algunas desventajas como:

- Heterogeneidad de plataformas
- Necesidad de estandarización

Sección 3

Modelos arquitectónicos

Introducción

En esta sección se presentan diferentes modelos arquitectónicos de computación distribuida. El enfoque se centra en describir como se organizan los elementos que componen cada uno de los modelos, así como presentar sus principales características.

Modelo Cliente/Servidor

El modelo Cliente/Servidor se establece sobre el paradigma de paso de mensajes. En este modelo los procesos pueden adquirir uno de los siguientes dos roles:

- El proceso Servidor, el cual encapsula el acceso a ciertos recursos mostrándose como un servicio de red. Este proceso se encuentra esperando de forma pasiva la llegada de peticiones de los procesos clientes y gestiona el acceso a dichos recursos en su nombre.
- El proceso cliente, el cual solicita el acceso a un recurso mediante peticiones al proceso Servidor.

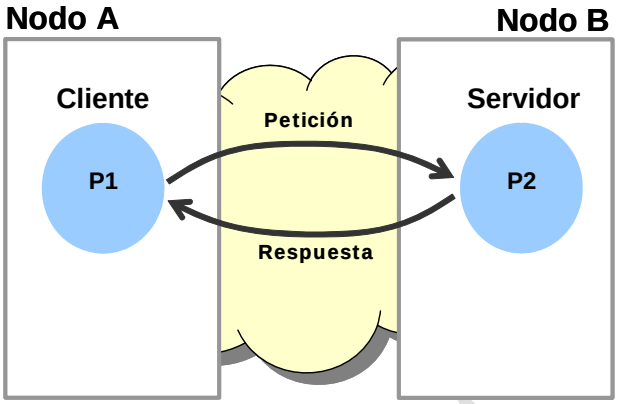


FIGURA 3.1: MODELO CLIENTE/SERVIDOR

Algunos requisitos que se deben tener en cuenta en el diseño de este tipo de modelos son:

- Son necesarios mecanismos de concurrencia que permitan al Servidor gestionar el acceso de múltiples clientes al mismo tiempo.
- En algunas implementaciones puede ser necesario el mantenimiento de forma continua de los accesos que un determinado cliente realiza al servidor, mantenimiento de la sesión.
- El número de clientes accediendo al mismo tiempo puede afectar al rendimiento del sistema, por lo que serán necesarios mecanismos que permitan escalar dicho sistema.

La mayoría de los protocolos de Internet conocidos hoy en día se sustentan sobre este modelo (HTTP, DNS, FTP, SMTP, etc.).

El esfuerzo que requiere el diseño de las aplicaciones basadas en el modelo Cliente/Servidor es elevado

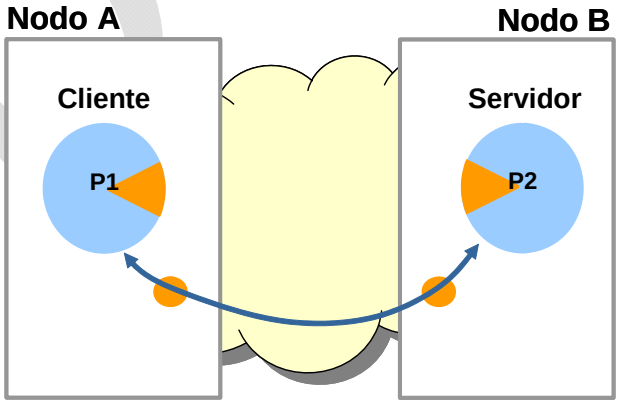


FIGURA 3.2: MODELO CLIENTE/SERVIDOR

Modelo Peer-to-Peer

El modelo P2P se presenta como una evolución del modelo anterior. En este caso todos los procesos participantes se comportan de una forma semejante, desempeñando a la vez el rol de cliente y de servidor.

- Como cliente puede enviar de peticiones y recibir respuestas.
- Como servidor puede recibir solicitudes, procesar de solicitudes, enviar respuestas y propagar solicitudes.

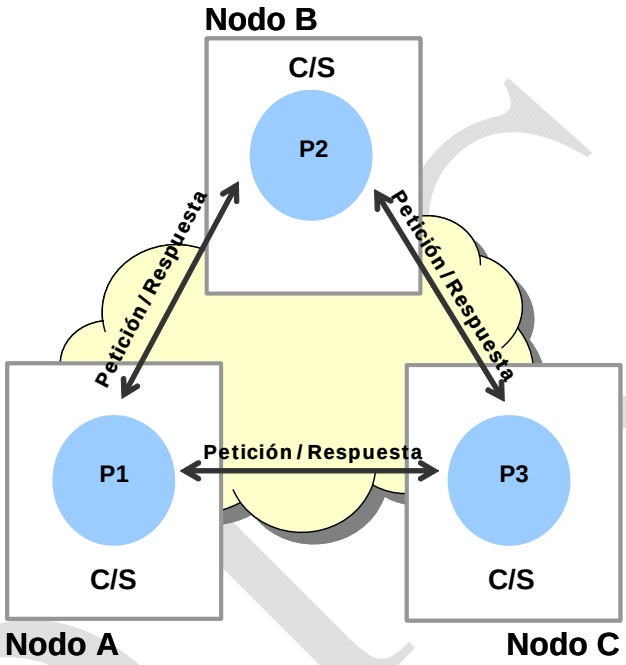


FIGURA 3.3: MODELO P2P

El modelo P2P es apropiado para aplicaciones de tipo: mensajería instantánea, transferencia de ficheros, video conferencia y trabajo colaborativo.
Como ejemplos de aplicaciones P2P podemos nombrar:

- Napster es un servicio de distribución de archivos de música que implementa una arquitectura P2P centralizada. Todas las transacciones entre pares que conforman la red se realiza a través de un nodo central.
- Gnutella es un protocolo que permite la distribución de archivos de manera distribuida implementando un sistema P2P puro (sistema descentralizado). Todos los nodos que conforman la red tienen la misma importancia, peso y funcionalidad.
- BitTorrent es un protocolo diseñado para la distribución de archivos que implementa un modelo P2P de tipo híbrido o mixto. A diferencia de otros sistemas del mismo tipo, Bittorrent busca como objetivo la distribución eficiente de archivos. Existe un servidor central que simplemente gestiona la comunicación entre los pares de manera que sea eficiente, pero si éste fallase, la comunicación se mantiene entre los pares. El servidor central no

actúa como distribuidor de información ni almacena ningún tipo de datos. Simplemente distribuye la conexión en función de los pares que permitan una mejor distribución.

Existen herramientas y plataformas que facilitan el desarrollo de aplicaciones basadas en el modelo de pares como JXTA. Se trata de un framework desarrollado por SUN, que se compone de un conjunto de protocolos basados en XML que permite que los participantes conectados intercambien mensajes entre sí. Este tipo de herramientas facilitan el desarrollo de aplicaciones P2P, disminuyendo el esfuerzo que debe realizar el programador.

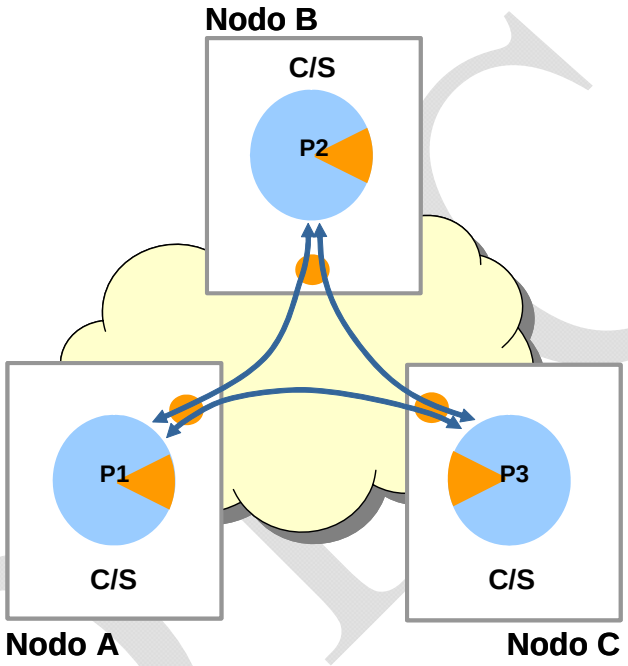


FIGURA 3.4: MODELO P2P

Modelo de sistema de mensajes

El modelo de sistema de mensajes es una elaboración del sistema de paso de mensajes convencional. La comunicación entre el emisor y el receptor del mensaje se produce de una forma completamente desacoplada. Este modelo es utilizado en intercambio de mensajes de forma asíncrona, donde el emisor no necesita una respuesta inmediata del receptor tras enviar el mensaje y puede continuar su procesamiento.

Los elementos que participan en este modelo son:

- El proceso emisor que es responsable de la emisión de un mensaje.
- El proceso intermediario que se encarga de almacenar los mensajes enviados por el emisor. El agente intermediario, generalmente es un middleware que incluye un conjunto de servicios que permiten el tratamiento de los mensajes. Este middleware puede poseer las siguientes

características: gestión de prioridades de mensajes, temporizadores para la gestión de mensajes, gestión de formatos de mensajes, gestión de seguridad, gestión de la persistencia de los mensajes.

- El proceso consumidor que se conecta al proceso intermediario para obtener los mensajes y procesarlos.

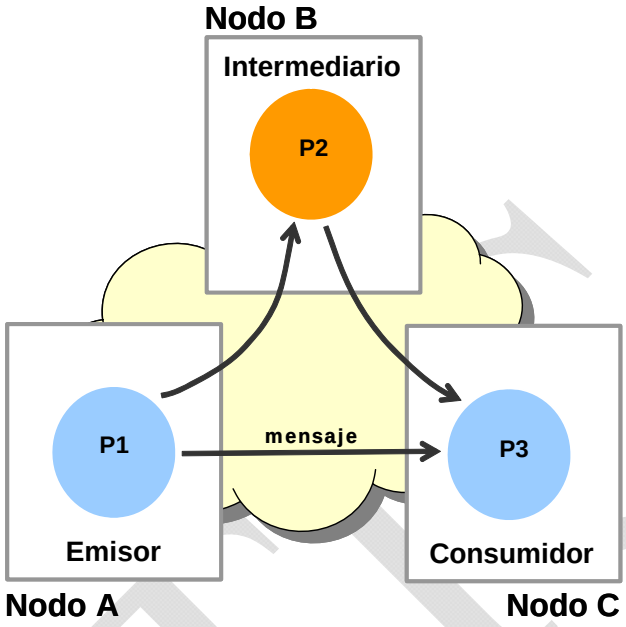


FIGURA 3.5: MODELO DE SERVICIO DE MENSAJERÍA

Es un modelo adecuado cuando se quiere establecer un único punto de entrada a aplicaciones y sistemas donde se establece una comunicación asíncrona entre las partes (por ejemplo, solicitudes de reservas o de ofertas, etc.).

Como ejemplos de tecnologías basadas en este modelo se pueden enumerara: JMS de SUN, MSMQ de Microsoft o MQSeries de IBM.

Existen dos variantes de este modelo.

- El modelo punto a punto el cual se caracteriza por que cada mensaje enviado por el emisor únicamente será procesado por un proceso consumidor. Una el agente consumidor obtiene el mensaje y lo procesa, el mensaje es borrado del agente intermediario.

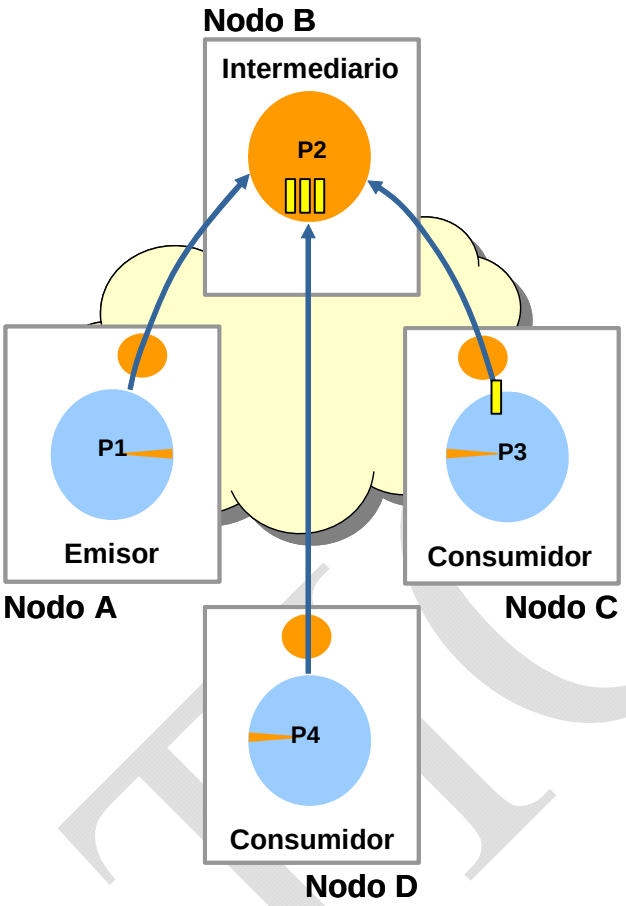


FIGURA 3.6: MODELO DE SERVICIO DE MENSAJERÍA: PUNTO A PUNTO.

- El modelo publicación/suscripción se caracteriza porque un mensaje publicado por el emisor (enviado al agente intermediario) será procesado por todos los agentes consumidores que se hayan suscrito al proceso intermediario.

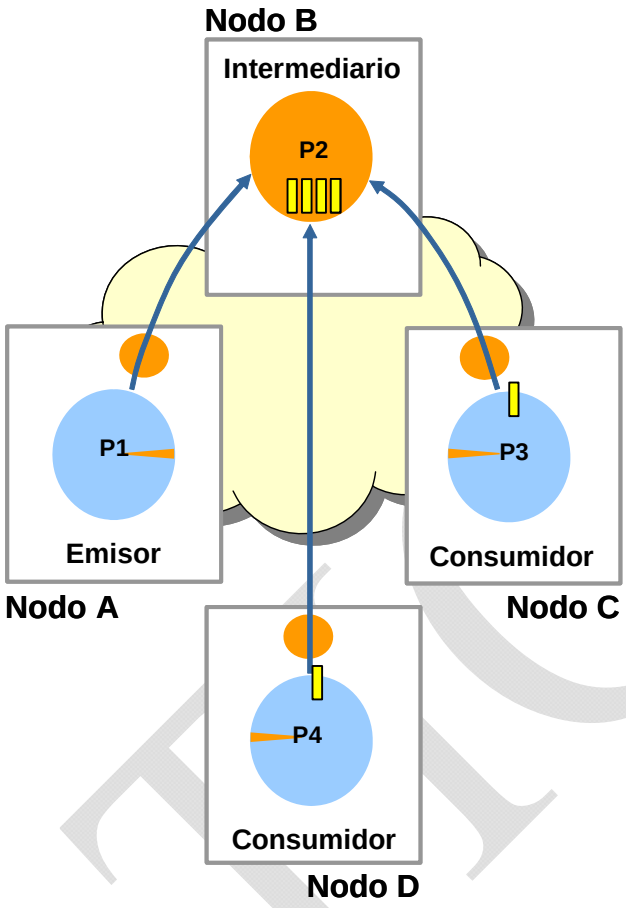


FIGURA 3.7: MODELO DE SERVICIO DE MENSAJERÍA: PUBLICACIÓN- SUSCRIPCIÓN

Modelo de Arquitectura Orientada a Servicios

El modelo de arquitecturas orientadas a servicios o SOA presenta la gestión de recursos de un sistema distribuido como servicios de red que son publicados y descubiertos por los procesos clientes para su consumo (aplicaciones, ficheros, bases de datos, servidores, sistemas de información, dispositivos). Un aspecto clave en este modelo es la aparición de un agente intermediario (servicio de directorio) que posibilita la localización de los servicios disponibles.
Los elementos que componen este modelo son:

- Proceso proveedor del servicio que expone el acceso a un recurso como un servicio de red. Este proceso se encarga de publicar la información en el servicio de directorio que permita localizar el servicio y acceder a él.
- Proceso consumidor del servicio que accede a un recurso a través de un servicio de red. El proceso consumidor se conecta al servicio de directorio para localizar el servicio que ofrezca la funcionalidad que requiere y obtiene la información necesaria, descubrimiento, para acceder a un servicio concreto.

- El servicio de directorio almacena la información necesaria sobre los servicios para que un proceso consumidor puede localizar y descubrir los servicios y consumirlos. Este elemento provee una característica clave del modelo, transparencia de localización. Permite establecer un modelo completamente desacoplado entre el proceso que requiere un servicio y el proceso que lo provee. El proceso consumidor solicita una funcionalidad y se puede conectar a cualquiera de los servicios que la ofrezcan, sin conocer a priori ningún dato del proceso que provee dicho servicio.

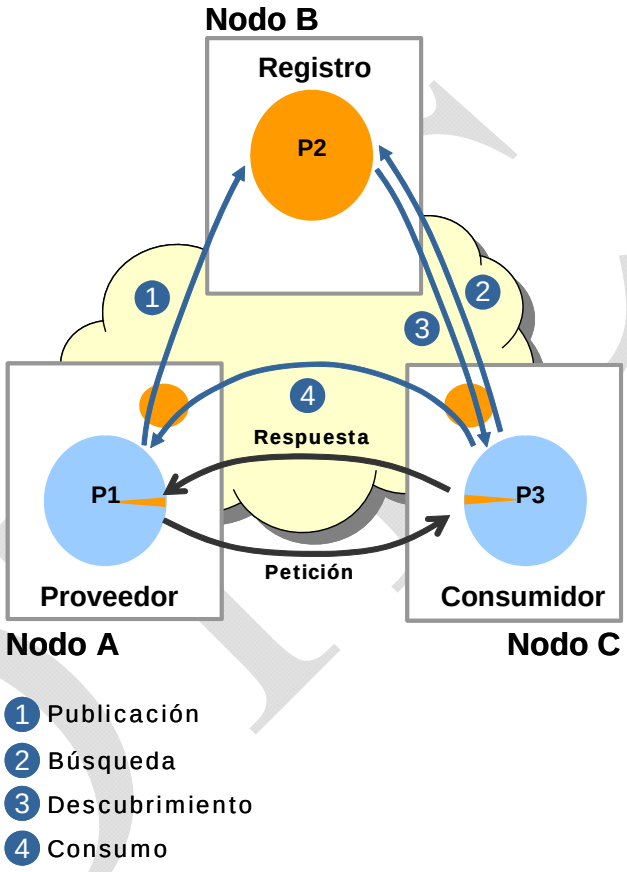


FIGURA 3.8: MODELO DE ARQUITECTURA ORIENTADA A SERVICIOS

Dos características claves de este modelo son la interoperatividad y la reutilización. Las tecnologías basadas en este modelo permiten sin gran esfuerzo diseñar aplicaciones complejas a partir de la composición de servicios heterogéneos.

Ejemplos de tecnologías basadas en el modelo SOA son:

- UPNP
- JINI
- Servicios Web, aunque esta última no ha surgido como una tecnología a partir del modelo SOA, se ha comprobado que se acopla perfectamente a

dicho modelo y se está convirtiendo en el estándar de facto para el desarrollo de este tipo de aplicaciones.

Modelo de Cluster/Grid

Estos modelos tienen como objetivo aprovechar los recursos de sistemas independientes conectados a la red para obtener una infraestructura con mayor capacidad de procesamiento y almacenamiento.
De forma muy conceptual se podría decir que estos modelos me permiten a partir de un conjunto de computadores independientes obtener un super computador. Como ejemplo, de manera muy general, se podría decir que:

- 1000 computadores 1GB RAM → 1 computador 1000GB RAM
- 1000 computadores 40GB HD → 1 computador 40000GB HD

Aunque el objetivo inicial para los dos modelos es similar existen ciertas diferencias entre el Grid y el Cluster.

Cluster

Un Cluster se caracteriza porque los elementos que lo componen se encuentran fuertemente acoplados a nivel de hardware o de software.
Los nodos que forman el cluster son interconectados a través de redes de alta velocidad que permita una comunicación eficiente.
Por esta razón el modelo de cluster, generalmente, suele aplicarse a un problema concreto y en un entorno dedicado.
En la mayoría de los casos, los cluster presentan las siguientes características:

- alta disponibilidad
- balanceo de carga
- escalabilidad
- alto rendimiento

En la actualidad las técnicas de cluster son aplicadas en servidores Web y de aplicaciones, sistemas de información y para resolución de problemas con necesidades de supercómputo.
Otra característica importante de los cluster, a diferencia de los Grids, es que existe una pérdida de independencia en cuanto al procesador. Esto significa que cuando un proceso es enviado a uno de los nodos del cluster consume procesador, y puede ralentizar la ejecución de otras aplicaciones en ese mismo nodo.En un cluster existe un elemento que se encarga de la gestión de los recursos del sistema global y de repartir la carga entre los diferentes componentes. De este modo se muestra una visión de los recursos como uno único.

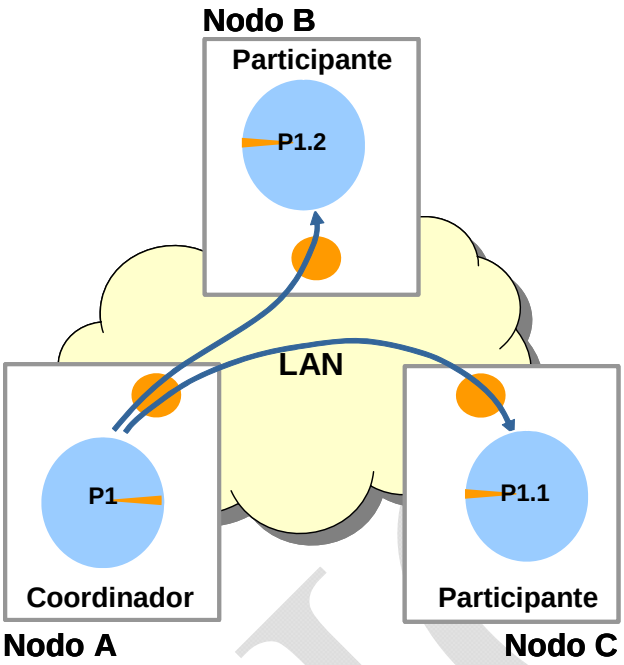


FIGURA 3.10: MODELO DE CLUSTER

Como ejemplos de implementaciones de cluster orientados a sistemas de alto rendimiento podemos nombrar MOSIX, OpenMosix, Heartbeat, Beowulf.

Grid

El modelo de Grid se caracteriza porque los elementos que lo componen son totalmente heterogéneos conectados a través de redes de área amplia completamente distribuidas como Internet. En la mayoría de los casos, los recursos (procesamiento y almacenamiento) son cedidos por usuarios particulares para lograr el objetivo del sistema.
Estas propiedades hacen necesarios la existencia de mecanismos que garanticen la seguridad de acceso y transmisión de información entre otros.
Existe un completo desacoplamiento entre los nodos que conforman el Grid.
Una característica importante del Grid, a diferencia del cluster, es que un computador no suele perder la independencia de procesamiento, ya que el Grid consume recursos aprovechando los tiempos muertos del sistema. Cada nodo perteneciente al Grid es responsable de la gestión de sus recursos, obteniendo una gestión y planificación completamente distribuida. A diferencia del Cluster no se busca ofrecer una visión única del sistema.

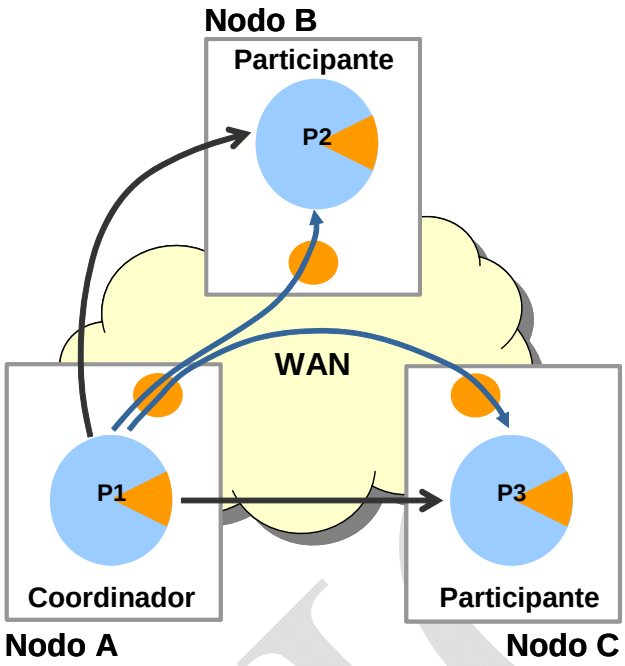


FIGURA 3.11: MODELO DE GRID

Existen propuestas, como OGSA (Open Grid Service Architecture), para lograr una arquitectura estándar de la tecnología Grid que normalice los servicios propios de este modelo. Esta arquitectura se sustenta sobre el modelo SOA, visto anteriormente, usando tecnología de servicios Web como base para lograr una mayor interoperación en la comunicación de los componentes del Grid.

Un ejemplo de plataforma que facilite el desarrollo de aplicaciones tipo Grid es Globus Toolkit 4.0. Se trata de un sistema de código abierto que implementa las especificaciones definidas por el modelo OGSA.

El proyecto Seti es un ejemplo de aplicación Grid desarrollada con el fin de analizar señales procedentes del espacio exterior en busca de vida inteligente.

Sección 4

Mecanismos de Comunicación

Fundamentos de comunicación

En esta sección se describen algunas de las principales características y requisitos para establecer la comunicación entre procesos distribuidos.
Además, se presentan los principales mecanismos de comunicación distribuida describiendo sus principales propiedades.
Por último se realiza un breve resumen de las consideraciones que se deben tomar a la hora de utilizar los diferentes mecanismos de comunicación y modelos arquitectónicos distribuidos.

Mecanismos IPC

El mecanismo más simple de comunicación consiste en el intercambio de información entre dos procesos, un emisor y un receptor, a través de la red de comunicaciones mediante un protocolo que establezca las reglas de la comunicación.

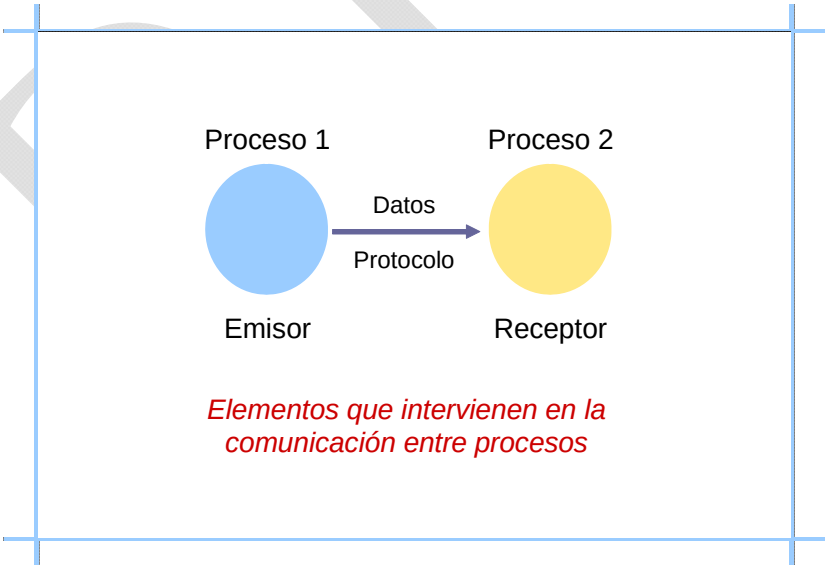


FIGURA 4.1:ELEMENTOS DE LA COMUNICACIÓN.

Al igual que un sistema operativo convencional ofrece una serie de mecanismos que permiten la comunicación entre procesos locales, como son los mecanismos

de colas de mensajes, semáforos y memoria compartida, hoy en día se proporcionan mecanismos de comunicación de procesos de más alto nivel que permiten la interoperación entre procesos distribuidos con un menor esfuerzo de desarrollo. Estos mecanismos pueden ubicarse dentro o por encima del sistema operativo como librerías.

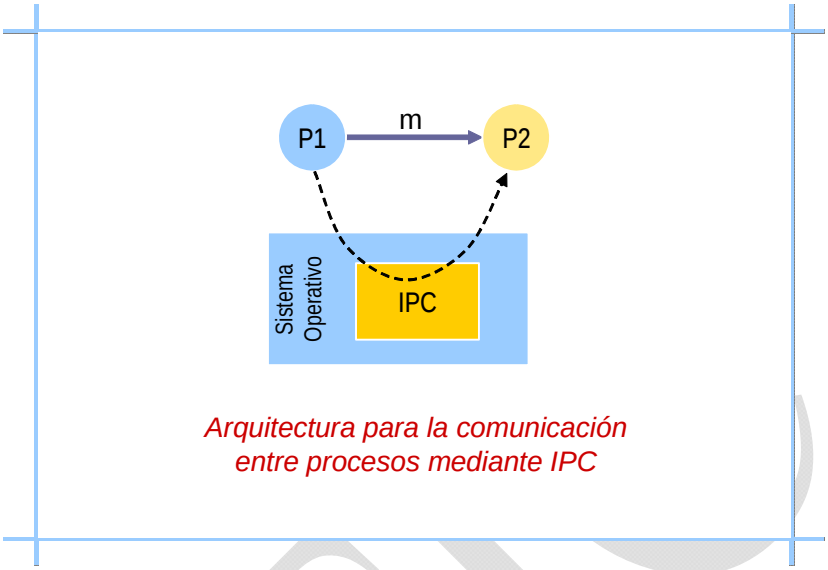


FIGURA 4.2: MECANISMOS IPC.

La comunicación se puede realizar desde un único proceso emisor a un único proceso receptor. Este modelo de comunicación se conoce como comunicación por unidifusión. Si el proceso de comunicación se establece desde un proceso emisor a varios procesos receptores, se conoce como comunicación por multidifusión.

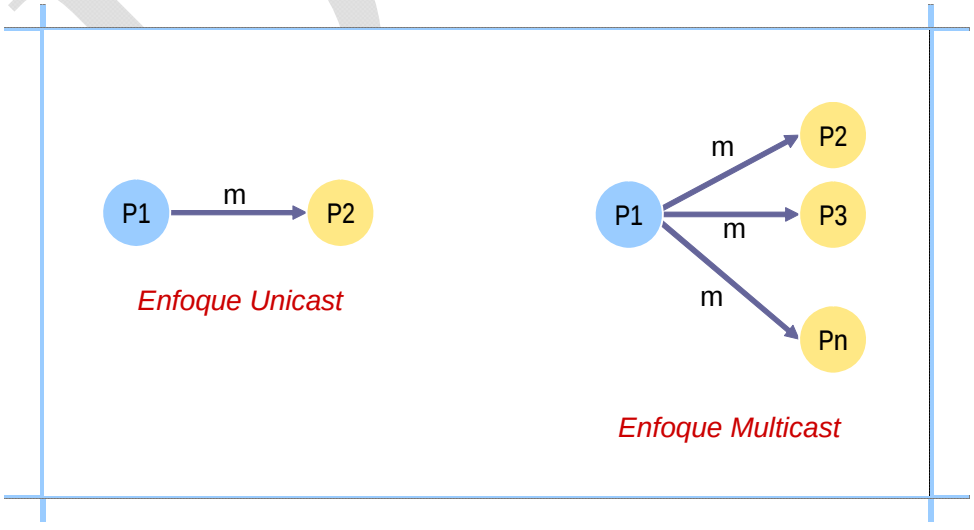


FIGURA 4.3: MODELOS DE COMUNICACIÓN.

Los mecanismos IPC deben definir interfaces que permitan establecer la comunicación entre procesos distribuidos. Las primitivas incluidas en los mecanismos IPC pueden variar el rango de complejidad en función de la abstracción que ofrezcan.
Un ejemplo de interfaz de programación que proporciona el mínimo nivel de abstracción para lograr la comunicación entre procesos puede contener las siguientes primitivas:

- Enviar
- Recibir
- Conectar (solicitar-conexión/aceptar-conexión)
- Desconectar

Existen ciertos aspectos en la comunicación de procesos que se deben tener en cuenta en el diseño de interfaces de programación para la comunicación de procesos.

- Sincronización. Uno de los problemas que se presenta en la comunicación de procesos es que, al ser independientes, cada uno de ellos desconoce el estado en el cual se encuentra el resto de procesos con los que puede establecer una comunicación. Para poder sincronizar los eventos que se producen en la comunicación entre procesos, una técnica sencilla es el uso de operaciones bloqueantes. Cuando un proceso invoca una operación su procesamiento queda suspendido hasta que la operación invocada haya finalizado. Las operaciones bloqueantes se denominan a menudo operaciones síncronas. Cuando un proceso invoca una operación no bloqueante, de forma que no es necesario que esta termine para que el proceso siga ejecutándose se denominan operaciones asíncronas.
- Las operaciones bloqueantes que facilitan la sincronización pueden llevar a un proceso a un estado de bloqueo permanente o temporalmente no aceptable.
    - Una solución es utilizar primitivas de comunicación entre procesos que introduzcan temporizadores con el objetivo de evitar bloqueos indefinidos o de tiempos desmesurados.
    - Otra solución consiste en el uso de procesos hijos o hilos de ejecución que se encarguen de la comunicación, permitiendo que el proceso padre o el hilo principal continué su ejecución.
- Como causa de diseños de protocolos erróneos, en la comunicación de procesos se pueden producir Interbloqueos (bloqueos indefinidos). Por ejemplo dos procesos que quieren establecer una comunicación, podrían comenzar a enviar información a la vez y que dar bloqueados de forma indefinida.

Representación de datos

La comunicación entre dos procesos distribuidos, en el nivel más bajo (nivel de red), se realiza mediante el intercambio de un flujo binario. La heterogeneidad de los entornos de red y los computadores conectados a ella implica que la

información transmitida puede ser interpretada de manera inadecuada por el receptor. Dos ejemplos pueden verse en la comunicación entre dos maquinas, en las cuales una posee una arquitectura de 32-bits con una representación *big-endian* y la otra una arquitectura de 16-bits con una representación *little-endian*.

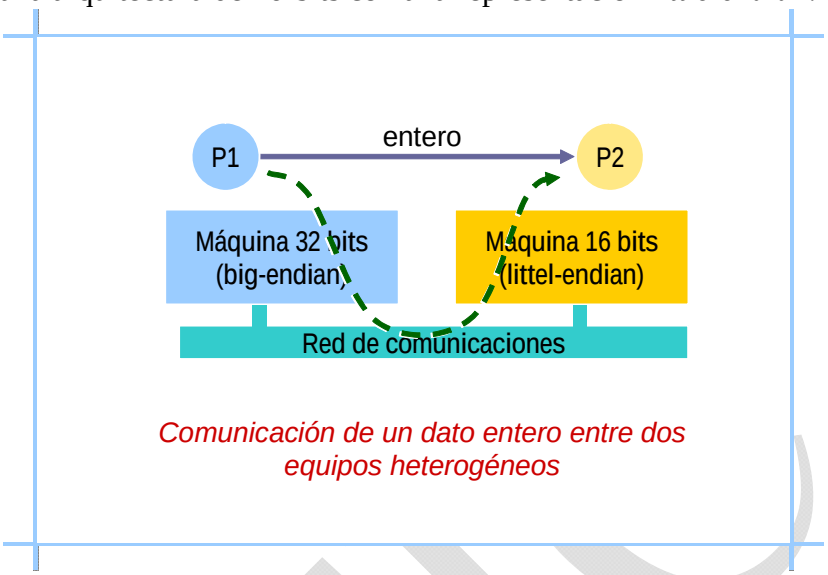


FIGURA 4.4: TRANSMISIÓN ENTRE ARQUITECTURAS DIFERENTES.

El otro ejemplo sería el de dos sistemas cuya representación de caracteres varía. Mientras una representa los caracteres mediante ASCII, la otra podría representarlo mediante el formato UNICODE.

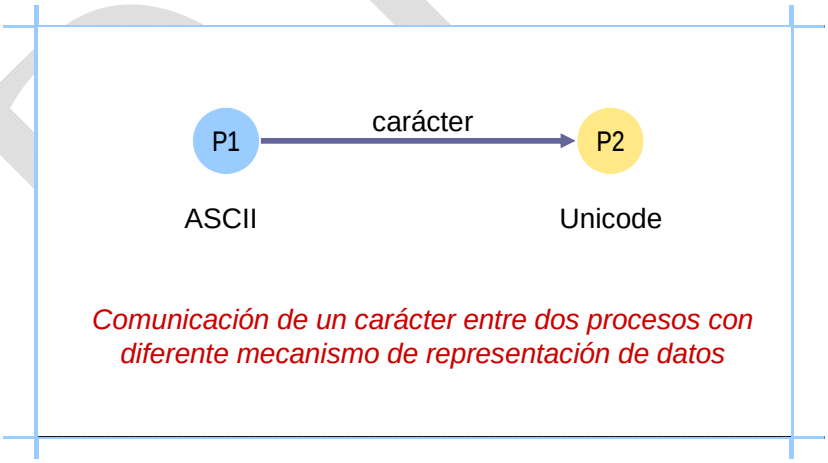


FIGURA 4.5: TRANSMISIÓN ENTRE PROCESOS DE REPRESENTACIÓN DE CARACTERES DIFERENTES.

En ambos casos se podría perder o alterar la información transmitida desde el emisor.
Existen tres tipos de soluciones, más o menos factibles, para resolver esta problemática:

- El proceso emisor (P1) adapta antes de enviar los datos al sistema de el proceso receptor (P2). Esta solución no es muy flexible puesto que siempre se debería conocer el formato de representación del proceso receptor.
- El proceso receptor (P2) adapta el formato del proceso emisor (P1) a su representación interna. En esta solución, junto con la información enviada se debería incluir información del formato de representación del proceso emisor.
- La última solución es el uso de una representación externa que negocien los procesos emisor y receptor. De esta forma el proceso emisor, antes de enviar la información debe transformarla al formato acordado y el proceso receptor, cuando reciba dicha información, debe transformarla del la representación externa a su formato interno. La mayoría de plataformas middleware de comunicación entre procesos utilizan hoy en día esta solución.

Otro aspecto a tener en cuenta es la transmisión de información de tipos de datos complejos o tipos no básicos. Además, de utilizar los mecanismos de representación externa son necesarios mecanismo que permitan organizar este tipo de datos de manera que puedan ser transmitidos por la red desde el emisor y reconstruidos por el receptor una vez recibido.

El concepto de empaquetamiento de datos o marshalling surge para resolver esta problemática. El marshlling se compone de dos etapas:

- La primera se conoce como serialización. En esta etapa las estructuras o tipos complejos de datos son transformados en un formato que permita trasmitirlo a través de la red de comunicaciones para su posterior reconstrucción.
- La segunda etapa se refiere a la codificación de los datos a una representación externa antes de ser transmitidos por la red.

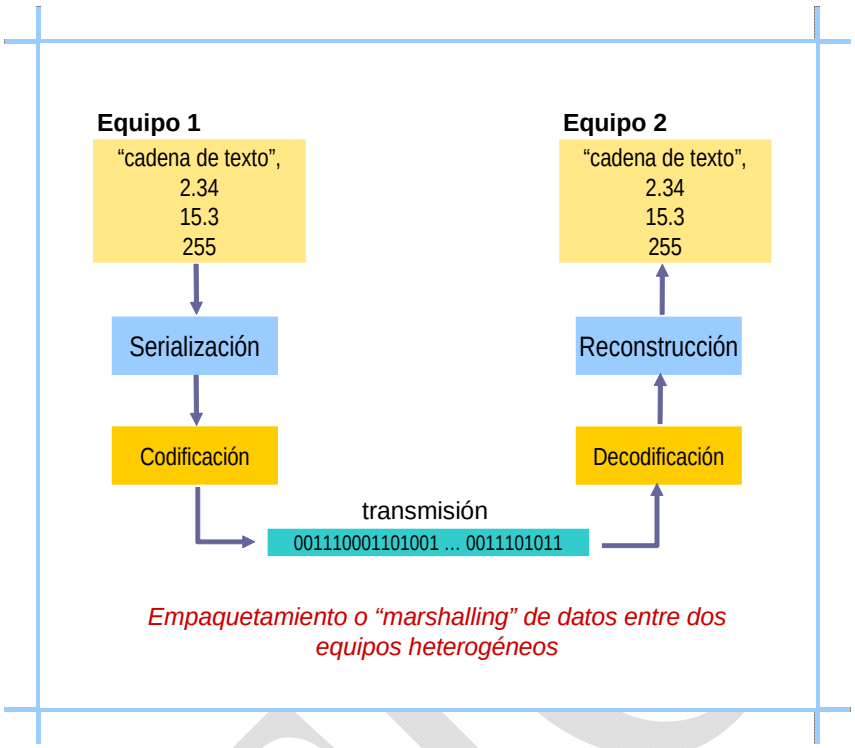


FIGURA 4.6: PROCESO DE MARSHALLING.

El envío de objetos a través de la red requiere un proceso más complejo y recibe el nombre de serialización de objetos. Además de realizar un proceso de serialización de datos, es necesario transmitir información sobre el estado del objeto y los métodos del mismo.

La codificación a un lenguaje de representación externa común puede ser acordado entre el emisor y el receptor. Generalmente, y con el objetivo de flexibilizar y desacoplar la relación entre el emisor y el receptor, se suelen utilizar lenguajes de representación externa que se han convertido en estándares.

Ejemplos de lenguajes estándares de representación externa de datos son:

- XDR (eXternal Data Representation). Creado por SUN para los mecanismos RPC.
- ASN.1 (Abstract Syntax Notation). Estandar OSI y que es utilizado por ejemplo en protocolos como LDAP o DNS.
- XML (eXtensible Markup Language). Hoy en día es uno de los más utilizados por su capacidad expresiva y flexibilidad.

```
<agenda>

   <contacto>
      <nombre>Virgilio</nombre>
      <apellidos>Gilart Iglesias</apellidos>
      <localidad>Alicante</localidad>
      <teléfono>555 77 9999</teléfono>
      <email>vgilart@dtic.ua.es</email>
   </contacto>

  <contacto>
      <nombre>Diego</nombre>
      <apellidos>Marcos Jorque</apellidos>
      <localidad>Elche</localidad>
      <teléfono>555 66 8888</teléfono>
      <email>dmarcos@dtic.ua.es</email>
   </contacto>

</agenda>
```

Archivo XML de ejemplo con la definición de una agenda personal de contactos

FIGURA 4.7: EJEMPLO DE REPRESENTACIÓN CON XML.

Protocolos de aplicación

Para que dos procesos puedan establecer una comunicación es necesario el establecimiento de un protocolo o conjunto de reglas que especifiquen el intercambio de datos. Cuando el protocolo permite la comunicación entre dos aplicaciones se realiza a nivel de aplicación se denomina protocolo de aplicación.
Los protocolos basados en texto son sencillos de manejar y de interpretar. El empaquetamiento de los datos a ser transmitidos supone el caso más simple. El esfuerzo para diseñar protocolos de este tipo (http, FTP, etc.) requiere un menor esfuerzo por parte de los desarrolladores que el requerido en el diseño de protocolos de aplicación de tipo binario (LDAP, DHCP, etc.).
Algunos ejemplos de protocolos basados en texto son: HTTP, SMTP o POP3.
Un tipo importante de protocolos son los basados en solicitud-repuesta, en los cuales un proceso emisor envía una solicitud al procesos receptor y este le devuelve una respuesta. El envío de peticiones y la solicitud de repuestas puede ser constante hasta que se complete la tarea deseada. Algunos ejemplos típicos de este tipo de protocolos son: FTP, HTTP o SMTP.
Otra clasificación de los protocolos de aplicación puede realizarse alrededor del concepto de conexión. Se pueden diseñar protocolos orientados a la conexión, en los cuales se establece una conexión lógica entre el emisor y el receptor. Una vez

realizada dicha conexión, se envía la información a través del canal lógico establecido. Este tipo de protocolos garantiza la transmisión de los datos aunque conllevan una mayor recarga de la red. Ejemplos de protocolos que implementan este modelo son HTTP o FTP. En los protocolos no orientados a la conexión, los datos se transmiten por medio de paquetes independientes sin establecimiento de conexión previa. En este caso no se garantiza que el mensaje alcance su destino pero mejora el rendimiento en la comunicación entre procesos. Ejemplos de protocolos que implementan este modelo son DHCP o DNS.
Por último, mencionar que un protocolo puede mantener el estado de la comunicación entre el proceso emisor y el receptor, es decir mantener una relación entre las distintas conexiones entre un proceso emisor y un receptor. Los protocolos sin estado no mantienen relación entre las diferentes conexiones entre los procesos que tienen lugar en la comunicación. Un ejemplo de protocolo sin estado es HTTP. Ejemplos de protocolos con estado son FTP o SMTP.

Paso de Mensajes

El paradigma de paso de mensajes es el modelo de comunicación entre procesos distribuidos más básico y sobre el cual se sustentan el resto de mecanismos. En el paradigma de paso de mensajes el proceso emisor envía información, que representa un mensaje, a un proceso receptor a través de un canal de comunicaciones.
La interfaz mínima requerida en este tipo de mecanismo debe ofrecer las operaciones enviar y recibir para poder transmitir los datos entre el emisor y el receptor. Además el mecanismo puede estar orientado a la conexión, con lo cual la interfaz incluirá también las operaciones necesarias para tal fin: conectar y desconectar.

Sockets

Los sockets son un mecanismo de comunicación entre procesos que fueron originalmente desarrollados como una biblioteca de programación en la versión del sistema operativo UNIX de Berkeley (BSD).

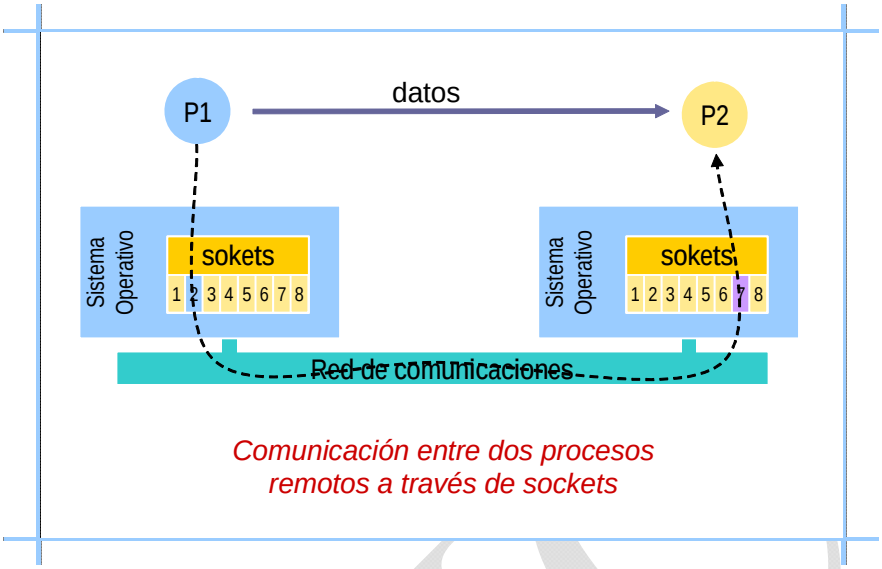


FIGURA 4.8: MODELO DE SOCKETS.

Este API se fundamenta en el modelo de paso de mensajes visto anteriormente.
Hoy en día la mayoría de los sistemas operativos (UNIX, Linux, Windows, OS/2…) ofrecen este mecanismo de comunicación entre procesos como una librería de programación con un conjunto de primitivas que permite al programadora abstraerse, de una forma sencilla, de los aspectos básicos de comunicación.
Aunque el API de sockets permite la comunicación entre procesos ubicados en una misma máquina, en máquinas diferentes y a través de familias de diferentes protocolos, la versión más utilizada y que más repercusión ha tenido es la basada en la familia INET, la cual permite la comunicación a través de la familia de protocolos TCP/IP.
Se puede establecer una analogía entre la gestión de archivos en UNIX y el manejo de sockets. Cuando se crea un socket se establece un descriptor similar al obtenido al crear un archivo. Desde este momento las primitivas utilizadas para trabajar con los sockets y transmitir información son similares a las primitivas que provee un sistema operativo para la lectura y escritura en un archivo.
El API de socket ofrece primitivas que permiten orientar la comunicación a la conexión o no orientarla a la conexión. Como vimos anteriormente, en el caso de establecer una comunicación orientada a la conexión se garantiza la confiabilidad y el orden del mensaje. En la familia INET la comunicación de este tipo se realiza a través del protocolo TCP y en ella se establece una comunicación entre el emisor y el receptor de flujo de datos como si se tratase de una tubería. En el caso de no orientarse a la comunicación la información se envía como paquetes discretos y no existe garantía de que la información llegue al receptor o que llegue en el orden adecuado. INET hace uso de protocolo UDP para establecer este tipo de comunicación.

Llamadas a métodos remotos

De la misma forma que en una aplicación local escrita en un lenguaje procedimental se puede realizar una llamada a una subrutina, el modelo de llamadas a procedimientos remotos (RPC) busca poder invocar un procedimiento ubicado en una máquina remota. Este modelo se basa en el modelo de protocolo petición/repuesta. Cuando desde un proceso se realiza una petición para invocar un procedimiento remoto, el proceso emisor suspende su ejecución hasta que recibe una respuesta del proceso remoto.

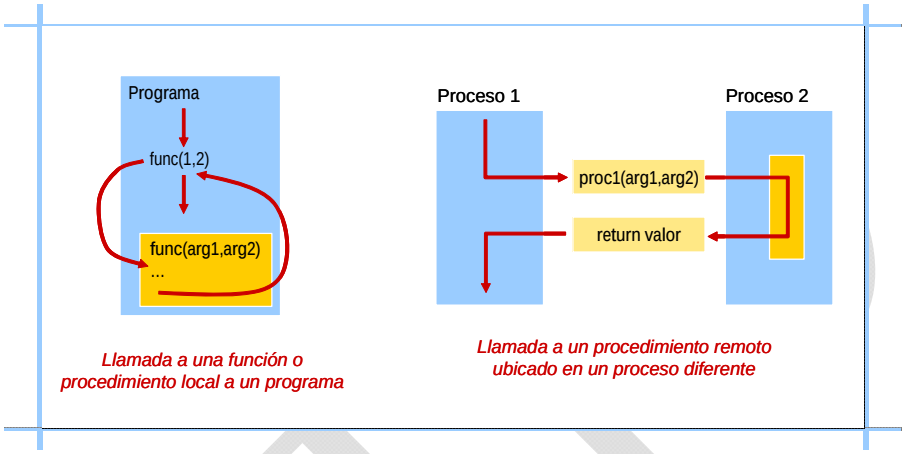


FIGURA 4.9: MODELO RPC.

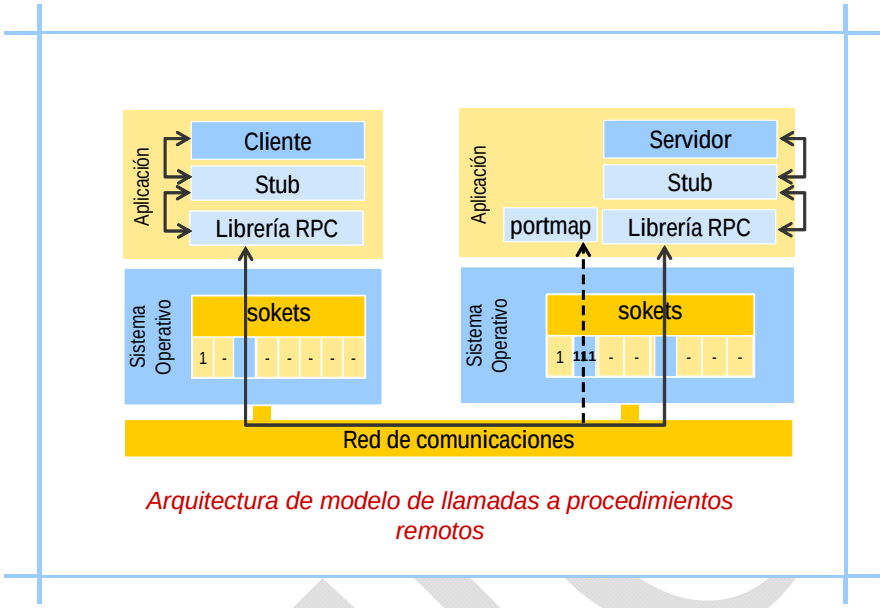
Una de las primeras implementaciones de este modelo fue el desarrollado por SUN, que posteriormente se convirtió en un estándar abierto especificado en las RFC1831 y RFC1057.

RPC permite al desarrollador abstraerse de los detalles de la comunicación a través de la red: de la representación de la información, la localización del proceso a invocar, los parámetros de entradas y valores devueltos desde el procedimiento remoto.

Esta transparencia se consigue a través de unas librerías denominadas Stubs. Se trata de aplicaciones proxies con las que realmente establece la comunicación la aplicación cliente. Estas se encargan de codificar la información a un lenguaje de representación intermedio (XDR), ordenar la secuencia de datos, realizar el proceso de marshalling, establecer la comunicación, etc.

Algunas implementaciones como la de SUN ofrecen herramientas que permiten de manera automática, a partir de una interfaz definida mediante un lenguaje de definición de interfaces (XDR) obtener las librerías proxies del cliente y del servidor.

Ejemplos de implementaciones de llamadas a procedimientos remotos son: SUN RPC, DCE RPC. Una evolución de este modelo que está teniendo gran repercusión en la comunicación de servicio Web es XML-RPC. Se trata de una especificación que define el mecanismo de llamadas a procedimientos remotos usando XML como lenguaje de representación de datos.



Arquitectura de modelo de llamadas a procedimientos remotos

FIGURA 4.10: ARQUITECTURA RPC.

Invocación de métodos remotos

La invocación a métodos remotos es un mecanismo IPC similar al plateado en el modelo RPC pero orientado a la invocación de métodos de objetos en lugar de a procedimientos de aplicaciones.

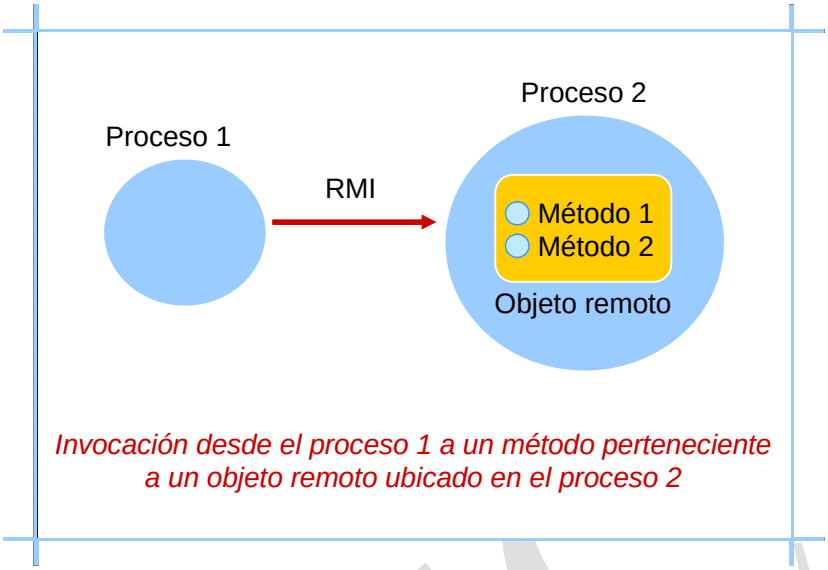


FIGURA 4.11: MODELO RMI.

De la misma forma que en RPC se establecen librerías que permiten abstraer la desarrollador de los aspectos relacionados con la comunicación a través de la red, en la invocación a métodos remotos se establecen elementos o librerías similares llamadas Stubs (en el cliente) y Skeletons (en el servidor). No obstante como vimos en la serialización a objetos la funcionalidad que desempeñan estas librerías es más compleja debido a las características intrínsecas de los objetos.

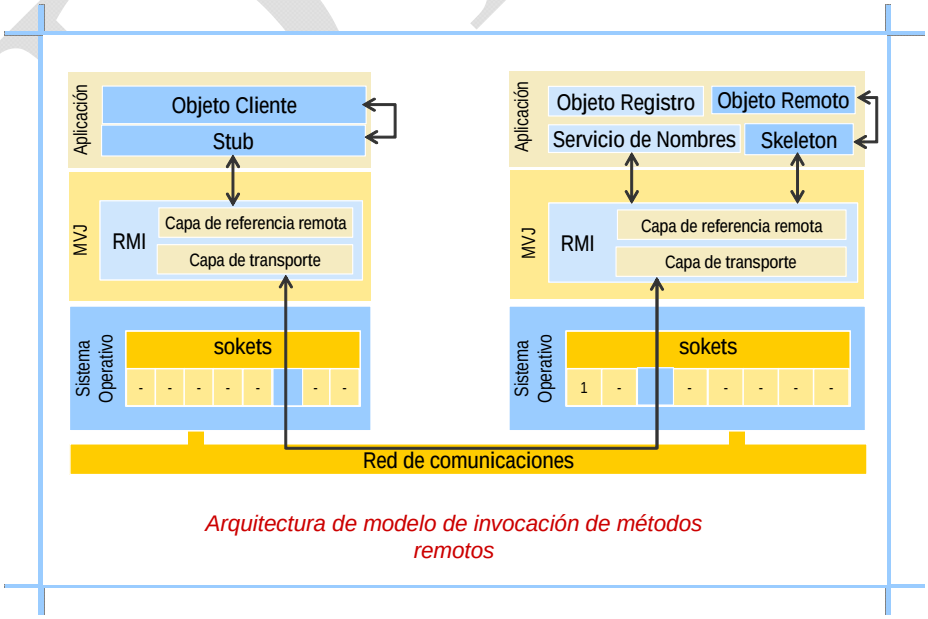


FIGURA 4.12: ARQUTECTURA RMI.

La propuesta de SUN para este modelo se llama RMI. RMI utiliza como protocolo de transporte binario JRMP. En las sesiones de prácticas se ofrecen ejemplos guiados que permita al alumno familiarizarse con esta tecnología.
Otras implementaciones de este paradigma son las propuestas por MicroSoft

- DCOM
- .NET Remoting (VS.NET)

Una de las ventajas de .NET Remoting es que permite utilizar como protocolo de transporte el protocolo HTTP. Este enfoque facilita el envío de información a través de entornos de red donde existan firewalls.
Un problema de la invocación a métodos remotos es que la comunicación se realiza únicamente entre objetos escritos en el mismo lenguaje o plataforma.

Intermediario de petición objetos

Se trata de uno de los mecanismos de más alto nivel que a diferencia de los anteriores, como principal objetivo establece la comunicación entre objetos escritos en diferentes lenguajes y diferentes plataformas.
Este modelo es la base de la arquitectura CORBA (Commom Object Request Brorker Architecture) definida por el OMG.

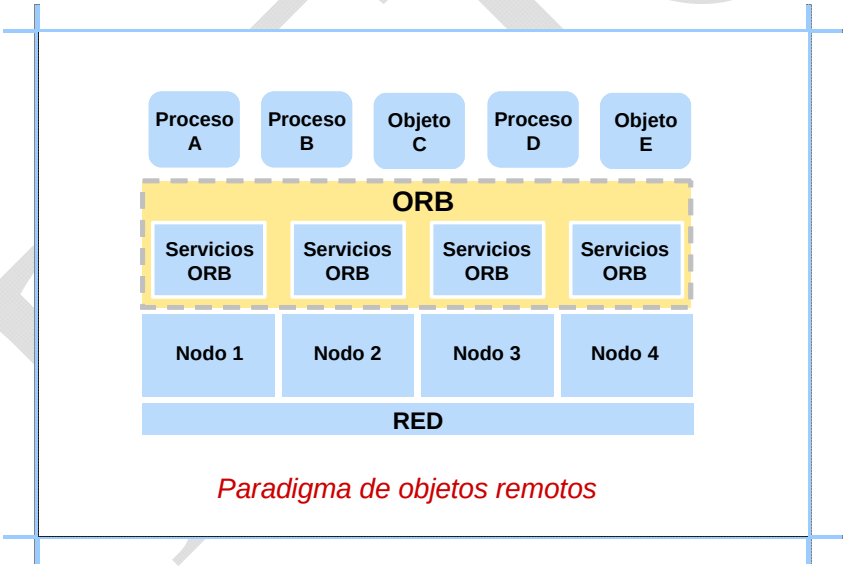


FIGURA 4.13: MODELO DE INTERMEDIARIO DE PETICIÓN DE OBJETOS.

Los elementos básicos que conforman esta arquitectura son:

- ORB (agente intermediario para la gestión de objetos)
- IDL y CDR (definición de interfaces y representación externa)
- GIOP (protocolos de comunicación)
  - IIOP → TCP/IP
  - HTIOP → http

Existen diferentes implementaciones de este modelo como la incluida en la plataforma J2EE o .NET, o implementaciones menos complejas como microORB. Este modelo se presenta con más detalle en el tema 2 de la asignatura, Tecnologías Web y Middleware.

Servicios Web

La tecnología de servicios Web se ha establecido como una de las grandes promesas en la computación distribuida. Los servicios Web permiten, al igual que sucede en el modelo anterior, la interoperación de aplicaciones escritas en diferentes lenguajes y plataformas pero utilizando protocolos de un nivel de abstracción mucho mayor y los cuales se fundamentan en el uso de XML como lenguaje de representación.
Los servicios Web a diferencia del modelo CORBA permite un completo desacoplamiento entre las aplicaciones.
Los principales elementos o protocolos que conforman dicha tecnología son:

- SOAP es el protocolo de transporte. Evolución de XML-RPC.
- WSDL es el lenguaje de definición de interfaces.
- UDDI es el servicio que almacena la información de los servicios y que permite la publicación, localización y consumo de servicios. Ofrece los servicios en términos de negocio.

Hoy en día existen implementaciones de servicios Web en casi todos los lenguajes. Como ejemplo encontramos la implementación de J2EE, .NET, cSOAP entre otras.
Actualmente se está trabajando en servicios sobre la tecnología de servicios Web que permitan resolver problemas derivados de la computación distribuida como las transacciones (WS-transaction), seguridad (WS-security), direccionamiento (WS-Addressing), etc. Este modelo se conoce como servicios Web 2.0 o WS-* y pretende proveer un middleware de servicios similar a la arquitectura CORBA. Además una de las ventajas de esta tecnología es que debido al nivel de desacoplamiento que ofrece, su uso es idóneo en modelo como la gestión de procesos de negocios (BPM).

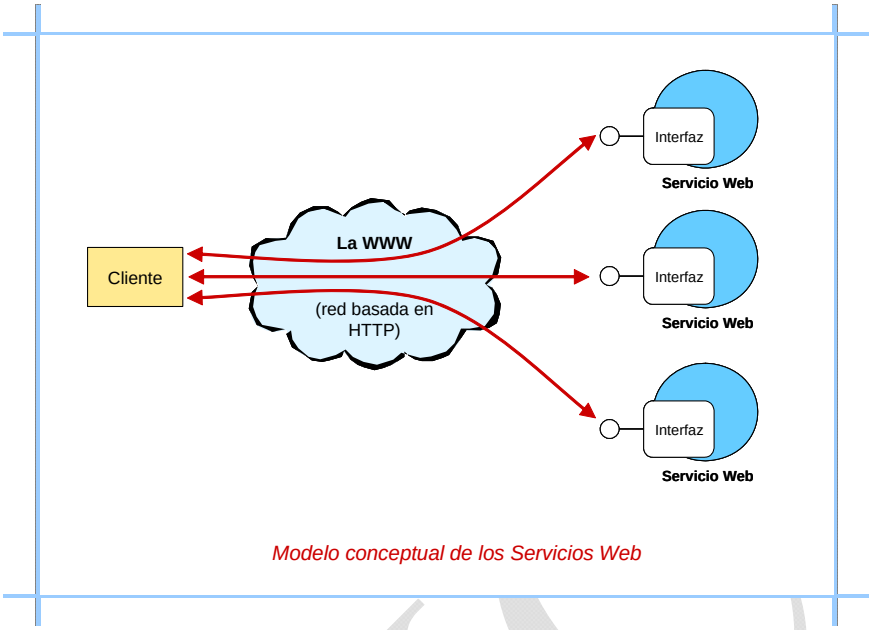


FIGURA 4.14: MODELO DE SERVICIOS WEB.

Al igual que en el modelo ORB este mecanismo será estudiado con mayor detalle en el tema 2.

Decisiones de diseño

Una vez revisados los modelos, paradigmas y mecanismos de comunicación se deben tener en cuenta ciertos aspectos de dichos elementos a la hora de diseñar un sistema distribuido para un problema dado de una forma adecuada.
Un aspecto importante es la decisión de usar mecanismos de mayor nivel de abstracción que faciliten el desarrollo y mejoren la productividad pero que pueden producir una sobrecarga en el diseño final.
Otro aspecto radica en la escalabilidad que pueden ofrecer los diferentes paradigmas y permitir crecer al sistema de forma transparente sin que para ellos haya que realizar grandes esfuerzos de modificación o rediseño de la aplicación.
Un aspecto importante hoy en día es que la mayoría de los sistemas son heterogéneos y un criterio que se debe valorar al elegir los paradigmas o herramientas adecuadas radica en la posibilidad de poder portar las aplicaciones a otras plataformas o la integración de la aplicación con aplicaciones en diferentes lenguajes y plataformas.
Otros criterios adicionales que se deben tener en cuenta se listan a continuación:

- Madurez, estabilidad de la tecnología y disponibilidad de herramientas de desarrollo
- Tolerancia a fallos ofrecida por la herramienta
- Mantenibilidad y Reutilización de código

Sección 5

Conclusiones

En la unidad presentada se ha descrito y justificado de manera conceptual la que se ha producido desde las aplicaciones basadas en diseños monolíticos hasta llegar a los modelos de computación distribuida existentes en la actualidad.
Se ha hecho hincapié en esta evolución en la necesidad de establecer mecanismos comunes a los sistemas que abstraigan de la complejidad surgida en la gestión de la red recomunicaciones y de procesos de intercambio de información entre aplicaciones remotas y minimicen el esfuerzo de desarrollo de los programadores en estos entornos.
Además se han descrito y justificado la existencia de diferentes modelos, mecanismos y herramientas de forma que se puedan seleccionar de manera adecuada y con un espíritu crítico en función del problema que se deba afrontar.