Resumen TEMA 1.
Fundamentos de computación distribuida

Melanie Mariam Cruz Morgado y Pedro Giménez Aldeguer.

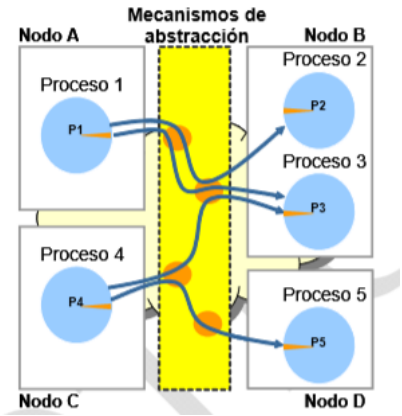
1 PARADIGMAS DE COMPUTACIÓN DISTRIBUIDA

1.1 EVOLUCIÓN DE LOS SISTEMAS DE COMPUTACIÓN

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

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. Facilitaba el mantenimiento de las aplicaciones y la modificación de pequeños módulos de la aplicación sin que para ello sea necesario modificar la aplicación original. La división implicaba la posibilidad de que esos módulos puedan ser utilizados por diferentes aplicaciones mejorando la productividad en el desarrollo de aplicaciones gracias a la reusabilidad. (Procesos ubicados en el mismo procesador.)

Los computadores personales y las arquitecturas multiprocesadores surge la posibilidad de ubicar los diferentes módulos en distintos procesadores. 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.



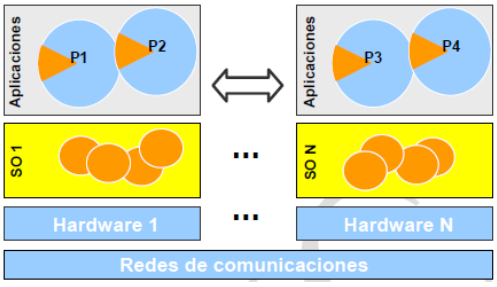
En un principio, toda la lógica necesaria para establecer la comunicación en un entorno distribuido se incluía en la propia aplicación, lo que suponía un gran esfuerzo por parte de los programadores. Poco a poco esta lógica se fue extrayendo y ubicando en módulos comunes a las aplicaciones ubicadas en el mismo procesador.

Finalmente, los mecanismos que permiten la comunicación entre aplicaciones distribuidas son ubicados 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.

1.2 ENFOQUES DE LOS SISTEMAS OPERATIVOS PARA LOS SISTEMAS DISTRIBUIDOS

1.2.1 Sistemas Operativos en Red

Los mecanismos de abstracción de comunicaciones se ubican generalmente en el propio SO y se implementan de forma **diferente** en cada **SO**. Además, esta propuesta es la más adecuada actualmente.



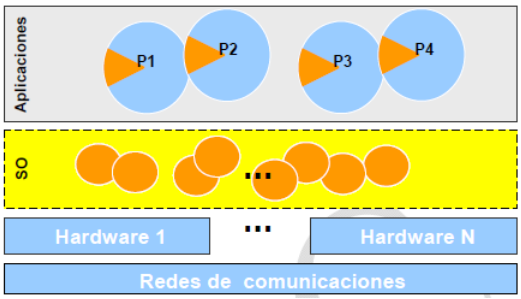
Ventajas:

- Flexibles. No existe una fuerte dependencia a nivel de software entre los nodos que conforman la red.
- Adecuados para entornos donde la gestión de la red se encuentra dispersa y es incontrolable (Internet).
- SO convencionales. Técnicas usadas para su diseño poseen un algo grado de madurez.

Desventajas:

- No transparentes en la gestión de los recursos distribuidos. El usuario ve un conjunto de máquinas independientes.
- Mayor esfuerzo de los desarrolladores en la implementación de aplicaciones distribuidas que trabajen conjuntamente (protocolos comunes).

1.2.2 Sistemas Operativos Distribuidos



Los mecanismos de abstracción de comunicaciones se ubican en el SO y son comunes a todos los nodos que conforman el sistema, puesto que comparten un **único SO**.

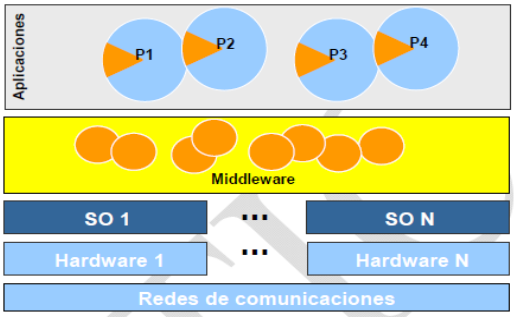
Ventajas:

- Transparente integración entre los nodos.
- Transparente escalabilidad del sistema.
- Ofrece una visión única a la hora de compartir y gestionar recursos.

Desventajas:

- Técnicas de diseño más complejas.
- Difícil de aplicar en entorno de red de gran tamaño y con gestión distribuida.
- Necesidad de un entorno de red de alta velocidad.

1.2.3 Middleware



Los mecanismos de abstracción de comunicaciones en una capa situada por encima del sistema operativo convencional y es común a los elementos que conforman la red.

Ventajas: Flexibilidad, transparencia, integración, madurez y escalabilidad.

Desventajas: Heterogeneidad de plataformas y necesidad de estandarización.

2 MODELOS ARQUITECTÓNICOS

2.1 MODELO CLIENTE/SERVIDOR

Se establece sobre el paradigma de paso de mensajes y el esfuerzo que requiere el diseño es elevado. Los protocolos de Internet más conocidos son HTTP, DNS, FTP y SMTP. Los procesos pueden adquirir uno de los 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 Servidos.

Requisitos en el diseño:

- Mecanismos de **concurrencia** que permitan al Servidor gestionar el acceso de múltiples clientes al mismo tiempo.
- **Mantenimiento** de forma continua de los accesos que un determinado cliente realiza al servidor, es decir, mantenimiento de la sesión.
- Mecanismos de **escalabilidad** necesarios para que el número de clientes accediendo al mismo tiempo no afecten al rendimiento.

2.2 MODELO PEER-TO-PEER (P2P)

Evolución del modelo Cliente/Servidor. Todos los procesos participantes desempeñan tanto el rol de cliente como el de servidor. Como cliente puede enviar peticiones y recibir respuestas; y como servidor puede recibir solicitudes, procesarlas y propagarlas, además de enviar respuestas.

Este modelo es apropiado para aplicaciones de tipo: mensajería instantánea, transferencia de ficheros, vídeo conferencia y trabajo colaborativo.

Ejemplos de aplicaciones:

- **Napster** utiliza una arquitectura centralizada, es decir, todas las transacciones entre pares que conforman la red se realizan a través de un nodo central.
- **Gnutella** utiliza una arquitectura descentralizada, es decir, todos los nodos que conforman la red tienen la misma importancia, peso y funcionalidad.
- **BitTorrent** utiliza una arquitectura híbrida o mixta, es decir, existe un servidor central que gestiona la comunicación entre los pares de manera eficiente sin almacenar ningún tipo de dato, pero si falla la comunicación se mantiene entre los pares.

JXTA es un framework que permite que los participantes conectados intercambien mensajes entre sí, disminuyendo el esfuerzo que debe realizar el programador.

2.3 MODELO DE SISTEMA DE MENSAJES

Elaboración del sistema de paso de mensajes convencional, donde la comunicación entre emisor y receptor es de forma completamente desacoplada. Se utiliza en intercambio de mensajes de forma asíncrona y se emplea cuando se quiere establecer un único punto de entrada.

Elementos:

- El proceso **emisor** es responsable de la emisión de un mensaje.
- El proceso **intermediario** almacena los mensajes enviados por el emisor. El agente suele ser un middleware con las siguientes características: gestión de prioridades de mensajes, temporizadores para la gestión de mensajes, gestión de formatos de mensajes, gestión de seguridad y gestión de la persistencia de los mensajes.
- El proceso **consumidor** se conecta al proceso intermediario para obtener los mensajes y procesarlos.

Ejemplos de tecnologías basadas en este modelo: JMS de SUN, MSMQ de Microsoft o MQSeries de IBM.

Variantes:

- **Punto a punto**. 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.
- **Publicación/suscripción.** un mensaje publicado por el emisor (enviado al agente intermediario) será procesado por todos los agentes consumidores que se hayan suscrito al proceso intermediario.

2.4 Modelo de Arquitectura Orientada a Servicios (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. un agente intermediario posibilita la localización de los servicios disponibles.

Elementos:

- **Proceso proveedor del servicio** que expone el acceso a un recurso como un servicio de red. Siendo el que se encarga de publicar la información en el servicio del 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. Siendo el que se conecta al servicio de directorio para localizar el servicio que ofrezca la funcionalidad que requiere.
- **Servicio de directorio** almacena la información necesaria sobre los servicios para que un proceso consumidor pueda localizar, descubrir y consumir los servicios. De esta forma, se consigue la característica clave del modelo, la transparencia de localización.

Características claves:

- Interoperatividad.
- 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 este modelo: UPNP, JINI y Servicio Web (no ha surgido como SOA, pero se acopla perfectamente).

2.5 Modelo de Clúster/Grid

Su objetivo es aprovechar los recursos de sistemas independientes conectados a la red para obtener una infraestructura hardware y software con mayor capacidad de procesamiento y almacenamiento. Se basa en la unión de un conjunto de computadores que se unen para dar forma a un supercomputador.

A continuación, se exponen las características distintivas de cada uno de ellos:

2.5.1 Clúster

Los componentes se encuentran fuertemente acoplados a nivel de hardware o software, consiguiendo homogeneidad, por lo que se encuentran en redes locales de alta velocidad. El coordinador decide qué procesos se ejecutan en cada participante (gestor de recursos centralizado) y presentan alta disponibilidad, balanceo de carga, escalabilidad y alto rendimiento.

Además, existe pérdida de independencia en cuanto al procesador, por lo que puede ralentizar de aplicaciones que se estén ejecutando en el mismo nodo del clúster que está consumiendo del procesador.

Algunas aplicaciones son: servidores web y de aplicaciones, sistemas de información y resolución de problemas con necesidad de supercómputo.

2.5.2 Grid

Los elementos son heterogéneos conectados a través de redes de área amplia completamente distribuidas, como Internet, por lo que es más flexible. El coordinador coordina los procesos que se ejecutan en los participantes sin decidir cuál se ejecuta en cada participante. No suele perder independencia de procesamiento y no busca ofrecer una visión única del sistema.

3 MECANISMOS DE COMUNICACIÓN

3.1 MECANISMOS IPC

Es el mecanismo de comunicación básico y simple de los SD ya que 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.

Al igual que un SO convencional ofrece una serie de mecanismos que permiten la comunicación entre procesos locales: mecanismos de colas de mensajes, semáforos y memoria compartida. Actualmente se proporcionan mecanismos de comunicación de procesos de más alto nivel que permiten la interoperación entre procesos distribuidos con un menor esfuerzo de desarrollo y pueden ubicarse dentro o por encima del SO como librerías.

Aspectos a tener en cuenta en el diseño de interfaces de programación para la comunicación de procesos:

- Los modelos básicos son unidifusión (**Unicast**) y multidifusión (**Multicast**).

- **Sincronización**.
  Problema: Los procesos son independientes y cada uno de ellos desconoce el estado en el cual se encuentra el resto.
  Solución: Uso de operaciones bloqueantes o síncronas, ya que cuando un proceso invoca una operación, su procesamiento queda suspendido hasta que esta haya finalizado.
  * Cuando se invoca a una operación no bloqueante o asíncrona no es necesario que esta termine para que el proceso siga ejecutándose.

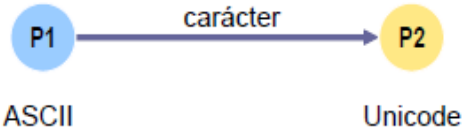
- **Operaciones bloqueantes**.
  Problema: Pueden llevar a un proceso a un estado de bloqueo permanente o temporalmente no aceptable.
  Soluciones:
  - Utilizar primitivas de comunicación entre procesos que introduzcan temporizadores, para evitar bloqueos indefinidos o tiempos desmesurados.
  - Uso de procesos hijos o hilos de ejecución que se encarguen de la comunicación, permitiendo que el proceso padre o hilo principal continué su ejecución.

- Los **Interbloqueos (Bloqueos indefinidos)** se pueden producir a causa de diseños de protocolos erróneos.

3.2 REPRESENTACIÓN DE DATOS

El intercambio de datos en el nivel más bajo (nivel de red) se realiza en el nivel de red mediante 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.

Ejemplos:



Comunicación de un carácter entre dos procesos con diferente mecanismo de representación de datos

Comunicación de un dato entero entre dos equipos heterogéneos

Problema: En ambos casos se podría perder o alterar la información transmitida desde el emisor.

Soluciones:

- P1 (Proceso emisor) adapta los datos antes de enviarlos al sistema del P2 (Proceso receptor), aunque no es muy fiable ya que siempre se debería conocer el formato de representación del P2.
- P2 adapta el formato del P1 a su representación interna y junto con la información enviada se debería incluir información del formado de representación del P1.
- Uso de una representación externa que negocien P1 y P2, y de esta forma el P1 antes de enviar la información, debe transformarla al formato acordado y el P2, cuando la reciba, debe transformarla de la representación externa a su formato interno. Esta solución es la usada por la mayoría de plataformas middleware actualmente.

En la transmisión de información de tipos de datos complejos…

Problema: Además de utilizar los mecanismos de representación externa, son necesarios mecanismos que permitan organizar este tipo de datos de manera que puedan ser transmitidos por la red y reconstruidos una vez recibidos.

Solución: Empaquetamiento de datos o marchalling que se compone de dos etapas:

- Serialización. Las estructuras o tipos complejos de datos son transformados en un formato que permita transmitirlo a través de la red de comunicaciones para su posterior reconstrucción.
- Codificación. Estos se codifican a una representación externa antes de ser transmitidos por la red.

El envió de objetos a través de la red requiere un proceso más complejo, serialización de objetos. Además de este, 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, pero generalmente se usan estándares, con el objetivo de flexibilizar y desacoplar la relación entre ambos.

Ejemplos de lenguajes estándares de representación externa de datos:

- XDR. Creado por SUM para los mecanismos RPC.
- ASN.1. Estándar OSI utilizado n protocolos como LDAP o DNS.
- XML. Uno de los más usados actualmente por su capacidad expresiva y flexibilidad.

3.3 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 (HTTP, FTP, SMTP, POP3) son más sencillos de diseñar, manejar y de interpretar que los de tipo binario (LDAP, SHCP).

Un tipo importante de protocolo son los basados en solicitud-respuesta, donde el envío de peticiones y la solicitud de respuestas puede ser constante hasta que se complete la tarea deseada (FTP, HTTP, SMTP).

Los protocolos de aplicación se pueden clasificar según si están orientados a la **conexión** o no. En los que **sí** (HTTP, FTP), se establece una conexión lógica entre emisor y receptor, y una vez realizada, se envía la información a través del canal lógico establecido, esto garantiza la transmisión de los datos, aunque conlleva una mayor recarga de la red. Pero en los que **no** (DHCP, DNS), los datos se transmiten por medio de paquetes independientes sin establecimiento de conexión previa y no se garantiza que el mensaje alcance su destino, aunque mejora el rendimiento en la comunicación.

Además, un protocolo puede mantener el estado de la comunicación entre emisor y receptor (FTP, SMTP), es decir, la relación entre las distintas conexiones entre ambos. Pero los protocolos sin estados (HTTP) no mantienen esta relación.

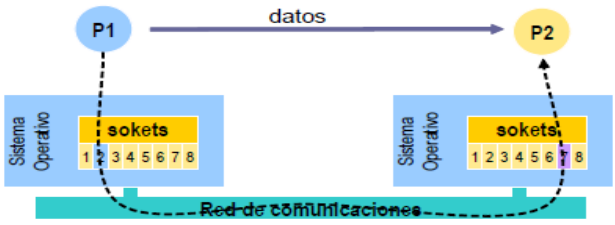
3.4 PASO DE MENSAJES

El modelo más básico de comunicación entre procesos distribuidos es el paradigma de paso de mensajes y es sobre el cual se sustentan el resto de mecanismos. En este, la interfaz mínima debe ofrecer las operaciones enviar y recibir, y las operaciones necesarias para conectar y desconectar, en el caso de que esté orientado a la conexión.

3.4.1 Sockets

Mecanismo de comunicación entre procesos que fueron originalmente desarrollados como una biblioteca de programación en la versión del SO UNIX de Berkeley (BSD).

La mayoría de SO ofrecen este mecanismo como una librería de programación con un conjunto de primitivas que permita al programador abstraerse, de una forma sencilla, de los aspectos básicos de comunicación.



Comunicación entre dos procesos remotos a través de sockets

Esta API se fundamenta en el modelo de paso de mensajes y ofrece primitivas que permiten orientar, o no, la comunicación a la conexión. Permite la comunicación entre procesos ubicados en la misma máquina, en máquinas diferente y a través de familias diferentes de protocolo. La versión más utilizada y con más repercusión es la basada en la familia INET, que permite la comunicación a través de la familia de protocolos TPC/IP. En esta, la orientada a conexión se realiza a través del protocolo TCP y la no orientada a conexión, a través del UDP.

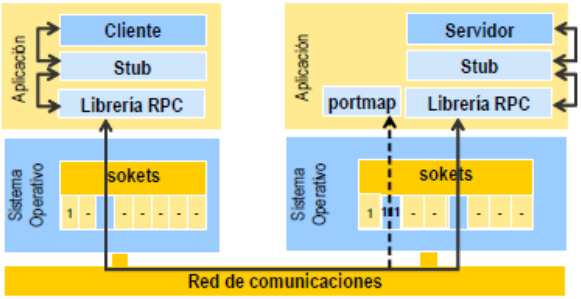
Además, se puede establecer una analogía entre la gestión de archivos de UNIX y el manejo de shockets. De manera que, cuando se crea un socket, se establece un descriptor similar al obtenido al crear un archivo y desde este momento las primitivas utilizadas para trabajar con los sockets y transmitir información son similares a las primitivas que provee un sistema operativo para la lectura y escritura en un archivo.

3.5 LLAMADAS A MÉTODOS REMOTOS

El modelo de llamadas a procedimientos remotos (RPC) busca poder invocar un procedimiento ubicado en una máquina remota y 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. Además, permite al desarrollador abstraerse de los detalles de la comunicación a través de la red.

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.

SUN-RPC, DCE-RPC, XML-RPC son ejemplos de implementaciones de llamadas a procedimientos remotos.

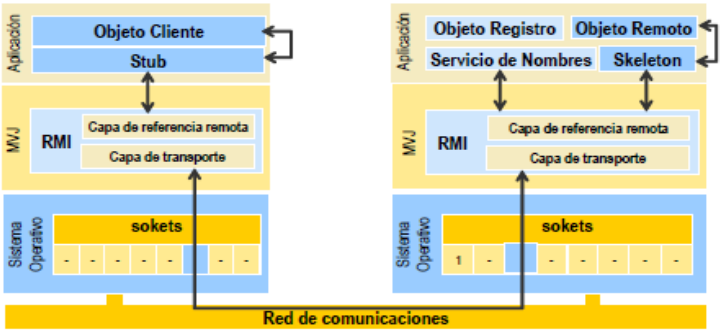


Arquitectura de modelo de llamadas a procedimientos remotos

3.6 INVOCACIÓN DE MÉTODOS REMOTOS

Mecanismo IPC similar al planteamiento RPC, pero orientado a la invocación de métodos de objetos en lugar de a procedimientos de aplicaciones.

Se establecen librerías que permiten abstraer el desarrollador de los aspectos relacionados con la comunicación a través de la red y, en la invocación a métodos remotos, se establecen elementos o librerías similares llamadas **Stubs** (en el cliente) y **Skeletons** (en el servidor). La funcionalidad que desempeñan estas es más compleja debido a las características intrínsecas de los objetos.



Arquitectura de modelo de invocación de métodos remotos

La propuesta de SUN para este modelo se llama RMI. RMI utiliza como protocolo de transporte binario JRMP. 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.

3.7 INTERMEDIARIO DE PETICIÓN OBJETOS

Uno de los mecanismos de más alto nivel. Principal objetivo: Establecer la comunicación entre objetos escritos en diferentes lenguajes y diferentes plataformas.

Este modelo es la base de la arquitectura CORBA definida por el OMG.

Elementos básicos que conforman esta arquitectura:

- 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

3.8 Servicios Web

Permiten la interoperación de aplicaciones escritas en diferentes lenguajes y plataformas, pero utilizando protocolos de un nivel de abstracción mucho mayor, los cuales se fundamentan en el uso de XML como lenguaje de representación.

A diferencia del modelo CORBA, permite un completo desacoplamiento entre las aplicaciones.

Principales elementos o protocolos que conforman esta tecnología:

- 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, que permite la publicación, localización y consumo de servicios, y que ofrece los servicios en términos de negocio.

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. 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).

3.9 Decisiones de diseño

- 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.
- 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.
- La posibilidad de poder portar las aplicaciones a otras plataformas o la integración de la aplicación con aplicaciones en diferentes lenguajes y plataformas, mediante los paradigmas o herramientas adecuadas.

Criterios adicionales a tener en cuenta:

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