Sistemas Operativos Distribuidos

Integración de Servicios

<Visión General>

Contenidos

1. Arquitecturas distribuidas: Conceptos
2. Middleware:
   - Abstracciones y modelos de programación
   - Infraestructuras
3. Tipos de middlewares:
   - Middlewares basados en objetos
   - Middlewares basados en transacciones
   - Middlewares basados en mensajes
4. Arquitecturas orientadas a servicios:
   - Integración: *Legacy systems y otras tecnologías*
   - Orquestación y coordinación de servicios

Sistemas Operativos Distribuidos

Arquitecturas Distribuidas

- **Sistema**
- **Arquitectura**

Desarrollo de Grandes Sistemas

- Sistema: Conjunto de elementos (distribuidos),relacionados entre sí, que llevan a cabo una función o tarea, no realizable por ellos por separado.
- Arquitectura: La estructura organizativa de un sistema:
  - Descrita de forma *top-down.*
  - Formada por una serie de elementos de los cuales se define:
    - Sus propiedades (externas): ¿qué hacen?
    - Las relaciones entre sí.
- Tecnología: Herramientas / entornos de programación para acometer el desarrollo del sistema. <<Su infraestructura>>.

Objetivos de Diseño de una Arquitectura

- El diseño de la arquitectura debe proporcionar las funcionalidades requeridas por el sistema:
  - Funcionalidades de negocio.
  - Funcionalidades de sistema.
  - Requisitos derivados: No son funcionales, pero sí de sistema y afectan a muchos elementos del mismo.
  - Prestaciones y otras características.
- Debe cumplir propiedades de integridad y robustez.
- Debe permitir la evolución y mantenimiento del sistema.

¿Cómo se Diseña una Arquitectura?

- El diseño se basa en vistas (a la UML):
  - Lógica, funcional, procesos, implementación, …
  - Despliegue: Más próxima a su implantación física, que considera dónde se ejecuta cada elemento del sistema y cómo estos elementos se envían las peticiones.
  - La forma de poder realizar el despliegue depende lo que tengamos por debajo (la "plataforma" sobre la cual desarrollamos). Para ello necesitamos dos partes:
    - El modelo conceptual de cada uno de los elementos. Abstracciones tales como: objetos, servicios, mensajes, …
    - La infraestructura de apoyo, la tecnología sobre la cual desarrollar la posible solución.

Sistemas Operativos Distribuidos

Middleware: Conceptos

- **Modelos de Programación**
- **Infraestructuras**

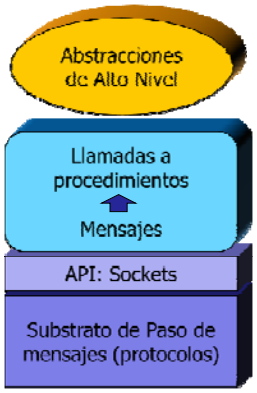
Modelos de Programación

El desarrollo de aplicaciones distribuidas requiere el uso de nuevos modelos (paradigmas) de programación:

- Todos ellos arrancan de un substrato común: "*Intercambio de paquetes de mensajes en base a protocolos de comunicaciones*".
- Pero, por encima de eso hacen falta de finir una serie de abstracciones, orientadas a:
  - Simplificar el desarrollo (detección de errores y verificación)
  - Ocultar detalles de bajo nivel (hardware y protocolos)
  - Aproximar las técnicas y herramientas de desarrollo a las características del dominio de aplicación
- Estas abstracciones se pueden encontrar implementadas en una o varias tecnologías.

Modelos de Programación

- La evolución del desarrollo de sistemas (distribuidos) ha ido progresando en diferentes etapas de abstracción:
  - Funcionalidades básicas (intercambio de mensajes en base a un protocolo)
  - Encapsulamiento en llamadas a procedimientos remotos (RPCs)
  - Modelos de programación con un mayor grado de abstracción:
    - ¿Qué elementos definen?
    - ¿Cómo interactúan?



Modelos de Programación

- Abstracciones/modelos de programación de alto nivel:
  - **Modelos de objetos distribuidos:** Extensión del modelo RPC sobre la teoría de objetos.
  - **Modelo transaccional distribuido:** Control de atomicidad de una serie de llamadas remotas.
  - **Modelo de mensajes asíncronos:** El procesamiento de peticiones está diferido, apoyándose sobre colas de peticiones pendientes.

Infaestructura

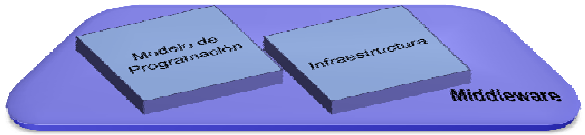
- La posibilidad de desarrollar un modelo de programación concreto se basa en la existencia de una **infraestructura** de apoyo que lo implemente.
- Esta infraestructura:
  - Implica nuevas capas de software que soporten las correspondientes abstracciones.
  - Define (hasta cierto punto) aspectos como los lenguajes de programación o las estructuras de datos.
  - Debe mantener estructuras de información de gestión (metadatos).
  - Debe considerar aspectos no funcionales: prestaciones,robustez, administración, mantenimiento, …
  - Se apoyan en la estandarización de sus componentes y herramientas.

Infraestructura

- Las infraestructuras asociadas a los diferentes modelos de programación son:
  - Modelos de objetos distribuidos → Brokers de objetos
    - Por ejemplo: CORBA
  - Modelo transaccional distribuido → Monitores transaccionales
    - Por ejemplo: Tuxedo
  - Modelo de mensajes asíncronos → Brokers de mensajes
    - Por ejemplo: MQSeries

Middleware

- Middleware: Consiste en una infraestructura adaptada para un modelo de programación concreto.
- Puede incluir:
  - Se concibe como un entorno de desarrollo y ejecución completo.
  - Servicios y funcionalidades adicionales.
  - Aspectos como hardware, sistema operativo o lenguajes deben abstraerse. El middleware debe ser consciente de ellos, soportarlos pero nos los oculta.



Sistemas Operativos Distribuidos

Middleware: Tipos

- **DOM**
- **TOM**
- **MOM**

MIddleware

- Las infraestructuras asociadas a los diferentes modelos de programación son:
  - *Distributed Object Middleware* (DOM):
    - Modelos: Objetos distribuidos
    - Infraestructura: Brokers de objetos
  - *Transaction Oriented Middleware* (TOM):
    - Modelo: Transacciones distribuidas
    - Infraestructura: Monitores transaccionales
  - *Message Oriented Middleware* (MOM):
    - Modelo: Mensajes asíncronos
    - Infraestructura: Brokers de mensajes

DOM: Motivación

- La extensión de los mecanismos de RPC a una programación orientada a objetos dio lugar a los modelos de objetos distribuidos.
- Esto permite que:
  - Los métodos remotos están asociados a objetos remotos.
  - Solución más natural para desarrollo orientado a objetos.
  - Admite modelos de programación orientada a eventos.

DOM: Modelo de Programación

Características:

- Nivel de abstracción para la comunicación de los objetos distribuidos. Oculta:
  - Localización de objetos.
  - Protocolos de comunicación.
  - Hardware de computadora.
  - Sistemas Operativos.
- Modelo de objetos distribuidos: Describe los aspectos del paradigma de objetos que es aceptado por la tecnología: Herencia, Interfaces, Excepciones, Polimorfismo, …
- Recogida de basura (*Garbage Collection):* Determina los objetos que no están siendo usados para a liberar recursos.

DOM: Conceptos Clave

- Facetas de un objeto:
  - La descripción del interfaz: ¿Qué puede hacer?
    - Lenguajes de definición: IDL
    - Repositorios de interfaces (descubrimiento dinámico [reflexión] y validación de referencias).
    - En la fase de compilación de un interfaz IDL se genera un identificador de interfaz.
  - Implementación de los métodos: ¿Cómo lo hace?
    - Resguardos: Stub/Skeleton
    - Repositorios de implementaciones (activación bajo demanda, asocia nombres e implementaciones)
    - La información depende del estado del objeto:
      - Activado: Nombre →Servidor/Puerto o Dir. Memoria
      - No Activado: Nombre →*Path* al ejecutable o biblioteca
    - Tiene un efecto importante en la gestión de ciclos de vida.

DOM: Conceptos Clave

- Facetas de un objeto (cont.):
  - Atributos y variables de estado: ¿Cómo se instancia?
    - Hace uso de referencias a objetos (implementación + estado).
    - En CORBA se controlan por medio de adaptadores de objetos.
    - Tipos según persistencia:
      - Referencias volátiles: Válidas mientras se ejecuta el código que la implementa (proceso, por ejemplo)
      - Referencias persistentes: Válidas a lo largo de varias ejecuciones. Su utilización requiere:
        1. Asociación (referencia, implementación). La referencia contiene la información necesario para consultar el repositorio de implementaciones.
        2. El estado de la instancia se tiene que recuperar: Externalización + reencarnación.

DOM: Tecnologías

- Actualmente existen varias tecnologías de desarrollo de sistemas distribuidos basados en objetos:
  - ANSA (1989-1991) fue el primer proyecto que intentó desarrollar una tecnología para modelizar sistemas distribuidos complejos con objetos.
  - DCOM/COM+ de Microsoft.
  - CORBA de OMG.
  - Tecnologías Java de Sun Microsytems:
    - *Remote Method Invocation (RMI).*
    - *Enterprise Java Beans (EJB).*
    - Jini.
  - Diferentes entornos de trabajo propietarios.

TOM: Motivación

- Los modelos básicos de interacción distribuida (e.g., RPCs) presentan una serie de problemas:
  - Si una operación está compuesta por 2 o más servidores las solicitudes del cliente a ellos son consideradas independientemente (a pesar de ser parte de la misma operación de alto nivel).
    - Cliente → servidor 1
    - Cliente → servidor 2
  - La recuperación ante errores la debe articular particularmente cada cliente.
  - Es difícil asegurar la robustez del sistema ante diferentes escenarios de fallo.

TOM: Modelo de Programación

- Descripción de abstracciones de más alto nivel (soportadas por una sintaxis específicas) denominadas transacciones
- Cumplen las propiedades ACID:
  - *Atomicity* (Atomicidad): La transacción se realiza completa o no se realiza nada.
  - *Consistency* (Consistencia): Los estados anterior y posterior a la transacción son estados estables(consistentes).
  - *Isolation* (Aislamiento): Los estados intermedios de la transacción son sólo visibles dentro de la propia transacción.
  - *Durability* (Durabilidad): Las modificaciones realizadas por una transacción completada se mantienen.

TOM: Conceptos Clave

- Estrategias de control transaccional:
  - Arbitraje por medio de marcas de tiempo.
    - Utilización de cerrojos
    - Concurrencia optimista
- Problemática:
  - Interbloqueos (detección y prevención)
  - Pérdida de rendimiento, granularidad y consumo de recursos
  - Inanición
- Protocolos de control de transacciones:
  - Transacciones que afectan de forma atómica a objetos residentes en varios servidores.
  - Transacciones distribuidas: (compromiso de servidores)
  - Protocolo clásico *two-phase-commit (2PC)*

TOM: Tecnologías

- Monitores transaccionales:
  - Tuxedo (BEA Systems)
  - MTS (Microsoft)
- Especificaciones estándar:
  - X/Open
- Por lo general, los middleware existentes soportan funcionalidades adicionales para su integración con otro tipo de plataformas.

MOM: Motivación

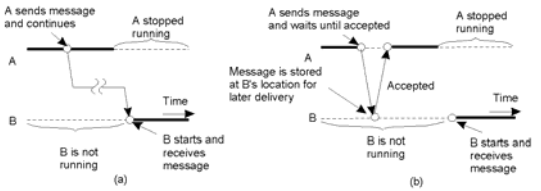
- ¿Qué ocurre si cliente y servidor no se encuentran ejecutando a la vez?
  - Requiere un elemento que recepcione la petición, la guarde y se la despache al servidor cuando éste se encuentre disponible.
  - Se pueden implementar políticas de prioridad, caducidad de mensajes o filtrado de los mismos.
  - Permite modelos de negocio cuya construcción es una secuencia de pasos (ejecutados por diferentes servidores) para resolver una tarea: *Workflows.*

MOM: Modelo de Programación

- Dentro de los que es un modelo asíncrono de paso de mensajes se pueden ver tres posibles submodelos:
  - Modelo de mensajes asíncronos: Un elemento del sistema envía un mensaje a un único receptor (que puede ser entregado a posteriori):
    - Cardinalidad 1:1
  - Modelo observador/observable: Se informa a determinados elementos del sistema cuando otro elemento cambia su estado:
    - Cardinalidad 1:N
  - Modelo publicación/subscripción: Los elementos no se relacionan entre sí, se inscriben a canales de eventos (como generador o receptor de mensajes).
    - Cardinalidad N:M

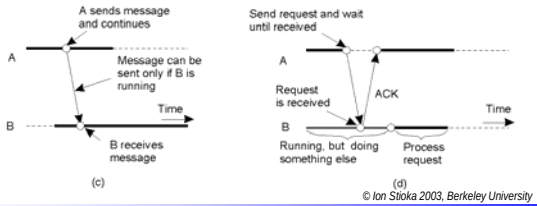
MOM: Modelo de Programación

- También hay determinados niveles de sincronización y persistencia:
  a) Persistente asíncrono: El mensaje se encola en un almacenamiento externo hasta que puede ser entregado, pero el emisor no tiene certeza de su entrega.
  b) Persistente síncrono: El mensaje se almacena en la localización donde está el receptor (aunque no esté activado), y se confirma al emisor la entrega



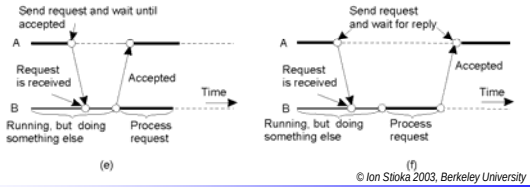
MOM: Modelo de Programación

- Niveles de sincronización y persistencia (cont.):
  c) Volátil asíncrono: El envío del mensaje sólo es posible si el receptor existe y está listo para recibir el mensaje.
  d) Volátil síncrono (en recepción): El emisor se bloquea hasta que se confirma la recepción del mensaje (el receptor debe existir, en cualquier caso)



MOM: Modelo de Programación

– Niveles de sincronización y persistencia (cont.):
  e) Volátil síncrono (en entrega): El emisor se bloquea hasta que se confirma que al receptor se le ha entregado el mensaje (el receptor debe existir, en cualquier caso)
  f) Volátil síncrono (en respuesta): El emisor se bloquea hasta que el receptor responde al mensaje (el receptor debe existir, en cualquier caso).



MOM: Conceptos Clave

Direccionamiento:
– El direccionamiento de los mensajes se hace por identificador de cola.
– Permite una gran escalabilidad con "estafetas" intermedias en el envío de los mensajes.
– Se pueden articular mecanismos de tolerancia a fallos en los receptores.

Respaldo del *broker* de mensajes:
– El *broker* puede usar almacenamiento estable para no perder mensajes si se cae.
– Tolerancia a fallos a nivel de cola de mensajes.
– Permite gestionar acuses de recibos o control transaccional.

MOM: Tecnologías

- Middleware completos de colas de mensajes:
  – MQSeries (IBM)
  – MSMQ (Microsoft)
- Servicios de colas de mensajes de otros entornos:
  – JMS (Java)
  – COSS Event/Notification (CORBA)
- Un caso especial de los middleware orientados a mensajes son los SOM (Stream Oriented Middlewares):
  – Distribución de *streams* de datos (e.g., vídeo o audio).
  – Gestión de QoS.
  – Problemas de distribución multicast.
  – Soporte RSVP o similares.

Sistemas Operativos Distribuidos

Middlewares: Arquitecturas Orientadas a Servicios

- **SOA**
- **ESB**
- **EAI**

Concepto de Servicio

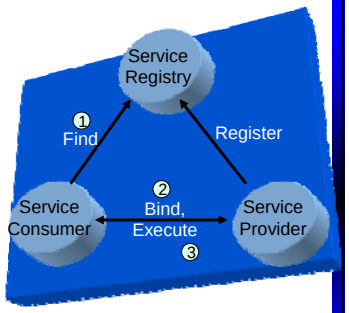
- Un servicio es una tarea (de negocio) que cumple una serie de características:
  - Es repetible (se puede realizar varias veces).
  - Es auto-contenida (sin dependencias visibles con otros servicios, poco acoplada).
  - Está disponible (permanece a la espera hasta que se activa).
  - Es de *grano-grueso* (tiene una entidad apreciable).
- Que tiene asignadas una serie de propiedades:
  - Calidad del servicio.
  - Coste del servicio.
  - Qué debe general el servicio (pero no cómo debe hacerlo).

Contexto y Estado de un Servicio

- Los servicios deben ser independientes de contexto:
  - No quiere decir que no tengan estado (todo lo contrario por lo general).
  - Quiere decir que sean *débilmente acoplados* (reutilizables en diferentes contextos).
- Esta característica hace que se puedan combinar en diferentes escenarios:
  - La combinación de servicios en sí puede ser otro servicio:
    - Reservar un vuelo
    - Reservar un hotel
    - Reservar un coche de alquiler

Interacción con Servicios (I)

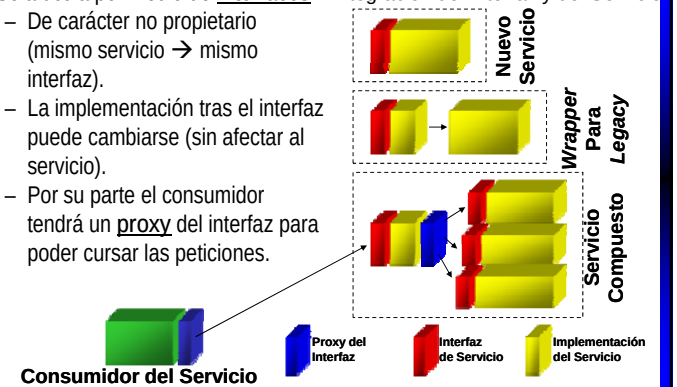
- Cada servicio encapsula un proceso de negocio (unidad básica de negocio).
- Participantes:
  - Proveedor de Servicio:
    - Servicio sin contexto y transparente (localización)
  - Consumidor del Servicio
    - Usa los servicios del proveedor para sus procesos de negocio
  - Registro de Servicios
    - Pone en contacto a proveedor y consumidor.
- El consumidor del servicio debe realizar: *Find, Bind, Execute*



Estructura genérica de los participantes en una solicitud de servicios

Interacción con Servicios (II)

- Se articula por medio de interfaces:
  - De carácter no propietario (mismo servicio → mismo interfaz).
  - La implementación tras el interfaz puede cambiarse (sin afectar al servicio).
  - Por su parte el consumidor tendrá un proxy del interfaz para poder cursar las peticiones.



Interacción con Servicios (y III)

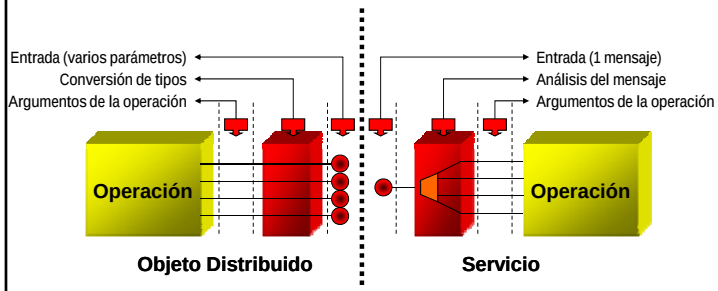
- El modelo de interacción *tiende a ser* más flexible que *Request/Response*:
  - El consumidor de servicios expresa sus intenciones (*Intent*)
  - El proveedor de servicios indica su oferta (*Offers*).
  - De alguna forma se articula una mediación para ajustar intenciones y ofertas. Se puede hacer por medio de:
    - Buscar la oferta (entre varias disponibles) que mejor se ajusta a la intención del consumidor.
    - Anunciar cada oferta de diferentes formas (parámetros) de forma que pueda ajustarse a diferentes intenciones.
- Los protocolos *Intent/Offers* requieren la existencia de directorios de servicios.

Escenario Ideal

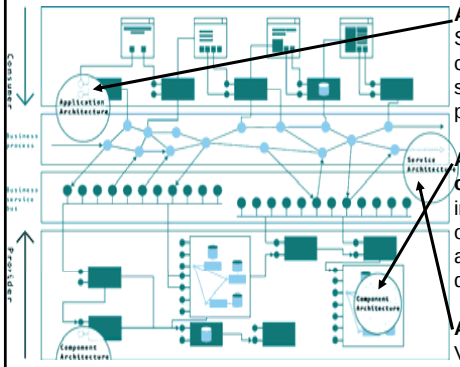
- La orientación a servicios permite:
  - A los solicitantes:
    - Combinar con libertad escenarios y contextos en los cuales se usen y combinen servicios.
    - Expresar sus intenciones sobre una necesidad de servicio más que una petición (sin flexibilidad).
  - A los proveedores:
    - Optimización del servicio como centro de atención en el desarrollo. Mejor calidad de servicio y robustez.
    - Oferta de sus servicios a un espectro más amplio de clientes.
    - Resolución de problemas de localización y conexión.
- Respuesta ideal para B2B o B2C y otras interacciones.

SOA vs. Objetos Distribuidos

- A nivel conceptual SOA difiere de los objetos distribuidos en que el interfaz de acceso a un servicio es genérico recibe un mensaje en un formato complejo, en lugar de más o menos parámetros predeterminados por el interfaz.



Arquitecturas



Arquitectura de aplicación: Solución de cara al negocio con consume uno o varios servicios de diferentes proveedores.

Arquitectura de componentes: La implementación de los objetos de negocio asociado a una tecnología determinada.

Arquitectura de servicios: Visión lógica de los servicios disponibles.

© David Sprott and Lawrence Wilkes , CBDI Forum

¿Cómo se Implementa una SOA?

- SOA es una arquitectura software genérica no una tecnología, su implementación puede hacerse por medio de tecnologías concretas:
  - Servicios Web
  - CORBA
  - REST
  - Tecnologías Java (JBI, JES)
  - Otras: (ESB,
- Atención: Ni toda SOA está hecha en WS ni todo WS implementa una SOA.

Tecnologías/Conceptos Asociados (I)

Los otros conceptos relacionados son:

- ESB (*Enterprise Service Bus*) tipo de arquitectura basada en estándares que proporciona un motor de mensajes dirigido por eventos. Puede no implementarse con servicios web y puede valer como plataforma para SOA (aunque no exclusivamente). Implica una tecnología de implantación.
- SOMA (*Service-Oriented Modelling and Architecture*) enfoques formales y metodologías para el desarrollo de soluciones SOA (propuesto por IBM).
- EII (*Enterprise Information Integration*) proceso de abstracción de datos para el acceso uniforme a los mismos. Incluye y engloba diferentes tecnologías de acceso a datos heterogéneas (ODBC, JDBC, ADO.NET, XQuery) para diferentes procesos de consumo de los mismo desde los clientes.

Tecnologías/Conceptos Asociados (y II)

Otros conceptos relacionados son:

- BPM (*Business Process Management*) estrategias de gestión de procesos de negocio. Por medio de un sistema de reglas (o redes) se define la dinámica del negocio, por medio de un motor dichas reglas se consultan y aplican. Por último se definen reglas de monitorización del proceso.
- EAI (*Enterprise Application Integration*) propuesta arquitectónica de integración de servicios (anterior a SOA), orientada a los sistemas de información de la propia empresa. La finalidad es articular aplicaciones del tipo CRM, BI o *Supply Chain Management*. Se basan en dos tipos de modelos:
  - Hub: Sistema central de recepción de información y reenvío.
  - Bus: En un sistema de mensajería del tipo MoM (*Message oriented middleware*)

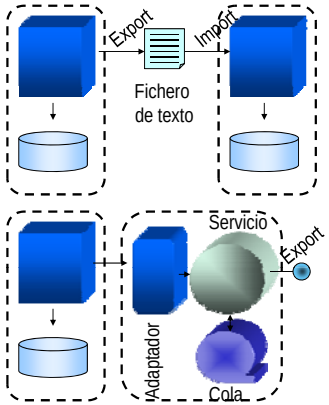
Sistemas Operativos Distribuidos

Integración de Tecnologías

- ***Legacy Systems***
- **Integración a nivel de empresa**

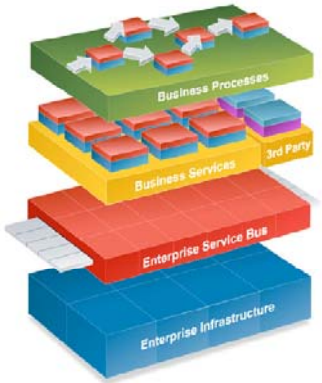
Legacy Systems

- Problemática:
  - El grado de acoplamiento es diferente.
  - Los modelos originales de interacción son en base a ficheros.
  - En muchos casos son modelos *batch*.
  - Otros casos, sustitución de llamadas tipo RPC por WS.
  - Se pueden dar casos de accesos directos a bases de datos (nuevos servicios).



Integración a Nivel de Empresa

- Requiere:
  - Resolución de los problemas de interacción (desarrollo con tecnologías más avanzadas).
  - Definición de la secuencia de operaciones de un proceso de negocio.
  - Empaquetado de las operaciones unitarias en servicios.
  - Adopción de una tecnologías de integración de servicios (ESB).



© Cape Clear, 2005

Sistemas Operativos Distribuidos

Coordinación de Servicios

- **Orquestación**
- **BPMS**
- **BPEL**

Orquestación vs. Coreografía

Existen dos términos relacionados en la gestión de servicios:

- Orquestación (*Orchestation*): Representa la ordenación y gestión de servicios desde la perspectiva de un participante (un proceso de negocio). Existe un solo coordinador.
- Coreografía (*Choreography*): Tiene un ámbito más amplio e implica la coordinación de todos los participantes de un sistema complejo entero. Existe una política en la que varios elementos se coordinan y se ajustan entre sí.

- Una diferencia muy sutil (en el plano teórico).
- En ambos casos representan definiciones declarativas de cómo se deben realizar uno o varios procesos, denominadas a veces como reglas de negocio (*business rules*)

Gestión de Negocio (I)

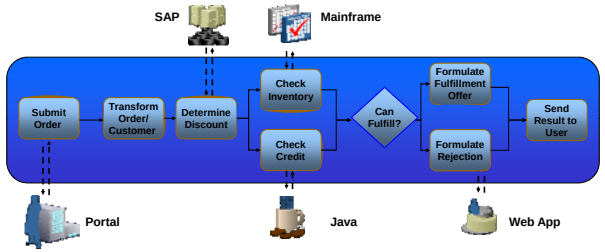
- Los sistemas que implementan BPM, denominados habitualmente Business Process Management System (BPMS) utilizan lenguajes de descripción de procesos:
  - BPEL (*Business Process Execution Language*) lenguaje XML de orquestación de servicios. Extensión de:
    - WSFL (de IBM)
    - XLANG (de BizTalk-Microsoft).

    Actualmente estandarizado por OASIS.
  - Otros lenguajes son (BPML – *Business Process Modeling Language* [anterior], y WS-CDL – *Web Services Choreography Description* [sin implementación]).

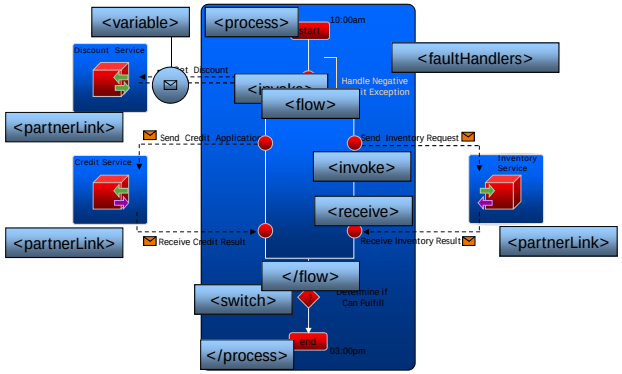
Gestión de Negocio (II)

BELP:

- Define procesos de negocio interoperables y protocolos de negocio.
- Permite componer servicios nuevos a partir de otros.
- Define estructuras de control (if…then…else, while, sequence, flow)
- Gestiona variables del proceso y mensajes (entrantes y salientes).

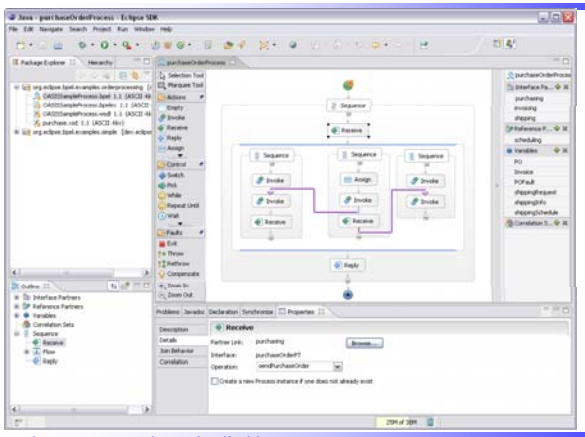


Gestión de Negocio (III)

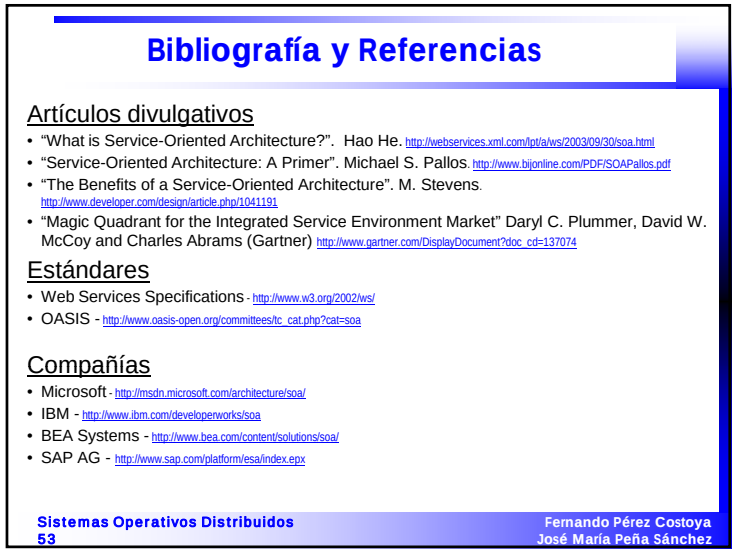


© Oracle, SOA – Oracle Development Day

Gestión de Negocio (y IV)



Edición de procesos BPEL por medio de un entorno gráfico (Eclipse)
© BPEL project
http://www.eclipse.org/bpel/



Bibliografía y Referencias

Artículos divulgativos

- "What is Service-Oriented Architecture?". Hao He. http://webservices.xml.com/lpt/a/ws/2003/09/30/soa.html
- "Service-Oriented Architecture: A Primer". Michael S. Pallos. http://www.bijonline.com/PDF/SOAPallos.pdf
- "The Benefits of a Service-Oriented Architecture". M. Stevens. http://www.developer.com/design/article.php/1041191
- "Magic Quadrant for the Integrated Service Environment Market" Daryl C. Plummer, David W. McCoy and Charles Abrams (Gartner) http://www.gartner.com/DisplayDocument?doc_cd=137074

Estándares

- Web Services Specifications - http://www.w3.org/2002/ws/
- OASIS - http://www.oasis-open.org/committees/tc_cat.php?cat=soa

Compañías

- Microsoft - http://msdn.microsoft.com/architecture/soa/
- IBM - http://www.ibm.com/developerworks/soa
- BEA Systems - http://www.bea.com/content/solutions/soa/
- SAP AG - http://www.sap.com/platform/esa/index.epx

Sistemas Operativos Distribuidos
53
Fernando Pérez Costoya
José María Peña Sánchez