CAPÍTULO 2
Conceptos de servicios en los lenguajes de programación

Dalila Romero y Patricia Bazán

El concepto de servicios y su uso dentro de los lenguajes de programación conlleva analizar la evolución tecnológica desde los objetos distribuidos, sus verdaderos antecesores. Se ha introducido este concepto en [Bazán P, 2017] y analizado los objetos en el marco de los sistemas distribuidos.

En este capítulo se amplían algunos de los conceptos allí planteados; se analizan en detalle los estándares y tecnologías habilitantes que precedieron a los servicios y de qué manera estos cobran protagonismo en los lenguajes de programación como piezas constructivas de software.

A medida que fue aumentando la complejidad de las soluciones distribuidas fue necesario incorporar conceptos, metodologías y buenas prácticas en la construcción de software y llevarlas al entorno distribuido.

El concepto de objeto en el sentido clásico de la Programación Orientada a Objetos y que tiene sus raíces principales en el lenguaje Smalltalk, se mantiene a nivel de diseño de aplicaciones, pero cambia algunas características cuando se lo lleva a un entorno distribuido.

Un objeto distribuido es un objeto (componente funcional con estado interno, comportamiento, heredable y encapsulado), que por estar en un entorno distribuido también debe ser:

- Transaccional: los cambios de estado que produce se deben ejecutar completamente o no ejecutarse.
- Seguro: debe evitar vulnerabilidades que corrompa el estado del sistema.
- Lockeable: debe fijar un cerramiento que garantice su ejecución correcta ante accesos concurrentes.
- Persistente: su estado interno debe conservarse más allá de su ciclo de vida.

Esta definición de objeto distribuido nos conduce a la necesidad de contar con una infraestructura que facilite la comunicación entre este tipo de componentes, logrando a su vez que logren esta comunicación a través de distintas plataformas de ejecución, sistemas operativos y lenguajes de programación.

Así surge el estándar CORBA (*Common Object Request Broker Architecture*)[1], definido por la OMG (*Object Management Group*) y que surge de la iniciativa de un consorcio de empresas interesadas en establecer una arquitectura interoperable sobre la base del paradigma orientado a objetos. CORBA se constituye así en un ejemplo de sistema distribuido basado en objetos.

Entre los elementos distintivos del estándar - y que se puede decir que sentaron las bases para lo que se conoce actualmente como web services - encontramos:

- Un lenguaje de especificación de interface, IDL *Interface Definition Language*, que define los límites de los componentes o sea las interfaces contractuales con los potenciales clientes.
- Un bus de objetos u ORB (*Object Request Broker*) que comunicar objetos independientes de su locación. El cliente se despreocupa de los mecanismos de comunicación con los que se activan o almacenan los objetos servidores.

Provee un vasto conjunto de servicios de middleware distribuido y tiene claramente un mecanismo de comunicación mucho más complicado que los clásicos RPC (*Remote Procedure Call*) y MOM (*Message Oriented Middleware*).

La Figura 13 muestra el mecanismo de interoperabilidad entre cliente y servidor a través de un bus de objetos. Por ejemplo, un componente escrito en C, expone sus interfases en IDL y se comunica con el cliente que genera un requerimiento al ORB. Por su parte, del lado del servidor, puede haber un componente escrito en Java que también expone sus interfases y se comunica con el ORB como el proveedor de un servicio. El ORB es un middleware que vincula cliente con servidor.

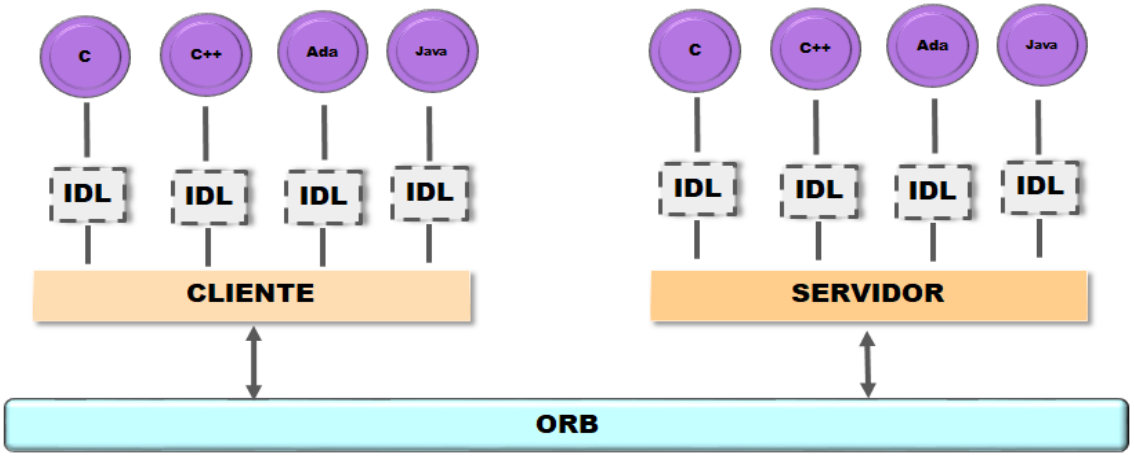

Figura 13. *ORB e interoperabilidad cliente/servidor*

[1] http://www.corba.org/

2.1 - CORBA - Arquitectura de Referencia

Hacia mediados de 1990 surge la arquitectura de referencia CORBA para gestionar objetos en un entorno distribuido. Los 4 elementos de la arquitectura incluyen: 1- ORB que define el bus de objetos de CORBA, 2- *Common Object Services* define los objetos a nivel sistema operativo que extienden el bus, 3- *Common Facilities* define aplicaciones horizontales y verticales usadas de manera directa por los objetos de negocio, 4- *Application Objects*, son los objetos de negocio y aplicaciones que usan la infraestructura.

La Figura 14 muestra gráficamente la arquitectura descripta.

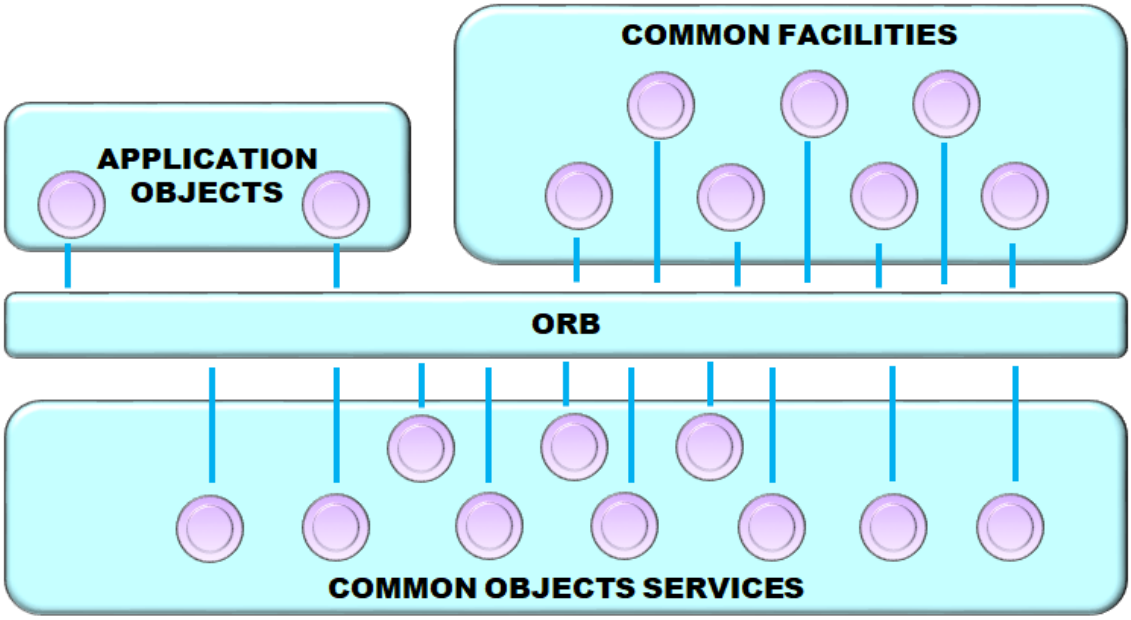

Figura 14. *Arquitectura de Gestión de Objetos Distribuidos*

El ORB en particular constituye un middleware basado en mecanismos de invocación remota similares al RPC. La diferencia entre ellos es equivalente a la diferencia entre el mecanismo *call-return* y *request-reply* específicos de la vinculación programa-subrutina y objeto-mensaje del paradigma procedural y orientado a objetos, respectivamente. En la Figura 15 se observa la diferencia entre la invocación a una función específica con RPC versus la invocación a un método de un objeto específico. En este sentido, el polimorfismo de los objetos permitirá que la respuesta sea diferente según el objeto que contenga el método en cuestión.

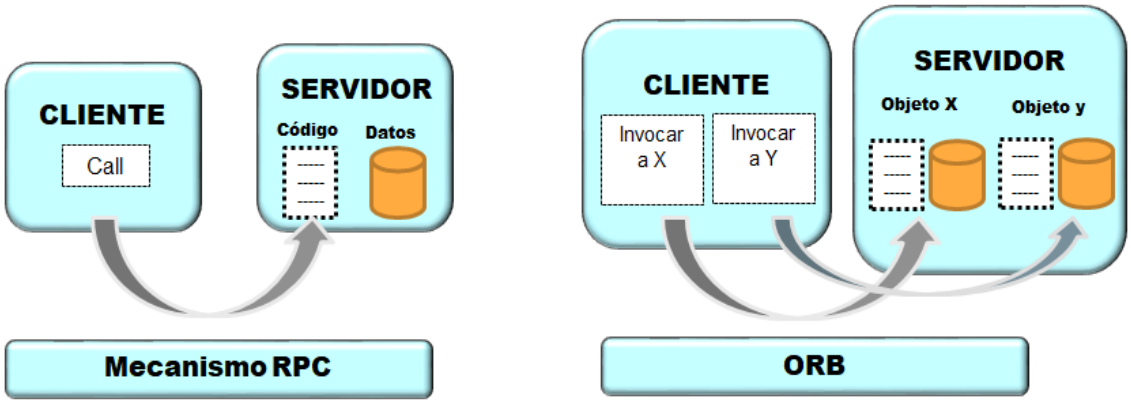

Figura 15. *RPC vs ORB*

Por su parte, un objeto de negocio se define como un activo de software en el dominio de aplicación, que posee, nombre, definición, atributos, comportamiento, relaciones y restricciones. En este sentido, y a la luz del concepto actual de "servicio" un objeto de negocio puede considerarse un antecedente del mismo.

La Figura 16 ayuda a comprender la ubicación del objeto de negocio en la pila de infraestructura de los objetos distribuidos.

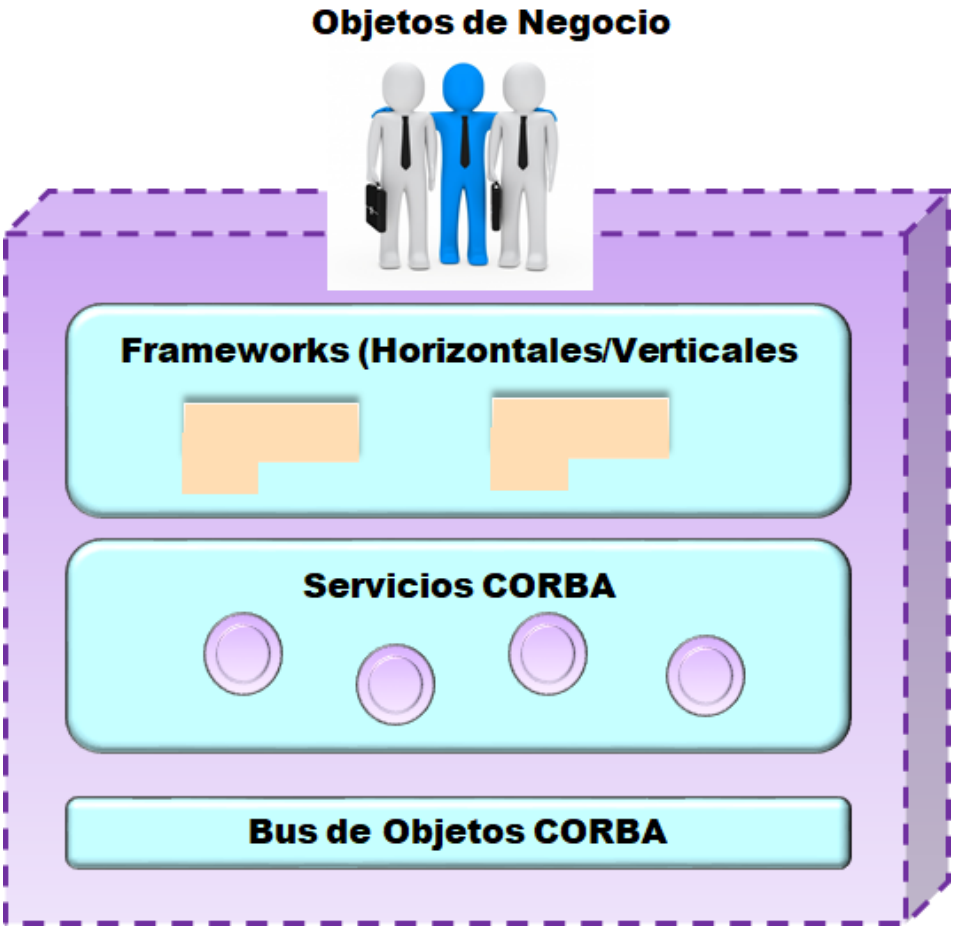

Figura 16. *Los objetos de negocio en la pila de CORBA*

2.2 - Servicios en los lenguajes de programación

Además de la construcción de estándares como CORBA para mejorar la interoperabilidad, el advenimiento de las plataformas basadas en Web favoreció ampliamente el desarrollo de aplicaciones distribuidas a gran escala y volvió a poner en evidencia la necesidad de integrarse e interoperar.

Como se ha desarrollado ampliamente en el Capítulo 3**,** la integración punto a punto, por adaptadores y por mediador de mensajes conforman una arquitectura accidental (se construye para cada caso), de mantenimiento costoso y lento y difícil de gestionar, monitorizar y extender.

El concepto de servicio y su arquitectura de referencia (SOA - *Services Oriented Architecture*) representa un cambio radical en la relación entre el mundo del negocio y el área de tecnología de la información. SOA constituye mucho más que un conjunto de productos aglutinados por una tecnología. Es un nuevo enfoque en la construcción de sistemas de IT que permite a las empresas aprovechar los activos existentes y abordar fácilmente los inevitables cambios en el negocio.

Si bien la industria del software ha venido enfocándose en una arquitectura orientada a servicios desde hace más de 20 años con la noción de reusabilidad y su aplicación a la construcción de software, lo cierto es que en los últimos años esto se ha fortalecido con la definición de estándares y la conformación de consorcios que participan en su definición.

Características de SOA	
Reuso de componentes de software existentes	Este concepto se encuentra vinculado fundamentalmente con la idea de federación de aplicaciones. Un entorno informático federado es aquel en el cual los recursos y aplicaciones se encuentran unidos manteniendo autonomía y autogobierno. La exposición de servicios mediante SOA facilita la federación y por ende el reuso.
Interoperabilidad entre aplicaciones y tecnologías heterogéneas	Este concepto con la capacidad de compartir información entre aplicaciones y también con la independencia de las plataformas. La exposición de servicios favorece la interoperabilidad y permite el uso de la funcionalidad que otorga un proveedor con independencia de la plataforma de implementación.
Flexibilidad para componer, integrar y escalar soluciones	La flexibilidad de una solución se vincula con un mayor alineamiento entre el dominio o negocio y la tecnología. Este alineamiento busca que el sistema reaccione rápidamente a los cambios del entorno. Los servicios de grano fino facilitan la evolución del sistema y otorgan agilidad a la organización.
Conceptos SOA	
Servicios	Piezas funcionales que resuelven un aspecto del negocio. Pueden ser simples (almacenar los datos de un cliente) o compuestos (el proceso de compra de un cliente).
Enterprise Service Bus (ESB)	Infraestructura que habilita una alta interoperabilidad entre servicios en un contexto distribuido.
Bajo acoplamiento	Reducción de dependencias entre sistemas.

Recapitulando, CORBA ha constituido una base conceptual muy importante en la aparición de los conceptos de SOA.

El concepto de objeto y el de servicio guardan muchas similitudes en lo que se refiere a su modularidad y capacidad de reuso, pero también se diferencian, fundamentalmente en el modelo de interacción que presentan.

Los objetos se definen con gran cohesión, en el sentido que existe gran asociación entre los métodos que atienden el objeto y una fuerte ligadura funcional, pero con bajo acoplamiento para minimizar el impacto de los cambios de una clase.

Los servicios, por su parte, necesitan un menor acople y no requieren que se conozca su nombre para utilizarlos porque poseen abundante meta-información.

Dentro de un servicio existen operaciones, no habiendo, a priori, asociaciones entre ellos, por lo tanto, existe menos cohesión que en los objetos.

Un servicio difiere de un objeto o un procedimiento porque se define en función de los mensajes que intercambia con otros servicios.

Los Web Service constituyen una manera apropiada (no la única) de implementar interfaces de aplicaciones basadas en servicios y permiten que diferentes aplicaciones, realizadas con diferentes tecnologías, y ejecutándose en una variedad de entornos, puedan comunicarse e integrarse.

En este sentido, los Web Services constituyen un primer punto para construir software basado en servicios, siendo estos últimos, escritos en cualquier lenguaje de programación.

Para lograr un análisis más específico de los lenguajes de programación en el mundo de los servicios, vamos a introducirnos en BPEL (*Business Process Execution Language*) es un lenguaje concebido por Oracle, BEA Systems, IBM, SAP y Microsoft entre otros y estandarizado por OASIS para la composición de servicios web. La especificación 2.0 de este lenguaje fue descrita por el consorcio OASIS a finales de marzo del 2007, el cual se encarga de definir estándares a nivel mundial.

2.3 – BPEL Business Process Execution Language

El lenguaje BPEL es un lenguaje de alto nivel que lleva el concepto de servicio un paso adelante, al proporcionar métodos de definición y soporte para flujos de trabajo y procesos de negocio. Está definido en XML (*eXtensible Markup Language*) y diseñado para orquestar procesos de forma automática.

El origen de BPEL se remonta a 2001 cuando IBM y Microsoft definieron WSFL (*Web Services Flow Language*) y XLANG, respectivamente. El posterior crecimiento de iniciativas de código abierto y el exitoso *BPMI.org* -que define estándares para la gestión de procesos de negocio que abarca varias aplicaciones y dominios de negocio-, guió a IBM y Microsoft a combinar sus soluciones en un nuevo lenguaje BPEL4WS (*BPEL for Web Services*).

En abril de 2003 surge BPEL4WS y es enviado a OASIS para su estandarización quien en setiembre de 2004 dio origen a WS-BPEL 2.0.

En junio de 2007, otro importante conjunto de empresas publicaron BPEL4People que describe cómo especificar tareas humanas en procesos BPEL.

Con estos antecedentes los objetivos principales que persigue BPEL se pueden resumir en los siguientes:

- Es una plataforma para ejecutar procesos BPEL que se exponen a sus clientes como servicios Web y que requieren la definición de su WSDL.
- Para comenzar el proceso, el cliente deberá invocar una operación del mismo que normalmente será una sola (único punto de entrada).
- Cada invocación genera una nueva instancia como mecanismo básico de ciclo de vida y que posee operaciones para su soporte como “suspender”, “continuar”, “terminar”.
- Provee funciones de manipulación simple de datos solo para mantener el flujo de datos y el flujo de control.
- Soporta un método de identificación de instancias de procesos que permita la definición de identificadores de instancias a nivel de mensajes de aplicaciones. Los identificadores de instancias deben ser definidos por socios (*partner*) y pueden cambiar [BPEL, 2020].

BPEL para orquestar servicios

Con respecto a la orquestación de servicios, es BPEL el encargado de consumir varios servicios en un orden especificado y realizar una función muy concreta, para entenderlo mejor, veamos un ejemplo:

Supongamos que estamos haciendo uso de un sitio de compras on line que nuclea diferentes proveedores y que también ofrece distinto tipo de servicios.

El cliente que hace uso del sitio solo debe indicar qué producto o servicio comprar y elegir un medio de pago. El sistema responde con un mail indicando el éxito de la compra y, eventualmente, brinda un número de rastreo. Cabe mencionar que se está considerando el caso de una compra exitosa, dado que se pretende mostrar el flujo de operaciones subyacentes a la acción de compra de un producto o servicio.

Más allá de si las validaciones y operaciones siguientes se realizan en el momento o de manera diferida, está claro que desde el sitio de compras se deben realizarlas y que no todas dependen de él.

El siguiente diagrama muestra la secuencia de operaciones que el sitio realiza ante la compra de un producto o servicio.

1. Registrar la compra
2. Reservar stock con el proveedor
3. Gestionar pago con el medio de pago
4. Confirmar la compra

Tanto la operación 2 como 3 no dependen del sitio de compras sino de servicios que deben ser expuestos tanto por el proveedor como por el medio de pago, a través de, por ejemplo, un Servicio Web. La Figura 16 muestra la interacción de las cuatro operaciones.

La reserva de stock y la gestión del pago son operaciones externas al sitio de compras y que deben ser resueltas por sus respectivos propietarios (proveedor o medio de pago). Eventualmente, ambos también requerirán una respuesta del sitio de compras para convalidar la reserva del stock y el pago, pero estos dos escenarios no están siendo considerados por este ejemplo para simplificarlo.

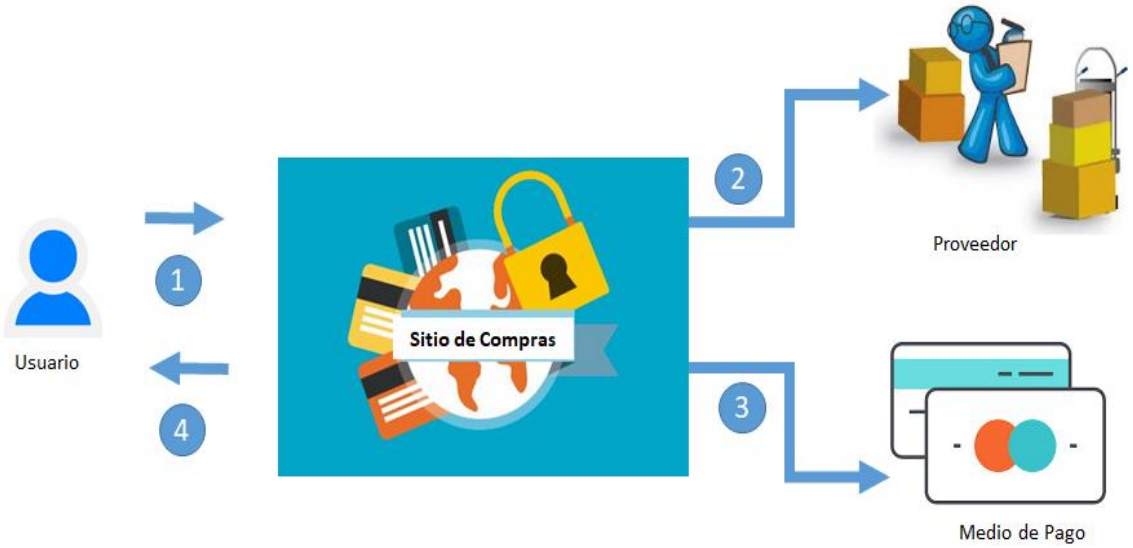

Figura 17. *Operaciones de compra en un sitio de compras con proveedores y medios de pago externos.*

¿Cómo encaja BPEL en este escenario?

Con BPEL se puede crear un solo servicio que reciba todos los parámetros necesarios para realizar la operación como los datos de la compra y los datos del medio de pago, para realizar la reserva de stock y la gestión del pago, guardar el registro de la compra y regresar al usuario la confirmación de su compra.

Si bien esto te puede parecer algo que cualquier lenguaje pueda realizar la realidad es que BPEL está diseñado para estos escenarios lo que permite una programación mucho más rápida.

BPEL como lenguaje de programación

Un proceso BPEL especifica el orden exacto en el que deben invocarse los servicios web participantes, tanto de forma secuencial como en paralelo. Con BPEL se puede expresar un comportamiento condicional, por ejemplo la invocación de un servicio web puede depender del valor de una invocación previa. También es posible construir bucles, declarar variables, copiar y asignar valores, y definir manejadores de fallos, entre otras cosas. Combinando todas estas construcciones se definen procesos de negocio complejos de una forma algorítmica. De hecho, debido a que los procesos de negocio son esencialmente grafos de actividades, podría ser útil expresarlos utilizando diagramas de actividad UML.

Para entender mejor el comportamiento de BPEL como lenguaje de programación, es necesario hacer hincapié en las siguientes características:

Transformar al XML en un lenguaje ejecutable

La máquina de servicios BPEL es un componente JBI (*Java Business Integration*) que satisface el estándar JSR 208 (*Java Specification Request*) y proporciona servicios para ejecutar procesos de negocio desarrollados con WS-BPEL 2.0. Para desplegar un proceso BPEL, es preciso añadirlo como un módulo JBI en un proyecto de composición de aplicaciones (*Composite Application Project*) [Orquestación BPEL, 2012].

La máquina de servicios BPEL se inicia juntamente con el servidor de aplicaciones, por lo que para desplegar y ejecutar procesos BPEL será necesario tener en marcha el servidor de aplicaciones. La máquina de servicios BPEL está representada como el componente JBI sun-bpel-engine [BPEL User Guide, 2007] del servidor de aplicaciones.

Una vez desarrollado el proceso BPEL, el último paso es su despliegue en una BPEL engine. Una alternativa mostrada en [Orquestación BPEL, 2012] es usar NetBeans como herramienta para desarrollar y desplegar procesos BPEL, por lo tanto, vamos a explicar cómo se realiza el proceso de despliegue y pruebas, utilizando NetBeans 7.1

Netbeans utiliza el entorno de ejecución JBI. Dicho entorno de ejecución incluye varios componentes que interactúan utilizando un modelo de servicios. Dicho modelo está basado en el lenguaje de descripción de servicios web WSDL 2.0. Los componentes que suministran o consumen servicios dentro del entorno JBI son referenciados como máquinas de servicios (*Service Engines*). Uno de estos componentes es la máquina de servicios BPEL (BPEL *Service Engine*), que proporciona servicios para ejecutar procesos de negocio. Los componentes que proporcionan acceso a los servicios que son externos al entorno JBI se denominan *Binding Components*. Toda esta descripción se grafica en la Figura 18.

Estructura de un programa BPEL

Un proceso BPEL puede ser síncrono o asíncrono. Un **proceso BPEL síncrono** bloquea al cliente (aquél que usa el proceso BPEL) hasta que finaliza y devuelve el resultado a dicho cliente. Un **proceso asíncrono** no bloquea al cliente. Para ello utiliza una llamada *callback* que devuelve un resultado (si es que lo hay). Normalmente utilizaremos procesos asíncronos para procesos que consumen mucho tiempo, y procesos síncronos para aquellos que devuelven un resultado en relativamente poco tiempo. Si un proceso BPEL utiliza servicios Web asíncronos, entonces el propio proceso BPEL normalmente también lo es.

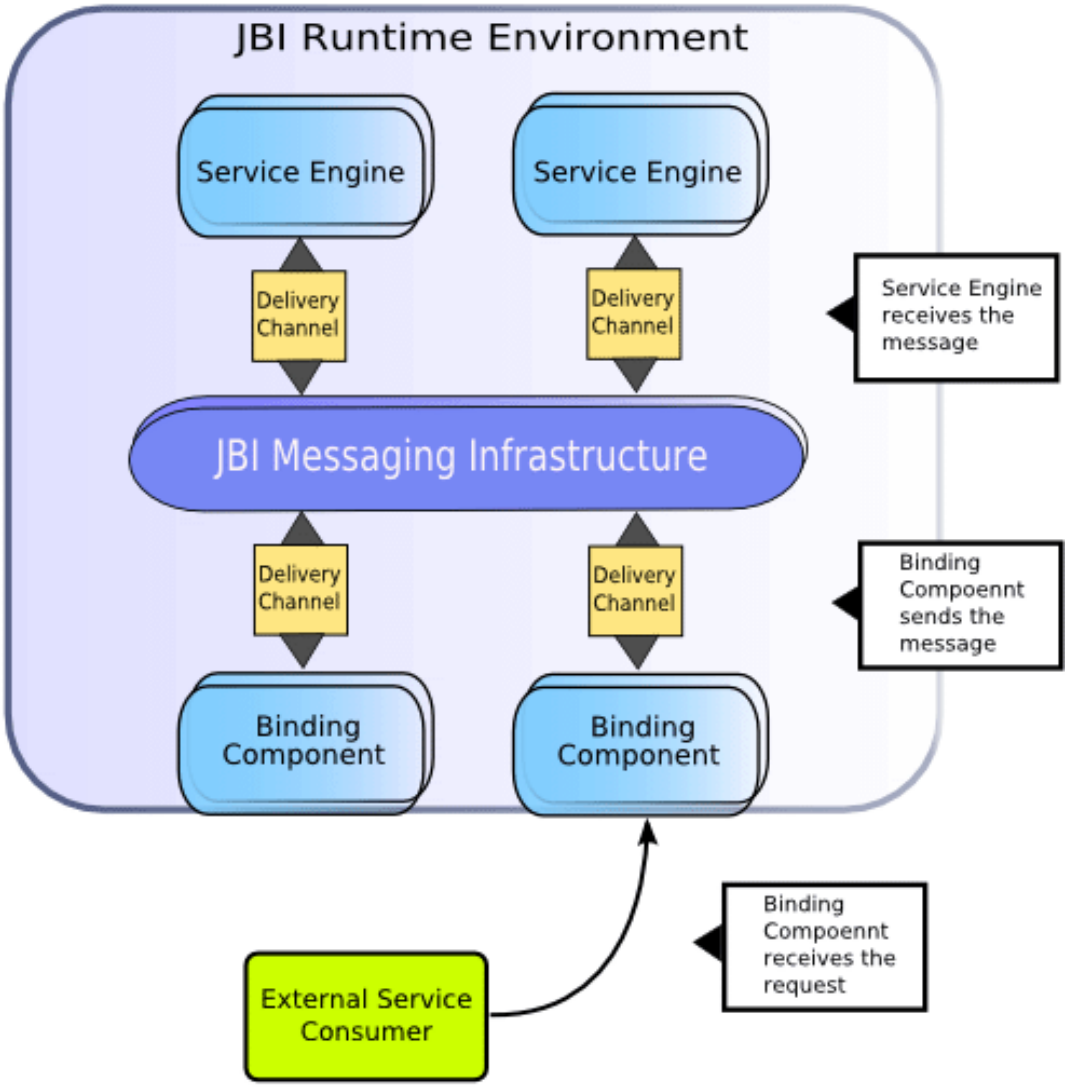

Figura 18. *Ejemplo de entorno de ejecución JBI*

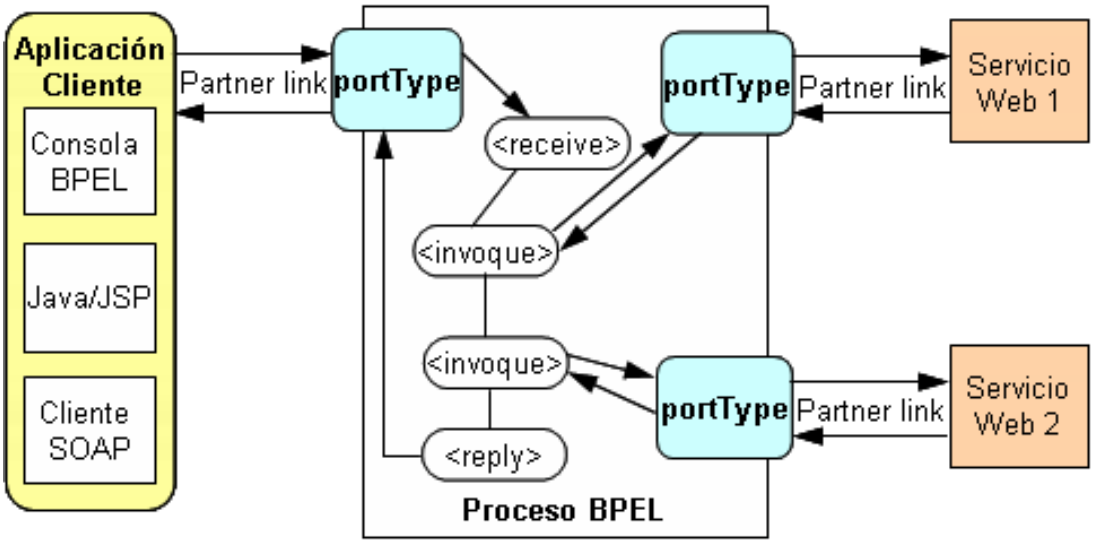

Figura 19. *Estructura de un Proceso BPEL*

La estructura básica de un documento (fichero con extensión **.bpel**) que define un proceso BPEL es la siguiente:

```
<process name="nameProcess" ... >

 <partnerLinks>
  <!-- Declaración de partner links -->
 </partnerLinks>

 <variables>
  <!-- Declaración de variables -->
 </variables>

 <sequence>
   <!-- Cuerpo principal de la definición del proceso BPEL -->
 </sequence>

</process>
```

En la Tabla 3 se resumen algunas de las etiquetas que estructuran un proceso BPEL, su sintaxis y su descripción -en [BPEL User Guide, 2007] se encuentra una guía completa-.

Tabla 3. Resumen de etiquetas principales que estructuran un proceso BPEL

Etiqueta	Sintaxis	Descripción
<process>	<process name="nameProcess" targetNamespace= "http://..." xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/" xmlns:sw1="http://..." <!-- namespace del servicio Web sw1 --> xmlns:sw2="http://..." <!-- namespace del servicio Web sw2 --> ... > ... </process>	Incluyen los espacios de nombres, tanto el objetivo (**targetNameSpace**), como los *namespaces* para acceder a los WSDL de los servicios Web a los que invoca. También se declara el *namespace* para todas las etiquetas y actividades BPEL.
<partnerLinks>	<partnerLinks> <partnerLink name="ncname" partnerLinkType="qname" myrole="ncname" partnerRole="ncname"> </partnerLink> </partnerLinks>	Cada proceso BPEL tiene al menos un partner link cliente, debido a que tiene que haber un cliente que invoque al proceso BPEL. Por otro lado, un proceso BPEL tendrá (normalmente) al menos un partner link a quién invoque.
<partnerLinksType>	<!-- Extracto de Saludo.wsdl --> <partnerLinkType name="MyPartnerLinkType">	Especifica la relación entre dos servicios, definiendo el rol que cada

	<role name="ProveedorServicioSaludo"> portType="SaludoPortType"/> </role> </partnerLinkType> <!-- Extracto de Saludo.bpel --> <partnerLinks> <partnerLink name="cliente" partnerLinkType="MyPartnerLinkType" myRole="ProveedorServicioSaludo"/> </partnerLinks>	servicio implementa. Es decir, declara **cómo** interaccionan las partes y lo que cada parte ofrece. Los nombres de los roles son cadenas de caracteres arbitrarias. Cada **rol** especifica exactamente un tipo *portType* WSDL que debe ser implementado por el servicio correspondiente.

<variables>	<variables>
	<variable name="nombreVar"
	messageType="qname" //tipo mensaje
	type="qname" //tipo simple
	element="qname"/> /tipo elemento
	</variables>
		Almacenan y transforman mensajes que contienen el estado del proceso. Sus tipos son:
		• Mensaje WSDL: indicado mediante el atributo *messageType*
		• Esquema XML: indicado mediante el atributo *type*
		• Elemento de esquema XML: indicado mediante el atributo *element*

IMPORTANTE

Es fácil confundir los *partner link* y los *partner link types*, sin embargo:

- Los *partner link types* y los roles son extensiones especiales de WSDL definidas por la especificación BPEL. Como tales, dichos elementos se definen en los ficheros WSDL, no en el fichero del proceso BPEL.
- *Partner link* es un elemento BPEL 2.0. Por lo que se define en el fichero del proceso BPEL.

Actividades en BPEL

Un proceso BPEL está formado por una serie de pasos. Cada uno de los pasos se denomina **actividad**. BPEL soporta dos tipos de actividades: primitivas y estructuradas.

Algunas de las actividades primitivas y su sintaxis, están descriptas en la Tabla 6 -en [BPEL User Guide, 2007] se encuentra una guía completa.

Tabla 4. Resumen algunas Actividades Primitivas

Actividad	Sintaxis	Descripción
receive	<receive partnerLink="ncname" portType="qname" operation="ncname" variable="ncname" createInstance="yes|no"> </receive>	Especifica un *partnerLink*, un *portType* y una operación que puede ser invocada. Permite que el proceso quede a la espera del arribo de un mensaje.

reply	<reply partnerLink="ncname" portType="qname" operation="ncname" variable="ncname"> </reply>	Responde a una actividad receive, se usa para interacción sincrónica y específica el mismo *partnerLink*, un *portType* y operación que invocó al proceso.
invoque	<invoque partnerLink="ncname" portType="qname" operation="ncname" inputVariable="ncname" outputVariable="ncname"> </invoque>	Invoca a otro servicio Web que se haya definido como *partner*. Puede ser ser síncrona (operación de tipo *request-response*) o asíncrona (operación de tipo *one-way*).

Las **actividades estructuradas** permiten combinar las actividades primitivas para especificar exactamente los pasos de los procesos de negocio. Entre las principales actividades estructuradas

- Ejecución secuencial (<*sequence*>), declara un conjunto de actividades que engloba se ejecutarán de forma secuencial.
- Ejecución paralela (<*flow*>) declara que las actividades que engloba se realizarán en paralelo.
- Selector múltiple (<*switch*>) implementa condicional sobre la siguiente actividad a ejecutar.
- Iterador (<*while*>) permite iterar sobre un conjunto de actividades.
- Selección de ejecución (<*pick*>) permite seleccionar un camino de ejecución.
- Fallas (<*throw*>) permite lanzar excepciones.
- Manejador de excepciones (<*faulthandler*>) permite tratar las excepciones que se producen.
- <*scope*> Determina el alcance de un grupo de actividades.
- <*terminate*> Finaliza la ejecución de un proceso de negocio.

2.4 - Conclusiones

El enfoque sobre procesos de negocios modernos más el bagaje de los lenguajes WSDL y XLANG, llevaron a BPEL a adoptar los Servicios Web como su mecanismo de comunicación externa.

Las facilidades de mensajería BPEL dependen del uso del WSDL para describir los mensajes entrantes y salientes y su mecanismo de sincronización se basa en las propiedades de los mismos.

Los contextos de ejecución de BPEL contemplan mecanismos de serialización para controlar el acceso a las variables garantizando la integridad ante la concurrencia.

Un proceso BPEL puede ser ejecutado sobre diferentes motores de ejecución, otorgándole portabilidad, lo cual es muy importante en el desarrollo B2B (*Business to Business*).

El diseño específico de BPEL para el desarrollo de procesos, lo hace eficiente en la ejecución de procesos de larga duración (ya sea minutos, horas, días).

BPEL soporta una característica de “deshacer” actividades previas del proceso de negocio. Es posible definir manejadores para estos casos indicando el alcance de los mismos dentro del proceso de ejecución. Estas características son conocidas como *Long-Running-Transactions* (LRT).