CAPÍTULO 5
Arquitecturas Distribuidas

Patricia Bazán

En capítulos anteriores se presentaron distintos estilos arquitectónicos distribuidos y como se detalló en el Capítulo 1, las arquitecturas distribuidas buscan simplificar y abstraer las funciones de cada componente definiendo su intercomunicación y estableciendo su ubicación y plataforma de ejecución.

También se ha detallado la evolución de los sistemas distribuidos que fue guiada en gran medida por la misma evolución de construcción de aplicaciones y el crecimiento de Internet como mecanismo de comunicación (middleware) a nivel de red.

En la evolución presentada en el Capítulo 2 se describió someramente el concepto de arquitecturas orientadas a servicios (SOA), como evolución de los objetos distribuidos y como un paradigma adecuado para la integración de aplicaciones.

En este capítulo se profundizan estos conceptos y se analizan los Web Services como una implementación posible de una arquitectura orientada a servicios.

El capítulo se organiza de la siguiente manera: en la Sección 1 se describen las arquitecturas de n niveles, ya analizadas en el Capítulo 4, pero con tecnología CGI. En la sección 2 se presenta la Arquitectura Orientada a Servicios SOA, detallando componentes, características y motivaciones de uso. En la Sección 3 se analizan los Web Services como un medio para implementar un SOA, junto con sus variantes tecnológicas que se describen en la Sección 4. Finalmente, se arriban a algunas conclusiones.

5.1. Arquitectura n niveles. Tecnología CGI

Las llamadas aplicaciones Cliente/Servidor de misión crítica dejaron de ser un elemento teórico. Cada día, millones de transacciones de comercio electrónico se inician desde las PC's y se ejecutan en servidores distribuidos. Con Cliente/Servidor se mezcla el procesamiento local y remoto en una aplicación única. Este es de alguna manera el fundamento de Internet, intranets y extranets. Sin embargo el Cliente/Servidor clásico en dos niveles comenzó a resultar insuficiente.

Si bien el concepto de tres niveles fue inicialmente dirigido por la necesidad de escalar las aplicaciones en dos capas, el mercado actual de servicios y componentes lo están tomando

como su paradigma. Como se mencionó en el Capítulo 2, la tecnología Web es un ejemplo de evolución del modelo Cliente/Servidor clásico a un modelo en tres niveles. Pero este no es el único caso de arquitectura en más de dos niveles.

Características del modelo de 3 niveles

El cliente provee la interface de usuario gráfica (GUI - *Graphic User Interface*) e interactúa con el servidor a través de servicios remotos y métodos de invocación.

La lógica de la aplicación vive en el nivel medio y se distribuye y administra en forma separada de la interface de usuario y de la base de datos.

Las aplicaciones en 3 niveles minimizan los intercambios en la red creando niveles de servicios abstractos.

Las aplicaciones en 3 niveles sustituyen unas pocas invocaciones al servidor por varias consultas y actualizaciones SQL.

Proveen mayor seguridad al no exponer la estructura física de los datos.

La siguiente tabla muestra una comparación de características entre 2 niveles y 3 niveles

Característica	2 capas	3 capas
Administración de sistema	Compleja (más lógica en el cliente)	Menos compleja (se centralizan las aplicaciones)
Seguridad	Baja (a nivel de datos)	Alta (a nivel de servicios y métodos)
Encapsulamiento de datos	Baja (tablas expuestas)	Alta (el cliente invoca servicios y métodos)
Performance	Pobre (muchas sentencias SQL viajan por la red)	Buena (se intercambian sólo preguntas y respuestas)
Escalabilidad	Pobre	Excelente
Reuso de aplicaciones	Pobre (aplicaciones monolíticas en el cliente)	Excelente (se reusan servicios y objetos)
Soporte de Internet	Pobre (limitaciones por ancho de banda)	Excelente (los "thin" clientes son fáciles de descargar)
Soporte de BD heterogéneas	NO	SI
Elección de mecanismo de comunicación	NO (sólo sincrónico y orientado a la conexión)	SI (mensajes, colas, broacasting)

Tecnología CGI

La tecnología CGI (Common Gateway Interface), define un escenario posible de implementación de arquitectura de 3 niveles basada en la tecnología Web detallada el Capítulo 2 donde se planteaba la necesidad de contar con alguna variante de ejecución computacional dentro del servidor Web y transformarlo en un servidor Web *dinámico*.

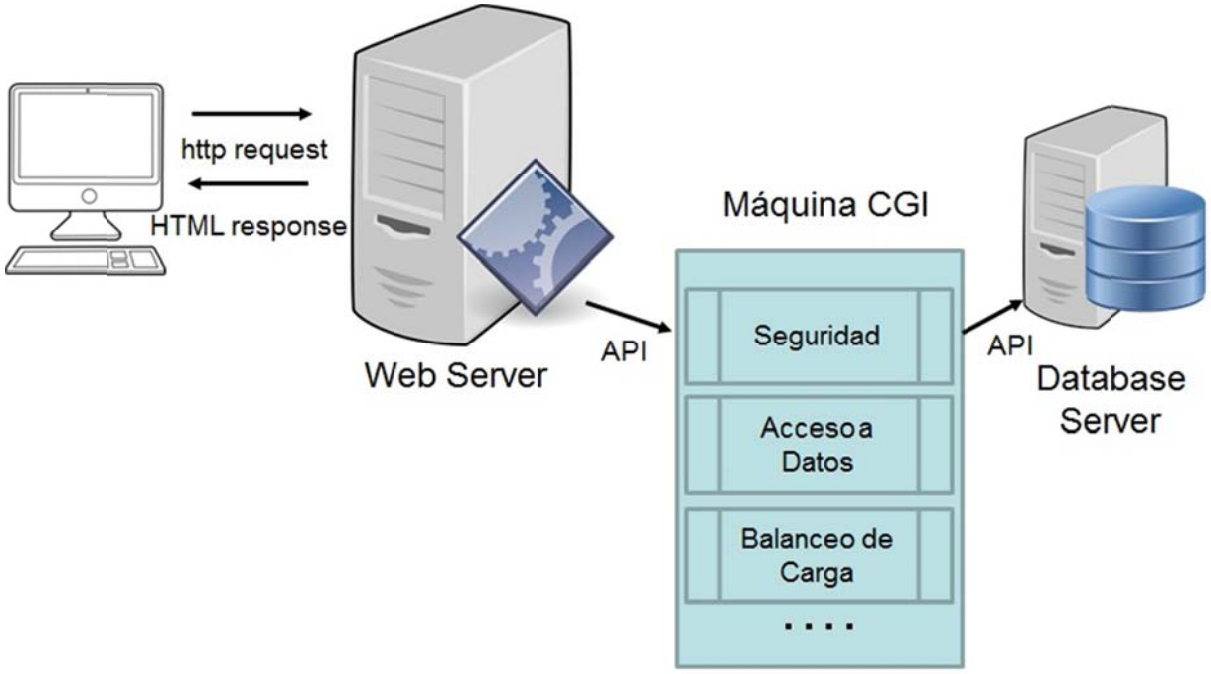

Fig. 43. Funcionalidades de una Máquina CGI

Como se detalló en el Capítulo 4, hay dos variantes para incorporar dinamismo a un servidor Web. Una de ellas, en modo nativo, a través de páginas dinámicas (la Tecnología Java es un ejemplo) o mediante una interface (API: Application Program Interface), tal es el caso de la Tecnología CGI.

La tecnología CGI es un mecanismo muy simple que permite que un servidor web ejecute un programa escrito en cualquier lenguaje de programación a través de una invocación definida por su API (implica salida al Sistema Operativo).

Para enviar datos al programa CGI se cuenta con la especificación CGI define cómo se empaquetan dichos datos. El URL determina cuál programa CGI se ejecutará.

La Figura 43 detalla algunas de las funciones que pueden llevar a cabo un motor o máquina CGI.

Debido a que el Web Server debe encaminar la invocación a la máquina CGI, sólo puede hacerlo a través de un tag HTML y en particular a través de los métodos GET o PUT.

Veamos con un ejemplo, cómo sería esta invocación utilizando una máquina CGI específica, PBCGI050.EXE que provee el lenguaje PowerBuilder.

<FORM ACTION = "/scripts/pbcgi050.exe/master/uo_inet_functions/f_handlelogin" METHOD="GET">

1- FORM le indica al WebServer que el texto que continúa es un FORM
2- ACTION le indica al WebServer que cuando este form sea enviado (submit) se realizará alguna acción. Esta acción puede ser bien un pedido de servicio HTTP o un llamado a una interface CGI para la ejecución de una función.
3- uo_inet_functions identifica el objeto no visual de master que contiene la definición de la función f_handlelogin.
4- /scripts/ determina el directorio donde encontrar la interface CGI y /pbcgi050.exe/ es el programa específico que implementa el CGI

Tecnología CGI: ventajas y desventajas

Las ventajas de usar tecnología CGI para lograr un Web Server dinámico son:

- Pueden escribirse en cualquier lenguaje de programación (Perl, C, PowerBuilder, VisualBasic).
- Un error en el programa CGI no afecta el servicio Web debido a que se ejecuta fuera de su contexto.
- Son programas fáciles de referenciar para un diseñador Web. Como se ve en el ejemplo, simplemente se incluye la invocación en una página HTML ya diseñada gráficamente.
- Dado que los programas CGI ejecutan su propia llamada al sistema operativo, no tienen conflicto de concurrencia con requerimientos que usen el mismo CGI.
- Todos los proveedores de servicios lo implementa.

Por su parte las desventajas son:

- Alto tiempo de respuesta, justamente por su ejecución fuera del contexto del Web Server.
- No es escalable. Si crece repentinamente la cantidad de requerimientos, el programa CGI debería hacer balanceo de carga.
- No permite un diseño modular ni separa la lógica de la presentación de la de negocio.

El nivel medio en muchas aplicaciones en 3 niveles no es implementado como un programa monolítico sino que es una colección de componentes utilizadas por varias transacciones.

Cada componente automatiza una función de negocio relativamente pequeña. Los clientes suelen combinar varias componentes del nivel medio dentro de una transacción única.

5.2. Conceptos de Arquitecturas Orientadas a Servicios

Según la definición de OASIS 9, SOA como un paradigma capaz de organizar y utilizar las capacidades distribuidas, que pueden estar bajo el control de distintas organizaciones, y de proveer un medio uniforme para publicar, descubrir, interactuar y usar los mecanismos oportunos para lograr los efectos deseados.

SOA es un modelo de referencia para entender las relaciones más significativas dentro del dominio de un problema concreto y facilitar el desarrollo de estándares o especificaciones. Se fundamenta en un pequeño número de conceptos para explicar el modelo a profanos y busca producir una semántica sin ambigüedades.

SOA es un modelo de referencia para:

- La creación y utilización de servicios a lo largo de su vida útil.
- La definición de la infraestructura que permita intercambiar datos entre diferentes aplicaciones.
- La participación de los servicios en los procesos de negocios independientemente del sistema operativo, los lenguajes de programación y si los procesos son internos o externos a la organización.

Los conceptos y sus relaciones, definidas en el modelo de referencia SOA, deben ser la base para describir la arquitectura.

Una arquitectura SOA concreta será el producto de aplicar la arquitectura de referencia desarrollada según el modelo de referencia y los patrones10 de esa arquitectura, así como los requerimientos necesarios, incluyendo los impuestos por los entornos tecnológicos.

La aplicación de un modelo de referencia para lograr una arquitectura completa, equivale a pasar de una etapa de análisis a una de diseño en analogía con las etapas del ciclo de vida del software. Implica dar un paso más en el nivel de detalle y comenzar a buscar metodologías para aplicar sobre los conceptos analizados.

Una arquitectura concreta se desarrolla en un contexto predefinido donde se fijan protocolos, perfiles, especificaciones y estándares. La plataforma SOA combina estos elementos a los efectos de generar un producto operativo. La Figura 44 (Bazán,2010) presenta un marco general entre la noción de marco de referencia, arquitectura de referencia y arquitectura específica, que muestra qué aporta y en qué consiste cada una de ellas, así como el paso de lo más abstracto hacia lo más concreto.

9 https://www.oasis-open.org/
10 Un patrón es una forma de realizar una tarea concreta basada en una generalización.

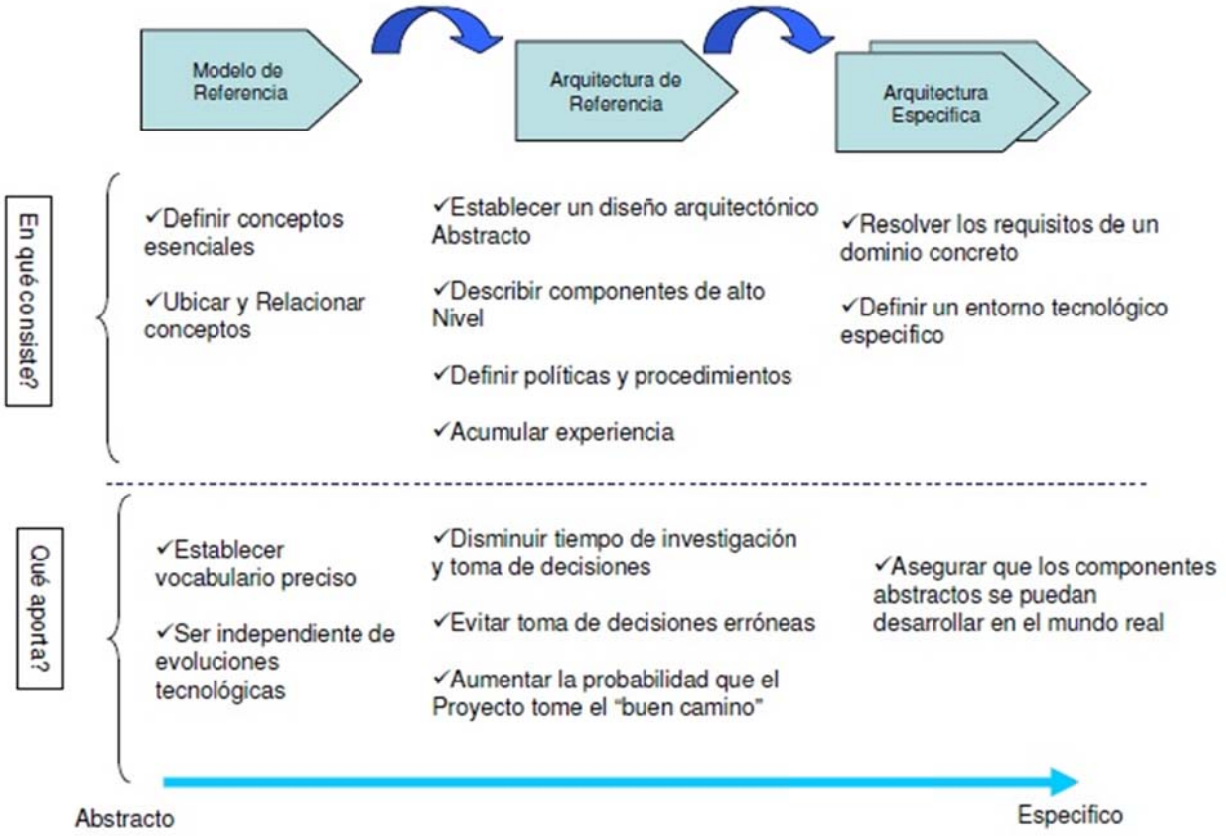

Fig. 44. Marco general entre modelo de referencia, arquitectura de referencia y arquitectura específica

Características de SOA

En conclusión con lo anterior, para lograr un SOA una empresa necesita entender cómo modelar procesos, servicios y componentes y cómo unir estos modelos de manera consistente

Las características o aportes fundamentales de SOA son:

- 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.
- Inteoperabilidad 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: Son 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)
- ESB (Enterprise Service Bus). Es la infraestructura que habilita una alta interoperabilidad entre Servicios en un contexto distribuido.
- Bajo acoplamiento. Concepto de reducir las dependencias entre sistemas. Es una idea muy difícil de desplegar, mantener y corregir.

SOA y el Modelo de Interacción

Según los conceptos subyacentes a SOA descriptos con anterioridad, el modelo de interacción es uno de los más relevantes dado que aporta la manera en que los servicios se comunican entre sí.

Este modelo de interacción sigue el paradigma triangular de Publicar-Ligar-Ejecutar como se muestra en la Figura 45 (Bazán,2010).

En este paradigma, los proveedores registran sus servicios en un registro público. Los consumidores utilizan este registro para buscar servicios que cumplan con cierto criterio. Si el registro posee tal servicio, el mismo provee al consumidor con un contrato y un punto de acceso para el servicio.

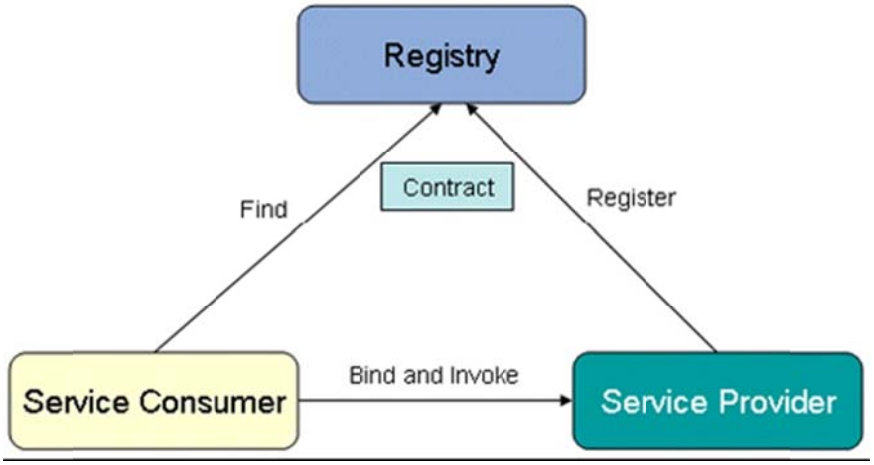

Fig. 45. Modelo de Interacción Triangular SOA

- Publicar (*Register*): para que un servicio sea usado, su descripción de servicio debe ser publicada
- Descubrir / Encontrar (*Bind and Invoke*): el consumidor busca una descripción de servicio directamente, o hace una búsqueda de servicios. La operación se puede hacer en dos puntos del ciclo de vida del cliente: durante el diseño o durante la ejecución.

- Enlazar (*Find*): el cliente del servicio invoca o inicia la interacción con el mismo durante la ejecución, contando con los detalles en la descripción del servicio para localizarlo, contactarlo e invocarlo.

Acerca de los servicios

Un servicio es un elemento que se comprende en términos de la utilidad que brinda, por lo tanto, no puede apartarse del negocio o problema para el cual debe ser útil.

Los servicios son tareas computacionales débilmente acopladas que se comunican vía una red (en el caso de los web services por Internet), y que juegan una relación cada vez más creciente en las interacciones B2B (Bussines To Bussines).

Un servicio captura funcionalidad con un valor de negocio, y está listo para ser usado. Es provisto por servidores, para lo cual requiere de una descripción que pueda ser accedida y entendida por potenciales clientes. Los servicios de software son servicios provistos por sistemas de software (Erl, 2007).

Con estas definiciones podemos decir que el conjunto de conceptos que describen los servicios son: descripción del servicio, contratos y normas y contexto de ejecución. A través de la descripción del servicio se obtiene la información que un consumidor necesita para considerar si usa o no el servicio.

Con esta información se puede informar al consumidor: si el servicio existe (alcanzabilidad), qué funciones realiza (funcionalidad), qué restricciones se aplican a su uso y cómo se debe interactuar con el servicio (interfaz) tanto en cuanto a formato como a secuencias.

Los contratos y normas representan el acuerdo entre las partes, las condiciones de uso, las restricciones y el despliegue de los servicios.

El contexto de ejecución es el conjunto de elementos técnicos y de negocios que conforman la vía para que proveedores y consumidores puedan interactuar.

La posibilidad de extender sistemas anexándole nuevos servicios y haciendo reuso de los ya existentes, con el objetivo de lograr interoperabilidad entre tecnologías y aplicaciones heterogéneas, nos permite prolongar la vida de los sistemas. Los beneficios de SOA son a largo plazo. La existencia de un único servicio, no tendrá valor si no tiene forma de complementarse con otros servicios.

Este estilo de arquitectura nos permite el desarrollo de aplicaciones débilmente acopladas, las cuales pueden ser accedidas a través de la red.

Enterprise Service Bus (ESB)

El enfoque SOA busca construir aplicaciones mediante la combinación poco acoplada de servicios interoperables. En tal sentido, el hecho de que un servicio pueda utilizarse amplia-

mente en toda la empresa por muchas aplicaciones, implicaría dar lugar a los siguientes riesgos para la infraestructura de IT de una organización:

- Tiempos de respuesta poco aceptables tanto para los usuarios como para procesos de negocio.
- Infracciones de seguridad.
- Pérdida de niveles de servicio para funciones críticas del negocio.
- Incumplimiento de normas de la industria y regulaciones gubernamentales.
- Gestión de servicios insuficiente.

El área de tecnología de una organización se ha centrado desde hace mucho tiempo en la gestión de la infraestructura como un activo para apoyar las aplicaciones y las unidades de negocio. Con SOA la atención se enfoca hacia la gestión de los servicios que prestan apoyo a los procesos de negocio.

Así, resulta evidente la necesidad de contar con una infraestructura de IT que apoye la gestión de los servicios.

Sobre una arquitectura SOA se puede definir un bus de servicios empresariales (Enterprise Service Bus o ESB) como una plataforma de software que da soporte a muchas funcionalidades resueltas a nivel de la capa de aplicación en los enfoques tradicionales de construcción de aplicaciones.

Tales funcionalidades son:

- La comunicación: el ESB se ocupa del ruteo de los mensajes entre los servicios.
- La conectividad: el ESB resuelve la conectividad entre extremos mediante la conversión de protocolos entre solicitante y servicio.
- La transformación: es responsabilidad del ESB resolver la transformación de formatos de mensajes entre solicitante y servicio.
- La portabilidad: los servicios serán distribuidos independientemente del lenguaje de programación en el que estén escritos y del sistema operativo subyacente.
- La seguridad: el ESB posee la capacidad de incorporar los niveles de seguridad necesarios para garantizar servicios que puedan autenticarse, autorizarse y auditarse.

Actualmente existen dos tendencias mayoritarias para la implementación de un ESB: los que requieren un servidor de aplicaciones y los que son totalmente distribuidos y, por lo tanto, no lo requieren (Bazán,2010).

Funcionalmente, cualquiera de las dos tendencias conserva sus propias características según se detalla en el cuadro *Servidores de aplicaciones Vs ESB*, pero se puede afirmar que un ESB totalmente distribuido que no requiere de un servidor de aplicaciones tendrá mayor independencia de la plataforma y será capaz de ofrecer una mayor ubicuidad de los servicios que gestiona.

Las siguientes características permiten determinar qué es y que no es un ESB:

- Está desarrollado sobre una arquitectura orientada a servicios, por lo tanto, es una implementación de SOA.
- Se basa en la utilización de estándares en un 100%, pero admite también elementos propietarios.
- Normalmente son multi-plataforma y multi-lenguaje.

- Son netamente distribuidos en el sentido de que no es necesario conocer la ubicación física de los servicios y que un mismo servicio puede existir en más de una localización en forma simultánea.
- Presenta un mecanismo de comunicación basado en mensajes con enrutamiento inteligente (basado en reglas o basado en contenido).
- Implementa un modelo se seguridad estandarizado con lo que se denomina nivel C2 de seguridad garantizando autenticación, autorización y auditoría.

5.3. Web Services como la Evolución Natural de la Computación Distribuida

Como hemos visto, 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.

Servidores de aplicaciones Vs ESB

Los servidores de aplicaciones manejan gran parte de las interacciones entre la capa cliente y la capa de persistencia a datos en un modelo de 3-capas. Proveen una colección de servicios de middleware junto con un ambiente de ejecución para desplegar las componentes de lógica de negocios (el container). La mayoría de los servidores de aplicaciones soportan Web Services, ORB's, sistemas de mensajes, manejo transaccional, seguridad, balanceo de carga y gestión de los recursos. Proveen una solución integral a las necesidades de los sistemas de información empresariales (a gran escala).
Constituyen una excelente plataforma de integración. Hoy la mayoría de los productos comerciales posicionan sus servidores de aplicaciones como máquinas de integración o los especializan agregando conexiones a back-end y sistemas legados posicionando sus productos como servidores de integración.
Si bien este tipo de servidores puede facilitar considerablemente la configuración de los diferentes productos de middleware, todavía vale la pena pensar en lo que hay debajo. Sean utilizado para desarrollo de aplicaciones o para integración, los servidores de aplicaciones son ***plataformas de software***: la combinación de tecnologías de software necesarias para ejecutar aplicaciones. En este sentido, ellos definen la infraestructura de todas las aplicaciones desarrolladas y ejecutadas en los mismos.
Un ESB, por su parte, es una ***infraestructura de software*** en sí misma, que actúa como middleware intermediario y que dirige los requerimientos extendidos que generalmente no pueden ser cubiertos por los Web Services, como la integración entre los mismos y la inclusión de otras tecnologías de middleware y servicios de valor agregado como robustez, seguridad, gestión y control de servicios.
Un ESB dirige estos requerimientos y agrega flexibilidad para la comunicación entre servicios, haciendo posible conectar los servicios implementados en diferentes tecnologías (tales como EJB, sistemas de mensajes, componentes CORBA y sistemas legados).

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.

A cambio, se hace imprescindible contar con un marco de referencia o arquitectura, que facilite su registro y publicación para conocer su existencia (Arsanjani,2004).

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

Por último, los Web Services surgieron en forma paralela a la idea de SOA. La plataforma SOA ordenó su uso sistemático combinando protocolos, perfiles, especificaciones y estándares (Pulier, 2006).

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.

Los estándares vinculados con los Web Services son:

- WSDL (Web Service Description Language). Describe en formato XML la funcionalidad brindada por el Web Service.
- SOAP (Simple Object Acces Protocol). Facilita el intercambio de información estructurada como XML. Constituye el protocolo de comunicación estándar.
- UDDI (Universal Description, Discovery and Integration). Es un estándar para facilitar el registro y descubrimiento de Web Services.

Estos estándares instancian un caso particular de modelo de interacción de servicios para Web Services, tal como se muestra en la Figura 46.

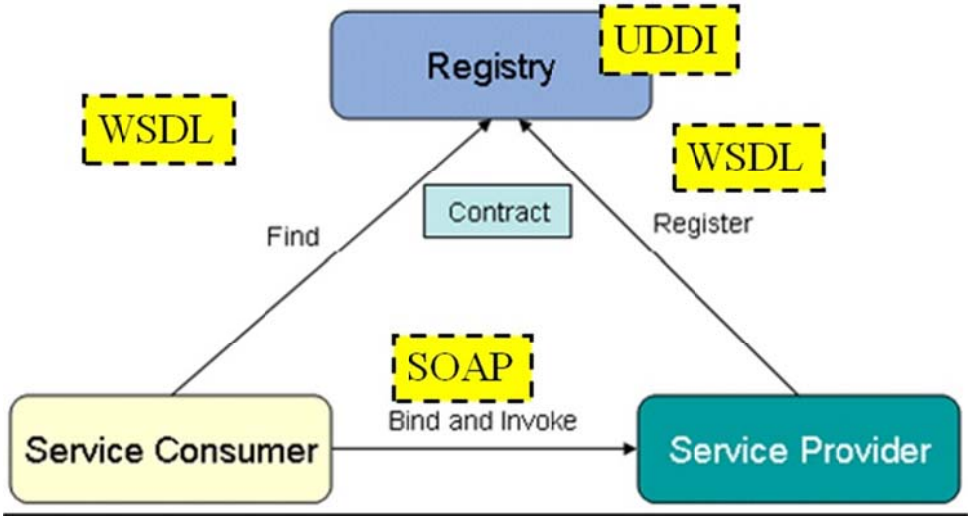

Fig. 46. Modelo de Interacción Triangular Web Services

5.4. Web Services en distintas tecnologías: REST y SOAP

Existen dos maneras de implementar Web Services y se diferencian respecto del uso o no del estándar para invocar y transportar el servicio. Estos son SOAP o REST.

SOAP (Simple Object Access Protocol)

Presentan una interface de invocación a una función (o método) distribuido similar a RPC. Comúnmente, la unidad básica de este estilo de Web Service es la operación WSDL, es decir, la especificación, en XML de lo que hace un servicio web, donde se encuentra y la forma de ser invocado. Este provee información muy importante para los desarrolladores, este lenguaje describe el formato de los mensajes que utiliza y a cuales puede responder.

REST (Representational state transfer)

Los Servicios Web RESTful emulan HTTP y protocolos similares limitando la interface a un conjunto de operaciones bien conocidas y estándares (p.e: GET, PUT, DELETE). Acá el foco esta puesto sobre la interacción con recursos, más que mensajes u operaciones. RESTful Web services pueden usar WSDL para describir mensajes SOAP sobre HTTP, lo cual define las operaciones, o pueden ser implementados puramente como una abstracción por encima de SOAP (ej: WS-Transfer).

	REST	SOAP
Formato del mensaje	XML	XML dentro de SOAP
Definición del interfaz	No es necesario	WSDL
Transporte	HTTP	HTTP, JMS, FTP, etc.

En resumen:

5.5. Conclusiones del Capítulo

En este Capítulo completamos la descripción de arquitecturas distribuidas desde su evolución en la Web a través de la tecnología CGI, hasta la aparición de las arquitecturas orientadas a servicios SOA.

Un SOA presenta un verdadero cambio en la concepción de aplicaciones pero que se ve favorecido en su instrumentación si proviene de un desarrollo basado en tecnología Web. La