



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 15

1. Arquitectura J2EE.
2. Arquitectura .NET.





CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 15

Arquitecturas de desarrollo basadas en componentes. Arquitectura J2EE. Arquitectura .NET.

1. ARQUITECTURA J2EE.

A finales de los 90, gran número de desarrollos corporativos empezaron a aprovechar los estándares, lenguajes y protocolos surgidos alrededor de la web, creándose el concepto de las arquitecturas multicapa, que se presentaban como alternativa a la arquitectura cliente/servidor clásica, que permitiría utilizar un entorno cliente ligero y, por lo tanto, evitar el problema de la distribución y mantenimiento del software de los PCs cliente.

Java es uno de estos lenguajes que aparecieron alrededor de Internet. Inicialmente, los applets de Java contribuyeron a enriquecer las interfaces de usuario, posteriormente, tecnologías como los servlets y los JSPs, facilitaron el desarrollo de la lógica de negocio compleja para los servidores.

Con la publicación de las especificaciones J2EE (Java 2 Enterprise Edition) se dio el impulso definitivo para la entrada de esta tecnología en el ámbito empresarial, pensada para la problemática de complejos desarrollos corporativos y se ha convertido en la base para los próximos años de la arquitectura de desarrollo para la web con tecnología Java.

La arquitectura de desarrollo multicapa se caracteriza por dividir el sistema en distintos niveles o capas que se comunican a través de interfaces:

1. Capa de presentación o cliente: es la encargada de mostrar la interfaz de usuario al cliente e interactuar con él.
2. Capa de negocio: es la parte en la que se reciben las peticiones de usuario y se ejecuta toda la lógica transaccional de la aplicación.
3. Capa de almacenamiento de datos: es fundamentalmente la formada por el sistema gestión de base de datos, aunque en ocasiones también incluye cierta funcionalidad de manejo de esos datos.



Esta arquitectura busca por un lado aprovechar las especiales características de los diferentes lenguajes o tecnologías de desarrollo y por otro independizar unas capas de otras tratando de conseguir que un cambio en una capa no afecte al resto de capas del sistema. Sin embargo, una arquitectura multicapa, por sí sola, no nos permitirá alcanzar estos objetivos. Se necesitará además una buena formación de los equipos de trabajo, un buen diseño y una buena definición de las interfaces entre las capas, sólo así lo tendremos un poco más fácil.

Si analizamos qué caracteriza a las aplicaciones empresariales, podríamos destacar los siguientes aspectos:

1. Alta escalabilidad y rendimiento. Las aplicaciones corporativas son complejas y requieren tiempos de respuesta razonables, así como adaptarse a los picos de trabajo sin sufrir una degradación apreciable en el nivel de servicio.
2. Integridad transaccional. Las aplicaciones de gestión requieren seguridad en las transacciones que garanticen la integridad del modelo de negocio gestionado.
3. Robustez. El trabajo diario de las organizaciones depende de estos sistemas de información, por lo que se debe proporcionar una calidad de funcionamiento y unos niveles de disponibilidad adecuados.
4. Interoperatividad. Deben integrar sistemas heterogéneos y distribuidos.
5. Seguridad. La información gestionada por estos sistemas, en la mayoría de las ocasiones, contiene información sensible para la que se debe garantizar tanto su confidencialidad, como su integridad.

Esto obliga a incorporar a la arquitectura nuevos servicios a los clásicos elementos del desarrollo web tradicional. Así, aparecen nuevas capas, como la capa de integración o la capa de gestión sistemas transaccionales. Este conjunto de nuevas funcionalidades complica aún más el entorno y aparecen más problemas: la gestión de sistemas distribuidos, la interoperatividad de sistemas, los nuevos entornos de desarrollo, etc.

El patrón de diseño MVC es un modo de dividir la funcionalidad de una aplicación entre distintos componentes de forma ordenada. Esta división se hace con el objetivo de que unos componentes se encarguen de la lógica de negocio (modelo), otros del flujo de control de la aplicación (controlador) y otros de la presentación o interfaz de usuario (vista), siguiendo las características anteriormente expuestas para una arquitectura multinivel, es decir, permitiendo la utilización de entornos especializados en cada capa y minimizando el acoplamiento entre los componentes de distintas capas.

Repasemos cada uno de estos tres módulos en que el patrón MVC divide una aplicación. En primer lugar, veamos qué es el modelo. El modelo representa los componentes que implementan el acceso a los datos de la aplicación y las reglas de negocio que gestionan su acceso. El modelo es, en definitiva, el punto por el que cualquier sistema que desee acceder a los datos de la aplicación deberá pasar. Para el desarrollo del modelo se utilizan normalmente lenguajes de propósito general como Java, con librerías de acceso a sistemas de gestión de base de datos e incluso lenguajes para desarrollar procedimientos almacenados integrados en los propios sistemas de gestión de base de datos.

La vista incluye los objetos encargados de presentar la interfaz al usuario. Estos componentes mostrarán al usuario el resultado de las transacciones realizadas con el modelo. Como ya hemos di-

cho, si el diseño se ha realizado bien, el cambio del modelo no debe afectar a los componentes de la vista. Del mismo modo, si se decide cambiar la presentación o permitir varios tipos de presentación, es responsabilidad de la vista permitirlo sin necesidad de modificar el modelo. En el desarrollo de los componentes que implementan la vista se utilizan normalmente tecnologías como XML/XSL, páginas JSP, páginas ASP; es decir, tecnologías distintas a las utilizadas para desarrollar el modelo y más enfocadas a permitir el diseño de interfaces gráficas de usuario.

El controlador o controladores decidirá qué vistas se muestran, dependiendo de los clientes o los resultados proporcionados por el modelo. Todo esto es transparente para los componentes del modelo que únicamente responden a peticiones del controlador. En las aplicaciones web, el controlador se utiliza como punto central en el que se reciben las peticiones del usuario y se traducen en llamadas al modelo y a las vistas que generan la interfaz para el cliente. Como veremos más adelante, una tecnología muy adecuada para implementar estos controladores son los servlets.

Java 2 Enterprise Edition.

El lenguaje Java, que empezó siendo un lenguaje para sistemas empotrados y que saltó a la fama como lenguaje para el desarrollo de applets, ha pasado a ser en estos momentos un lenguaje maduro, moderno, para el que se han construido diferentes entornos de desarrollo dependiendo de los objetivos perseguidos. En estos momentos, Java dispone de 3 plataformas de desarrollo:

1. J2ME (Java 2 Micro Edition). Es la plataforma dirigida al desarrollo de aplicaciones para dispositivos como teléfonos móviles o PDA's. Como el resto de las plataformas, está formada por un conjunto de APIs definidas por grupos de trabajo en los que han colaborado empresas del sector de los dispositivos y empresas de software.
2. J2SE (Java 2 Standard Edition). Consiste en el entorno de ejecución estándar de Java. Como el resto, está formado por un conjunto de APIs para crear distintos tipos de aplicaciones como applets que se ejecutan en el entorno del navegador y aplicaciones cliente que se ejecutan sobre una máquina virtual, que se distribuye tanto con el entorno de desarrollo como con el entorno de ejecución creado para la plataforma en la que la aplicación debe funcionar.
3. J2EE (Java 2 Enterprise Edition). Consiste en un conjunto de especificaciones dirigidas a facilitar el desarrollo de sistemas empresariales en arquitecturas multicapa.

J2EE aparece con el objetivo de minimizar los problemas asociados al desarrollo de aplicaciones multicapa en Internet (entornos heterogéneos, integración de tecnologías, complejidad, servidores complejos y propietarios). Sun Microsystems, con el apoyo de otras grandes empresas del software, impulsó una iniciativa de recopilación y elaboración de una serie de especificaciones y recomendaciones para crear una arquitectura estándar que permitiera reducir el coste y la complejidad de estos desarrollos. Las especificaciones Java 2 Enterprise Edition recogen:

- Especificaciones básicas para los servidores de aplicaciones Java que permiten la movilidad entre distintos proveedores y, por lo tanto, la creación de un mercado competitivo entre las empresas de software que provoca un abaratamiento de sus productos.
- Un conjunto extenso de librerías sobre las que poder desarrollar sistemas complejos.
- Un conjunto de manuales de buenas prácticas que permitan una formación sólida entre los profesionales dedicados al desarrollo de este tipo de sistemas.

Las especificaciones básicas forman lo que se conoce como la plataforma J2EE, que se ha convertido en la referencia sobre la que la práctica totalidad de fabricantes de servidores de aplicaciones desarrollan sus sistemas, y con la que el primer objetivo de crear un gran mercado competitivo se está consiguiendo.

La arquitectura de J2EE se representa en el siguiente diagrama:

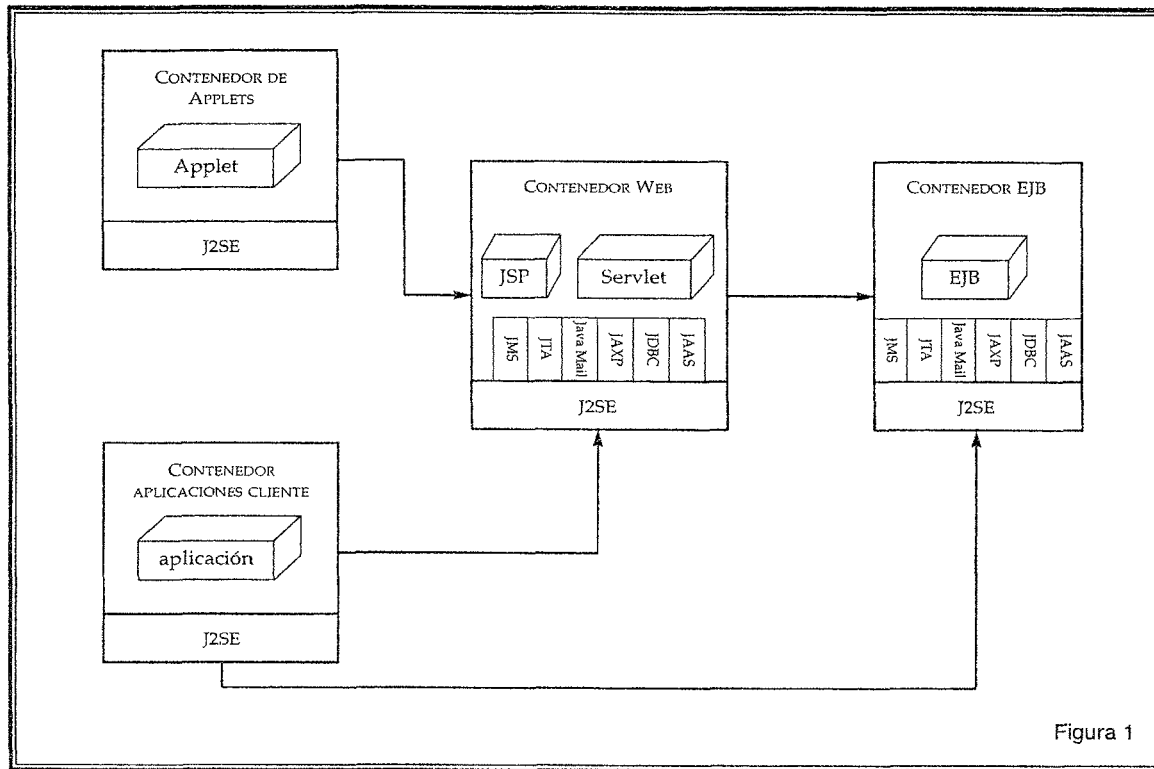


Figura 1

La arquitectura propuesta es una arquitectura multicapa en la que se definen cuatro grandes bloques identificados por el contenedor en el que se ejecutan. Se definen dos tipos de contenedores que se encuentran en las máquinas cliente y dos tipos de contenedores que se encuentran en los servidores.

Si recordamos el patrón MVC, podríamos decir que los contenedores de cliente y parte de los componentes del contenedor web son la vista, en el contenedor web está el controlador y en el contenedor EJB está el modelo. Evidentemente, esta es una aproximación, ya que puede existir parte del modelo en el contenedor web o incluso en el contenedor de aplicaciones cliente.

Servlets.

De forma resumida, podemos decir que los servlets son componentes web desarrollados en Java y utilizados para generar contenido dinámico: clases Java que extienden la interfaz `HttpServlet` y que son invocadas a través de una URL pudiendo generar dinámicamente diversos tipos de contenido como texto, imágenes y sonido, aunque normalmente es texto con formato HTML o XML. Este contenido es devuelto al cliente mediante el protocolo HTTP.

La clase `HttpServlet`.

La clase `HttpServlet` es una clase abstracta de la que heredarán todos los servlets destinados a dar servicios desde un servidor web. Cada servlet debe sobrecargar al menos un método que normalmente es uno de los siguientes:

- `doGet()`. Se ejecuta cuando se recibe una petición HTTP GET.
- `doPost()`. Se ejecuta cuando se recibe una petición HTTP POST.
- `doPut()`. Se ejecuta cuando se recibe una petición HTTP PUT.
- `doDelete()`. Se ejecuta cuando se recibe una petición HTTP DELETE.
- `init()` y `destroy()`. Gestionan el ciclo de vida del servlet, ejecutándose en la creación y destrucción del servlet.
- `getServletInfo()`. Lo utiliza el servlet para proporcionar información acerca de sí mismo.
- `service()`. Que maneja la petición HTTP para redirigir al método `doXXX` que corresponda.

Los métodos `doXXX` reciben como parámetros objetos de la clase `HttpServletRequest` con la información de la solicitud HTTP y `HttpServletResponse`, donde se incluye la información de respuesta a la petición HTTP. En ellos se implementa normalmente la funcionalidad del servlet ante los distintos tipos de solicitudes HTTP.

Como ya se ha comentado en este punto, los servlets normalmente se ejecutan en servidores multihilo, por lo que un servlet debe gestionar la concurrencia y tener en cuenta los recursos compartidos tanto en memoria (variables de clase) como en ficheros, bases de datos, etc.

Un servlet se instala, gestiona y ejecuta en un componente del sistema denominado contenedor web. Más adelante en este tema veremos qué es un contenedor dentro de la plataforma J2EE, pero de momento lo que nos interesa es que este contenedor va a proporcionar al servlet independencia de la plataforma, ya que el desarrollador se podrá encargar sólo de cumplir con la especificación, mientras que el contenedor se ocupará de su ejecución independientemente del entorno.

Para entender cómo funciona un servlet vamos a analizar el ciclo de vida del servlet desde su instalación hasta su desinstalación o parada.

1. El servlet es instalado en el contenedor. Básicamente, esta instalación consiste en indicar en el fichero de configuración del contenedor la URL a la que responderá y la clase que implementará y, por supuesto, copiar la clase en el directorio del servidor.
2. Cuando el contenedor recibe por primera vez una petición de un servlet, el contenedor crea un objeto de la clase Java que lo implementa. Al ser Java un entorno multihilo, los servlets están diseñados para permitir dar respuesta a un alto número de solicitudes concurrentes.
3. Para cada petición:

- El cliente (normalmente un navegador web) envía la petición HTTP al servidor web.
- El servidor web identifica la petición como una URL que está asociada con algo que se debe ejecutar en el contenedor web y reenvía la petición al contenedor.
- El contenedor (o motor) de servlets tiene una configuración en la que se indica para cada URL solicitada qué objeto debe crear. Por lo tanto construye o, si ya estuviese creado por una petición anterior, utiliza el objeto de la clase correspondiente a la petición de la URL.
- Una vez creado el objeto, ejecuta el método doGet() o doPost() (según se indique en la petición HTTP) del objeto. Todo servlet tendrá estos métodos, puesto que lo obliga la interfaz HttpServlet.
- El servlet analiza la información que viene en el request de la petición HTTP y realiza las llamadas pertinentes a otros componentes Java, como son los objetos de negocio, de cara a obtener la información necesaria.
- Finalmente, el servlet genera la respuesta en el response construyendo la página HTML, XML o lo que sea que el servidor web vaya a enviar al cliente.

Los servlets se han mostrado adecuados para determinadas funcionalidades dentro de las aplicaciones corporativas, ya que tienen toda la versatilidad de un lenguaje de propósito general como Java, que permite estructurar secuencias de control complejas sin obtener un código excesivamente difícil de mantener, funcionalidades como la implementación de controladores dentro de arquitecturas del tipo MVC, así como para la generación de salidas binarias (imágenes, sonido, vídeo, etc.).

Sin embargo, desde los inicios del desarrollo con servlets se comprobó que, aunque los servlets eran un excelente sistema para gestionar las peticiones web de contenido dinámico, no eran un sistema cómodo para generar la interfaz HTML de la respuesta, ya que había que codificar mucho para conseguir un nivel aceptable de presentación. Por ello, veremos que se han ideado otras tecnologías Java como JSP que aligeran esta fase.

Un JSP es en realidad un servlet orientado a la presentación pues, como veremos, antes de ser ejecutado el motor que maneja las JSP preprocesa su contenido y las convierte en auténticos servlets. Esta tecnología no es más que una aproximación a la tecnología servlet de una forma más natural para la creación de la parte estática de una página web.

Una página JSP puede contener códigos en diferentes lenguajes:

Código para interpretar en la máquina cliente:

1. HTML.
2. Un lenguaje interpretado por navegadores como JavaScript o VBScript.

Código para ejecutar en el servidor:

1. Directivas JSP.
2. Código Java insertado entre las etiquetas «<%» y «%>».

3. Expresiones de Java entre las etiquetas «<%= ... %>», que se ejecutan y devuelven un valor que se escribe directamente en la salida.
4. Etiquetas JSP estándar. Son etiquetas que realizan funciones básicas del manejo de las páginas JSP como, por ejemplo, la carga de librerías, la iteración de colecciones, acceso a beans, etc. Ejemplo: `<jsp:useBean id="Bean" scope="request" class="Clase"/>`
5. Etiquetas personalizadas (custom tags). Estas etiquetas son en realidad funcionalidades complejas encapsuladas en clases y publicadas en unos ficheros «.tld» para que se puedan invocar desde las páginas JSP con una sintaxis muy similar a la de las etiquetas HTML. Ejemplo: `<mlibreria:mitag nombre="ejemplo" />`

La diversidad de lenguajes que se pueden agrupar en un JSP ha sido el origen de la principal crítica a esta tecnología. Siempre se argumenta que, aunque evidentemente hacen que el contenido estático se pueda desarrollar más fácilmente, cuando existe mucho contenido dinámico y las aplicaciones tienen funciones avanzadas de configuración, personalización, etc., las páginas JSP acaban siendo una mezcla de muchos lenguajes muy difíciles de seguir por los propios desarrolladores y, por lo tanto, tienen un alto coste, no sólo de desarrollo sino sobre todo de mantenimiento.

Una página JSP se puede ver como una página HTML con trozos de código Java que se ejecutan cuando se solicita la página. Pero lo que es importante entender es lo que sucede cuando se solicita una página JSP, ya que, como se ha dicho más arriba, un JSP acaba siendo un servlet.

Existen dos fases claramente diferenciadas durante la ejecución de un JSP: la fase de generación de código y compilación del servlet generado y la fase de carga de la clase obtenida por la compilación del servlet:

1. El cliente (normalmente un navegador web) hace una petición HTTP al servidor web de una página JSP.
2. El servidor web identifica la petición como una URL que está asociada con algo que se debe ejecutar en el contenedor web y reenvía la petición al contenedor.
3. El contenedor «sabe» que la petición de una página JSP, en realidad, es la petición del servlet generado a partir de la página, por lo que busca la clase compilada y generada a partir del JSP.
 - a) Si no existe esa clase, el contenedor interpreta la JSP para generar un servlet, que es, en ese momento, compilado.
4. Una vez que encontrado ese servlet, el contenedor crea un objeto de la clase correspondiente a dicho servlet generado y compilado y, a partir de ese instante, continúa la ejecución como si fuera un servlet normal.

Inicialmente podríamos pensar que la ejecución de un JSP puede llegar a resultar muy lenta, ya que tenemos que realizar una generación de código y una compilación durante el tiempo que transcurre desde que se hace la petición hasta que se inicia la respuesta del sistema, pero, en realidad, el contenedor de JSP se encarga de optimizar este proceso realizando esas tareas sólo cuando es necesario; algunos, incluso, permiten realizar esta generación y compilación en el mismo momento de la instalación, con lo que incluso el rendimiento de la primera petición del usuario es exactamente igual al que tendría un servlet.

Los Enterprise JavaBeans son componentes de servidor que encapsulan la lógica de negocio de una aplicación y se pueden ejecutar de forma distribuida. Debido a que los beans se ejecutan en contenedores EJB, aprovechan los servicios de bajo nivel que éstos proporcionan, permitiendo al desarrollador centrarse en resolver los problemas del negocio.

Los componentes EJB pueden encontrarse físicamente ubicados en máquinas distintas de las que albergan los componentes que les invocan. Para gestionar esta comunicación los componentes hacen uso del protocolo de llamada remota RMI/IIOP. Estas operaciones de invocación remota son bastante pesadas, por lo que habrá que evaluar si realmente nos interesa pagar este precio para obtener la mejora de rendimiento en otros aspectos.

Existen tres tipos de Enterprise Beans: session, entity y message-driven.

Session beans.

Representa la transacción de un cliente en el servidor de aplicaciones J2EE. El bean desarrolla una tarea específica del negocio en el servidor. Dentro de este tipo de EJB podemos encontrar dos subtipos Stateless y State Full.

- Los stateless session beans son componentes sin estado, es decir, no guardan información entre distintas llamadas de un cliente. Esto permite que el número de objetos dando servicio sea menor que el número de clientes y que, por lo tanto, se puedan utilizar lo que se llaman pool de objetos (se define un conjunto de objetos en memoria y van siendo compartidos por todas las peticiones de los usuarios).
- Los state full session beans son componentes que sí que mantienen el estado del cliente. Por lo tanto, este tipo de EJB se consideran muy pesados, ya que tienen que mantenerse en memoria durante todo el tiempo que dura la conversación entre el cliente y el EJB. Además, el rendimiento puede verse alterado si durante la conversación el número máximo de EJB se alcanza o se acaba la memoria; en ese caso, el contenedor EJB se encarga de escribirlo en disco para poder recuperarlo cuando sea necesario, lo que permitirá al sistema seguir funcionando, aunque con un rendimiento muy inferior.

Entity beans.

Representan las entidades del negocio que perduran en el tiempo, ya sea en una base de datos o cualquier otro sistema de almacenamiento persistente. Reflejan en memoria los datos que maneja la aplicación. Normalmente estos objetos se corresponden con una tabla de una base de datos relacional y su objetivo es trabajar como caché de la base de datos manteniendo la sincronización entre la memoria y la base de datos. Dentro de este tipo de EJB podemos diferenciar dos subtipos:

- Componentes EJB con persistencia gestionada por el bean (bean managed persistence), que son aquellos beans que mantienen la sincronización con el sistema de almacenamiento persistente (normalmente un sistema de gestión de base de datos) gracias a instrucciones explícitas del programador a los métodos de la interfaz EntityBean.

- Componentes EJB con persistencia gestionada por el contenedor (container managed persistence), que son los beans que realizan la sincronización de forma declarativa, es decir, sólo hay que definir el descriptor para que el contenedor tenga la información necesaria para gestionar la persistencia. Cada vez se utilizan más este tipo de beans, aunque al principio los programadores no se fiaban mucho de ellos.

Message driven beans.

Los message driven beans actúan como session beans pero de forma asíncrona. En realidad, son una especie de componentes de tipo listener que se emplean para el procesamiento asíncrono de mensajes. Este tipo de EJB se utiliza cuando las operaciones que va a realizar el EJB son muy costosas en tiempo o cuando el tiempo de respuesta es indeterminado y es mejor dar una respuesta asíncrona que tener esperando al cliente durante mucho tiempo.

Contenedores.

Las especificaciones J2EE definen un contenedor como el software responsable de ofrecer una serie de servicios de ejecución a los componentes J2EE de aplicación. Estos servicios los prestan a través del conjunto de APIs que define la especificación J2EE. Sin embargo, estas llamadas a las API nunca se realizan directamente: se hacen siempre a través del contenedor; de este modo, el contenedor puede incluir la funcionalidad declarada en los ficheros de configuración proporcionados en la instalación de los componentes tales como gestión de transacciones, seguridad, gestión de estados y conexiones.

La especificación J2EE define cuatro tipos de contenedores: contenedores de applets, de aplicaciones Java en cliente, de componentes web y de EJB. Los dos primeros están relacionados con la parte de la arquitectura que se ejecuta en la plataforma cliente y se ha dejado su tratamiento para el tema «Entorno de desarrollo Java para software con distribución».

Contenedor Web.

Un contenedor web es el encargado de ejecutar los componentes web (servlets y JSPs) y de proporcionarles los servicios necesarios de seguridad, concurrencia, transacciones, implantación, etc.

Es importante no confundir un contenedor web con un servidor web: este último está optimizado fundamentalmente para servir páginas HTML estáticas. Un contenedor de web actúa como una capa intermedia de software entre el servidor web y el servlet. El servidor web transmite la petición HTTP al contenedor, éste crea un objeto que implementa la interfaz `HttpServletRequest` y pasa la petición al servlet. El servlet ejecuta su lógica, haciendo las llamadas oportunas a la lógica de negocio y genera una respuesta que el contenedor transmite al servidor web. Por cada petición HTTP el contenedor es responsable de crear y gestionar un nuevo hilo de ejecución que sirva la petición.

Contenedor EJB.

El contenedor de EJB es el lugar donde residen y se ejecutan los EJB instalados en un servidor de aplicaciones J2EE. Proporciona a los EJB un conjunto de servicios como caché, concurrencia, persistencia de los objetos, seguridad, gestión de transacciones, bloqueos, etc. Un contenedor de EJB se arranca automáticamente cuando se arranca el servidor de aplicaciones. En un mismo contenedor pueden residir varios beans.

La plataforma J2EE define una serie de servicios o API que debe proporcionar un servidor de aplicaciones para facilitar y estandarizar el desarrollo de aplicaciones Java basadas en esta arquitectura.

- **JDBC.** JDBC es la API estándar para conectar programas desarrollados en Java con sistemas de gestión de bases de datos (SGBD). Esta API permite realizar peticiones a los SGBD mediante instrucciones SQL. JDBC es muy similar en su concepción al otro estándar de acceso a bases de datos utilizado ampliamente en la industria denominado ODBC. De hecho, el primer nivel de compatibilidad de JDBC es un bridge de acceso JDBC-ODBC para acceder a bases de datos desde programas Java a través de drivers ODBC.
- **Servlet.** La API dedicada a Servlet no es más que el conjunto de funciones explicado de forma genérica en el apartado de componentes Java para el desarrollo web. En cada especificación J2EE se exige la compatibilidad con un estándar concreto (p. ej. en J2EE 1.3 se exige compatibilidad con Servlet 2.3).
- **JSP.** Al igual que con los servlets, la API de JSP es la explicada de forma genérica en el apartado de componentes Java para el desarrollo web. En la especificación J2EE se exige la compatibilidad con un estándar concreto (p. ej. en J2EE 1.3 se exige compatibilidad con JSP 1.2).
- **Java Message Service.** La API JMS es un estándar que pretende facilitar a los componentes J2EE la creación, envío, recepción y lectura de mensajes que se intercambian entre componentes software o aplicaciones. La utilización de mensajes permite una comunicación débilmente acoplada y asíncrona, en la que un componente emisor puede enviar un mensaje a un destino y continuar con la ejecución de la tarea, mientras que el componente receptor recogerá ese mensaje del destino. De este modo, el emisor no tiene que saber nada del receptor ni viceversa. Lo único que ambos deben conocer es el formato del mensaje. Es importante diferenciar este tipo de intercambio de información de las comunicaciones entre componentes fuertemente acoplados como RMI; que requiere que un componente conozca la interfaz que define los servicios que el componente remoto ofrece, así como de la comunicación a través de correo electrónico, que está más preparada para la comunicación entre personas o entre aplicaciones y personas, que entre componentes. La API JMS minimiza los conceptos que un programador debe aprender antes de utilizar un producto de mensajería que proporcione todas las características necesarias para construir una aplicación compleja. Además, maximiza la portabilidad de estas aplicaciones entre diferentes proveedores de productos de mensajería. La API JMS permite comunicación asíncrona y fiable, ya que asegura que cada mensaje llega una y sólo una vez al receptor. Existen dos modos de trabajo con mensajes: basado en colas de mensajes y basado en suscripciones. Ambos modos están soportados por la API JMS.
- **Java Transaction API.** La API JTA está diseñada para permitir a los componentes J2EE gestionar sus propias transacciones, permitiendo que varios componentes participen en una transacción común. Esta API la podemos dividir en dos niveles.
- **Java Mail Technology.** La API Java Mail permite a las aplicaciones incorporar la funcionalidad de envío y recepción de mensajes de correo electrónico de una forma sencilla.
- **Java API for XML (JAXP).** La API JAXP nos facilita el acceso, la generación y en general el desarrollo de nuestros componentes o programas para acceder a ficheros XML.

- **J2EE Connector API.** Se utiliza, fundamentalmente, por integradores de sistemas, para la creación de enlaces entre diferentes sistemas de información con sistemas en arquitectura J2EE. La tecnología J2EE connector habilita a los componentes J2EE para interactuar con otros sistemas de información de la organización como los sistemas integrados de gestión empresarial (ERP), procesadores de transacciones de mainframes (p. ej. CISC), etc. Cada sistema de información que quiera conectarse requiere de un componente J2EE que implementa la tecnología de conexión para ese sistema. Estas implementaciones se denominan resource adapters y serían equivalentes a un driver JDBC, ya que ambos proporcionan un API para el acceso a una aplicación externa al servidor J2EE.
- **Java Authentication and Authorization Service (JAAS).** La API JAAS proporciona un sistema de autenticación y autorización de usuarios y grupos para una aplicación. Su función principal es proporcionar independencia entre los sistemas básicos de autenticación y autorización utilizados y los mecanismos que las aplicaciones basadas en J2EE utilizan para dar acceso a las diferentes funcionalidades. Por ejemplo, permite que una aplicación gestione las funciones reservadas a los administradores de la misma, independientemente de si la autenticación estaba basada en Kerberos, usuario y password o una tarjeta criptográfica.

Webservices en J2EE.

Los webservices son servicios proporcionados por programas que residen en un servidor web a otros programas que pueden acceder a dicho servidor a través del protocolo HTTP.

El objetivo principal de los webservices es poder responder a peticiones de cualquier tipo de cliente, independientemente de su plataforma, que cualquier cliente pueda localizar y llamar a cualquier webservice, independientemente de la plataforma en la que se haya implementado. Por ello, se han definido una serie de estándares que permiten esta interacción entre plataformas heterogéneas.

Basados en XML, se han desarrollado una serie de estándares que permiten el funcionamiento de los webservices. Los estándares más importantes son:

- **UDDI (Universal Description, Discovery and Integration)** es un registro basado en XML para listar los webservices que se ponen a disposición de otros en Internet o la intranet. El objetivo es crear un directorio de servicios web de forma que un sistema pueda buscar los servicios disponibles. Empresas líderes del mercado del software como Microsoft e IBM lideraron esta iniciativa.
- **WSDL (Web Services Description Language).** Como el resto, es un lenguaje basado en XML y cuyo objetivo es describir servicios y permitir acceder a los mismos automáticamente. WSDL es la base de UDDI.
- **SOAP (Simple Object Access Protocol)** es un protocolo basado en XML que se utiliza como un modo de intercambio de información estructurada a través de mensajes entre sistemas que pueden estar en distintas máquinas, posiblemente sobre sistemas operativos distintos.

SOAP se divide en tres partes:

1. El sobre SOAP. Básicamente define el contenido del mensaje y el destinatario del mismo, aunque puede incluir más información.

2. Unas reglas de codificación de SOAP. Es la especificación funcional del intercambio de información de tipos de datos definidos por la aplicación.
3. Una representación RPC SOAP. Define el modo de realizar las llamadas a procedimientos remotos y las respuestas.

SOAP se puede utilizar sobre diferentes protocolos existentes como HTTP o SMTP.

Los mensajes SOAP están escritos en XML y definen dos esquemas: uno para el sobre y otro para las reglas de codificación, por lo tanto, el mensaje no contiene el DTD asociado. Los mensajes SOAP pueden llevar documentos anexos.

El principal objetivo de SOAP es la creación de forma fácil de servicios web distribuidos y que la interoperabilidad entre plataformas heterogéneas fuese transparente, evitando, en la medida de lo posible, la dificultad planteada por otros sistemas como CORBA o RMI.

La plataforma J2EE 1.4 está muy enfocada a integrar los estándares de las tecnologías webservices en la plataforma J2EE, permitiendo que las aplicaciones puedan proporcionar sus servicios a través de SOAP/HTTP. Las API proporcionadas por esta plataforma son:

- La librería Java API for XML Processing (JAXP) es una API para facilitar el análisis y procesamiento de documentos XML. Aunque tiene su propia implementación de un analizador gramatical (parser), permite la utilización de cualquiera que cumpla las especificaciones. Permite tratar documentos XML tanto a través del modelo SAX como a través del modelo DOM. El API JAXP está en el paquete `java.xml.parsers`.

El modelo SAX procesa los documentos en serie, convirtiendo los elementos de un documento XML en una serie de eventos. Cada elemento del documento XML genera un evento concreto. El desarrollador proporciona los métodos manejadores para esos eventos que realizarán las acciones deseadas. El procesamiento con SAX es más rápido por su modo de acceso y menor utilización de memoria.

El modelo DOM de procesamiento de documentos XML genera un árbol a partir del documento. Aunque requiere mucha más memoria para almacenar el árbol, hace que la codificación sea más sencilla.

- Otra librería importante es la Java API for XML-based RPC (JAX-RPC). Esta librería permite la llamada a procedimientos remotos utilizando un protocolo XML. En realidad, lo que se pretende es permitir a un desarrollador Java realizar llamadas a webservices a través del protocolo SOAP. De este modo se permite la conexión de un programa Java con un webservice aunque éste no esté desarrollado en Java.
- La Java API for XML Registries (JAXR) es la API encargada de facilitar el acceso a UDDI o ebXML para acceder a los registros de negocio.
- SOAP with Attachments API for Java (SAAJ) permite a los desarrolladores Java generar y recoger mensajes según la especificación SOAP y SOAP con ficheros adjuntos, que pueden ser documentos XML o ficheros en formato MIME. Otras API como JAX-RPC utilizan SAAJ como base.

Hasta este punto hemos visto las diferentes API que permiten utilizar los estándares de webservices. Pero la integración con la plataforma J2EE es mayor, ya que también define cómo se deben integrar estas tecnologías con los contenedores web y los contenedores EJB. Un webservice en J2EE 1.4 se puede implementar:

1. Utilizando un punto de acceso JAX-RPC que se ha desarrollado con una clase en el contenedor web.
2. Utilizando un punto de acceso a EJB. La implementación del servicio debe ser un session bean sin estado en un contenedor EJB.

2. ARQUITECTURA .NET.

Los objetivos del diseño de una aplicación son los siguientes:

- Disponibilidad (Availability).

En las aplicaciones Web es necesario poder trabajar las 24 horas del día, 7 días a la semana durante todo el año. Ello implica una infraestructura mucho más compleja que en el paradigma cliente servidor.

- Gestionabilidad (Manageability).

Es necesario poder manejar de una manera sencilla y ágil muchos componentes de hardware, de comunicaciones y poder gestionar los diversos permisos para redes con miles de usuarios.

- Rendimiento (Performance).

Desde el punto de vista del usuario, implica una calidad en el tiempo de respuesta de la aplicación, lo cual nos lleva a un contraste directo con el tema de coste que queremos soportar para obtener un nivel determinado de calidad.

- Predictibilidad (Reliability).

En estas aplicaciones Web el coste de un fallo de la misma es muy alto, por ello es básico lograr diseñar, implementar y poner en ejecución sistemas predecibles y fiables.

- Escalabilidad (Scalability).

El concepto básico de escalabilidad consiste en tener necesidad de recursos adicionales para una carga adicional sin tener que modificar sustancialmente la aplicación.

- Seguridad (Securability).

La mayoría de los puntos vulnerables de una aplicación no suelen estar en el propio código en cuanto a seguridad sino en la forma como se ha diseñado la aplicación para tener en cuenta estos temas.

En el proceso de planificación de una aplicación distribuida, los desarrolladores deben realizar numerosas decisiones sobre el diseño y la tecnología a emplear. Deben responder siempre estas decisiones a dos preguntas básicas: ¿esto funciona? Y ¿cómo funcionaría mejor? Para ello se utilizan los siguientes recursos.

Tenemos como herramienta de modelado Microsoft Visio for Enterprise Architects. Así, se puede comenzar a codificar con Visual Basic, entre otros lenguajes, y se puede efectuar reingeniería inversa para crear diagramas UML y continuar refinando el modelo a partir de ahí.

Visual Studio .NET tiene la posibilidad de crear plantillas (Enterprise Templates) para definir la infraestructura de aplicaciones complejas. También tiene la posibilidad mediante Visual Database Tools de crear objetos de una base de datos como tablas, procedimientos almacenados etc. Tiene una herramienta de impresión Standard como Crystal Report para poder crear informes para la aplicación cliente tanto en entorno Web como en el clásico entorno Windows.

Las aplicaciones de empresa requieren manejar peticiones de múltiples sitios remotos, accesos a datos de una gran variedad de almacenamientos y también colaboración de otras aplicaciones externas e internas de la empresa. Vamos ver una relación de productos de Microsoft que pueden ayudar a un mejor diseño de una aplicación.

Application Center 2000.

Es una herramienta para despliegue y gestión de aplicaciones complejas distribuidas compuestas por múltiples sitios Web, Web Services y componentes COM. Permite coordinar una imagen unificada del despliegue y del estatus de los diversos componentes distribuidos. También puede detectar fallos de hardware y software, efectuar automáticamente su recuperación y enviar mensajes de correo con avisos.

SQL Server 2000.

Es una base de datos relacional que permite almacenar, recuperar, analizar y gestionar masa de datos.

SQL Server 2000 Windows CE.

Es la base datos citada en el párrafo anterior adaptada para dispositivos móviles y Pockets PC.

BizTalk Server 2000.

Es un producto compuesto de varias herramientas y servicios para construir soluciones de intercambio de datos entre aplicaciones externas e internas de Internet empleando XML. Consta de herramientas XML para crear y manejar esquemas, enlazar campos origen y destino, así como transformaciones entre ellos y finalmente permite efectuar workflow de tareas.

Exchange 2000 Server.

Consta de una infraestructura de manejo de mensajes y de almacenamiento de datos diseñado para dar servicios de mensajería y colaboración en tiempo real.

Commerce Server 2000.

Es una aplicación parametrizable de e-commerce para crear negocio on-line.

Content Management Server 2001.

Es una herramienta para construir, desplegar y mantener sitios Web dinámicos, tanto para documentos internos como sitios Web comerciales. Contiene plantillas para páginas Web, circuitos de aprobación, planificación de las publicaciones, creación dinámica de páginas, manejo de revisiones y archivo de versiones.

Internet Security an Acceleration Server 2000.

Su función básica es ser un Firewall de empresa y una Web caché de alto rendimiento. Es una protección de la red de la empresa de accesos no autorizados, inspección de tráfico e inicia avisos de ataques a la red.

Windows 2000 DataCenter Server.

Es el sistema operativo para ejecutar aplicaciones críticas de gran volumen y en tiempo real.

Host Integration Server 2000.

Permite conectar aplicaciones .NET a aplicaciones mainframe a procesos y almacenamiento de datos. Con este producto se puede acceder a DB2 y a ficheros planos de un mainframe, AS/400, UNIX y a otros sistemas Windows.

Mobile Information 2001 Server.

Permite extender las aplicaciones .NET a dispositivos móviles y Pockets PC.

SharePoint Portal Server 2001.

Este producto está formado por un conjunto de herramientas y servicios que ayudan a crear y mantener un punto central para publicar información del negocio. Provee con capacidades de almacenamiento de documentos, manejo de búsquedas, permite trabajo en equipo y está integrado con Office.

Componentes y niveles en aplicaciones y servicios.

Se ha convertido en un principio ampliamente aceptado en el diseño de aplicaciones distribuidas, la división de la aplicación en componentes que ofrezcan servicios de presentación, empresariales y de datos. Los componentes que realizan tipos de funciones similares se pueden agrupar en capas, que en muchos casos están organizados en forma de apilamiento para que los componentes que se encuentran por «encima» de una capa determinada utilicen los servicios proporcionados por ésta, y un componente específico utilizará la funcionalidad proporcionada por otros componentes de su propia capa, y otras capas «inferiores», para realizar su trabajo.

Esta visión dividida de una aplicación también se puede aplicar a los servicios. Desde un punto de vista de alto nivel, se puede considerar que la solución basada en servicios está formada por varios servicios, los cuales se comunican entre sí pasando mensajes. Desde el punto de vista conceptual, los

servicios se pueden considerar como componentes de la solución global. Sin embargo, internamente el servicio está formado por componentes de software, al igual que cualquier otra aplicación, los cuales se pueden agrupar de forma lógica en servicios de presentación, empresariales y de datos. Los aspectos importantes que se deben tener en cuenta son los siguientes:

1. Los servicios se diseñan generalmente para comunicarse entre sí con el mínimo grado de acoplamiento. El uso de la comunicación basada en mensajes ayuda a desacoplar la disponibilidad y escalabilidad de los servicios, y basarse en los estándares de la industria, como los servicios Web XML, permite la integración con las demás plataformas y tecnologías.
2. Cada servicio está formado por una aplicación que dispone de sus propios orígenes de datos, lógica empresarial e interfaces de usuario. Un servicio puede presentar el mismo diseño interno que una aplicación tradicional de tres niveles.
3. Puede generar y exponer un servicio que no disponga de una interfaz de usuario directamente asociada (un servicio diseñado para que lo invoquen otras aplicaciones a través de una interfaz de programación). Observe que los componentes que forman un servicio y los componentes que componen las capas empresariales de una aplicación se pueden diseñar de forma similar.
4. Cada servicio encapsula sus propios datos y administra las transacciones atómicas con sus propios orígenes de datos.

Es importante tener en cuenta que las capas son simplemente agrupaciones lógicas de los componentes de software que conforman la aplicación o servicio. Ayudan a diferenciar entre los distintos tipos de tareas que realizan los componentes, facilitando el diseño y la reutilización de la solución. Cada capa lógica contiene un número de tipos de componentes discretos agrupados en subcapas, cada una de las cuales realiza el mismo tipo de tarea específica. Al identificar los tipos genéricos de componentes que existen en la mayoría de las soluciones, puede construir un mapa coherente de una aplicación o servicio y, a continuación, utilizar este mapa como plano técnico para el diseño. En la figura siguiente se muestra una visión simplificada de una aplicación y sus capas.

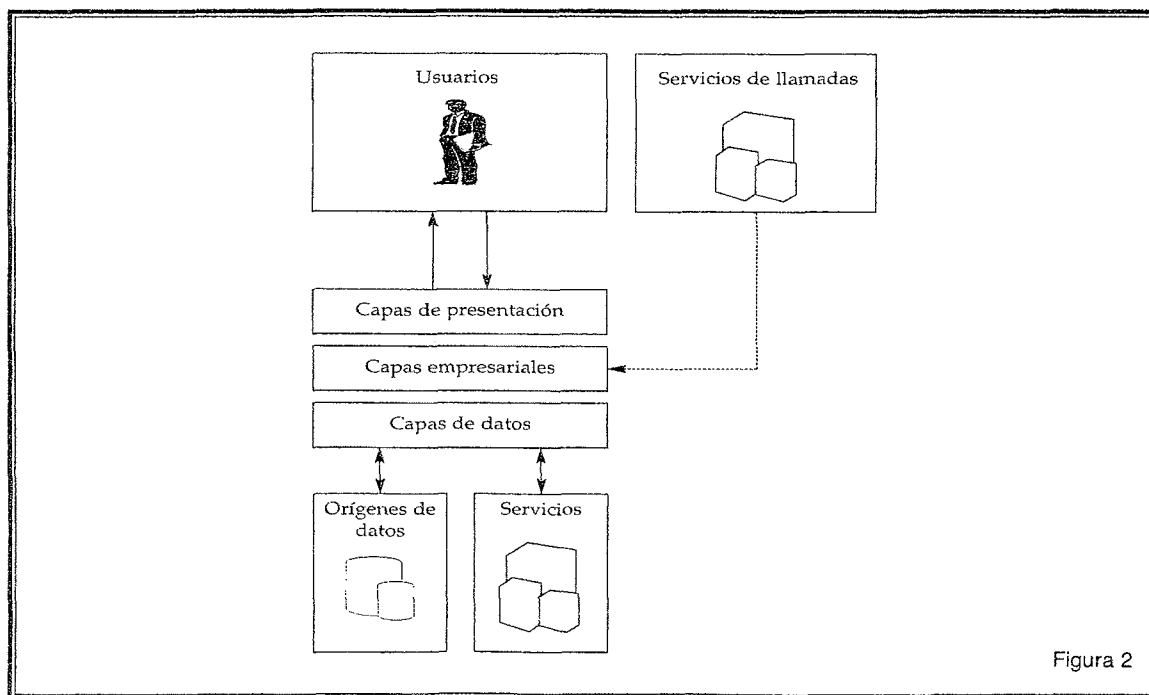


Figura 2

En el apartado anterior se ha descrito cómo una aplicación o un servicio se compone de varios componentes, así como el modo en que cada uno de los cuales realiza una tarea diferente. Todas las soluciones de software contienen tipos de componentes similares, independientemente de las necesidades empresariales que deban cubrir. Por ejemplo, la mayoría de las aplicaciones contienen componentes que tienen acceso a datos, encapsulan reglas empresariales y controlan la interacción con el usuario, entre otros. La identificación de los tipos de componentes que se encuentran normalmente en las soluciones de software distribuidas facilitará la elaboración de un plano técnico para el diseño de aplicaciones o servicios.

El análisis de la mayoría de las soluciones empresariales basadas en modelos de componentes por capas muestra que existen varios tipos de componentes habituales. En la figura siguiente se muestra una ilustración completa en la que se indican estos tipos de componentes.

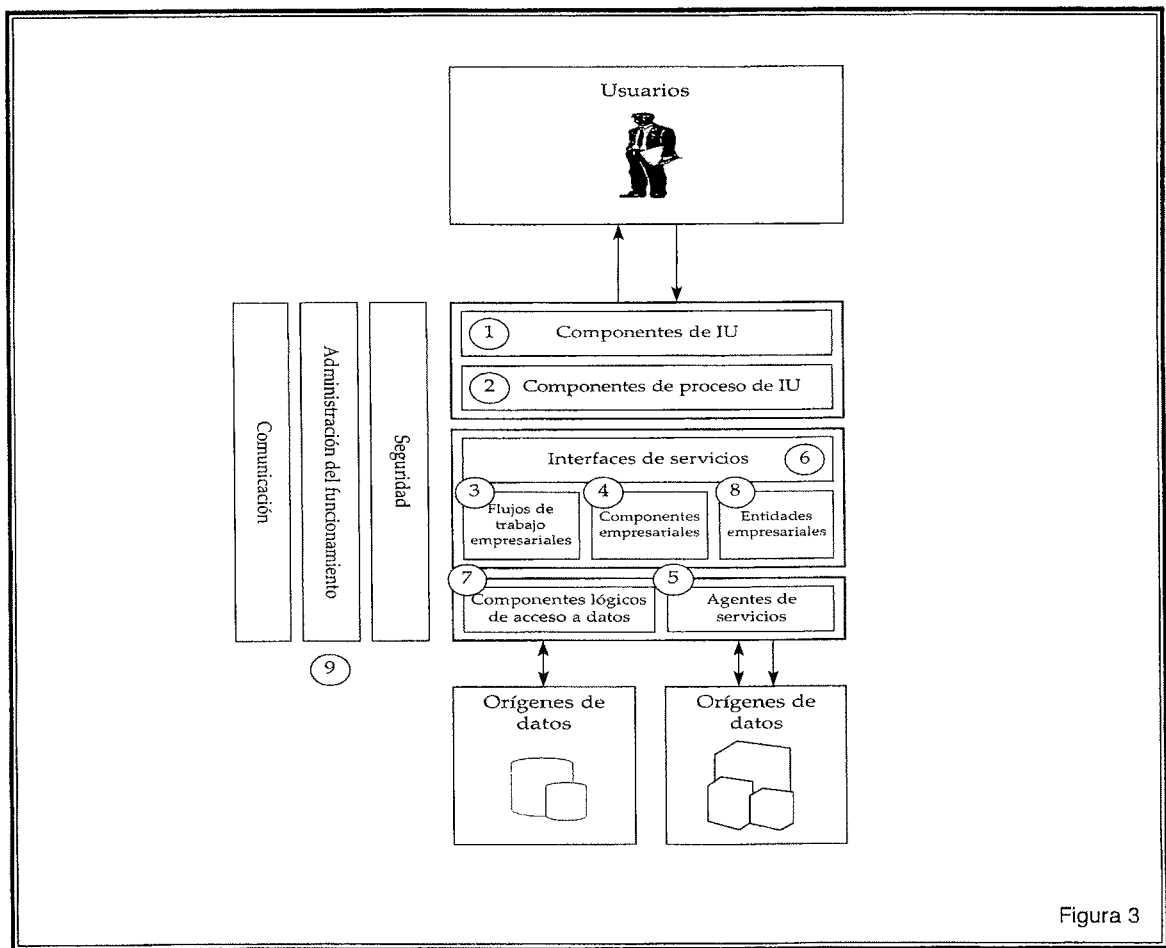


Figura 3

Los tipos de componentes identificados en el escenario de diseño de ejemplo son:

1. Componentes de interfaz de usuario (IU). La mayor parte de las soluciones necesitan ofrecer al usuario un modo de interactuar con la aplicación. En un ejemplo de aplicación comercial, un sitio Web permite al cliente ver productos y realizar pedidos, y una aplicación basada en el

entorno operativo Microsoft Windows® permite a los representantes de ventas escribir los datos de los pedidos de los clientes que han telefonado a la empresa. Las interfaces de usuario se implementan utilizando fundamentalmente páginas Microsoft ASP .NET, que permiten procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de éstos.

2. Componentes de proceso de usuario. En un gran número de casos, la interacción del usuario con el sistema se realiza de acuerdo a un proceso predecible. Por ejemplo, en la aplicación comercial, podríamos implementar un procedimiento que permita ver los datos del producto. De este modo, el usuario puede seleccionar una categoría de una lista de categorías de productos disponibles y, a continuación, elegir uno de los productos de la categoría seleccionada para ver los detalles correspondientes. Del mismo modo, cuando el usuario realiza una compra, la interacción sigue un proceso predecible de recolección de datos por parte del usuario, por el cual éste en primer lugar proporciona los detalles de los productos que desea adquirir, a continuación los detalles de pago y, por último, la información para el envío. Para facilitar la sincronización y organización de las interacciones con el usuario, resulta útil utilizar componentes de proceso de usuario individuales. De este modo, el flujo del proceso y la lógica de administración de estado no se incluye en el código de los elementos de la interfaz de usuario, por lo que varias interfaces podrán utilizar el mismo «motor» de interacción básica.
3. Flujos de trabajo empresariales. Una vez que el proceso de usuario ha recopilado los datos necesarios, éstos se pueden utilizar para realizar un proceso empresarial. Por ejemplo, tras enviar los detalles del producto, el pago y el envío a la aplicación comercial, puede comenzar el proceso de cobro y preparación del envío. Gran parte de los procesos empresariales conllevan la realización de varios pasos, los cuales se deben organizar y llevar a cabo en un orden determinado. Por ejemplo, el sistema empresarial necesita calcular el valor total del pedido, validar la información de la tarjeta de crédito, procesar el pago de la misma y preparar el envío del producto. El tiempo que este proceso puede tardar en completarse es indeterminado, por lo que sería preciso administrar las tareas necesarias, así como los datos requeridos para llevarlas a cabo. Los flujos de trabajo empresariales definen y coordinan los procesos empresariales de varios pasos de ejecución larga y se pueden implementar utilizando herramientas de administración de procesos empresariales, como BizTalk Server Orchestration.
4. Componentes empresariales. Independientemente de si el proceso empresarial consta de un único paso o de un flujo de trabajo organizado, la aplicación requerirá probablemente el uso de componentes que implementen reglas empresariales y realicen tareas empresariales. Por ejemplo, en la aplicación comercial, deberá implementar una funcionalidad que calcule el precio total del pedido y agregue el costo adicional correspondiente por el envío del mismo. Los componentes empresariales implementan la lógica empresarial de la aplicación.
5. Agentes de servicios. Cuando un componente empresarial requiere el uso de la funcionalidad proporcionada por un servicio externo, tal vez sea necesario hacer uso de código para administrar la semántica de la comunicación con dicho servicio. Por ejemplo, los componentes empresariales de la aplicación comercial descrita anteriormente podrían utilizar un agente de servicios para administrar la comunicación con el servicio de autorización de tarjetas de crédito y utilizar un segundo agente de servicios para controlar las conversaciones con el servicio de mensajería. Los agentes de servicios permiten aislar las idiosincrasias de las llamadas a varios servicios desde la aplicación y pueden proporcionar servicios adicionales, como la asignación básica del formato de los datos que expone el servicio al formato que requiere la aplicación.

6. Interfaces de servicios. Para exponer lógica empresarial como un servicio, es necesario crear interfaces de servicios que admitan los contratos de comunicación (comunicación basada en mensajes, formatos, protocolos, seguridad y excepciones, entre otros) que requieren los clientes. Por ejemplo, el servicio de autorización de tarjetas de crédito debe exponer una interfaz de servicios que describa la funcionalidad que ofrece el servicio, así como la semántica de comunicación requerida para llamar al mismo. Las interfaces de servicios también se denominan fachadas empresariales.
7. Componentes lógicos de acceso a datos. La mayoría de las aplicaciones y servicios necesitan obtener acceso a un almacén de datos en un momento determinado del proceso empresarial. Por ejemplo, la aplicación empresarial necesita recuperar los datos de los productos de una base de datos para mostrar al usuario los detalles de los mismos, así como insertar dicha información en la base de datos cuando un usuario realiza un pedido. Por tanto, es razonable abstraer la lógica necesaria para obtener acceso a los datos en una capa independiente de componentes lógicos de acceso a datos, ya que de este modo se centraliza la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma.
8. Componentes de entidad empresarial. La mayoría de las aplicaciones requieren el paso de datos entre distintos componentes. Por ejemplo, en la aplicación comercial es necesario pasar una lista de productos de los componentes lógicos de acceso a datos a los componentes de la interfaz de usuario para que éste pueda visualizar dicha lista. Los datos se utilizan para representar entidades empresariales del mundo real, como productos o pedidos. Las entidades empresariales que se utilizan de forma interna en la aplicación suelen ser estructuras de datos, como conjuntos de datos, DataReader o secuencias de lenguaje de marcado extensible (XML), aunque también se pueden implementar utilizando clases orientadas a objetos personalizadas que representan entidades del mundo real necesarias para la aplicación, como productos o pedidos.
9. Componentes de seguridad, administración operativa y comunicación. La aplicación probablemente utilice también componentes para realizar la administración de excepciones, autorizar a los usuarios a que realicen tareas determinadas y comunicarse con otros servicios y aplicaciones. Estos componentes no se tratarán en este tema.

Diseño de capas de presentación.

La capa de presentación contiene los componentes necesarios para habilitar la interacción del usuario con la aplicación. Las capas de presentación más simples contienen páginas Web de ASP .NET. Las interacciones más complejas conllevan el diseño de componentes de proceso de usuario que permiten organizar los elementos de la interfaz y controlar la interacción con el usuario. Los componentes de proceso de usuario resultan especialmente útiles cuando la interacción del usuario sigue una serie de pasos predecibles, como al utilizar un asistente para realizar una tarea determinada.

Las interfaces de usuario constan normalmente de una página con varios elementos que permiten mostrar datos y aceptar la entrada del usuario. Cuando un usuario interactúa con un elemento de la interfaz, se genera un evento que llama al código de una función de control. Ésta, a su vez, llama a componentes empresariales, componentes lógicos de acceso a datos o componentes de proceso de usuario para implementar la acción deseada y recuperar los datos que se han de mostrar. A continuación, la función de control actualiza los elementos de la interfaz.

Los componentes de la interfaz de usuario deben mostrar datos al usuario, obtener y validar los datos procedentes del mismo e interpretar las acciones de éste que indican que desea realizar una ope-

ración con los datos. Asimismo, la interfaz debe filtrar las acciones disponibles con el fin de permitir al usuario realizar sólo aquellas operaciones que le sean necesarias en un momento determinado. Al procesar datos, los componentes de interfaz de usuario:

- Adquieren y procesan los datos de los componentes empresariales o de los componentes lógicos de acceso a datos de la aplicación.
- Realizan el formateo de valores (como el formato adecuado de las fechas).
- Realizan las tareas de localización de los datos procesados (p. ej., utilizando cadenas de recursos para mostrar los encabezados de las columnas de una cuadrícula en el idioma correspondiente a la configuración regional del usuario).
- Normalmente, procesan los datos de una entidad empresarial. Estas entidades se suelen obtener del componente de proceso de usuario, aunque también se pueden obtener de los componentes de datos. Los componentes de IU pueden procesar datos a través del enlace a datos de su visualización con los atributos y colecciones adecuados de los componentes de la entidad, si ésta se encuentra disponible. Si se encuentra administrando los datos de una entidad como conjuntos de datos, esta tarea resulta bastante sencilla. Si ha implementado objetos de entidad personalizados, tal vez sea preciso implementar código adicional para facilitar el enlace a datos.
- Pueden personalizar el aspecto de la aplicación en función de las preferencias del usuario o el tipo de dispositivo de cliente utilizado.

La interacción del usuario con la aplicación puede seguir un proceso predecible. Por ejemplo, puede que la aplicación comercial requiera que los usuarios escriban los datos de los productos, vean el precio total, escriban los detalles de pago y, finalmente, escriban la información relativa a la dirección del pedido. Este proceso conlleva la visualización y aceptación de la entrada de un número de elementos de interfaz de usuario, y el estado del proceso (los productos solicitados y los detalles de las tarjetas de crédito, entre otros) se debe mantener en la transición de un paso a otro del proceso. Para facilitar la coordinación del proceso de usuario y controlar el mantenimiento del estado requerido al visualizar varias páginas de la interfaz de usuario, puede crear componentes de proceso de usuario.

Los componentes de proceso de usuario se implementan normalmente como clases .NET que exponen métodos a los cuales pueden llamar las interfaces de usuario. Cada método encapsula la lógica necesaria para realizar una acción específica en el proceso de usuario. La interfaz de usuario crea una instancia del componente del proceso de usuario y la utiliza para efectuar la transición a través de los pasos del proceso. Los nombres de las páginas ASP .NET que se deben visualizar para cada uno de los pasos del proceso se pueden insertar en el código del componente de proceso de usuario (enlazándolo estrechamente por tanto a implementaciones específicas de la interfaz de usuario) o se pueden recuperar de un almacén de metadatos, como un archivo de configuración (facilitando la reutilización del componente de proceso de usuario por parte de varias implementaciones de interfaz de usuario). El diseño de componentes de proceso de usuario para su uso por parte de varias interfaces da lugar a una implementación más compleja, debido al aislamiento de los problemas específicos de los dispositivos. No obstante, puede facilitar la distribución del trabajo de desarrollo de la interfaz de usuario entre varios equipos, cada uno de ellos utilizando el mismo componente de proceso de usuario.

Los componentes de proceso de usuario coordinan la visualización de los elementos de la interfaz. Se abstraen de la funcionalidad de procesamiento y adquisición de datos proporcionada por los componentes de la interfaz de usuario.

La separación de la funcionalidad de interacción del usuario en componentes de interfaz y proceso de usuario conlleva las siguientes ventajas:

- El estado de la interacción de usuario de ejecución larga se mantiene más fácilmente, lo que permite el abandono y la reanudación de la interacción, probablemente incluso utilizando una interfaz de usuario diferente. Por ejemplo, un cliente podría agregar varios elementos a una cesta de compra utilizando la interfaz de usuario basada en el Web y, a continuación, llamar a un representante de ventas para completar el pedido.

La partición del flujo de interacción de usuario de las actividades de procesamiento y recolección de datos puede aumentar la facilidad del mantenimiento de la aplicación y proporcionar un diseño limpio al que se puedan agregar fácilmente características aparentemente complejas, como la compatibilidad con el trabajo fuera de línea.

Los componentes de proceso de usuario:

- Proporcionan un modo simple de combinar los elementos de la interfaz de usuario en los flujos de interacción del usuario sin que sea necesario volver a desarrollar el flujo de datos ni la lógica de control.
- Separan el flujo de la interacción del usuario conceptual de la implementación o dispositivo en el que ocurre.
- Encapsulan el modo en que las excepciones pueden afectar al flujo de proceso de usuario.
- Realizan el seguimiento del estado actual de la interacción del usuario.
- No deben inicializar ni participar en transacciones. Mantienen los datos internos relacionados con la lógica empresarial de la aplicación y su estado interno, manteniendo los datos del modo adecuado.
- Mantienen el estado empresarial interno, normalmente aferrándose a una o varias entidades empresariales afectadas por la interacción del usuario. Puede mantener varias entidades en variables privadas o en una matriz interna o tipo de colección adecuado. En el caso de las aplicaciones basadas en ASP .NET, puede mantener las referencias a estos datos en el objeto Session, pero ello limitará la vida útil del proceso de usuario.
- Pueden proporcionar una característica de tipo «guardar y continuar más adelante» por la cual se puede reiniciar la interacción de un usuario en otra sesión. Puede implementar esta funcionalidad guardando el estado interno del componente de proceso empresarial de forma persistente y proporcionando al usuario el modo de continuar una actividad determinada en un momento posterior. Puede crear un componente de utilidad de administración de tareas personalizado para controlar el estado de activación actual del proceso. El estado del proceso de usuario se puede almacenar en una de las siguientes ubicaciones:
- Si el proceso de usuario puede continuar desde otros dispositivos o equipos, deberá almacenarlo de forma central en una ubicación como, por ejemplo, una base de datos.

- Si se encuentra en un entorno desconectado, el estado del proceso de usuario se deberá almacenar de forma local en el dispositivo del usuario.
- Si, por el contrario, el proceso de la interfaz de usuario se ejecuta en una batería de servidores Web, deberá almacenar el estado requerido en una ubicación de servidor central, de modo que se pueda continuar desde cualquiera de los servidores de la batería.
- Puede inicializar el estado interno llamando a un componente del proceso empresarial o a componentes lógicos de acceso a datos.

Diseño de capas de datos.

Casi todas las aplicaciones y servicios necesitan almacenar y obtener acceso a un determinado tipo de datos. Por ejemplo, la aplicación comercial descrita necesita almacenar datos de productos, clientes y pedidos. Al trabajar con datos debe determinar:

- El almacén de datos que utiliza.
- El diseño de los componentes utilizados para obtener acceso al almacén de datos.
- El formato de datos pasados entre componentes y el modelo de programación necesario para ello.

La aplicación o servicio puede disponer de uno o varios orígenes de datos, los cuales pueden ser de tipos diferentes. La lógica utilizada para obtener acceso a los datos de un origen de datos se encapsulará en componentes lógicos de acceso a datos que proporcionan los métodos necesarios para la consulta y actualización de datos. Los datos con los que la lógica de la aplicación debe trabajar están relacionados con entidades del mundo empresarial que forman parte de la empresa. En determinados escenarios, puede disponer de componentes personalizados que representan estas entidades, mientras que en otros puede decidir trabajar con datos utilizando directamente conjuntos de datos ADO .NET o documentos XML.

La mayoría de las aplicaciones utilizan una base de datos relacional como almacén principal de los datos de la aplicación. También se puede utilizar el almacén de Web Microsoft Exchange Server, bases de datos heredadas, el sistema de archivos o servicios de administración de documentos.

Cuando la aplicación recupera datos de la base de datos, puede hacerlo utilizando un formato de conjunto de datos DataReader. A continuación los datos se transfieren entre las capas y los distintos niveles de la aplicación y, finalmente, uno de los componentes los utilizará. Tal vez desee utilizar formatos de datos diferentes para recuperar, pasar y utilizar datos; por ejemplo, puede utilizar los datos de un conjunto de datos para llenar las propiedades de un objeto de entidad personalizado.

Independientemente del almacén de datos utilizado, la aplicación o el servicio utilizará componentes lógicos de acceso a datos para obtener acceso a los datos. Estos componentes abstraen la semántica del almacén de datos subyacente y la tecnología de acceso a datos (como ADO .NET) y proporcionan una interfaz simple de programación para la recuperación y realización de operaciones con datos.

Los componentes lógicos de acceso a datos separan el procesamiento empresarial de la lógica de acceso a datos. Cada uno de estos componentes suele proporcionar métodos para realizar operaciones Create, Read, Update y Delete (CRUD) relacionadas con una entidad empresarial determinada de la

aplicación (por ejemplo, Order). Los procesos empresariales pueden utilizar estos métodos. La interfaz de usuario puede utilizar las consultas específicas para procesar los datos de referencia (como una lista de tipos de tarjetas de crédito válidos).

Cuando la aplicación contiene varios componentes lógicos de acceso a datos, puede resultar útil utilizar un componente de ayuda de acceso a datos genéricos para administrar las conexiones de las bases de datos, ejecutar comandos y almacenar parámetros en caché, entre otros. Los componentes lógicos de acceso a datos proporcionan la lógica necesaria para obtener acceso a datos empresariales específicos, mientras que el componente de ayuda para el acceso a datos centraliza el desarrollo de API de acceso a datos y la configuración de la conexión a éstos, permitiendo de esta forma la reducción de código duplicado. Un componente de ayuda de acceso a datos bien diseñado no debe repercutir negativamente en el rendimiento y proporciona una ubicación central para el ajuste y optimización del acceso a datos:

1. Los componentes lógicos de acceso a datos exponen métodos para insertar, eliminar, actualizar y recuperar datos, incluyendo la provisión de funcionalidad de paginación al recuperar grandes cantidades de datos.
2. Se puede utilizar un componente de ayuda de acceso a datos para centralizar la administración de la conexión y todo el código relacionado con un origen de datos específico.
3. Se recomienda implementar las consultas y operaciones de datos como procedimientos almacenados (si es compatible con el origen de datos) para mejorar el rendimiento y la facilidad de mantenimiento.

Los componentes lógicos de acceso a datos proporcionan acceso simple a funcionalidad de bases de datos (consultas y operaciones de datos), devolviendo estructuras de datos simples y complejas. Ocultan las idiosincrasias de la invocación y el formato del almacén de datos de los componentes empresariales y las interfaces de usuario que las consumen. La implementación de una lógica propia de acceso a datos en los componentes lógicos de acceso a datos permite encapsular toda la lógica de acceso a datos de la aplicación completa en una única ubicación central, lo que facilita el mantenimiento y la extensibilidad de la aplicación.

Cuando se llaman, los componentes lógicos de acceso a datos realizan lo siguiente:

- Llevan a cabo asignaciones y transformaciones simples de argumentos de entrada y salida. De este modo, se abstrae la lógica empresarial de los esquemas de la base de datos y las formas de procedimientos almacenados.
- Obtienen acceso de un único origen. De este modo, aumenta la facilidad del mantenimiento desplazando toda la funcionalidad de agregación de datos a los componentes empresariales, donde los datos se pueden agregar en función de la operación empresarial específica que se está realizando.
- Actúan en una tabla principal y realizan operaciones en tablas relacionadas (los componentes lógicos de acceso a datos no tienen por qué encapsular necesariamente operaciones sólo en una tabla de un origen de datos subyacente). De este modo, se aumenta la facilidad de mantenimiento de la aplicación.

De forma opcional, pueden realizar las siguientes tareas:

- Utilizan un componente de utilidad personalizado para administrar y encapsular esquemas de bloqueo optimistas.
- Utilizan un componente de utilidad personalizado para implementar una estrategia de almacenamiento de datos en caché para los resultados de consultas no transaccionales.
- Implementan el enrutamiento dinámico de datos de sistemas de gran escala que proporcionan escalabilidad a través de la distribución de datos en varios servidores de bases de datos.

Los componentes lógicos de acceso a datos no deben:

- Invocar a otros componentes lógicos de acceso a datos. Un diseño en el que los componentes lógicos de acceso a datos no invocan a los otros componentes del mismo tipo facilita mantener la previsibilidad de los datos y, por tanto, aumenta la facilidad del mantenimiento de la aplicación.
- Inicializar transacciones heterogéneas. Debido a que cada uno de los componentes lógicos de acceso a datos sólo trata con un único origen de datos, no puede existir un escenario en el que uno de estos componentes constituya la raíz de una transacción heterogénea. En determinados casos, sin embargo, un componente lógico de acceso a datos puede controlar una transacción que conlleve varias actualizaciones en un único origen de datos.
- Mantener el estado entre llamadas a métodos.

Servicios web en .NET.

Los Servicios Web constituyen el siguiente paso en la evolución de la tecnología orientada a objetos y representan una revolución al alejarse de las arquitecturas tradicionales tipo cliente-servidor a nuevas arquitecturas distribuidas tipo igual-a-igual (peer-to-peer). Estos servicios consisten en un conjunto de estándares que permiten a los desarrolladores implementar aplicaciones distribuidas, utilizando herramientas muy distintas para crear aplicaciones que utilizan una combinación de módulos de software, que son llamados desde diversos sistemas distribuidos en regiones geográficas distintas.

La arquitectura de los servicios Web es una meta-arquitectura que permite que ciertos servicios de red sean dinámicamente descritos, publicados, descubiertos e invocados en un ambiente de cómputo distribuido.

Los servicios Web son aplicaciones auto-contenidas y modulares que pueden ser:

- Descritas mediante un lenguaje de descripción de servicio, como el lenguaje WSDL (Web Service Description Language).
- Publicadas al someter las descripciones y políticas de uso en algún Registro bien conocido, utilizando el método de registro UDDI (Universal Description, Discovery and Integration).
- Encontradas al enviar peticiones al Registro y recibir detalles de ligamiento (binding) del servicio que se ajusta a los parámetros de la búsqueda.
- Asociadas al utilizar la información contenida en la descripción del servicio para crear una instancia de servicio disponible o proxy.

- Invocadas sobre la red al utilizar la información contenida en los detalles de ligamiento de la descripción del servicio.
- Compuestas con otros servicios para integrar servicios y aplicaciones nuevas.

Las componentes de los servicios Web son:

- Servicio. La aplicación es ofrecida para ser utilizada por solicitantes que respetan los requisitos especificados por el proveedor de servicios. La implementación se realiza sobre una plataforma accesible en la red. El servicio se describe a través de un lenguaje de descripción de servicio. Tanto la descripción como las políticas de uso han sido publicadas de antemano en un registro.
- Proveedor de Servicio. Desde el punto de vista comercial, es quien presta el servicio. Desde el punto de vista de arquitectura, es la plataforma que provee el servicio.
- Registro de Servicios. Es un depósito de descripciones de servicios que puede ser consultado, donde los proveedores de servicios publican sus servicios y los solicitantes encuentran los servicios y detalles para utilizar dichos servicios.
- Solicitante de servicios. Desde el punto de vista comercial, la empresa que requiere cierto servicio. Desde el punto de vista de la arquitectura, la aplicación o cliente que busca e invoca un servicio.

Operaciones de Servicios Web:

- Publicar/cancelar. Los proveedores de servicios publican (publicitan) la disponibilidad de su servicio comercial (e-business) a uno o más Registros de servicios, o cancelan la publicación de su servicio.
- Búsqueda. Los solicitantes de servicios interactúan con uno o más Registros de servicios para descubrir un conjunto de servicios comerciales con los que pueden interactuar para encontrar una solución.
- Ligar, unir (Bind). Los solicitantes de servicios negocian con los proveedores de servicios para acceder e invocar servicios comerciales (e-business).

Entre las razones por las cuales los servicios Web jugarán un rol principal en la siguiente generación de sistemas distribuidos, están:

- Interoperabilidad. Cualquier servicio Web puede interactuar con cualquier otro servicio Web. El protocolo estándar SOAP permite que cualquier servicio pueda ser ofrecido o utilizado independientemente del lenguaje o ambiente en que se haya desarrollado.
- Omnipresencia. Los servicios Web se comunican utilizando HTTP y XML. Cualquier dispositivo que trabaje con éstas tecnologías puede suministrar o utilizar servicios Web.
- Barrera mínima de participación. Los conceptos detrás de los servicios de Web son fáciles de comprender y se ofrecen Herramientas de Desarrollo (ToolKits) tanto por parte de Mi-

Microsoft como por el resto de proveedores importantes del mercado (IBM, Sun Microsystems, etc.). Estas herramientas permiten a los desarrolladores crear e implementar rápidamente servicios web.

- Apoyo de las industrias. Todas las compañías apoyan el protocolo SOAP y la tecnología derivada de los servicios Web.

Retos de los servicios web.

Para que los servicios Web tengan éxito, se requiere vencer algunos retos técnicos, entre los cuales se encuentran:

- Descubrimiento. ¿Cómo se anuncia un servicio Web para ser descubierto por otros servicios? ¿Qué sucede si el servicio es modificado o se cambia una vez que ha sido anunciado? Existen dos estándares nuevos que están diseñados para ello, el WSDL (Web Services Definition Language) y el UDDI (Universal Description, Discovery and Integration).
- Confiabilidad. Algunos sistemas huésped de servicios Web serán más confiables que otros. ¿Cómo se puede medir ésta confiabilidad y ser comunicada? ¿Qué sucede cuando un huésped de servicio temporalmente se sale de línea? ¿Cómo se localiza o utiliza un servicio alternativo hospedado en otra compañía, o se pone en espera de que se reactive el servicio original? ¿Cómo se sabe en qué otra compañía se puede confiar?
- Seguridad. Algunos servicios Web estarán públicamente disponibles y con poca seguridad, pero la mayoría de los servicios comerciales utilizarán comunicaciones encriptadas con autenticación. Es probable que el protocolo HTTP sobre SSL proveerá la seguridad básica, pero los servicios individuales requerirán de un mayor nivel de especificidad. ¿Cómo autentifica un servicio Web a sus usuarios? ¿Cómo distingue un servicio los niveles de privilegios entre los diversos usuarios?
- Transacciones. Los sistemas tradicionales de transacciones utilizan un método de compromiso de dos fases -se recolectan todos los recursos participantes y se aseguran éstos hasta que se lleva a cabo la transacción completa-. Terminada la transacción, se liberan los recursos. Este método puede funcionar bien en ambientes cerrados donde las transacciones son de corta duración, pero no trabaja bien en ambientes abiertos donde las transacciones pueden tomar horas, si no es que días.
- Administración. ¿Qué tipos de mecanismos se requieren para administrar un sistema altamente distribuido? ¿Es posible delegar la administración de algunos servicios Web a otros?
- Contabilidad. ¿Cómo se define cuánto tiempo puede un usuario acceder y ejecutar un servicio Web? ¿Cómo se pueden cobrar los servicios Web? ¿Cómo será la comercialización del servicio, bajo suscripción o pago por evento?
- Pruebas. ¿Cómo se puede depurar un servicio Web que proviene de diferentes compañías, es hospedado en diferentes ambientes y en diversos sistemas operativos?

Estándares de Servicios Web.

Los servicios Web se registran y anuncian utilizando los siguientes servicios y protocolos. Mucho de estos estándares y otros están siendo desarrollados en el proyecto UDDI, un consorcio de industrias que coordina los esfuerzos de diseño y creación.

- XML (eXtensible Markup Language). Este estándar ha revolucionado la forma en que estructuramos, describimos e intercambiamos información. Independientemente de múltiples formas en que utiliza hoy en día el XML, todas las tecnologías de servicios Web se basan en XML. El diseño de XML se deriva de dos fuentes principales: SGML (Standard Generalized Markup Language) y de HTML (HyperText Markup Language).
- UDDI (Universal Description, Discovery and Integration) es un protocolo para describir los componentes disponibles de servicios Web. Este estándar permite a las empresas registrarse en un tipo de directorio «página amarilla» de Internet que les ayuda anunciar sus servicios, de tal forma que las compañías se puedan encontrar unas a otras y realizar transacciones en la Web. El proceso de registro y consultas se realiza utilizando mecanismos basados en XML y HTTP(S). En el proyecto UDDI se trabaja para proveer un método de acceso común a los metadatos necesarios para determinar si un elemento de código previamente elaborado es suficiente y, si lo es, cómo accederlo.
- SOAP (Simple Object Access Protocol) es un protocolo para iniciar las conversaciones con un servicio UDDI. El SOAP simplifica el acceso a los objetos, permitiendo a las aplicaciones invocar métodos objeto o funciones, que residen en sistemas remotos. Una aplicación SOAP crea una petición en XML, proporcionando los datos necesarios para el método remoto, así como la ubicación misma del objeto remoto.
- WSDL (Web Service Description Language) es el estándar propuesto para la descripción de los servicios Web, el cual consiste en un lenguaje de definición de interfaz (IDL, Interface Definition Language) de servicio basado en XML, que define la interfaz de servicio y sus características de implementación. El WSDL es apuntado en los registros UDDI y describe los mensajes SOAP que definen un servicio Web en particular.
- ebXML (e-business XML) define componentes centrales, procesos, registros y almacenajes comerciales, servicios de mensajes, acuerdos de intercambio comercial y seguridad.

Analogías y diferencias entre J2EE y .NET.

Como principal competidor de .Net se presenta J2EE, Java 2 Enterprise Edition, una versión de Java con librerías de clase añadidas, que usa la máquina virtual Java, y tiene muchas características similares a .Net. Java es un lenguaje bastante maduro, con soporte de cientos de librerías fuera de las básicas y con una comunidad bastante extensa. En ese sentido, C# vs. J2EE puede tratarse de una batalla «comunidad» frente a Microsoft, y no está claro quién la va a ganar. Lo que sí está claro es que Microsoft apuesta por .Net, como centro de su estrategia, y que cuando Microsoft apuesta por algo, suele acabar ganando. Es posible que coexistan las dos plataformas y es posible que se abran la una a la otra; por ejemplo, que haya intérpretes CLR que corran dentro de una JVM o viceversa.

La apuesta que no se puede perder es la apuesta por los servicios web y aplicaciones basadas en XML. Todos los grandes de la industria apuestan por ellas, y gran parte de las aplicaciones de cara al usuario, el middleware y los servidores entenderán y servirán XML. Es decir, que independientemente de la plataforma, XML será el vencedor.

Veamos de forma resumida las similitudes y diferencias tanto tecnológicas como de servicios entre Microsoft .NET y J2EE. Veremos que todos los conceptos aparecen en ambas arquitecturas.

CARACTERÍSTICA	.NET	J2EE
Tipo de tecnología	Producto	Estándar
Empresas que lo ofrecen	Microsoft	Más de 30
Librerías de desarrollo	.NET Framework SDK	Java core API
Intérprete	CLR	JRE
Páginas dinámicas	ASP .NET	Servlets, JSP
Componentes	.NET Managed Components	EJB
Acceso a bases de datos	ADO .NET	JDBC, SQL/J
Servicios Web	SOAP, WDSL, UDDI	SOAP, WDSL, UDDI
Interfaces gráficas	Win Forms y Web Forms	Java Swing
Herramienta de programación	Visual Studio .NET	Dependiente del fabricante
Transacciones distribuidas	MS-DTC	JTS
Servicios de directorios	ADSI	JNDI
Librería de encolado de mensajes	MSMQ	JMS 1.0
Lenguajes utilizados	C#, Visual Basic, C++, otros	Java
Lenguaje intermedio	IL	Bytecodes

Los parecidos entre las dos plataformas en los aspectos principales son evidentes:

Un lenguaje de programación que, unido a una librería de clases, compila a un código intermedio independiente de la máquina («Intermediate Language» en el caso de Microsoft .NET y «Bytecodes» en el caso de Java). Este código se ejecutará en máquina virtual, que proporciona un «entorno de ejecución» que transformará el lenguaje intermedio a código propio de la máquina en la que se corre la aplicación («Common Language Runtime (CLR)» en Microsoft .NET y «Java Runtime Environment (JRE)» en J2EE). Las dos plataformas disponen de multitud de servicios que facilitan la labor del programador, entre ellos, el acceso a bases de datos, los servicios de directorio, las transacciones distribuidas, etc.

