4 SOLUCIÓN PROPUESTA: J2EE

Java 2 Platform, Enterprise Edition (J2EE), es un estándar que define una plataforma de desarrollo empresarial basada en Java. Ofrece un modelo de aplicaciones distribuidas multicapa, con componentes reusables, un modelo de seguridad unificado, un control flexible de transacciones y soporte para Servicios Web. Proporciona soluciones al diseño, desarrollo, montaje y despliegue de aplicaciones empresariales, independientes del sistema operativo, en un entorno distribuido.

La plataforma J2EE ha sido diseñada en un proceso abierto, con conversaciones de una gran variedad de empresas informáticas para asegurar que aporta soluciones al más amplio rango posible de requisitos de las aplicaciones empresariales. Propone una arquitectura distribuida multicapa de servidor intermedio con un modelo basado en componentes.

La especificación de J2EE incluye un conjunto de especificaciones relacionadas de tecnologías utilizadas, una implementación de referencia J2EE (RI), un test de compatibilidad (Compatibility Test Suite, CTS) y un conjunto de guías y prácticas recomendadas (EnterpriseJava Blueprints). Existen varias versiones de J2EE, por ejemplo, la especificación JRS-151 que define la versión 1.4 de J2EE, y recientemente se ha aprobado la versión Java EE 5 (que, entre otros, introduce un cambio en la nomenclatura) en la especificación JRS-244.

El conjunto de tecnologías que incluye la especificación de J2EE, definen la arquitectura principal de la plataforma. Las especificaciones claves de este conjunto son la especificación de Java Servlets, la especificación de JavaServer Pages (JSP) y la especificación de Enterprise JavaBeans (EJB). La mayor parte de las especificaciones de las distintas tecnologías consisten en definiciones de interfaces de programación (API) para el desarrollo de aplicaciones. Las

abstracciones de las API corresponden a interfaces y clases abstractas. Esto permite la existencia de múltiples implementaciones de distintos fabricantes, algunas de ellas libres.

Una aplicación J2EE, puesto que se desarrolla de acuerdo a un estándar, es decir, sólo utiliza las APIS estándares, no depende de una implementación particular de la plataforma. Aunque existen múltiples implementaciones de distintos fabricantes, con todas ellas la aplicación será totalmente compatible. El J2EE Compatibility Test Suite ayuda a maximizar la portabilidad de las aplicaciones validando el cumplimiento de la especificación de los productos de la plataforma J2EE. Se ha creado para que los desarrolladores puedan comprobar que sus aplicaciones trabajan conforme al estándar y por tanto, que sus componentes serán portables en cualquier plataforma J2EE. Además, puesto que esta plataforma está basada en Java, las aplicaciones podrán ejecutarse en cualquier sistema operativo que soporte una máquina virtual de Java (JMV).

Sun Java 2 SDK Enterprise Edition es la implementación de referencia. Es una implementación completa del estándar J2EE que representa una definición operacional de la plataforma y es proporcionada por Sun Microsystems. Esta implementación no es eficiente, pero es especialmente útil a los fabricantes de servidores J2EE, además, es la plataforma estándar para ejecutar el test de compatibilidad CTS. Junto a esta implementación, existe un gran número de fabricantes que venden servidores de aplicaciones certificados J2EE, en la página de Sun MicroSystems existe una lista completa.

Las guías y prácticas incluidas en la especificación J2EE, EnterpriseJava Blueprints, conforman la documentación más completa y útil de la plataforma. Junto a esta documentación ha surgido mucha documentación complementaria como manuales, ayudas, artículos e información compartida por comunidades de usuarios.

Existen otros estándares relacionados con J2EE. El estándar J2EE (Java 2 Platform, Enterprise Edition) se centra en el desarrollo de aplicaciones empresariales. Este estándar se apoya en J2SE (Java 2 Platform, Standard Edition), que está enfocado al desarrollo de aplicaciones y API en general. Algunas API que en un principio pertenecían a J2EE con el tiempo se han pasado a J2SE. En muchos casos, en el desarrollo de nuestras aplicaciones utilizaremos API pertenecientes a J2SE. Por último, hacemos referencia al estándar J2ME (Java 2 Platform, Micro Edition), centrado en aplicaciones de dispositivos móviles.

El J2EE SDK proporcionado por Sun no es un producto comercial. Se ofrece gratuitamente a la comunidad de desarrolladores para ayudar a agilizar la adopción del estándar por parte de éstos. Es útil para el desarrollo de demos y prototipos. El J2EE SDK también incluye verificación de la aplicación y herramientas de despliegue para simplificar el desarrollo.

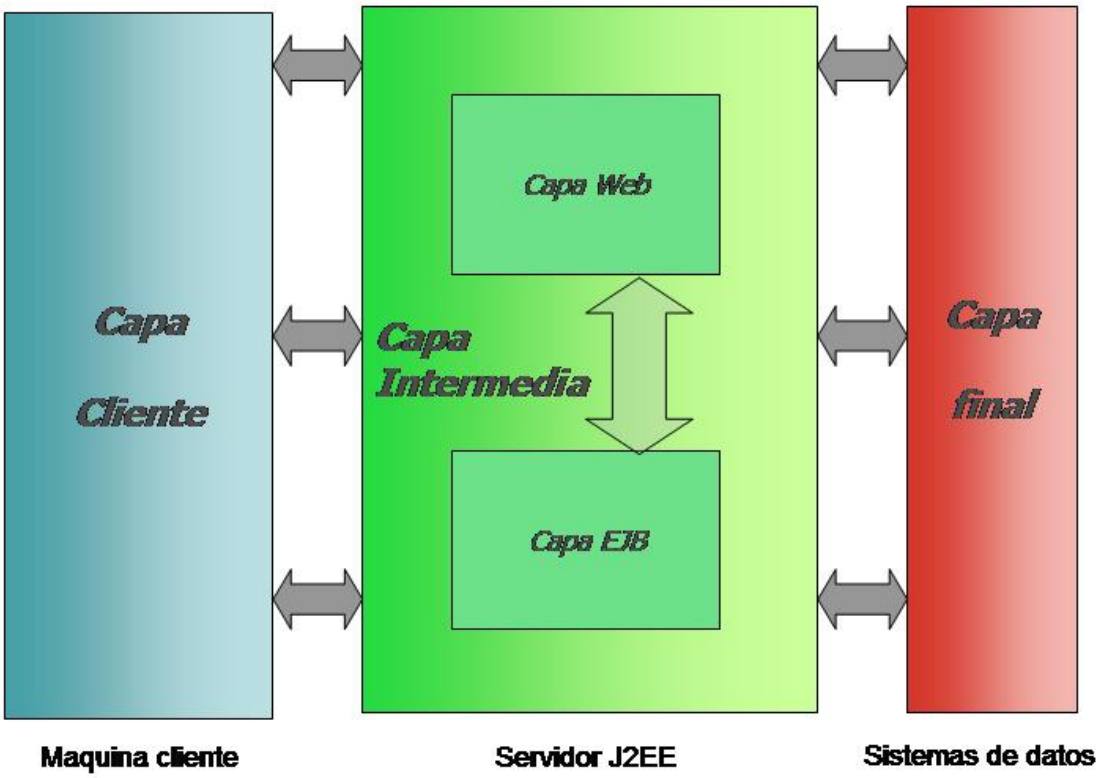
Aunque el J2EE SDK no es una herramienta comercial, existe gran variedad de implementaciones comerciales. Algunos de los servidores J2EE más importantes son: BEA WebLogic Server, Inprise/Borland AppServer, IBM WebSphere ApplicationServer, IONA iPortal Application Server, Sun ONE Application Server, Macromedia JRun Server, Sun Java 2 SDK Enterprise Edition.

Puesto que J2EE es una tecnología basada en componentes como Servlets, JavaServer Pages (JSP) o Enterprise JavaBeans (EJB), también existen implementaciones parciales de la plataforma. Por ejemplo, Tomcat es una implementación de un contenedor de servlets y JSP, y algunas de las implementaciones de contenedores de EJB más importantes son JBoss, JOnAS y OpenEJB.

4.1 ARQUTECTURA FÍSICA MULTICAPA

La plataforma J2EE proporciona modelo de aplicación multicapa distribuido. Esto significa que distintas partes de una aplicación pueden ejecutarse en diferentes máquinas.

La arquitectura J2EE define un modelo multicapa de servidor intermedio formado por una capa cliente, una capa intermedia, y una capa final con los sistemas de datos. La capa cliente soporta variedad de tipos de clientes, tanto fuera como dentro del firewall corporativo. La capa intermedia está formada por la capa Web y la capa de negocio también llamada de Enterprise JavaBeans (EJB), y soporta los servicios de clientes y los de la lógica de negocio. En la capa final los sistemas de información empresariales son accesibles a través de interfaces de programación estándares.



F. 4-1: Arquitectura multicapa J2EE.

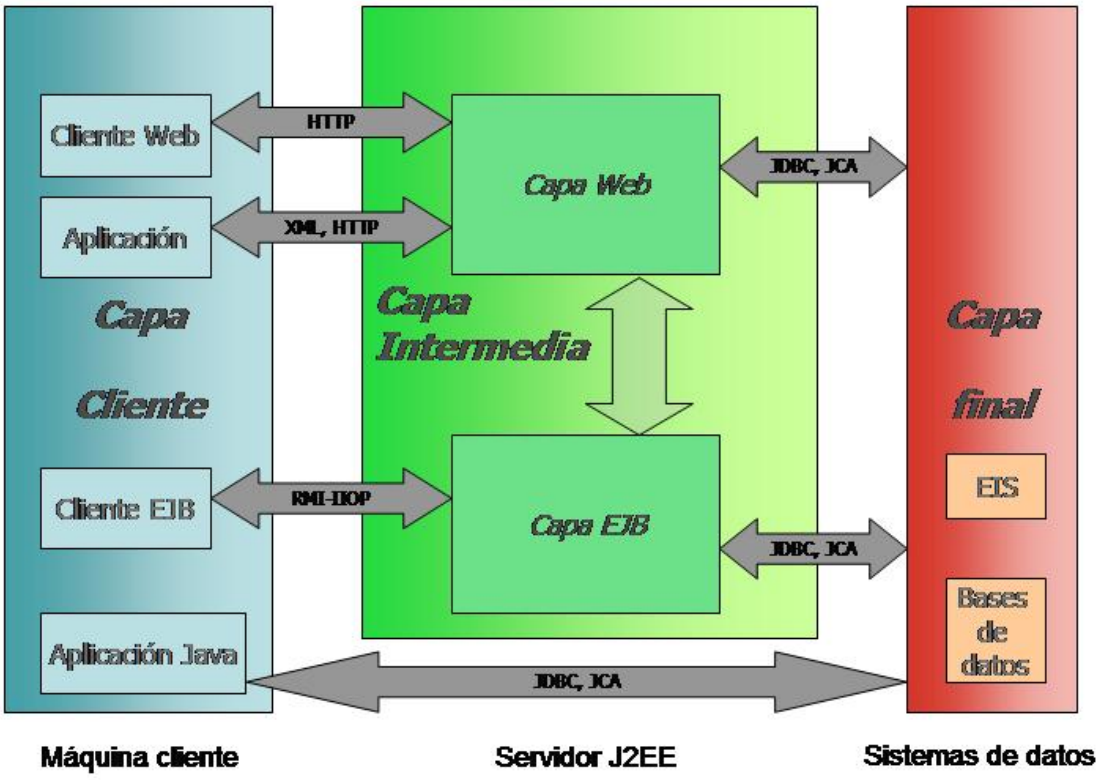
Con esta plataforma se hacen posibles distintos escenarios de aplicación. El modelo de programación es suficiente flexible para aplicaciones que soportan variedad de tipos de clientes, refleja un rango posible de configuraciones incluyendo casos donde el cliente actúa en solitario con la capa Web, donde los clientes interactúan directamente con la capa EJB, y un modelo de aplicaciones multicapa completo.

Las especificaciones de J2EE potencian la diversidad de arquitectura de las aplicaciones y no favorecen implícitamente a un escenario u otro, por lo que un producto J2EE debe soportar cualquiera de estos escenarios. Aunque está diseñada para la construcción de entornos distribuidos también admite configuraciones no distribuidas. Podríamos encontrar clientes en distintas máquinas y el servidor J2EE y los sistemas de datos en otra, o bien, todas las capas en

una sola máquina. Además, también se consideran configuraciones con varios clientes, varios servidores y varios sistemas de datos distintos.

La especificación J2EE no obliga a una configuración determinada, es suficientemente flexible para soportar configuraciones de aplicación que satisfagan los requisitos de diseño específicos. Las decisiones y elecciones a nivel de aplicación son un compromiso entre riqueza funcional y complejidad. Gracias a la integración de capas, este modelo se anticipa al crecimiento y potencia la reusabilidad del código.

En la figura F. 4-2: Arquitectura J2EE completa., se muestra la arquitectura completa de una aplicación J2EE. Sin embargo, la flexibilidad de la arquitectura J2EE permite el desarrollo de aplicaciones más simples, detalladas a continuación.



F. 4-2: Arquitectura J2EE completa.

4.1.1 Capa cliente

La capa cliente se ejecuta normalmente en la máquina cliente y soporta gran variedad de tipos de clientes tanto con firewall corporativo o sin él.

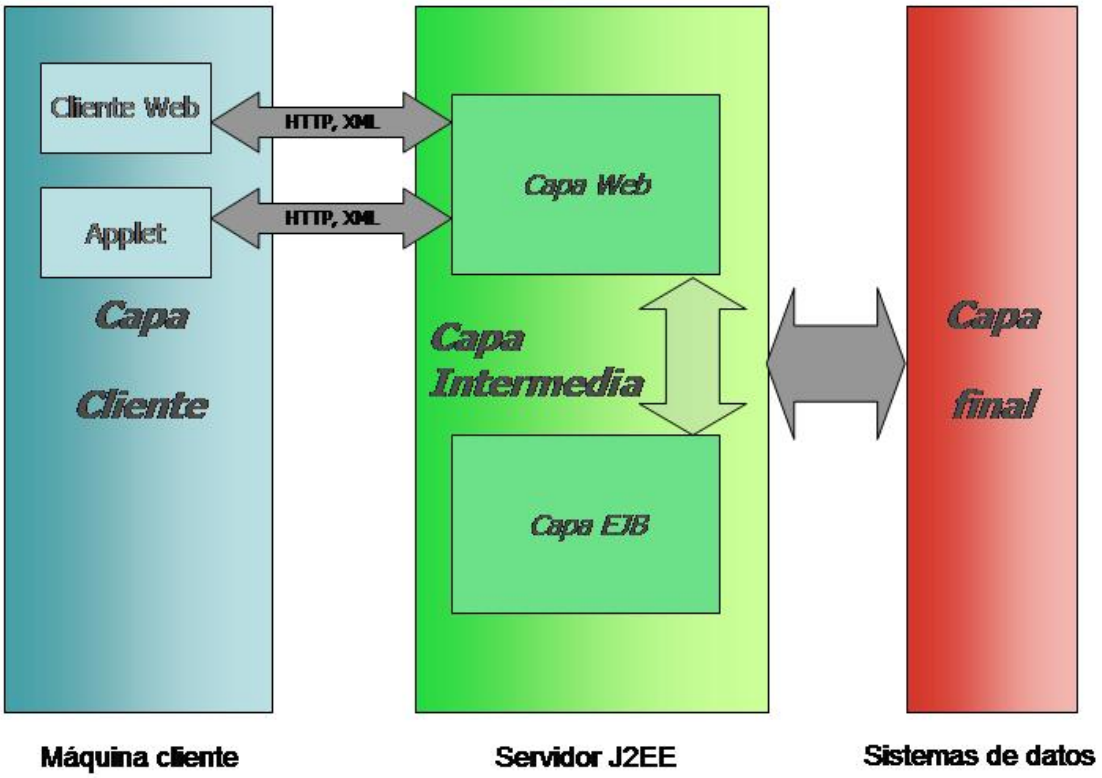
La figura F. 4-2: Arquitectura J2EE completa., muestra como los clientes pueden comunicase directamente con la capa de negocio del servidor J2EE o a través de la capa Web. Al diseñar aplicaciones J2EE es necesario decidir qué tipo de cliente utilizar. En esta decisión se tendrá en cuenta si se requiere mantener la funcionalidad de en la capa cliente, cerca del usuario, o incluir la mayor parte de la funcionalidad en el servidor. Cuanta más funcionalidad se implemente en el servidor mayor movilidad obtendremos ya que permitirá la utilización de una mayor diversidad de clientes en una misma aplicación.

4.1.1.1 Cliente Web y applets

El cliente más común en la estructura J2EE es el navegador usando páginas HTML o páginas JSP. Las páginas estáticas o dinámicas se ejecutan en la capa Web, y el navegador en la capa cliente.

Para interfaces de usuario más complejas a veces se proporcionan applets, que se descargan a través del navegador y se ejecutan en el cliente. Un applet es una pequeña aplicación cliente escrita en Java que se ejecuta en la máquina virtual del navegador. En muchas ocasiones este tipo de clientes necesitan tener instalado en el navegador el Java Plug-in para ejecutarse correctamente.

Normalmente los clientes puramente Web se prefieren al uso de applets, ya que no necesitan el Java Plug-in ni otra herramienta adicional ajena al propio navegador. Estos clientes además permiten un diseño más limpio y modular porque proporcionan un modo de separar la programación de la aplicación del diseño de las propias páginas Web, lo que permite una clara separación de roles entre diseñadores de página y programadores.

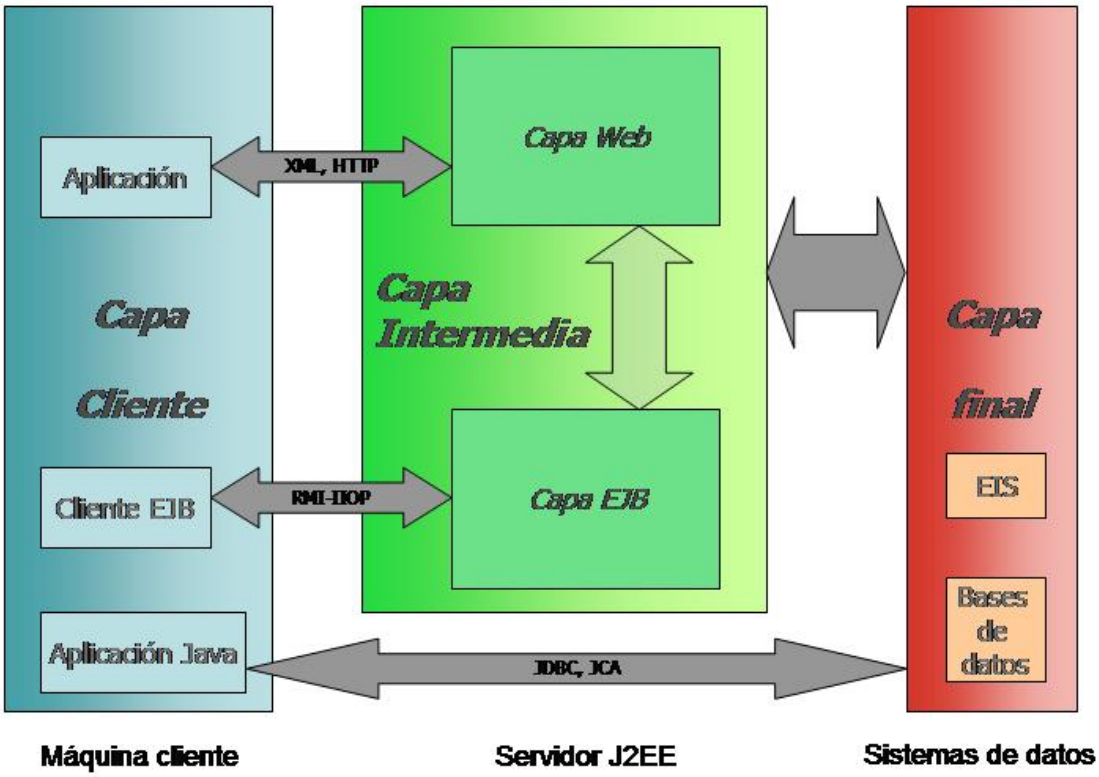


F. 4-3: Arquitectura multicapa J2EE. Clientes Web y Applets.

4.1.1.2 Aplicaciones clientes

Las aplicaciones clientes se ejecutan en la máquina cliente y proporcionan un modo para que los usuarios manejen tareas que requieren una interfaz gráfica más compleja que la que pueda obtenerse utilizando un lenguaje de etiquetas en una página Web.

Este tipo de aplicaciones normalmente incluyen una interfaz gráfica de usuario, aunque también es posible una interfaz de línea de comandos. La interfaz gráfica en aplicaciones Java normalmente se crea utilizando las API Swing o AWT (Abstract Window Toolkit).



F. 4-4: Arquitectura J2EE. Aplicaciones clientes.

Los clientes EJB pueden interactuar directamente con la capa de negocio (EJB) utilizando el protocolo de comunicación RMI-IIOP.

Si la aplicación está implementada en Java pueden también acceder directamente a los sistemas de información directamente utilizando JDBC (Java Database Connector, API de acceso a datos) o la arquitectura de conector J2EE (JCA). En este caso, presentación y lógica de negocio estarán integradas en una única capa.

Las aplicaciones, implementadas en Java u otro lenguaje de programación, pueden establecer una conexión HTTP para comunicarse con la capa Web y consumir contenido Web dinámico, normalmente mensajes XML. En este escenario la capa Web maneja esencialmente transformaciones XML y proporciona conectividad Web a los clientes. La lógica de negocio y el acceso a datos puede implementarse también en la capa Web, pero idealmente se hará desde la capa EJB.

Los clientes no-Java como aplicaciones desarrolladas en lenguajes de programación como Visual Basic u otros, pueden presentar servicios J2EE a los usuarios utilizando el protocolo HTTP. Esto es posible por que los servicios presentados por los componentes de la capa Web utilizan este protocolo estándar, por lo que es fácil acceder a ellos prácticamente desde cualquier programa que esté ejecutándose en cualquier sistema operativo.

A menudo las empresas proporcionan las aplicaciones clientes, escritas en leguaje de programación Java, a través de la descarga desde un navegador utilizando la tecnología Web Start. Este tipo de componentes han aumentado las características disponibles en la interfaz de usuario. La tecnología Java Web Start permite el despliegue de aplicaciones a través de la descarga de los procesos creados. Se comunican con el servidor utilizando el entorno J2SE para ejecutar XML sobre HTTP.

4.1.1.3 Cliente inalámbrico

Está basado en la tecnología MIDP (Mobile Information Device Profile), un conjunto de API de Java que proporciona un completo entorno J2ME para estructuras inalámbricas.

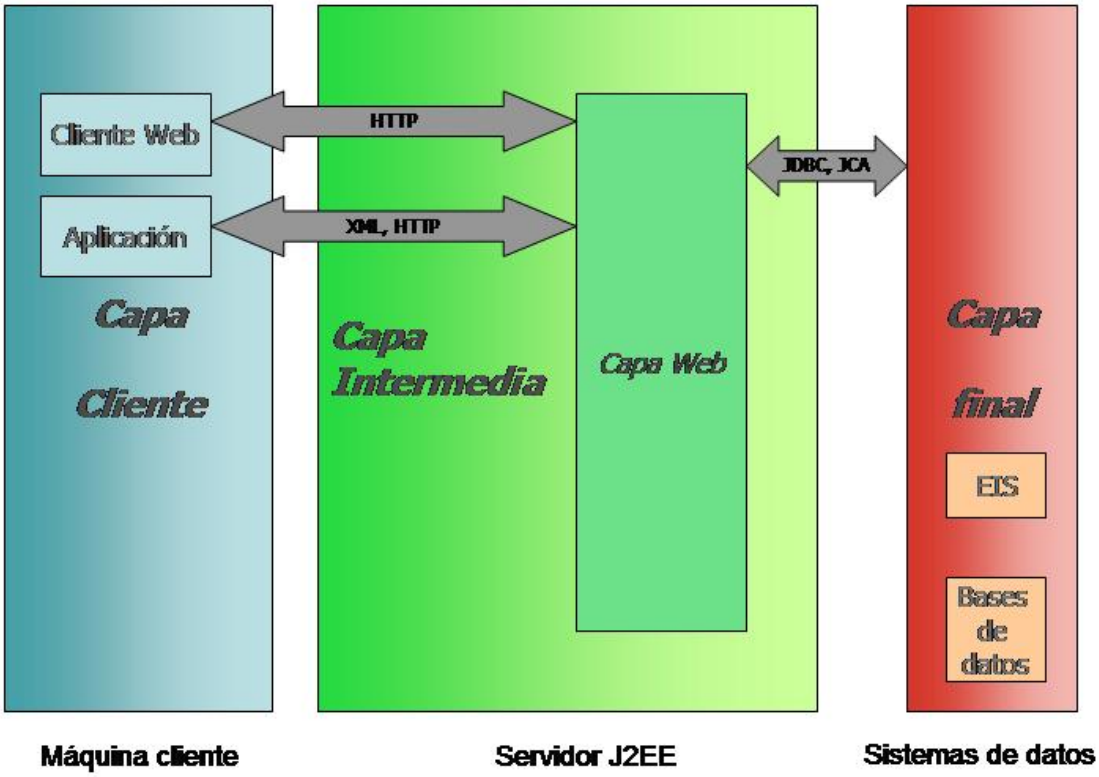
4.1.2 Capa intermedia

Una simple aplicación J2EE puede construirse en la capa cliente, pero normalmente es conveniente implementar la lógica de negocio en la capa intermedia. La capa intermedia se ejecuta en el servidor J2EE.

En una aplicación multicapa completa la capa Web se dedica casi exclusivamente al control de la presentación y control de flujo de la aplicación, mientras que la capa EJB se centra en el uso de los recursos de información.

4.1.2.1 Aplicaciones centradas en Web

Hay un gran número de escenarios en los que el uso de la capa EJB en la aplicación sería considerado un esfuerzo innecesario, excesivo. Este escenario sólo Web es ampliamente utilizado y con frecuencia se plantea como punto inicial de desarrollo de muchas aplicaciones J2EE. La capa Web controla la presentación y la lógica de negocio, y asume que el acceso a datos se hará a través de JDBC o una arquitectura de conector J2EE (JCA).

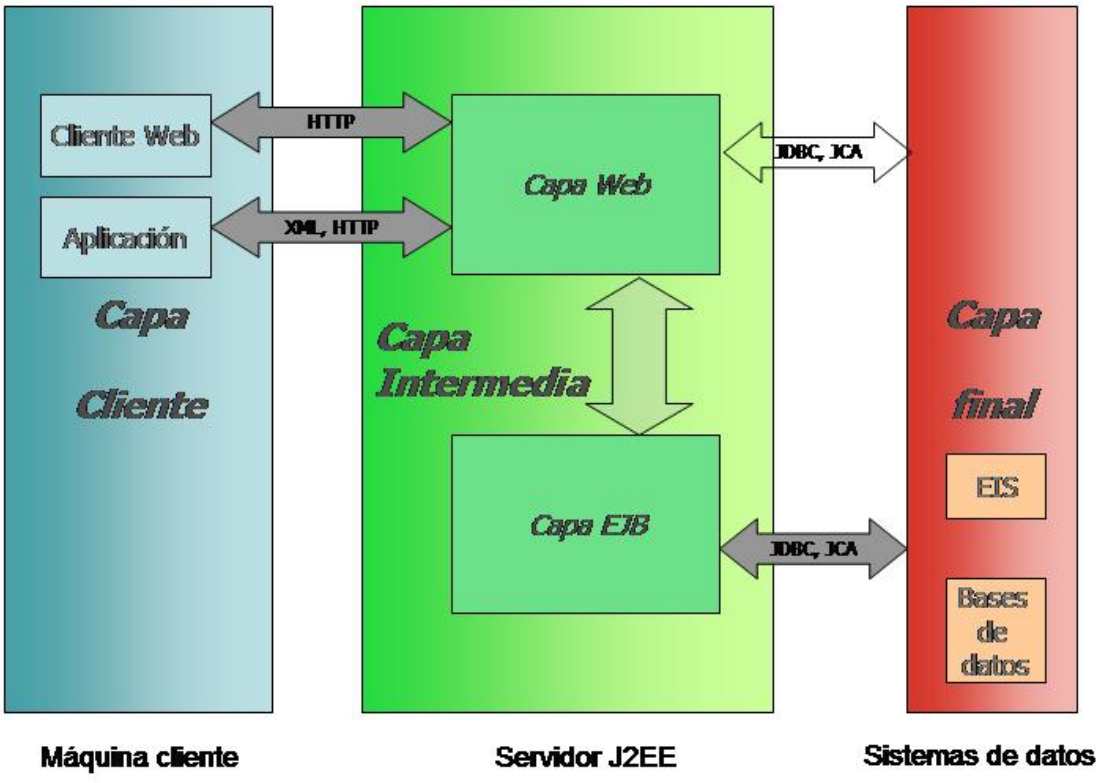


F. 4-5: Arquitectura J2EE. Aplicaciones centradas en Web.

4.1.2.2 Aplicaciones B2B

Esta es la solución natural para el desarrollo y despliegue de soluciones de comercio electrónico, se centra en las interacciones entre la capa Web y la capa EJB.

Aunque la capa Web puede acceder también a la capa final de sistemas de datos, lo normal es que ésta se dedique al control de flujo y presentación de la aplicación, y la capa EJB se centre en implementar la lógica de negocio y el acceso a los recursos de información.



F. 4-6: Arquitectura J2EE. Aplicaciones B2B.

4.1.3 Capa final

La capa final la forman los sistemas de información a los que acceden las aplicaciones. J2EE soporta multitud de fuentes de datos, ya sean bases de datos relacionales o no, ficheros planos, documentos XML o sistemas de información completos como EIS. Una aplicación J2EE podría necesitar acceder al sistema de información de la empresa para la conectividad a la base de datos.

Para las conexiones a bases de datos se suele utilizar la interfaz JDBC, aunque también es posible acceder a ellas a través de la arquitectura de conector J2EE (JCA). La arquitectura de conector J2EE está orientada a la conexión con cualquier fuente de datos, ya sean bases de datos, sistemas de información empresariales (EIS), ficheros planos o cualquier otro sistema informático de almacenamiento de información.

4.2 MODELO BASADO EN COMPONENTES

La flexibilidad de la plataforma se debe en gran medida a este modelo. Este modelo está basado en componentes bien formados que pueden automáticamente aprovecharse de los servicios que provee una plataforma sofisticada.

Un componente es una unidad funcional de software que está embebida en la aplicación con sus ficheros y clases asociadas y se comunica con otros componentes. Los componentes J2EE están escritos en lenguaje de programación Java y son compilados igual que otro programa de este lenguaje. Cuando estos componentes son ensamblados en una aplicación J2EE, se verifican para comprobar si son elementos bien formados y de acuerdo a la especificación. Una vez ensamblados pueden ser desplegados en cualquier servidor J2EE donde serán ejecutados.

Estos componentes al desarrollarse de acuerdo a unas guías estándares, pueden combinarse fácilmente en las aplicaciones, siendo reusables para una productividad máxima.

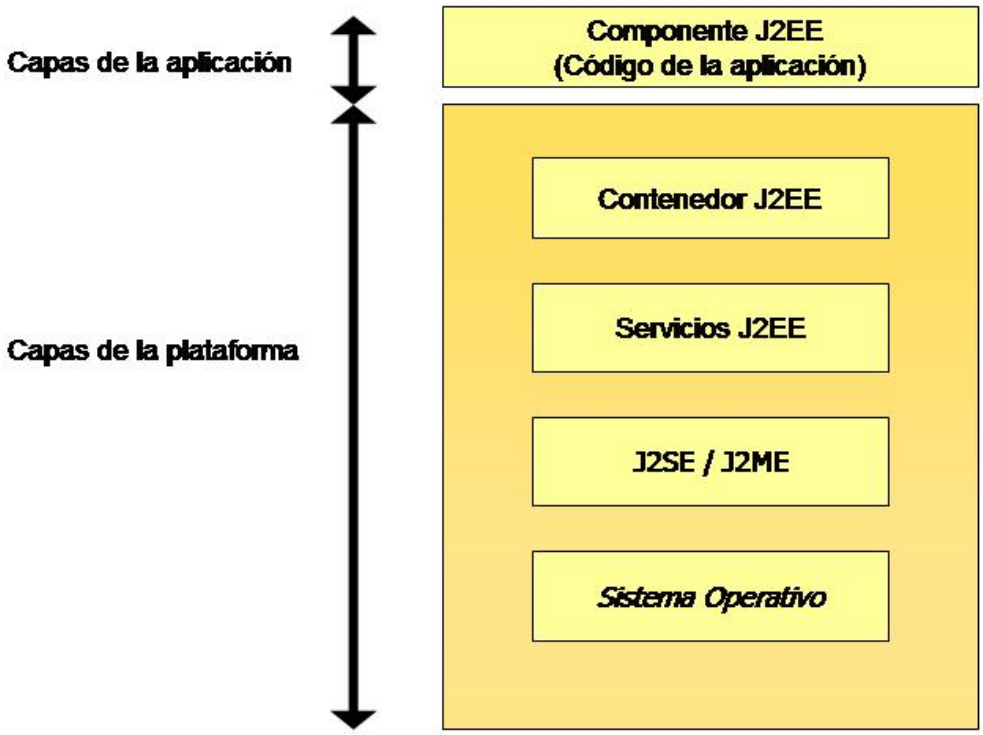
Todos los componentes J2EE dependen del soporte de ejecución del sistema llamado contenedor. En el modelo de desarrollo basado en componentes de la plataforma J2EE, es central la noción de contenedores.

Los contenedores son entornos estandarizados de ejecución que proveen de servicios específicos a los componentes. Los componentes pueden esperar que estos servicios estén disponibles en cualquier plataforma J2EE de cualquier vendedor.

Mientras que la especificación J2EE define los contenedores de los componentes que la plataforma debe soportar, y las relaciones entre contenedores y componentes, no especifica o restringe la configuración de los contenedores. Se define el ciclo de vida de cada componente, el comportamiento que cada componente debe implementar, y los servicios que el contenedor debe proveer a los componentes. De esta manera, todos los tipos de contenedores pueden ejecutarse en una única plataforma o la plataforma puede estar formada por múltiples contenedores en múltiples plataforma.

A continuación, se muestra las distintas capas del modelo de componentes. Cuando de desarrolla el código de la aplicación, en realidad lo que se hace es definir y desarrollar los componentes de la aplicación. Los componentes se ejecutan en los contenedores específicos a cada tipo de componente. Los contenedores proporcionan a los componentes los servicios J2EE y J2SE, y se soportan sobre la plataforma J2SE. En caso de los clientes inalámbricos, en vez de la plataforma J2SE y servicios J2SE, se utilizar la plataforma J2ME. La plataforma J2SE a su vez es soportada por un sistema operativo. Una de las características de la plataforma es la independencia del sistema operativo.

Los contenedores proporcionan servicios tales como control del ciclo de vida de los componentes, seguridad, despliegue y control de hilos. Además, los contenedores proporcionan un mecanismo de selección del comportamiento de la aplicación en tiempo de ensamblaje o despliegue de la aplicación. A través del uso de descriptores de despliegue, con ficheros de configuración en XML, que especifican el comportamiento de los componentes y del contenedor. Entre las características que pueden configurarse en estos descriptores de despliegues se incluyen seguridad, control de transacciones y otras responsabilidades de control. Los componentes pueden ser configurados para el entorno de un contenedor específico cuando se despliegan, en vez de realizar esta configuración en el código.



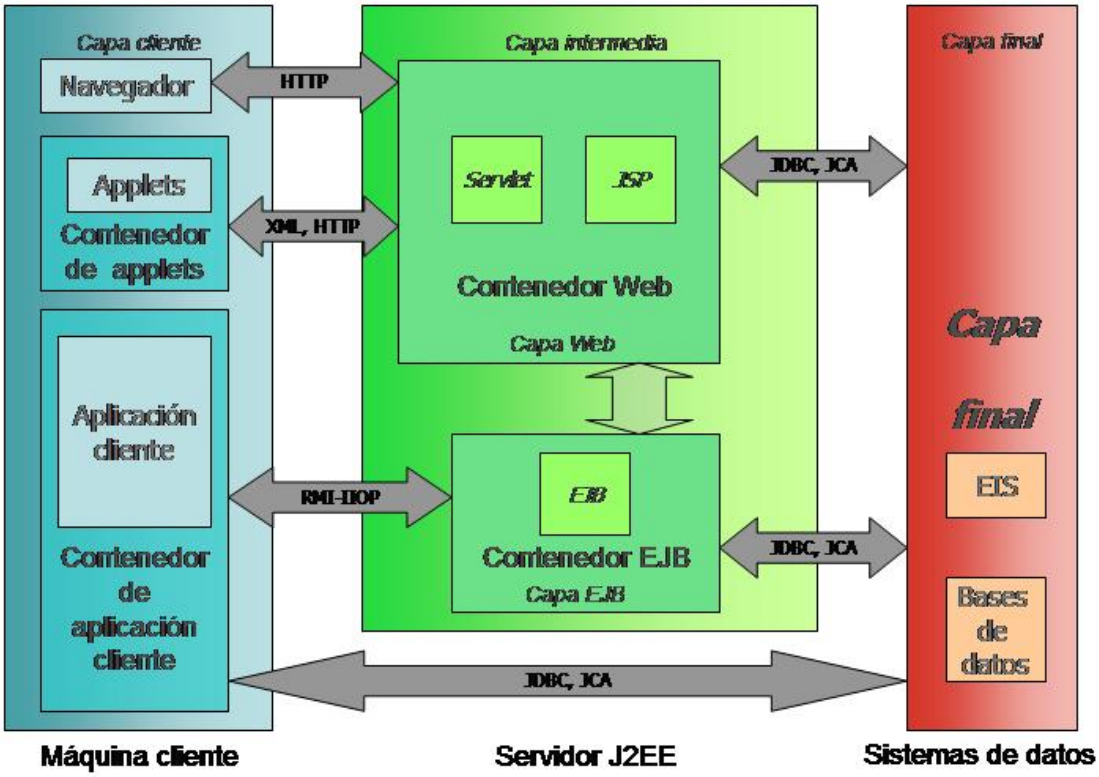
F. 4-7: Modelo de componentes.

La especificación J2EE define los siguientes componentes y contenedores:

a) Aplicaciones clientes y applets en la capa cliente que se ejecutan en un contenedor cliente.

b) Componentes de las tecnologías Java Servlet and JavaServer Pages (JSP) en la capa Web del servidor J2EE que se ejecutan en un contenedor de servlets o de JSP respectivamente, o en contenedor Web.

c) Componentes Enterprise JavaBeans (EJB) en la capa de negocio o de EJB del servidor J2EE que se ejecutan en un contenedor EJB.



F. 4-8: Modelo de componentes en la arquitectura J2EE.

Junto a los componentes definidos en J2EE, es importante resaltar el uso de otro tipo de componentes que forman parte de la plataforma J2SE, los JavaBeans. Estos componentes serán muy útiles en el desarrollo de aplicaciones.

4.2.2 JavaBeans

Las capas clientes e intermedia pueden también incluir componentes basados en la arquitectura de componentes JavaBeans. La función principal de estos componentes en la arquitectura J2EE es el manejo del flujo de datos entre una aplicación cliente, o applet, los componentes que se ejecutan en el servidor y la base de datos.

Los JavaBeans no son considerados componentes J2EE sino componentes de la plataforma J2SE.

Estos componentes son clases Java cuyas propiedades son accesibles con métodos `get` y `set`, que deben cumplir algunas convenciones de nombre y diseño.

4.2.3 Componentes clientes

4.2.3.1 Applets

Para soportar interacciones de usuario complejas en aplicaciones de servidor, puede ser necesario proporcionar funcionalidad directamente en la capa cliente. Esta funcionalidad es típicamente implementada por los componentes JavaBeans que interactúan con el servicio de la capa intermedia a través de los componentes de la capa Web. Los componentes JavaBeans de la capa cliente serán normalmente proporcionados por el servicio como un applet que es descargado automáticamente en el navegador del usuario y se ejecutan en el propio cliente. Los applets basados en el navegador se comunican con HTTP.

4.2.3.2 Aplicación cliente

Los componentes de la capa cliente pueden contener también aplicaciones clientes solitarias escritas en lenguaje de programación Java. Se ejecutan en un contenedor del propio cliente. Este tipo de programas de interfaz de usuario pueden interactuar directamente con la capa de EJB de una plataforma J2EE usando RMI-IIOP.

4.2.3.3 Contenedores de la capa cliente

Los contenedores de applets consisten en un navegador Web y un Plug-in de Java que manejan la ejecución de los applets. Normalmente, no es necesaria la plataforma J2SE para la ejecución de applets en la máquina cliente. El modelo de aplicaciones J2EE proporciona soporte especial para descargar automáticamente e instalar Java Plug-in para eliminar versiones no estandarizadas de la máquina virtual del navegador de usuario.

Para la ejecución de las aplicaciones clientes es necesario un entorno de ejecución para está. Este entorno de ejecución recibe el nombre de contendor de aplicación cliente y cliente es un conjunto de librerías y API que soporta el código cliente. Principalmente proporcionan a las aplicaciones clientes los servicios básicos que se ofrecen a través de la plataforma J2SE como la comunicación RMI-IIOP, aunque también proporciona otros servicios como búsquedas y mensajería asíncrona entre otros.

4.2.4 Componentes de la capa intermedia

En la capa intermedia podemos encontrarnos componentes Web y componentes Enterprise JavaBeans, que se ejecutan respectivamente, en un contenedor Web y un contenedor EJB.

Los componentes Web pueden ser servlets que se hospedan y ejecutan en los contenedores de servlets, y JavaServer Pages (JSP) que se hospedan y ejecutan en los contenedores de JSP. Tanto los servlets como las páginas JSP pueden ejecutarse en los contenedores Web, ya que un contenedor Web es la agrupación de un contenedor de servlets y un contenedor de JavaServer Pages.

Las páginas HTML estáticas, applets y algunas utilidades de servidor pueden estar inmersas en los componentes Web durante el ensamblaje de la aplicación, pero no se consideran componentes Web en la especificación J2EE.

La lógica que resuelve las necesidades de un entorno particular de negocio es manejada normalmente por los enterprise beans (EJB) ejecutados en la capa de negocio.

4.2.4.1 Servlets

Un servlet es un componente Web que extiende la funcionalidad de un servidor Web de un modo portable y eficiente gracias a un contenedor de servlets o contenedor Web. Son clases Java que procesan las peticiones y construyen las respuestas dinámicamente. Aunque pueden utilizarse para crear la vista de la aplicación, su función principal en la plataforma J2EE es la de control de flujo de peticiones.

Se hospedan y ejecutan en un contenedor de servlet o un contenedor Web. Cuando un servlet recibe una petición de un cliente, genera una respuesta, posiblemente invocando lógica de negocio. Entonces envía la respuesta al cliente, normalmente en un documento HTML o XML.

4.2.4.2 JavaServer Pages, JSP

La tecnología JSP proporciona un modo extensible para generar contenido dinámico para el cliente. Una página JSP es un documento textual que describe como procesar una petición para crear una respuesta adecuada. Las páginas JSP son documentos de texto que se ejecutan como los servlets pero permiten una aproximación más natural a la creación de contenido estático.

Una página JSP contiene una plantilla para dar formato al documento y elementos JSP para generar el contenido dinámico. A primera vista, una página JSP presenta una estructura similar a una página HTML ya que ambas utilizan un lenguaje de etiquetas. La mayoría de las páginas JSP utilizan JavaBeans o Enterprise JavaBeans para realizar el procesado más complejo requerido por las aplicaciones. Esta tecnología es extensible a través del desarrollo de etiquetas personalizadas.

Los componentes JSP se hospedan y ejecutan en un contenedor de JSP o un contenedor Web.

4.2.4.3 Enterprise JavaBeans

La arquitectura Enterprise JavaBeans es una tecnología de servidor para el desarrollo y despliegue de componentes que contienen la lógica de negocio de una aplicación empresarial.

Son componentes de la capa intermedia escalables, transaccionales y multi-usuario. Se hospedan en un contenedor EJB y pueden presentar interfaces locales o remotas. El contendor EJB proporciona a estos componentes una gran variedad de servicios de transacciones, persistencia y acceso a otros servicios J2EE y API de comunicación.

Los Enterprise JavaBeans, conocidos como EJB o enterprise beans, permiten a los desarrolladores de componentes o de aplicaciones concentrarse en la lógica de negocio mientras que las complejidades de una entrega de servicios fiables y escalables son controladas por el contenedor EJB.

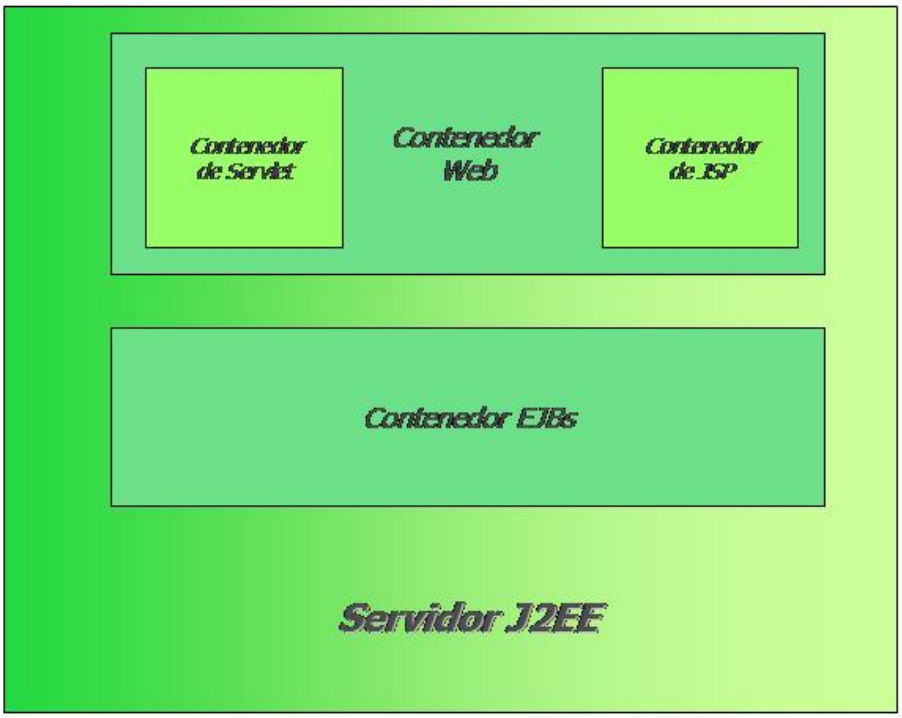
Un enterprise bean además obtiene la información de las fuentes de datos y la procesa si es necesario antes de enviársela al cliente.

En muchos sentidos, la plataforma J2EE y la arquitectura EJB tienen metas complementarias. El modelo de componentes EJB es el núcleo de arquitecturas de aplicaciones industriales en el modelo de programación J2EE. La plataforma complementa la especificación de EJB con una completa especificación de API que un desarrollador de beans empresariales puede usar para la implementación de éstos.

4.2.4.4 Contenedores del servidor J2EE

Puesto que el empaquetamiento y despliegue de las aplicaciones J2EE está estandarizado, una aplicación conforme a J2EE puede servirse, sin necesidad de recompilar el código y sin necesidad de un nuevo empaquetamiento de la aplicación, en cualquier servidor que cumpla con el estándar J2EE. Si la aplicación es puramente Web y no contiene componentes de la capa EJB, la aplicación puede ejecutarse en cualquier contenedor Web, no siendo necesaria una plataforma completa J2EE para su ejecución.

Un servidor J2EE completo está formado por el contendor Web y el contendor de EJB.



F. 4-9: Servidor J2EE.

Una aplicación Web se ejecuta en un contenedor Web. Los contenedores Web proporcionan soporte en tiempo de ejecución para responder a las respuestas del cliente, realizar peticiones en tiempo de procesamiento y retornar resultados al cliente. El contenedor maneja el ciclo de vida de cada componente Web, envía las peticiones de servicio a los componentes de la aplicación y proporciona una interfaz estándar al contexto de datos como un estado de sesión e información sobre la petición en curso.

Los componentes de la capa Web son los servlets y las páginas JSP. Aunque ambos son soportados por los contenedores Web, la arquitectura basada en componentes permiten implementaciones de contenedor de servlets y contenedor de JSP independientes. Un contenedor Web incluye un contenedor de servlet y un contenedor JSP.

Un contenedor de servlets, también llamado motor de servlet, es una extensión de un servidor Web que proporciona funcionalidad a los servlets. Les proporciona los servicios de red por los que las peticiones y respuestas se envían. También decodifican las peticiones y dan formato a las respuestas. Todo contenedor de servlet debe soportar HTTP como protocolo de comunicación, aunque también puede soportar otros protocolos como HTTPS.

Un contenedor JSP proporciona los mismos servicios que un contenedor de servlets a los componentes JSP. A la agrupación de contenedor de servlet y JSP se le da el nombre de contendor Web.

Podemos encontrar contendores Web capaces de trabajar stand-alone (sin necesidad de un servidor Web) e interpretar las páginas HTML y JSP. Para trabajar con otro tipo de páginas como PHP o ASP junto a las JSP, seguramente será necesario utilizar el contenedor Web trabajando conjuntamente con un servidor Web.

Los contenedores EJB proporcionan soporte automatizado para transacciones y control del ciclo de vida de los componentes EJB, así como servicios de entrega fiable y escalable. Proveen a los componentes EJB de un gran rango de servicios de persistencia y acceso a servicios J2EE y a las API de comunicación.

Tanto los contenedores Web como EJB proporcionan acceso estandarizados a los sistemas de información.

Es necesario recalcar que, aunque normalmente los proveedores de plataformas localizan sus contenedores Web y contenedores EJB ejecutándose ambos en la misma máquina virtual, éstos pueden ejecutarse sobre distintas máquinas virtuales y distinto hardware.

4.3 TECNOLOGÍAS DE LA PLATAFORMA J2EE

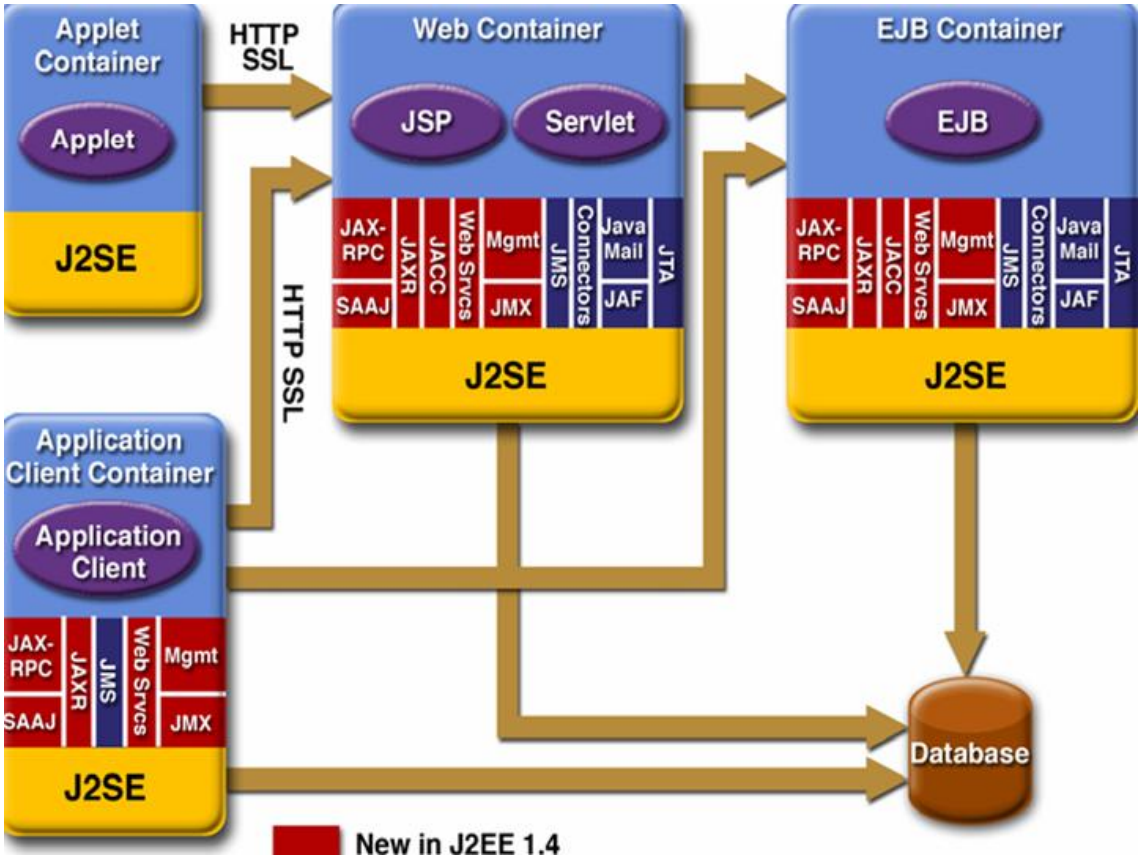
El estándar J2EE es un conjunto de especificaciones de otras tecnologías que integradas, dan lugar a una plataforma potente y flexible.

Las especificaciones definen un conjunto de librerías, para la construcción de aplicaciones empresariales. En su mayoría se corresponden a interfaces de programación (API) Java y clases abstractas. Normalmente, al diseñar aplicaciones empresariales, el desarrollador extenderá e implementará estas clases abstractas, o utilizará implementaciones de otros fabricantes.

Junto a los componentes que define la especificación, J2EE define también los servicios que los contenedores proporcionan a estos componentes. A continuación se muestran algunas de las tecnologías utilizadas en el desarrollo de aplicaciones. Es importante señalar que alguna de estas API no son especificadas por J2EE, sino por el estándar J2SE. Los contenedores proporcionan a los componentes un entorno de ejecución gracias a la plataforma J2SE que soporta estos contenedores. Además los contenedores ofrecen a sus componentes servicios incluidos en la plataforma J2SE.

Para clientes inalámbricos, los servicios y entorno de ejecución se soportan sobre la plataforma J2ME, y no sobre J2SE. La plataforma J2ME ofrece servicios específicos para este tipo de clientes.

El modelo de programación J2EE divide las responsabilidades entre los contenedores y los componentes, lo que implica una división en las responsabilidades los proveedores de componentes de aplicación y los proveedores de productos J2EE. Los proveedores de componentes de aplicación se centran en desarrollar la lógica de negocio, y los proveedores de productos J2EE se centran en proporcionar una infraestructura en la que los componentes de aplicación puedan desplegarse. Esta división conduce a la restricción de funcionalidad que los componentes. Un componente no podrá realizar funciones propias del contenedor que lo soporta. Por ejemplo, si los EJB manejaran hilos, la plataforma J2EE no podría manejar los ciclos de vida de los EJB en el contenedor de EJB, y el contendor no podría manejar apropiadamente las transacciones.



F. 4-10: API en la plataforma J2EE.

(The J2EE™ 1.4 Tutorial For Sun Java System Application Server Platform Edition 8.1 2005Q2 UR2)

En la figura F. 4-10: API en la plataforma J2EE., se muestra la relación lógica entre los elementos de la arquitectura J2EE. Estas relaciones no implican una partición física de los elementos en máquinas, procesos, espacios de direcciones o máquinas virtuales separadas.

En particular, el entorno de ejecución de applets debe ser compatible con 1.4. Sin embargo, ya que no todos los navegadores ofrecen aún tal soporte por defecto, los applets pueden usar el Java Plugin (API incluida en J2SE pero que también se puede utilizar de modo independiente) como entorno de ejecución.

La plataforma J2EE es una solución independiente del sistema operativo y de integración total de sistemas que crea un mercado abierto en el que cada vendedor puede vender a cada cliente. Este mercado anima a los vendedores a competir, no intentando encerrar clientes en sus tecnologías, sino intentando ofrecer cada uno mejores productos y servicios que puedan beneficiar a los clientes, tales como mejores implementaciones, mejores herramientas o mejor soporte al cliente. Las interfaces de J2EE facilitan la integración de aplicaciones y sistemas proporcionando un modelo de aplicación unificado con componentes enterprise beans, un mecanismo simple de petición respuesta con páginas JSP y servlets, un modelo fiable de seguridad con JAAS, integración con intercambios de datos basados en XML con JAXP, SAAJ y JAX-RPC, interoperabilidad simplificada con la arquitectura de conectores J2EE, fácil conectividad a bases de datos con la interfaz JDBC o la arquitectura de conector J2EE, e integración de aplicaciones empresariales con beans de mensajes, JMS, JTA y JNDI.

4.3.1 Tecnologías de componentes

Entre las tecnologías de componentes se encuentran las tecnologías expuestas en apartado 4.2, MODELO BASADO EN COMPONENTES. Los componentes más importantes, y en las que se centran la especificación J2EE son aquellos de la capa intermedia, expuestos en el apartado 4.2.4.

4.3.1.1 Servlets

La tecnología Java Servlet define un tipo de componente Web que se hospeda y ejecuta en un contenedor de servlet o un contenedor Web. Los servlet son clases Java que procesan las peticiones y construyen las respuestas dinámicamente. Un servlet interactúa con los clientes Web utilizando los mecanismos de petición/respuesta implementados por el contenedor que extiende las capacidades de los servidores que hospedan las aplicaciones. Los contenedores de servlets deben soportar el protocolo HTTP como protocolo de peticiones y respuestas y, aunque los servlets pueden responder a cualquier tipo de petición, se utilizan comúnmente para extender las aplicaciones hospedadas en los servidores Web. Se definen servlets específicos para HTTP.

El contendor controla el ciclo de vida de los servlets. Una vez instanciada la clase servlet es inicializada, maneja las peticiones de los clientes y se destruye. Este ciclo de vida se expresa en la interfaz `javax.servlet.Servlet` con los métodos `init`, `service`, y `destroy`. Todos los servlets deben implementar `javax.servlet.Servlet`, implementando sus métodos. Sin embargo, normalmente, los desarrolladores no implementan directamente esta interfaz. Utilizan la interfaz abstracta `javax.servlet.http.HttpServlet`. Esta otra interfaz ya implementa la interfaz anterior (implementan los métodos `init`, `service` y `destroy`) para servlets en un entorno de peticiones HTTP, aunque los desarrolladores tendrán que extender esta interfaz sobre escribiendo principalmente los métodos `doGet` o `doPost`. Para otro tipo de peticiones también puede utilizarse la interfaz abstracta `javax.servlet.GenericServlet`.

El contendor Web debe soportar descriptores de despliegue. Estos descriptores de despliegue son ficheros en los que se pueden configurar algunos elementos de la aplicación Web en formato XML. Entre las características que pueden configurarse se encuentran los parámetros iniciales del contexto de servlets, configuración de sesión, declaración de los servlets de la aplicación y mapeo a peticiones recibidas, y filtros.

Los paquetes que componen la API de Java Servlets son `javax.servlet` y `javax.servlet.http`. El paquete `javax.servlet` contiene las clases e interfaces que describen y definen las relaciones entre un servlet y el entorno de ejecución que le proporciona el contendor. El paquete `javax.servlet.http` contiene las clases e interfaces que describen y definen las relaciones entre un servlet y el entorno de ejecución que le proporciona el contenedor pero en un entorno específico HTTP.

Los servlets pueden generar la vista de una aplicación que incluya contenido dinámico. Sin embargo, en un entorno J2EE suelen utilizarse conjuntamente con las páginas JSP. En este entorno, las páginas JSP suelen centrarse en las funciones de presentación de contenido, y los servlets se centran principalmente en el control de flujo de la aplicación.

4.3.1.2 JSP

La tecnología JavaServer Pages (JSP) proporciona un modo extensible para generar contenido dinámico para el cliente. Como permite insertar partes del código, se puede insertar código utilizado en servlets directamente en un documento de texto. Una página JSP es un documento textual que contiene dos tipos de texto: dato estático que puede ser expresado en cualquier formato de cómo HTML, XML, WML, y elementos JSP que determina como construir el contenido dinámico. Los componentes JSP se hospedan y ejecutan en un contenedor de JSP o un contenedor Web.

Una página JSP tiempo de ejecución, aunque depende de la implementación concreta del contenedor, es normalmente representada por el contenedor por un servlet. En estos casos, las páginas JSP son traducidas por el contenedor por un tipo especial de servlets. El contenedor entrega las peticiones de los clientes a este servlet. La página JSP describe cómo crear la respuesta.

Algunas de las propiedades de las páginas JSP también pueden configurarse en el descriptor de despliegue de la aplicación.

Los paquetes que componen la API JavaServer Pages son `javax.servlet.jsp` y `javax.servlet.jsp.tagext`. El paquete `javax.servlet.jsp` contiene las clases e interfaces que describen y definen las relaciones entre las clases que implementan una página JSP, y el entorno de ejecución que proporciona el contendor JSP. Normalmente, el uso de estas clases se hace de un modo totalmente transparente al desarrollador. El desarrollador, normalmente no utiliza directamente esta interfaz ya que utiliza el lenguaje de etiquetas de las páginas JSP. El paquete `javax.servlet.jsp.tagext` describe el uso de etiquetas y otras clases de extensión de etiquetas. Los autores de páginas JSP pueden usar acciones personalizadas utilizando las clases contenidas en este paquete.

Las últimas versiones de JSP incluyen el paquete `javax.servlet.jsp.el` que contiene las clases e interfaces relacionadas con el lenguaje de expresión EL (Expresión Languaje). Este lenguaje fue inicialmente definido por la especificación JSTL (JSP StandardTag Library) 1.0, pero se ha incorporado a la especificación JSP desde la versión 2.0 y se ha extendido con nuevas características. Puede utilizarse en valores de atributos, y acciones JSP estándares o

personalizadas. La incorporación de este lenguaje a la tecnología JSP facilita el desarrollo de páginas sin scriptlets, ya que se permite el uso del lenguaje EL, pero no el uso de sriptlets, expresiones o declaraciones Java. La utilización de este lenguaje es opcional.

4.3.1.3 EJB

La tecnología Enterprise JavaBeans es una arquitectura para la programación basada en componentes para el desarrollo y despliegue de aplicaciones empresariales transaccionales.

Un componente Enterprise JavaBeans (EJB), o enterprise bean, es un cuerpo de código con propiedades y métodos que implementan, normalmente, la lógica de negocio que opera sobre los datos empresariales. El contendor de la capa de negocio, contenedor EJB, crea y controla las instancias de los EJB en tiempo de ejecución. Como el resto de componentes, los EJB pueden ser configurados y personalizados en el momento de despliegue gracias a un descriptor de despliegue.

Los EJB pueden presentar interfaces locales o remotas. El contendor EJB proporciona a estos componentes una gran variedad de servicios de transacciones, persistencia y acceso a otros servicios J2EE y API de comunicación.

Si un EJB utiliza sólo los servicios estándares que ofrece un contenedor EJB, puede ser desplegado en cualquier contenedor EBJ estándar y puede incluirse en la aplicación sin requerir cambios en el código o recompilación del componente.

Existen tres tipos de EJB: de sesión (session beans) que gestionan el flujo de la información en el servidor, de entidad (entity beans) que encapsulan los objetos de lado de servidor que almacenan los datos, y dirigidos por mensajes (message-driven beans) con funcionamiento asíncrono. Los EJB a menudo interactúan con las bases de datos. Uno de los beneficios de los EJB de entidad, es que no es necesario escribir código SQL, ni utilizar la interfaz JDBC directamente para acceder a la base de datos y realizar operaciones sobre ella; el contenedor EJB es el que maneja este tipo de operaciones. Sin embargo, si se sobrescribe, por alguna razón, la persistencia por defecto manejada por el contenedor, se podrá utilizar JDBC. Los beans de sesión utilizan JDBC para el acceso a datos.

Los Enterprise JavaBeans, permiten a los desarrolladores de componentes o de aplicaciones concentrarse en la lógica de negocio mientras que las complejidades de una entrega de servicios fiables y escalables son controladas por el contenedor EJB.
La especificación de Enterprise JavaBeans requiere el soporte de protocolos basados en CORBA/IIOP para invocaciones remotas desde los clientes J2EE de beans de sesión o de entidad. Además pueden soportar otros protocolos de invocación remota.

Los paquetes incluidos en la API EJB son `javax.ejb`, `javax.ejb.spi` y, en versiones antiguas de EJB, `javax.ejb.deployment`. El paquete `javax.ejb` contiene las clases e interfaces que definen las relaciones entre los enterprise beans y sus clientes, y entre los clientes y el contenedor. El paquete `javax.ejb.spi` define las interfaces que implementa el contenedor de EJB. Por último, en el paquete `javax.ejb.deployment` se definían las clases usadas por el contenedor para encapsular la información de los objetos EJB.

4.3.2 JSTL, biblioteca de etiquetas para JSP

JSTL (Java Standard Tag Library) especifica un conjunto de librerías de etiquetas basadas en la API JSP, un estándar que incluye numerosas librerías de etiquetas JSP. Esta API proporciona cuatro librerías de etiquetas independientes, cada una contiene acciones dirigidas a un área funcional específica.

Cada librería utiliza un conjunto de etiquetas que facilitan la tarea del desarrollador evitando la implementación directa en código java para operaciones comunes en las mayorías de las aplicaciones. Las librerías que JSTL ofrece son los siguientes:

a) La librería Core (c) contiene acciones de propósito general, acciones condicionales, iteración de objetos y operaciones relacionadas con URL. Incluye tareas rutinarias como manejo de variables, incluir o excluir una parte de una página dependiendo de una condición en tiempo de ejecución, hacer un bucle sobre una colección de ítems, manipular URL para

seguimiento de sesión, importar contenido de otros recursos y redireccionar la respuesta a una URL diferente.

b) La librería I18N&Formatting (`fmt`) soporta las acciones de Internacionalización y el formateo general. Preparar una aplicación para varios idiomas se llama internacionalización (comúnmente abreviado i18n), y hacer que el contenido esté disponible para un idioma específico se llama localización (o i10n). Para establecer las características cada localización puede ser necesario considerar, además del idioma, otras características como el formateo de datos, especialmente fechas y números. También podríamos necesitar adaptar los colores, las imágenes y otro contenido no textual.

c) La librería Database Access (`sql`) se puede utilizar para leer y modificar la información almacenada en una base de datos con las acciones proporcionadas.

d) La librería XML (`x`) contiene acciones para procesamiento XML, incluidas validaciones de documentos XML, y transformaciones usando XSLT. También proporciona acciones para extraer parte de un documento XML validado, hacer bucles sobre un conjunto de nodos, y procesamiento condicional basado en valores de nodos.

De este modo, JSTL responde a la demanda de los desarrolladores de un conjunto de etiquetas JSP personalizadas para manejar las tareas que necesitan casi todas las páginas JSP, incluyendo procesamiento condicional, internacionalización, acceso a bases de datos y procesamiento XML. Además de las librerías de etiquetas JSTL soportan el lenguaje EL para referenciar objetos y sus propiedades sin necesidad de código Java, y validadores de bibliotecas de etiquetas (Tag Library Validators, TLVs).

No siempre es recomendable el uso de estas librerías de etiquetas, al menos un uso abusivo. Normalmente representan una buena alternativa al uso de código Java inmerso las páginas JSP, ya que evitan el uso de scriptlets. Sin embargo, cuando la utilización de estas etiquetas no

simplifica la estructura de la página JSP, es conveniente plantearse el uso de etiquetas personalizadas. Estas etiquetas personalizadas serán diseñadas por los desarrollares y realizarán las operaciones mostrando los resultados en la página JSP. Por ejemplo, en general no se recomienda el uso de la librería Database Access, se considera como una mala práctica. En principio, todos los accesos a bases de datos se deberían realizar desde componentes Java puros en una aplicación basada en MVC en lugar de acceder directamente desde páginas JSP.

Hay aplicaciones que se cualifican como muy simples y donde el tiempo de desarrollo o las habilidades necesarias hacen impracticable la arquitectura MVC. Esta solución de librerías JSTL presenta una alternativa mucho más elegante que el uso de scriptlets ya que es una solución que proporciona mejor mantenimiento y desarrollo. Aún así, insistimos, en que si el equipo de desarrollo incluye programadores Java, debería considerarse seriamente encapsular el código en clases Java y utilizar JSP sólo para mostrar los resultados.

4.3.3 JSF, framework estándar basado en MVC

A menudo, para el desarrollo de aplicaciones según el patrón MVC, es recomendable utilizar un framework que implemente, principalmente, la capa de controlador y su interacción con el modelo y la presentación.

Los primeros frameworks que surgieron no están estandarizados. Ante la evidente necesidad de utilizar un framework que facilite la tarea de programación, se creó la especificación JavaServer Faces.

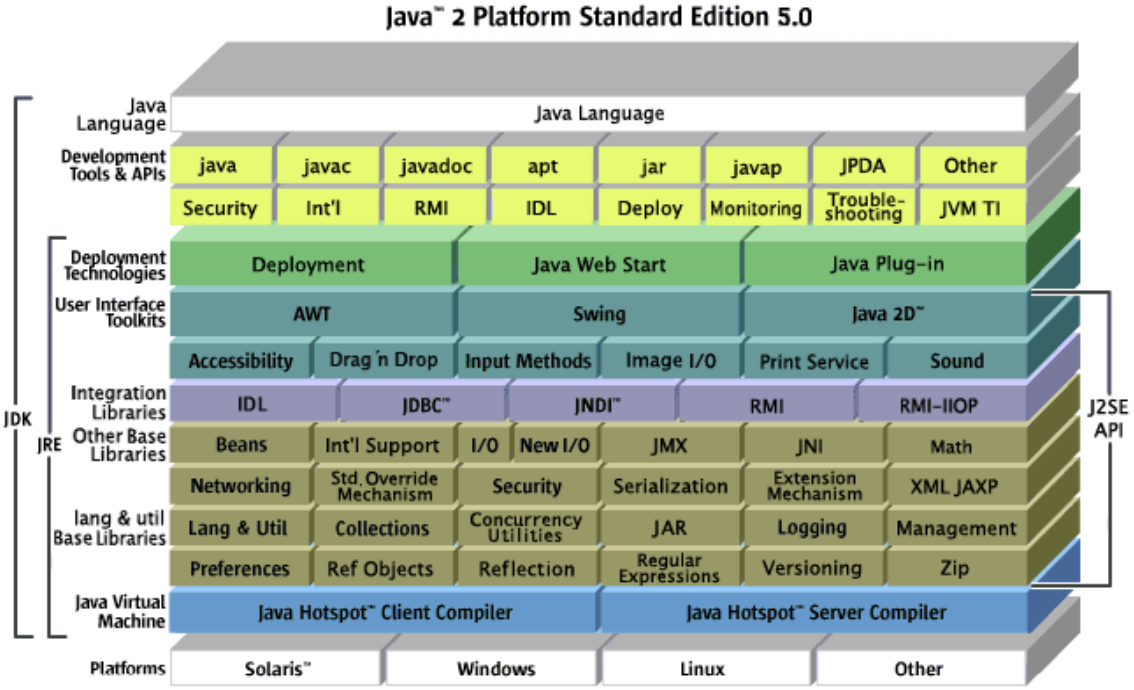
JavaServer Faces es por tanto un framework de desarrollo estandarizado basado en el patrón MVC cuya primera versión final surgió en 2004. La especificación inicial de JavaServer Faces es la JSR-127, aunque la versión actual es la JavaServer Faces 1.2, JSR-252.

4.3.4 Tecnologías de la plataforma J2SE

Los contenedores proporcionan a los componentes de la aplicación las interfaces de programación de J2SE, Java 2 Platform, Standard Edition, que incluyen las siguientes interfaces de programación de aplicaciones empresariales: Java IDL API, JDBC API, RMI-IIOP API, JNDI API, JAXP API, JAAS API.

J2SE nos proporciona un entorno completo para el desarrollo de aplicaciones en Java. Aporta operaciones las bases de seguridad, conectividad a bases de datos y otras muchas características necesarias para nuestras aplicaciones.

J2SE Run Environment (JRE) proporciona librerías, máquina virtual java y otras herramientas necesarias para ejecutar aplicaciones desarrolladas en Java. J2SE Development Kit (JDK) incluye el JRE además de herramientas de desarrollo como compilador.



F. 4-11: Plataforma J2SE.

(JDK™ 5.0 Documentation)

En la figura F. 4-11: Plataforma J2SE., se muestran los servicios que la plataforma J2SE proporciona. No todos los servicios que proporciona J2SE se utilizan en todos los componentes J2EE. Por ejemplo, como se comentó anteriormente, para la ejecución de applets en el cliente, es suficiente con la tecnología de desarrollo Java Plug-in sin necesidad de la plataforma J2SE completa.

4.3.4.1 JDBC, servicio de conexión a bases de datos

Esta tecnología proporciona acceso estandarizado a los sistemas de información. La tecnología JDBC es una interfaz que proporciona la conectividad a un amplio rango de bases de datos SQL.

La API JDBC proporciona acceso a datos relacionales utilizando el lenguaje de programación Java. Con JDBC, las aplicaciones escritas en Java pueden ejecutar sentencias SQL, obtener resultados, y modificar datos en la fuente de datos.

La interfaz JDBC permite invocar comandos SQL desde un programa Java. Se utiliza en los enterprise beans cuando se sobrescribe la persistencia por defecto controlada por el contenedor, o cuando se utilizan beans de sesión para acceder a las bases de datos. Con la persistencia que proporciona el contenedor EJB, las operaciones de acceso a las bases de datos son manejadas las implementaciones de los enterprise beans no contienen código JDBC o comandos SQL. También se puede utilizar la API JDBC desde un servlet o página JSP para acceder a las bases de datos directamente sin utilizar enterprise beans.

Esta interfaz permite conectar una aplicación con todos los datos corporativos incluso en ambientes distribuidos y heterogéneos, incluso con distintas fuentes de datos, con la utilización de los drivers adecuado.

Los fabricantes de bases de datos suelen desarrollar un driver JDBC para sus bases de datos. Este driver normalmente es una librería java que contiene una implementación de todas las interfaces del API JDBC. De esta forma el desarrollador de una aplicación podrá utilizar, para sus conexiones a las bases de datos, las interfaces que proporciona JDBC y configurar adecuadamente la utilización del driver. Idealmente, si necesitáramos cambiar la base de datos de nuestra aplicación, sería suficiente cambiar el driver por uno adecuado a la nueva fuente de

datos. Debido a que algunas bases de datos no utilizan un lenguaje SQL estándar, puede que esta solución no sea suficiente ante un cambio de fuente de datos, para evitar estos problemas se recomienda además, la utilización de ciertos patrones de diseño que se verán más adelante.

La interfaz JDBC tiene dos partes: una interfaz a nivel de aplicación utilizada por los componentes de la aplicación para acceder a las bases de datos, y una interfaz de proveedor de servicio para agregar un driver JDBC a la plataforma J2EE.

La API JDBC es parte de la plataforma Java, que incluye la plataforma J2SE y J2EE. La versión más actual de la especificación es JDBC 3.0 que incluye los paquetes `java.sql` y `javax.sql`. Ambos paquetes están incluidos tanto en la plataforma J2SE como en la J2EE. El paquete `java.sql` es el núcleo de la interfaz, y en versiones antiguas el paquete `javax.sql` se presentaba como opcional, aunque actualmente ambas conforman la API completa JDBC. Ya se está preparando la versión de JDBC 4.0.

4.3.4.2 JNDI, servicio de nombres y directorios

En la plataforma J2EE se pretende que la configuración de las aplicaciones esté disponible en servidores de información accesible vía JNDI, y no en ficheros planos.

La interfaz JNDI (Java Naming and Directory Interface) ofrece un servicio de nombres para que los componentes de la aplicación puedan localizar otros componentes o recursos. El servicio de nombres JNDI proporciona un contexto de nombres que es un conjunto de relaciones nombre-objeto. Todas las operaciones de nombres son relativas a este contexto. El contexto proporciona los métodos para relacionar nombres y objetos, romper estas relaciones, renombrar objetos, y listar relaciones. Además JNDI también proporciona funcionalidades de subcontexto, parecidas a las de un directorio en un sistema de archivos así como los métodos necesarios para la creación y destrucción de estos contextos. Esta estructura jerárquica permite una mejor organización de la información.

Todos los clientes de aplicaciones J2EE utilizan la facilidad JNDI para buscar y crear componentes. JNDI proporciona los métodos para acceder a los servicio de nombres y directorios que permite a los clientes obtener el contexto que contiene los nombres de los

componentes y objetos. Este contexto permanece válido mientras sea válida la sesión del cliente. El cliente proporciona el nombre registrado en JNDI del objeto requerido para obtener una referencia a un objeto administrado.

La arquitectura JNDI consiste en una interfaz de programación (Application Programming Interface, API) y una interfaz de proveedor de servicio (Service Provider Interface, SPI). La SPI permite conectar de forma transparente una gran variedad de servicios de nombres y directorios, por lo tanto permite a las aplicaciones Java usar el API JNDI para acceder a sus servicios. J2SE incluye tres proveedores de servicios: Lightweight Directory Access Protocol (LDAP), servicio de nombres de CORBA Common Object Services (COS) y Java Remote Method Invocation (RMI) Registry. Sin embargo, puede utilizarse otro proveedor de servicio si se desea.

Con JNDI, una aplicación J2EE puede guardar y recuperar cualquier tipo de objeto Java nombrado. Los servicios de nombre J2EE proporcionan los clientes de la aplicación, a los enterprise beans y a los componentes Web acceso a un entorno de nombres JNDI. Este entorno permite que los componentes sean configurados sin necesidad de acceder o cambiar el código fuente del componente.

Un contenedor implementa el entorno del componente y le proporciona un contexto de nombres JNDI. Un componente J2EE crea un contexto inicial y busca el contexto de nombre con `java:comp/env`. El entorno de nombre de un componente es guardado directamente en el contexto inicial o en algunos de sus subcontextos. Un componente J2EE puede acceder a objetos proporcionados por el sistema definidos por el usuario si están nombrados. Los objetos de nombres proporcionados por el sistema son guardados en el contexto de nombre.

La plataforma J2EE permite que un componente utilice objetos de nombre definidos por el usuario, tales como enterprise beans, entradas de entorno, objetos JDBC, y conexiones de mensaje. Un objeto deberá ser nombrado con un subcontexto de acuerdo al tipo de objeto. Por ejemplo, los enterprise beans son nombrados con el subcontexto `java:comp/env/ejb`, y un objeto JDBC DataSource se referencia en un contexto como `java:comp/env/jdbc`. Puesto que JNDI es independiente de cualquier implementación específica, las aplicaciones pueden usar JNDI para acceder a múltiples servicios de nombre y directorio, incluidos LDAP, NDS, DNS y NIS. Esto permite a las aplicaciones J2EE coexistir con otras aplicaciones y sistemas.

La versión más actual es JNDI 1.2, incluida en la plataforma J2SE 1.4. La interfaz JNDI está formada por los paquetes `javax.naming` contiene las clases e interfaces para acceder a los servicios de nombres, `javax.naming.directory` extiende el paquete anterior para proporcionar acceso a directorios, `javax.naming.event` contiene clases e interfaces para soportar la notificación de eventos en servicios de nombres y directorios, `javax.naming.ldap` contiene clases e interfaces para soportar LDAP v3 y `javax.naming.spi` que contiene las clases e interfaces para permitir que varios proveedores de servicios de nombres y directorios se conecten dinámicamente a la tecnología JNDI.

Para evitar colisiones con nombres proporcionados por otros recursos y problemas de portabilidad, todos los nombres en una aplicación J2EE, accesibles vía JNDI deben comenzar con la cadena `java:comp/env`.

4.3.4.3 JAXP, servicios XML

Java API for XML Processing (JAXP) soporta el procesado de documentos XML utilizando DOM, SAX y XSLT. JAXP permite a las aplicaciones parsear y transformar documentos XML independientemente de la implementación de procesado particular de los documentos XML. Además proporciona soporte de espacio de nombres que permite trabajar con esquemas que de otro modo causarían conflictos de nombre.

JAXP está diseñada para ser flexible y permite utilizar cualquier parser XML o procesador XML desde cualquier aplicación que soporte los esquemas W3C.

4.3.4.4 JAAS, servicio de autenticación y autorización

El servicio de autentificación y autorización de Java, Java Authentication and Authorization Service (JAAS), proporciona un modo autentificación y autorización de un grupo de usuarios a las aplicaciones J2EE.

Este framework, JAAS, es una versión en Java del estándar PAM, Pluggable Authentication Module, que extiende la arquitectura de seguridad de J2EE para soportar la autorización basada en usuario.

4.3.4.5 RMI, Java IDL y RMI-IIOP, tecnologías de comunicación

La especificación J2EE exige el soporte de protocolos de invocación de método remoto. RMI es un conjunto de API que permiten a los desarrolladores construir aplicaciones distribuidas en Java.

RMI forma parte de la plataforma J2SE. Utiliza interfaces de Java para definir objetos remotos y una combinación de la tecnología de serialización de Java y del protocolo JRMP. El protocolo de método remoto de Java, JRMP, es el mecanismo de transporte para la comunicación entre objetos de lenguaje Java en distintos espacios de direcciones. RMI convierte las invocaciones de métodos locales en invocaciones de métodos remotos.

Junto a los protocolos de invocación de método remoto, la plataforma J2EE soporta protocolos de grupo de manejo de objetos (OMG, Object Management Group). Estos protocolos permiten a los objetos soportados en la plataforma, el acceso a objetos remotos desarrollados en CORBA (Common Object Request Broker Arquitectura).

La interfaz de un componente de aplicación desarrollado en CORBA, se define usando el lenguaje de definición de interfaz IDL (Interface Definición Language). Un compilador IDL genera los stubs de cliente y servidor que conectan la implementación de objetos al ORB. El ORB (Object Request Broker) es una librería que permite a los objetos de CORBA localizarse y comunicarse unos con otros utilizando el protocolo IIOP (Internet Inter-ORB Protocol).

Las tecnologías OMG que la plataforma J2EE requiere son Java IDL y RMI-IIOP, ambas proporcionadas por la plataforma J2SE. Java IDL permite a los clientes Java invocar operaciones sobre objetos CORBA definidos con IDL e implementados en cualquier lenguaje con un mapeo de CORBA. RMI-IIOP es una implementación del protocolo RMI sobre IIOP que permite escribir interfaces remotas en lenguaje en Java. La interfaz remota puede ser convertida a IDL e implementada en cualquier otro lenguaje que sea soportado, por lo que clientes y servidores pueden ser escritos en cualquier lenguaje. Cuando una interfaz remota es definida como una

interfaz RMI Java, RMI-IIOP proporciona la interoperabilidad con los objetos CORBA implementados en cualquier otro lenguaje.

4.3.5 Tecnologías J2EE para Servicios Web

Las tecnologías SOAP, XML y HTTP son primordiales en la arquitectura de Servicios Web para integración de sistemas en Internet. Básicamente, SOAP es un mecanismo de encapsular los mensajes XML que se transmiten a través de HTTP. J2EE incorpora las API JAX-RPC, JAAS y JAXR que permiten la utilización de estas tecnologías para la creación de Servicios Web.

4.3.5.1 JAX-RPC, para el soporte de SOAP sobre HTTP.

Java API for XML-based RPC (JAX-RPC) utiliza los estándares SOAP y HTTP, por lo que los programas clientes pueden hacer llamadas remotas (RPC) basadas en XML en Internet. JAX-RPC también soporta WSDL, por lo que se puede importar y exportar documentos WSDL. Con JAX-RPC y WSDL, se puede fácilmente interoperar con clientes y servicios ejecutándose tanto en plataformas Java como no-Java, por ejemplo .NET. Por ejemplo, basándonos en un documento WSDL, un cliente Visual Basic .NET puede configurarse para usar un servicio Web implementado en Java, o un servicio Web puede configurarse para reconocer un cliente Visual Basic .NET.

JAX-RPC depende del protocolo de transporte HTTP y permite crear aplicaciones de servicio que combinen HTTP con versiones Java de los protocolos SSL y TLS para autentificación. SSL y TLS aseguran una integridad del mensaje proporcionando encriptación de dato con las capacidades de autentificación de cliente y servidor.

La autenticación es un modo controlado de verificar si una parte tiene acceso a cierta información para proteger los datos contra un uso fraudulento del sistema o una transmisión fraudulenta de transmisión de información. La información transportada por Internet es especialmente vulnerable a ser interceptada y mal usada, por lo que es muy importante configurar un servicio Web JAX-RPC para proteger los datos en transmisión.

4.3.5.2 SAAJ, soporte de JAX-RPC

SOAP with Attachments API for Java (SAAJ) es una interfaz de bajo nivel de la que depende JAX-RPC. SAAJ permite la producción y consumición de mensajes que conforman la especificación de SOAP 1.1 y SOAP con notas adjuntas. La mayoría de los desarrolladores no usan esta interfaz, ya que utilizan la interfaz de alto nivel JAX-RPC.

4.3.5.3 JAXR, tecnologías de registros para servicios Web.

Java API for XML Registries (JAXR) permite acceso a registros de negocio y de propósito general a través de la Web. Soporta el estándar de registro y repositorio ebXML Registry and Repository y las especificaciones de UDDI. Usando JAXR, los desarrolladores pueden aprender una interfaz simple y obtener acceso estas importantes tecnologías de registros. Adicionalmente, las empresas pueden suscribir material para compartir y buscar material que otros hayan registrado. Grupos estándares han desarrollado esquemas para algunos tipos en particular de documentos XML; dos empresas podrían, por ejemplo, ponerse de acuerdo para utilizar un esquema estándar para los formularios de pedidos. Debido a que este esquema estaría guardado en un registro estándar, ambas partes podrían utilizar JAXR para acceder a ellos.

4.3.6 Tecnologías J2EE de mensajería

Las tecnologías de mensajería proporcionan una interfaz asíncrona de envío y recepción de mensajes.

4.3.6.1 JMS

La interfaz Java Message Service (JMS API) es un estándar de mensajes que permite a los componentes de una aplicación J2EE crear, enviar, recibir y leer mensajes. Proporciona una interfaz para el manejo de peticiones, informes o eventos asíncronos que son consumidos por las aplicaciones empresariales. Los mensajes JMS son usados para coordinar estos servicios y la API facilita la comunicación distribuida fiable asíncrona.

Aunque pueden utilizarse otras interfaces para la notificación asíncrona, JMS suele utilizarse cuando se requiere primordialmente fiabilidad y velocidad.

4.3.6.2 JavaMail API

Las aplicaciones J2EE usan JavaMail API para enviar notificaciones de e-mails. La interfaz JavaMail proporciona la interfaz para el envío y recepción de mensajes entre usuarios. Esta interfaz tiene dos partes: la interfaz del nivel de aplicación que es usada por los componentes de aplicación para enviar correos, y la interfaz de proveedor de servicio. La plataforma J2EE incluye JavaMail con un proveedor de servicio que permite a los componentes de aplicación enviar correo por Internet.

JavaMail soporta formato de datos MIME gracias a la integración del framework JAF.

4.3.6.3 JAF, framework de activación de JavaBeans

JavaBeans Activation Framework (JAF) se incluye porque JavaMail lo utiliza. No es una interfaz, es ya una implementación. JAF proporciona servicios estándar para determinar el tipo de cualquier pieza de dato, encapsular el acceso a él, descubrir las operaciones que se pueden realizar sobre él y crear un componente JavaBeans apropiado para realizar estas operaciones.

4.3.7 JTA, Servicio de transacciones

La interfaz Java Transaction (JTA API) proporciona una interfaz estándar para el manejo de transacciones. La arquitectura J2EE proporciona por defecto el modo AUTO COMMIT al manejar transacciones, este modo implica que otras aplicaciones que están visualizando datos podrán ver las actualizaciones de los datos después de la operación de escritura en cada base de datos. Sin embargo, si la aplicación realiza operaciones en dos bases de datos separadas en las que dependen unas de otras, se utilizará la interfaz JTA para realizar la transacción completa.

4.3.8 Arquitectura de conector J2EE

La arquitectura de conector de J2EE (J2EE Connector Arquitecture) es usada por vendedores de herramientas e integradores de sistemas para crear adaptadores que soporten el acceso a sistemas de información que pueden ser integrarse en cualquier producto J2EE. Los adaptadores de recursos son componentes software que permiten a los componentes de una aplicación J2EE acceder e interactuar con el controlador de recursos subyacentes de EIS. Los adaptadores de recursos son específicos de cada recurso, normalmente hay diferentes adaptadores por casa tipo de base de dato o sistema de información EIS. La arquitectura de conector proporciona una integración segura, orientada a interpretación, escalable y transaccional de Servicios Web basados en J2EE con los sistemas EIS existentes que pueden ser asíncronos o síncronos. Las aplicaciones existentes y los EIS integrados a través de la arquitectura de conector en una plataforma J2EE pueden ser expuestos como servicios Web basados en XML usando la interfaz JAX-RPC y los modelos de componentes J2EE. De este modo JAX-RPC y la arquitectura de conector J2EE son tecnologías complementarias para la integración de aplicaciones empresariales (Enterprise Application Integration, EAI) e integraciones B2B.

4.3.9 Otras tecnologías de comunicación

Las tecnologías de comunicación proporcionan mecanismos para la comunicación entre clientes y servidores, y entre los distintos objetos que colaboran en los distintos servidores. Junto a las tecnologías de comunicación proporcionadas por la plataforma J2SE, y las tecnologías de mensajería proporcionadas por J2EE, la especificación J2EE requiere soporte para los siguientes tipos de tecnologías de comunicación y formatos de datos:

4.3.9.1 Protocolos de Internet

Estos protocolos definen los estándares por los que las diferentes piezas de la plataforma J2EE se comunican entre ellas y con las entidades remotas. La plataforma J2EE soporta los siguientes protocolos de Internet:

a) TCP/IP: Protocolo de control de transporte sobre IP. IP no garantiza la entrega de paquetes de datos y ni orden de llegada. TCP nos proporciona la garantía de entrega y la llagada ordenada de paquetes.

b) HTTP: Protocolo de Transferencia de HiperTexto, es un protocolo del nivel de aplicación sin estado de petición/respuesta entre cliente/servidor. La comunicación Web se realiza normalmente con HTTP sobre TCP.

c) SSL: Protocolo de seguridad que proporciona privacidad en Internet. Los protocolos permiten a las aplicaciones de cliente y servidor mantener una comunicación segura. Los servidores siempre son autenticados y los clientes pueden ser opcionalmente autenticados.

4.3.9.2 Formatos de datos

Los formatos de datos definen los tipos de datos que pueden intercambiarse entre componentes. En las aplicaciones Web, cuando se quiere transferir información no ASCII, se utiliza el protocolo MIME. La plataforma J2EE exige soporte para: HTML, archivos de imágenes GIF y JPEG, archivos JAR, clases .class y archivos XML.

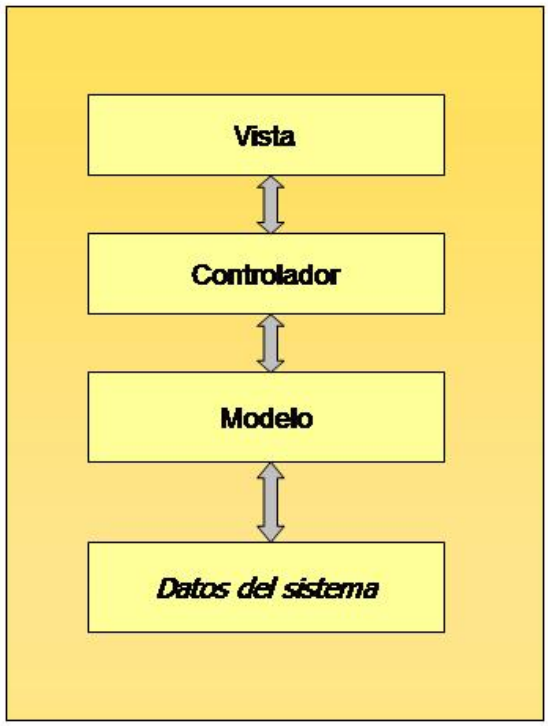
4.4 PATRÓN ARQUITECTÓNICO MODEL-VIEW-CONTROLLER, MVC

Junto a la estructura física multicapa, en el diseño de aplicaciones J2EE se utiliza el patrón arquitectónico Model-View-Controller, MVC. Un patrón arquitectónico es un patrón de alto nivel que fija la arquitectura global de la aplicación. Con el modelo MVC, se permite una fácil separación de la interfaz gráfica y del modelo de negocios gracias a un controlador que los mantiene separados.

El patrón MVC es, en gran parte, el soporte de la arquitectura multicapa típica de J2EE. La separación de presentación y lógica nos permite tener configuraciones en las que el cliente tan sólo disponga de la interfaz gráfica y acceda a un servidor donde se implemente el modelo.

La estructura de una aplicación según el patrón MVC se divide en tres partes. La parte de la vista representa la interfaz gráfica y generación de contenido dinámico. La vista no interactúa directamente con el modelo, sino que lo hace a través de un controlador que realizará las invocaciones remotas si son necesarias y mantendrá el control de flujo de las peticiones. La aplicación por último accederá a los datos a través del modelo, donde se ejecuta también la lógica de negocio.

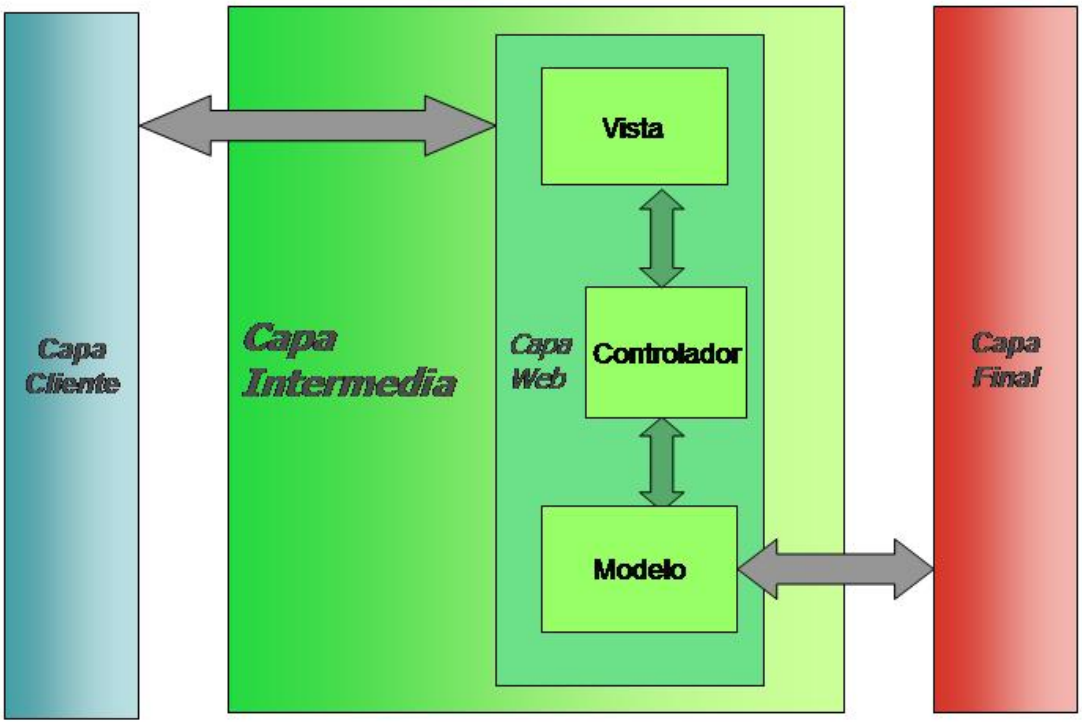
Esta división en la estructura de una aplicación permite a la plataforma J2EE distintas posibilidades en la configuración de la arquitectura física. La estructura de la aplicación según el patrón MVC y la arquitectura de la aplicación están íntimamente relacionadas. Según la situación de cada una de las partes del modelo MVC de la aplicación, obtendremos una u otra arquitectura física.



F. 4-12: Patrón arquitectónico Model-View-Controller, MVC.

En una aplicación J2EE completa el diseño más común es aquel en el que la vista y el controlador se implementan en la capa Web, y el modelo en la capa EJB desde donde se acceden a los datos del sistema. Sin embargo, la distribución de cada una de estas capas dependerá de la arquitectura física completa de la aplicación.

En las aplicaciones centradas en Web, al no tener capa EJB, todos los componentes se situarán en la capa Web. En las aplicaciones B2B, el modelo se encontrará en la capa EJB.

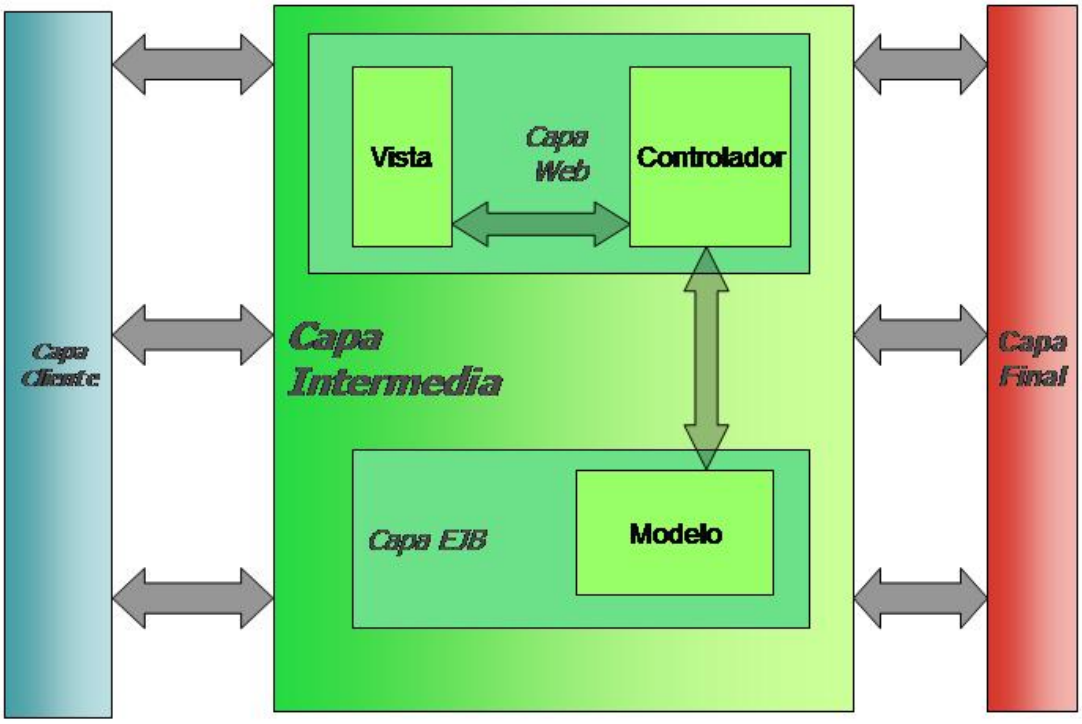


F. 4-13: Patrón MVC en la capa intermedia de una arquitectura J2EE centrada en Web.

En las configuraciones que involucran a la capa intermedia completa, la vista suele estar generada por los componentes JSP de la capa Web. En este tipo de configuraciones, los servlets normalmente implementan el controlador. En el caso de no existir la capa de negocio, EJB, los servlets también se encargarán de implementar el modelo y el acceso a datos.

Estas configuraciones son flexibles. Los servlets, pueden generar la vista de una aplicación, así como las páginas JSP a través de extensiones de etiquetas pueden implementar tanto el controlador, como el modelo incluido el acceso datos. Sin embargo, estas últimas configuraciones no son recomendables, más que en aquellos casos en los que la sencillez de la aplicación lo permita. Para una aplicación medianamente compleja, se recomienda la utilización

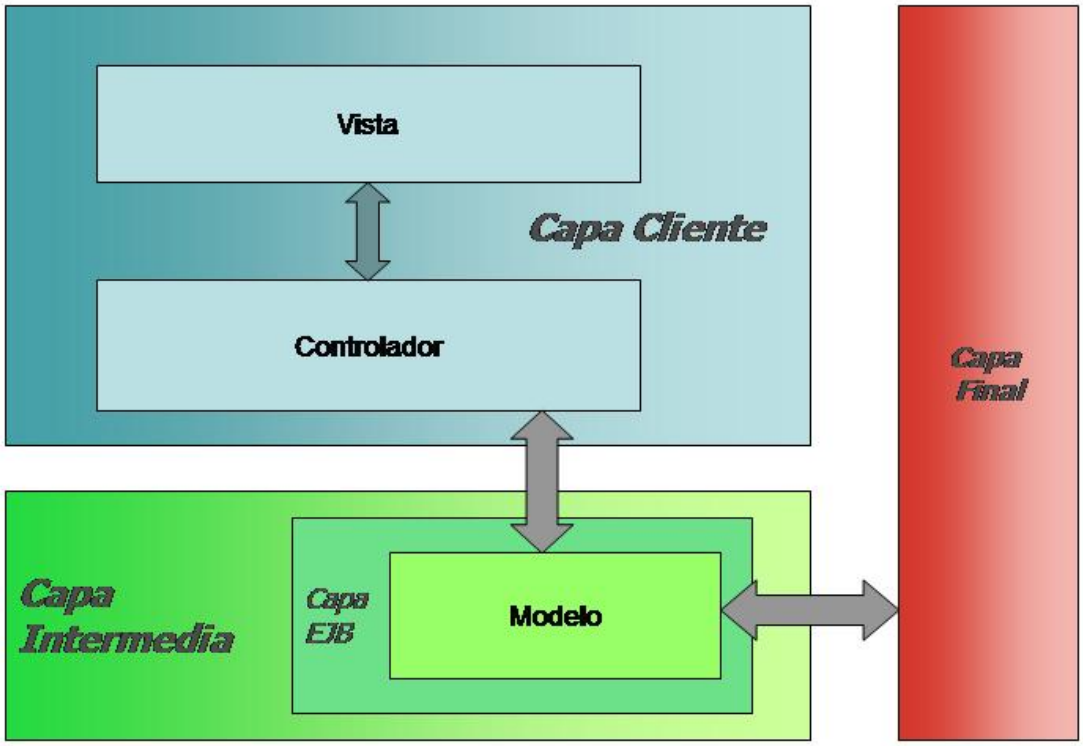
del patrón MVC implementando la vista con componentes JSP y el controlador con servlets. El uso o no de componentes EJB también depende de las exigencias y complejidad de la aplicación. En muchos casos es aceptable la implementación del modelo con servlets si el uso de componentes EJB no se considera justificado.



F. 4-14: Patrón MVC en la capa intermedia de una arquitectura J2EE B2B.

En aplicaciones clientes, típicamente la vista se situará en la capa cliente, mientras que situación de las capas controlador y modelo dependerá del resto de la estructura.

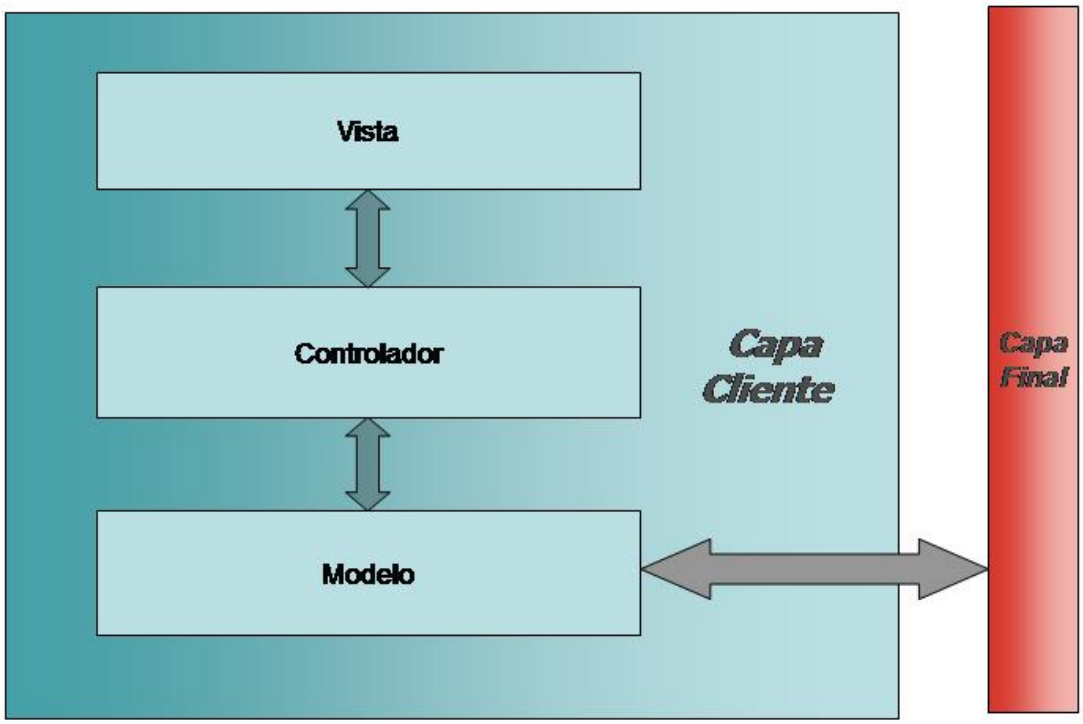
Las aplicaciones clientes EJB, implementarán la vista y el controlador, mientras que el modelo lo implementará la capa intermedia a través de componentes EJB.



F. 4-15: Patrón MVC en una aplicación cliente EJB.

Las aplicaciones clientes stand-alone, con acceso directo a los datos implementarán tanto las tres partes del modelo MVC, mientras que las aplicaciones clientes con acceso a la capa Web sólo implementarán la vista. En estas aplicaciones el modelo se implementará en la capa EJB o en la capa Web según se utilicen componentes EJB o no.

Las aplicaciones clientes que implementen la vista, normalmente lo harán utilizando las librerías de Swing o AWT.

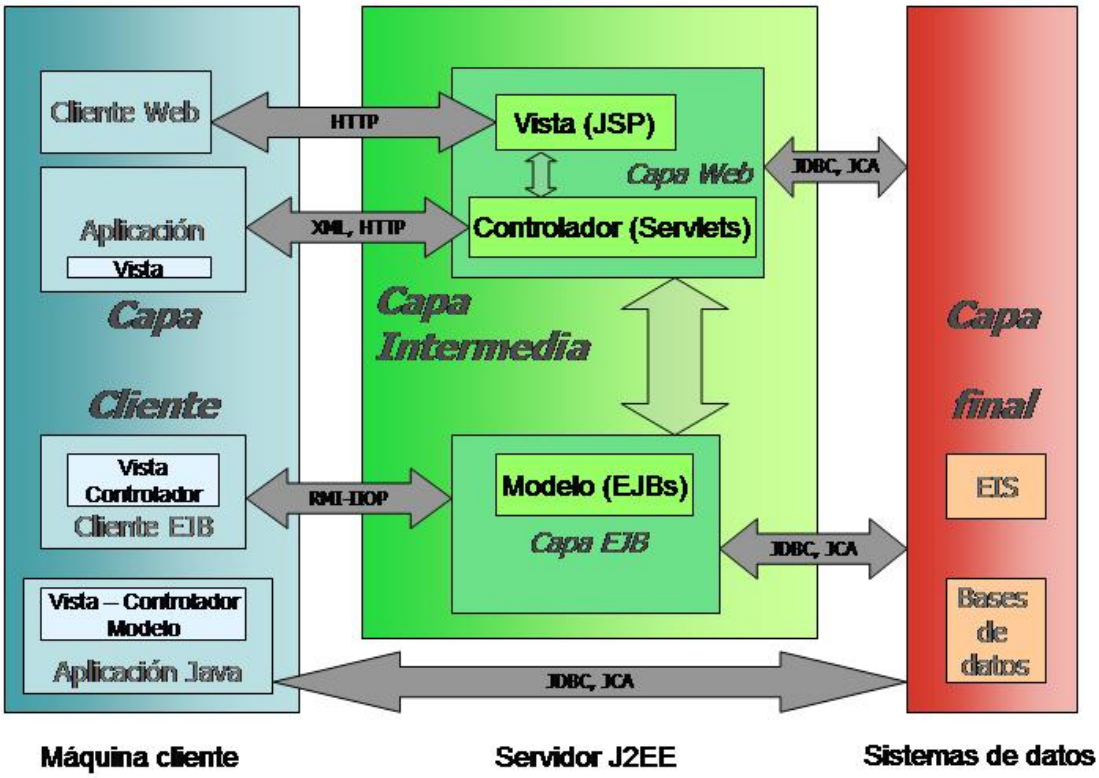


F. 4-16: Patrón MVC en una aplicación stand-alone con acceso directo a datos.

Además, J2EE utiliza el patrón arquitectónico Layers. Este patrón estructura el software en capas, por lo que dará soporte a la arquitectura MVC. Se podría considerar MVC como una abstracción de alto nivel del patrón Layers, cada capa del patrón MVC podría estar dividida en más subcapas.

La utilización de los modelos MVC y Layers nos permite hacer frente a actualizaciones o cambios en el modelo sin que los clientes se vean afectados. Además, este patrón proporciona otros beneficios como la separación de roles en el equipo de trabajo. Nos permite tener un grupo programadores para el desarrollo de la lógica de negocio y diseñadores Web para la presentación.

A continuación, como resumen, se muestra una implementación del patrón MVC en una arquitectura completa J2EE.



F. 4-17: Patrón MVC en una arquitectura J2EE completa.

4.5 PATRONES DE DISEÑO

Junto a las tecnologías de J2EE, es importante destacar el uso de patrones de diseño específicos de J2EE. Algunos de estos patrones son especializaciones de patrones generales aplicados al contexto de esta plataforma.

Sun Java Center ha ido identificando problemas similares que surgen en multitud de proyectos y aplicando soluciones similares a estos problemas. Estos problemas y sus soluciones recurrentes se han ido identificando, abstrayendo y documentado, y se ha creado lo que se conoce como Patrones de Diseño J2EE.

Los patrones de diseño presentan soluciones en distintos niveles de abstracción, análisis, diseño, arquitectura e implementación. También definen las relaciones de unos patrones con otros. La definición más extendida de patrones de diseño es la de “solución recurrente para un problema en un contexto”, donde el problema es cualquier cuestión insatisfecha, algo que se necesita investigar o resolver y que puede ser fácilmente especificado por un conjunto de causas y efectos. Los patrones pretenden comunicar soluciones de diseño a estos problemas a los desarrolladores.

Otra definición para el concepto de patrón de diseño, ya centrada en la estructura de las aplicaciones, clases y objetos, es la de descripciones de objetos y clases comunicándose, adaptadas para resolver un problema de diseño general en un contexto particular.

Si sólo nos centramos en el uso de las tecnologías J2EE podemos obtener modelos de alto coste de mantenimiento y grandes duplicaciones de código, sin embargo se puede mejorar enormemente la calidad de estas aplicaciones con la utilización de los Patrones de Diseño J2EE basados en años de experiencia en desarrollo y diseño.

El principal objetivo de los patrones de diseño es capturar buenas prácticas que nos permitan mejorar la calidad del diseño de sistemas. Con este fin se determinan objetos útiles en un contexto específico, encapsulando complejidad y haciéndolo más flexible. Estos patrones han demostrado ser una forma eficiente de reutilizar el diseño apartando beneficios a nivel de ahorro de inversión y calidad a la hora de desarrollo de aplicaciones.

Los Patrones de Diseño J2EE nombran, abstraen e identifican aspectos claves de estructura comunes de diseño descritos de una forma específica documental. Estos patrones, en general, hacen que la aplicación sea más fácil de mantener, sencilla y limpia. Algunas de las aplicaciones de estos patrones centralizan y encapsulan mecanismos como servicios de seguridad, de recuperación de contenidos y navegación, y eliminan gran cantidad de scriplets en las páginas JSP.

Al hablar de los Core J2EE Patterns se hace referencia a un catálogo de los Patrones de Diseño J2EE publicados por el Centro Java Sun. En este catálogo se describen los problemas típicos encontrados por los desarrolladores de aplicaciones empresariales y proveen soluciones para estos problemas. Además, en la documentación de estos patrones, se proponen distintas implementaciones combinando las tecnologías que J2EE proporciona de una forma eficiente. Muchos de estos patrones se basan en patrones aplicados al diseño de aplicaciones de software en general.

Un patrón J2EE esta documentado según una plantilla que define las siguientes secciones:

a) Contexto, entornos bajo los cuales el patrón existe.

b) Problema, describe el problema al que se enfrenta el desarrollador.

c) Motivación, razones y motivaciones que afectan al problema y a la solución, resaltando el porqué se debe elegir el patrón en discusión.

d) Solución, describe la solución en detalle. Se aportan estructuras y diagramas de clases que definen los objetos y relaciones entre ellos, y estrategias que describen las diferentes formas de cómo un patrón puede ser implementado intentando proporcionar detalles de bajo nivel.

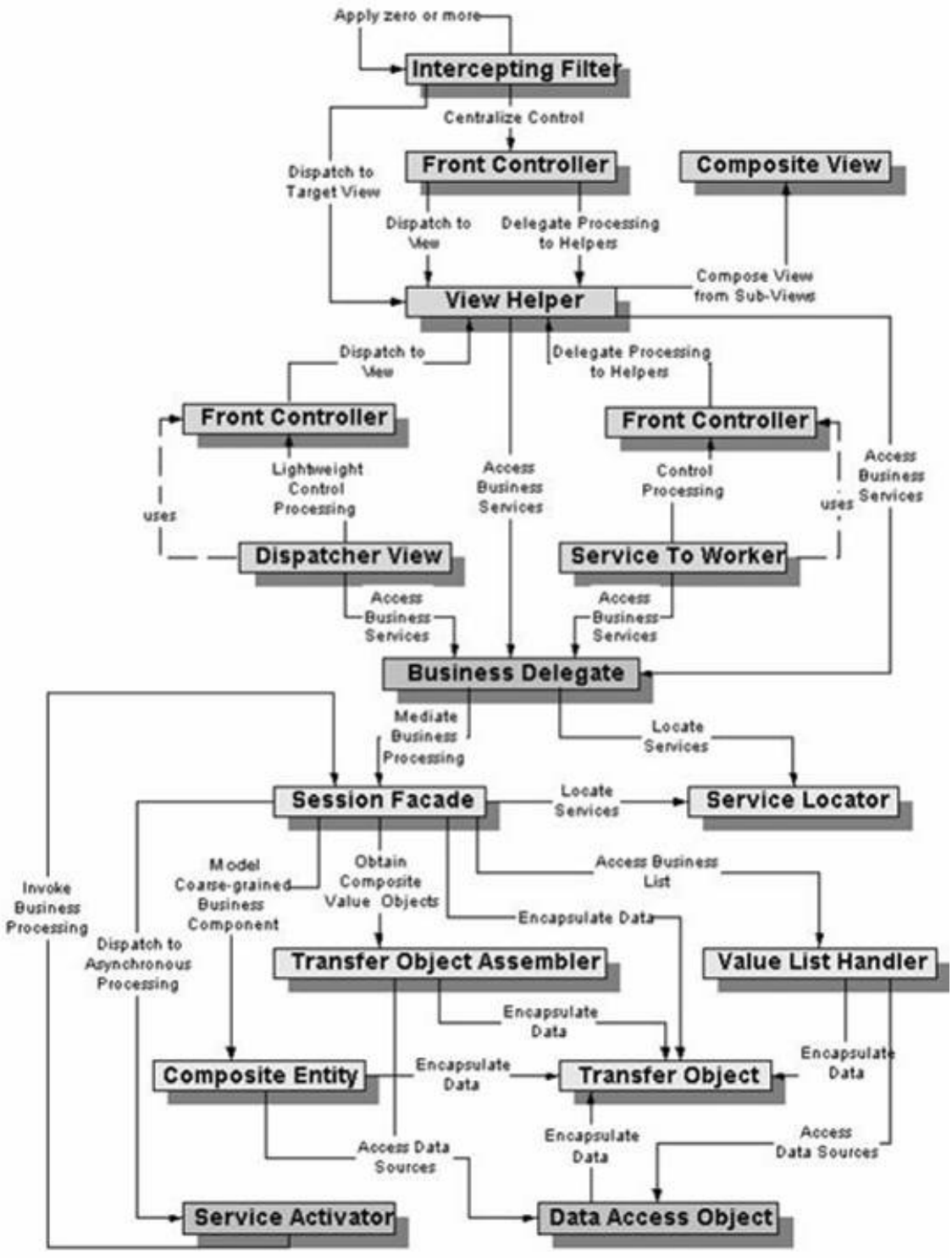
e) Consecuencias, las ventajas y desventajas de aplicar el patrón.

f) Patrones relacionados, lista de patrones y relación con el que se describe.

A continuación se muestra el catálogo de Patrones de Diseño J2EE, Core J2EE Patterns. Existen multitud de patrones soluciones a distintos problemas, nosotros nos centraremos en el uso de sólo algunos de ellos que se detallarán en capítulos posteriores.

Para el diseño de aplicaciones empresariales basadas en los Patrones de Diseño J2EE utilizamos ciertos elementos que combinados tienen la estructura Model – View – Controller, estructura multicapa de nuestras aplicaciones empresariales conforme a la plataforma J2EE, presentada en el apartado anterior, 4.4 PATRÓN ARQUITECTÓNICO MODEL-VIEW-CONTROLLER, MVC. Principalmente se definen patrones que componen la vista de la aplicación en la capa Web y la implementación de un controlador. Además se incluyen patrones para acceso las capas del modelo y el acceso a datos.

La estructura de clases de una aplicación puede derivarse fácilmente de este esquema de patrones y sus relaciones. Según los patrones que se decidan utilizar se implementarán las clases apropiadas obteniendo la estructura principal de la aplicación. Sin embargo, en muchos casos, en el diseño de una aplicación, se utilizarán frameworks que ya implementan estos patrones de diseño. Existen distintos proveedores de frameworks para aplicaciones J2EE, uno de los más utilizados es Struts, de Apache. El uso de frameworks, normalmente, es recomendable, ya que implementan funciones complejas de controlador y vista, y permiten configuraciones personalizadas en ficheros XML en el momento de despliegue al igual que los contenedores J2EE.



F. 4-18: Patrones de Diseño J2EE.

(Core J2EE Patterns: Best Practices and Design Strategies authored by architects from the Sun Java Center.)

4.6 XML EN LA PLATAFORMA J2EE

XML juega un papel muy importante en el escenario J2EE. Java y XML son tecnologías complementarias; Java nos proporciona portabilidad del código, XML nos proporciona portabilidad de datos.

El lenguaje de etiquetas eXtensible Markup Language, XML, nos permite escribir documentos de texto que expresan datos y no aspecto visual. Este lenguaje, extensible como su propio nombre indica nos permite definir un conjunto de etiquetas particulares según la aplicación, ya sean intercambio de datos entre aplicaciones heterogéneas, configuración de aplicaciones, generación de aspecto visual a partir de los datos o bases de datos entre otras.

J2EE ofrece un soporte estándar que usa XML para especificar la información de configuración de las aplicaciones Web y los componentes EJB. Permite configurar aspectos de los contenedores de aplicaciones Web y componentes EJB, así como configuración específica a la aplicación. El programador accede a la información de configuración específica de la aplicación mediante API de J2EE, normalmente mediante JNDI. Los contenedores de aplicaciones Web y componentes EJB también suelen usar XML para su propia configuración.

Los contenedores proporcionan un mecanismo de selección de comportamiento de la aplicación en tiempo de ensamblaje o despliegue de la aplicación a través de descriptores de despliegue. Los descriptores de despliegue son ficheros XML que especifican el comportamiento de los componentes y el contenedor. Gracias a estos descriptores de despliegue, los componentes pueden ser configurados para el entorno de un contenedor específico cuando se despliegan, no teniéndolo que hacer en el código de la aplicación. Entre las características que pueden configurarse se incluyen seguridad, control de transacciones y otras responsabilidades de control.

Por otra parte, la habilidad de producir y consumir mensajes de datos XML en el contenedor Web, es un modo extremadamente flexible de abarcar un conjunto diverso de tipos de clientes.

Junto al documento XML, aparecen otros documentos que validan (pero no dan formato) ese documento. Estos documentos de validación pueden ser las DTD o los esquemas (XSDL, XML

Schema Definition Languaje). Los esquemas XML permiten expresar más restricciones en cuanto a tipos de datos, que las DTD, aunque su sintaxis es más compleja que la de las DTD.

Las DTD (Data Type Definition) son un mecanismo de especificación del conjunto de etiquetas válidas en un documento, cómo están anidadas, si son vacías o no y los atributos (y su tipo) que tiene cada elemento. Cuando usamos DTD, un documento es válido si cumple con las reglas de su DTD asociado.

Para las configuraciones de contenedores, el fichero de configuración será un fichero que utiliza una serie de etiquetas con sus atributos. Las posibles etiquetas y los posibles valores de sus atributos quedan definidos en la DTD o esquema. Por ejemplo, en las configuraciones de los contenedores Web, hasta la versión de servlets 2.3 se utiliza para validar el archivo `web.xml` las DTD, a partir de la versión 2.4 se utilizan los esquemas.

Para poder validar un documento XML con su DTD necesitamos un *parser*. J2SE proporciona la API JAXP (Java API for XML Processing).

El conjunto de normas que define XML y la posibilidad de definir reglas con las DTD o los esquemas permite construir parsers genéricos que son capaces de comprobar que el documento XML "está bien formado" y es "válido". Estos parsers normalmente no habrá que crearlos. Existen parsers para los lenguajes más usuales, conformes a varias API estándares como DOM y SAX.

Un parser SAX, Simple API for XML, es un pequeño framework basado en eventos. El programador proporciona uno o varios objetos callback a los que el parser llamará cada vez que ocurra un evento de interés (apertura de una etiqueta, cierre de una etiqueta, un error, etc.). Se trata de un parser secuencial, con un consumo mínimo de memoria lo que lo hace más eficiente. No es fácil acceder a elementos anteriores, hermanos, etc. Puede ser difícil de usar en algunas situaciones.

Los parsers DOM, Document Object Model, aunque es más sencillo que SAX al construir una representación (un árbol) en memoria del documento, el consumo de ésta es mayor. El programador puede acceder a los datos del documento recorriendo el árbol ya que permite

acceder fácilmente a todos los detalles del documento. Permite construir y/o modificar árboles, y generar XML.

Existen implementaciones Java de éstos parsers en la plataforma J2EE. J2SE incluye JAXP (Java API for XML Processing), entre otros, los paquetes `javax.xml.parsers` y `javax.xml.transform`. En un principio JAXP formaba parte de J2EE, pero a partir de la versión 1.4 se movió a J2SE.

El paquete `javax.xml.parsers` incluye dos factorías abstractas para obtener instancias de parsers SAX (familia org.xml.sax) y DOM (org.w3c.dom). Aunque J2SE incluye una implementación por defecto de JAXP, se puede utilizar otra si se desea.

Para generar el aspecto visual dinámicamente las transformaciones XSL (XSLT). En este caso, las hojas de estilo acompañan al documento XML. Un fichero XSL (eXtensible Stylesheet Language) permite expresar reglas para transformar un documento XML en otro formato (por ejemplo en HTML, en otro documento XML, en PDF, etc).

También existen API de Java para estas transformaciones. El paquete `java.xml.transform`, que forma parte de JAXP, proporciona una factoría abstracta que permite obtener un transformador para realizar la transformación de un documento XML a partir de un documento XSL.

Otro ejemplo de uso de XML son los Servicios Web para la integración de aplicaciones heterogéneas que utilizan SOAP, un protocolo basado en XML sobre HTTP. Para ello J2EE ofrece las API JAX-RPC, JAAS y JAXR.

Esta tecnología de integración de aplicaciones heterogéneas aunque es reciente es muy utilizada. Los Servicios Web utilizan SOAP, un protocolo basado en XML, para el intercambio de información en un entorno distribuido. Conceptualmente SOAP permite enviar peticiones/respuesta en XML, normalmente sobre HTTP. Un Servicio Web es un servicio ofrecido a través del Web que envía información mediante SOAP.

Con la tecnología de Servicios Web, cliente y servidor pueden estar escritos en distintos lenguajes y corriendo distintas plataformas (hardware y sistema operativo). Cuando la comunicación es vía Internet normalmente se utiliza HTTP. Este protocolo es reconocido por todos los firewalls. Además, el servidor se suele implementar como una sencilla aplicación Web y el cliente sólo necesita conectarse a una URL, leer su respuesta, parsearla e introducirla en su base de datos.

Existen API basadas en RPC (ej.: JAX-RPC en J2EE), que permiten implementar un Servicio Web como un objeto cuyos métodos se pueden invocar remotamente, como si de un objeto local se tratase. Estas API ocultan el uso de XML y el intercambio de información se podría programar sin necesidad de parsear XML.

Con los Servicios Web es fácil por ejemplo contratar contenidos de forma que la información se nos pasara a través de un Servicio Web. Nuestra aplicación sólo tendría que invocar el Servicio Web que ofrece estos contenidos y generar dinámicamente, por ejemplo, HTML.

Otra forma más sencilla es que la empresa de contenidos proporcionara una URL que ofrece la información en XML. Nuestra aplicación, al acceder a dichos contenidos leería estos datos de esta URL y los transformaría dinámicamente a HTML mediante un documento XSL.

J2EE, a través de JSTL (Java Standard Tag Library), proporciona una biblioteca de etiquetas que implementan funciones de uso frecuente en aplicaciones JSP. Entre estas funciones ofrece etiquetas para realizar una transformación en una página JSP, y para parsear un documento XML y acceder a su información de una manera bastante parecida a XSL. Estas etiquetas representan una alternativa a XSL aunque menos potente.

4.7 ESTANDARIZACIÓN, JAVA Y J2EE.

Java no es tan sólo un lenguaje de programación orientado a objetos independiente de la plataforma, desarrollado por Sun Microsystems a principios de los noventa. Java hace referencia

a toda una plataforma virtual formada por el lenguaje Java, una máquina virtual (JRE) que proporciona la portabilidad en ejecución, y una biblioteca estándar para el lenguaje (API Java).

Aunque tiene origen como lenguaje propietario de Sun Microsystems, a mediados de la década se creó el Java Community Process, JCP, como un proceso abierto y participativo para desarrollar y revisar las especificaciones de la tecnología Java.

Actualmente Java es libre, aunque en los comienzos del JCP, Sun Microsystems publicó sus API, pero sólo permitía modificaciones a sus licenciatarios, por lo que no se podía considerar software libre. Actualmente grandes empresas informáticas como IBM, Oracle, SAP, Siebel, Nokia, BEA, Ericsson o HP, pertenecen al JCP. Cualquier interesado, sociedad o individual, puede formar parte de este proceso.

Las especificaciones y tecnologías de la plataforma Java son descritas formalmente a través de las Java Specification Requests, JSR, que, junto a la descripción detallada, incluyen una implementación de referencia (RI) de la tecnología y un test de compatibilidad (CTS).