UNIVERSIDAD CATOLICA “NUESTRA SEÑORA DE LA ASUNCIÓN”

Facultad de Ciencias y Tecnología

Trabajo Practico TAI 2

J2EE

Integrantes: Daniel Cricco
Julio Rey

Profesor: Juan de Urraza

Año 2004

El desafió del desarrollo n-tier

La tendencia actualmente en el desarrollo de aplicaciones empresariales es de tener un framework apuntado a la construcción de aplicaciones seguras y escalables. Para este propósito Sun Microsystem introduce Java 2 Enterprise Edition (J2EE), y Microsoft Corporation se aventura con .NET framework. Para entender como llegamos a necesitar este tipo de framework es preciso hacer un repaso de la evolución de los sistemas empresariales.

Observaciones

Tier: Se refiere a una maquina diferente
Layer: Se refiere a una capa lógica en términos de software, muchos layer pueden estar en una misma máquina.

Desarrollo Monolítico

En los días de la mainframe y cuando empezaron a aparecer las computadoras personales con aplicaciones standalone, ósea cuando la aplicación estaba en una sola maquina, era común encontrar lo que se conoce como programación monolítica, cuya característica es la de tener toda la funcionalidad de la aplicación en un grande, frecuentemente inmantenible, código (conocido como código spaghetti).

Todas lo referente a el input del usuario, la verificación, la lógica de negocios, y el acceso a los datos se encontraba junto. La modificación de una parte del programa requería recompilar todo, o peor aun, la modificación de una parte introducía bugs en otras secciones del software, por lo que se hacia cada vez mas difícil de mantener. La figura 1 muestra este tipo de aplicaciones.

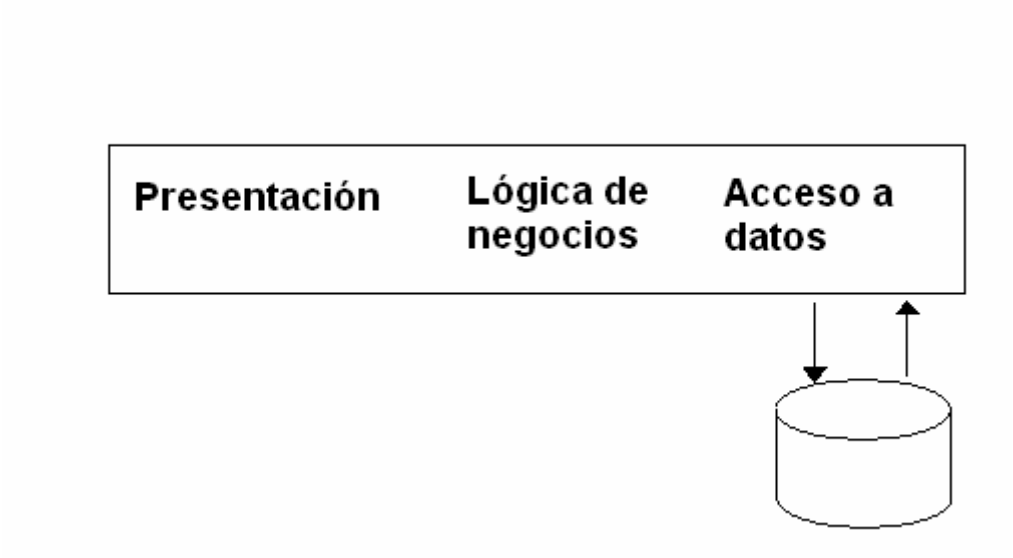

fig 1

La aparición del segundo tier

La aparición del segundo tier nace de el deseo de compartir datos entre múltiples aplicaciones instaladas en diferentes maquinas. Para hacer esto se necesita un servidor de datos.

La ventaja de tener el acceso a datos en otro ambiente físico no es solo la comparición de los datos, sino que cualquier cambio hecho a la lógica de acceso esta localizada en un segundo tier. De hecho, todo el segundo tier puede ser reemplazado con un servidor de base de datos diferente siempre que se conserven las interfaces entre los dos tier. Este modelo se presenta en la figura 2

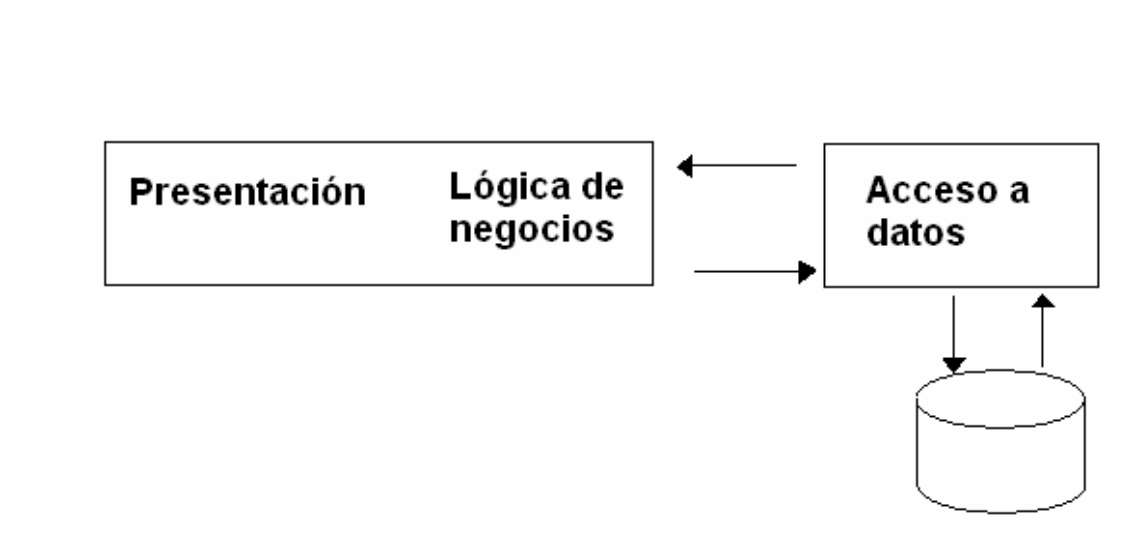

fig 2

Consecuencia del diseño 2-Tier

Uno de los problemas de esta arquitectura es que los clientes tienen el código de la lógica de negocios y necesitan tener los detalles de la localización de los datos. Existe una concentración de funcionalidades en el cliente por lo que estos se conocen como **thick client**. Thick client normalmente necesitan de ser actualizados cuando la aplicación cambia. Esto limita nuevamente los procesos de mantenimiento y la escalabilidad de la aplicación.

Con la llegada de Internet hubo un movimiento para separar la lógica de negocios de la interfaz de usuario. Los usuarios Web, necesitan acceder a aplicaciones sin instalar nuevo código en sus máquinas. De hecho se pretende usar siempre la misma aplicación (el web browser) para acceder a las diferentes aplicaciones que se encuentran en la web. Estos clientes ya no tienen la lógica del programa por lo que se los conoce como **thin client.** De esta manera llegamos al modelo 3-tier que se ha convertido en el modelo por defecto de las aplicaciones basadas en Web.

Esta separación le convierte a los sistemas muchos mas flexibles así las partes pueden ser cambiadas independientemente. Este modelo se muestra en la figura 3.

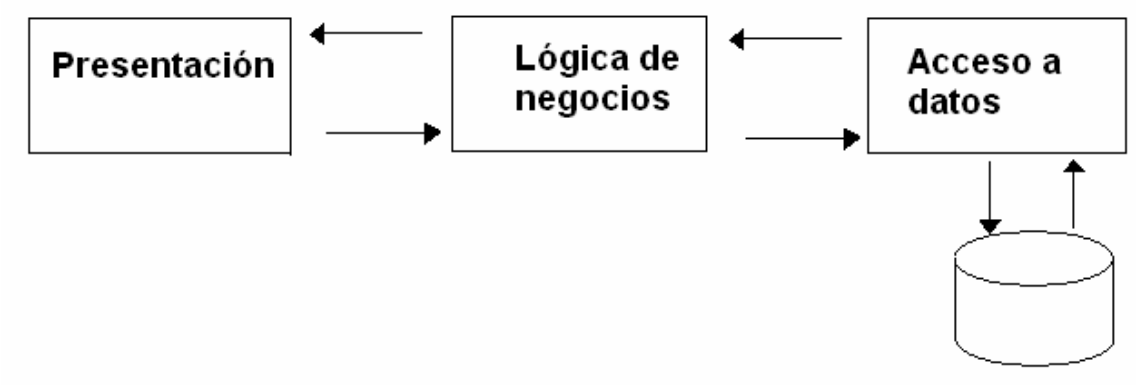

fig 3

Tecnología de componentes

Un componente es una unidad de funcionalidad que puede ser usada en un particular framework. Los componentes proveen soporte para simplificar el desarrollo de aplicaciones. Cuando se usa un framework orientado a componentes, un contenedor provee al componente de unos servicios Standard, como comunicación y persistencia.

De esta forma cada programador que sea mas hábil en una parte puede desarrollar los componentes por separado, o se pueden comprar componentes de terceros que sean conformes con el framework que usamos. Esto ahorra tiempo por lo tanto costos. El sistema se vuelve mas mantenible si las diferentes partes pueden ser actualizadas sin la necesidad de parar la ejecución del programa como un todo. La utilización de componentes permite desarrollar aplicaciónes débilmente acopladas por lo que la escalabilidad esta asegurada.

Java 2 Enterprise Edition (J2EE)

J2EE es un Standard para producir aplicaciones empresariales que sean seguras, escalables y altamente disponibles. El strandart dicta que servicios deben ser proporcionados por servidores que soporten J2EE. Estos servidores proveen contenedores en donde los componentes correrán. Los contenedores proveen un conjunto de servicios a los componentes. J2EE provee una especificación con la que se pueden construir servidores de aplicación J2EE en los que se pueden correr aplicaciones J2EE. A pesar de que se define el conjunto de servicios y tipos de componentes, no se provee una información de cómo organizar la arquitectura lógica en maquinas físicas, direcciones y ambiente.

Componentes y contenedores

J2EE establece que un servidor de aplicación que obedezca la especificación J2EE tiene que proveer un conjunto definido de contenedores para albergar a los componentes. Los contenedores proveen un ambiente de ejecución para los componentes. Debe haber un contenedor para cada tipo de componente:

- Applet Container
- Application Client Container
- Web Container
- EJB Container

Hay dos tipos de componentes que se ejecutan en el J2EE Server

Componentes Web: Este interactúa con un cliente basado en Web, como un Web Browser. Hay dos clases de componentes Web en J2EE – Servlet y JSP. Los dos se encargan de la presentación de los datos al usuario

Componentes EJB : Hay tres clases de EJB: Session bean, entity bean y message Driven Bean.

Servicios de servidores J2EE

Los contenedores tienen que proveer los siguientes servicios

Conectividad: Deben soportar conección a otros componentes y otros servidores de aplicación. Una forma de requerir conectividad es distribuir objetos a traves de Java Remote Method Invocation (RMI) y CORBA. La conectividad de internet se debe hacer a traves de Hypertext Transport Protocol (http) y su forma segura HTTPS.

Servicio de directorio: Debe de proveer un servicio de nombres en donde los componentes pueden ser registrados y descubiertos. Java Naming and Directory Interfaces (JNDI) provee una manera de acceder a este servicio.

Acceso a datos y persistencia: Esto se provee a traves de Java Database Connection API (JDBC).

Conectividad con sistemas de legado: Java Connector Architecture provee el soporte para integrar servidores de información empresarial y sistemas de legado.

Seguridad: Aquí se provee seguridad para la autenticación a traves de Java Authentication and Authorization Service (JAAS). Al igual que JAAs existen API`s para la parte de criptografia y todas las areas de seguridad

Soporte para XML: JAXP soporta el parseo de documentos XML usando Document Object Model (DOM).

Transacciones: Los limites de las transacciones deben ser especificados por el contenedor. El contenedor tiene la responsabilidad de llevar a cabo lastransacciones, y a traves de Java Transaction API (JTA) se permite a los componentes controlar su propia transacción.

Mensajes y e-mail: Java Message Service (JMS) permite a los componentes enviar y recibir mensajes asincronos, tipicamente dentro de los limites de una organización. Java Mail API provee la capacidad para enviar e-mail a traves de Internet, de recibir e-mail de algun servidor de mail. También soporta MIME. La figura 4 muestra la interaccion entre los contenedores.

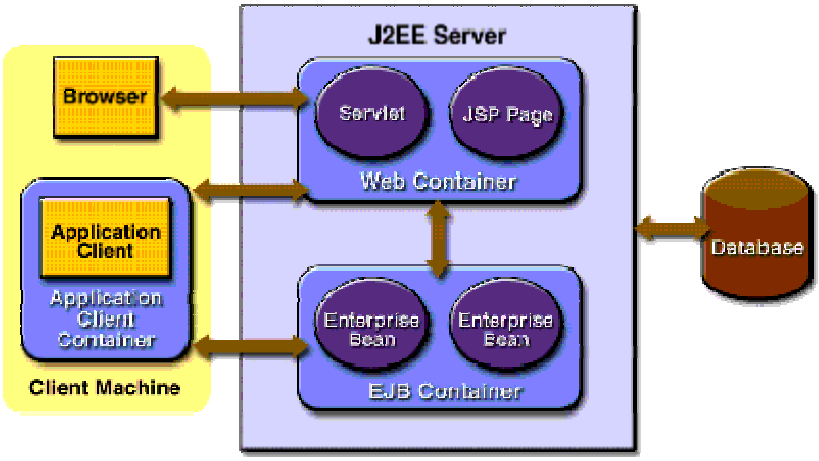

fig 4

Servidores de aplicación J2EE

BEA System: BEA Weblogic Server incluye soporte para Web Service, J2EE Connector Architecture, EJB 2.0, Servlet 2.3 y JSP 1.2

IBM: Websphere Commerce Business Edition soporta EJB, JSP, XMLy HTTP.

iPlanet: Soporta la plataforma J2EE, monitores de transaccion, provee iPlanet Web Server, an iPlanet Directory Server. Soporta XML, gíreles application protocols, SNMP, LDAP, CORBA y JDBC.

Jboss: Es freeware y provee una implementacion de EJB 2.0. No incluye un Web container, pero JBoss esta disponible con un freeware Web server

Peristence: PowerTier Release 7 para J2EE soporta EJB y esta integrado con Racional Rose.

Jonas: Es freeware. Soporta EJB 2.1, transacción y utiliza como Web server Tomcat 5 que tambien es freeware.

Presentación y logica de negocios

El modelo 3-tier parte una aplicación y la lógica de negocios queda en el medio, por eso generalmente la lógica de negocios se conoce como middle tier. El primer tier tiene la responsabilidad de proveer la interfaz entre el usuario y la aplicación. La mayoría del código de una aplicación J2EE reside en estos dos tier. Los componentes en diferentes tier deben estar débilmente acoplados, un componente no debe tener dependencia de un cliente u otro componente. Por ejemplo si tenemos un componente de negocios que procesa pagos de tarjeta de créditos. Si el componente esta contenido en si mismo, cualquier aplicación que se comunique a través de la interfaz adecuada puede utilizar el componente y este responderá adecuadamente.

Beneficios de los componentes de negocio

Incrementa eficiencia: La división de labores ayuda a que se desarrolle un aplicación más rápido. El uso de componentes permite a los programadores de presentación desarrollar el GUI, los encargados de la lógica de negocios se concentran en su sección y los expertos en acceso a datos se encargan de este aspecto, y todo concurrentemente.

Extensibilidad: Simplemente se puede agregar o quitar componentes de una aplicación para aumentar la funcionalidad.

Independencia del lenguaje: Un sistema modular permite escribir código en un lenguaje que se comunica con código de otro lenguaje. Por ejemplo se puede acceder a la funcionalidad de una aplicación J2EE desde un cliente CORBA usando IIOP o desde un cliente Microsoft COM usando Client Acces Services COM Bridge.

Actualización del sistema: Inevitablemente los procesos de una organización cambian y de acuerdo la lógica de negocios. El uso de componentes permite cambiar un componente sin afectar los otro componentes del sistema

Enterprise JavaBeans (EJB)

En una típica aplicación J2EE la lógica de negocios se encapsula dentro de EJBs. Es posible usar otros tipos de componentes o directamente objetos JAVA para implementar la lógica de negocios, pero EJBs provee un medio conveniente de encapsular y compartir lógica de negocios, y también se beneficia de los servicios que provee el contenedor.

Supongamos que tenemos que hacer una aplicación de e-commerce basada en Web. Podriamos tener el siguiente flujo:

1) Desplegar los producto al cliente
2) Permitir al cliente seleccionar uno o mas productos
3) Confirmar la orden y mostrar los detalles de envió
4) Realizar el cobro
5) Enviar la orden a la empresa encargada de la distribución

Todas estas etapas envuelven un grupo de lógica de negocios y acceso a datos. Por ejemplo en la etapa 1, la aplicación necesitara enviar paginas HTML que contengan información del producto al browser del cliente. Esta información debe ser sacada de algún lugar, la opcion obvia seria una base de datos.

Puede que sea posible que la información del producto no se pueda obtener directamente de la base de datos. Por ejemplo:

La información del producto esta distribuida en múltiples bases de datos. Peor aun, existe información que debe ser extraída de un sistema de legado. Hay procesos de negocios que deben ser aplicados durante la creación del catalogo. Esto puede variar desde precios especiales hasta sugerir al cliente productos parecidos. Si se debe identificar al cliente, entonces se puede obtener información de sus preferencias.

Como se puede notar si ponemos todo en una sola porción de código esto se va pareciendo mucho al código spaghetti. Lo mejor seria asociar la lógica de negocios con EJBs para que el user interface se concentre en su parte.

Luego de construir el EJB, este tiene que estar disponible esto se hace a través de RMI y JNDI. La funcionalidad del EJB esta disponible para el cliente a través de un interfaz RMI. Cuando un EJB es instalado en el servidor, su localización es registrada en el servicio de nombres.

Un cliente luego usa JNDI para localizar un EJB. El cliente va a interaccionar con una fabrica de objetos llamada EJB`s home para obtener una instancia del EJB. Cuando tiene la instancia puede usar sus funcionalidades.

Existen diferentes tipos de EJB ya que se necesitan diferentes comportamientos.

EJB Session Bean

Encapsula un conjunto de funciones de negocios comunes. Usando UML, Use Cases son identificados, estos representan una interacción entre el usuario y el sistema, esta interacción puede ser encapsulada dentro de un Session Bean.

Ofrece una interfaz sincrona a través de la cual el cliente puede usar la lógica de negocios. No están orientados a almacenar datos, generalmente no almacenan datos o si lo hacen son básicos y temporales.

Componentes: Entity Bean

Generalmente, es la representación de datos de negocios. Durante el análisis varios objetos son descubiertos, por ejemplo Cliente, Cuenta. Estos objetos a veces llamados entidades, estos probablemente se mapearán a entity bean.

Provee una interfaz sincrónica a través de la cual el cliente puede acceder sus datos y funcionalidades. Estos acceden algún tipo de fuente de datos

(frecuentemente una base de datos o un sistema ERP) para recolectar la información que representan. Este actúa como la representación dinámica de los datos, proveyendo métodos para actualizar y devolver de varias formas

Componentes: Message Bean

Ofrece una interfaz asíncrona a través de la cual el cliente puede interactuar. El Bean esta asociado con una cola de mensajes, y cualquier mensaje que llegue a la cola se le da a un Bean asociado con la cola. Como los Session bean, message-driven Bean están orientados a encapsular lógica de negocios en lugar de datos, o sea acceden a los datos a través de JDBC o Entity Bean.

Capa de Presentación

En la capa de Presentación J2ee provee de varias alternativas, Java Servlets, Java Server Page, Applets . Ver figura 5

fig 5

Java Servlets

Los servlets son componentes del servidor. Son piezas de código escritos en Java. Incrementan la funcionalidad de una aplicación web, así como del servidor, del mismo modo que un applet incrementa la funcionalidad de un Browser,. Se cargan de forma dinámica por el entorno de ejecución Java del servidor cuando se necesitan. Cuando se recibe una petición del cliente, el contenedor/servidor web inicia el servlet requerido. El servlet procesa la petición del cliente y envía la respuesta de vuelta al contenedor/servidor, que es enrutada al cliente. La interacción del cliente/servidor basada en Web usa el protocolo HTTP

Permiten una interacción entre diferentes clientes web, ya que pueden recibir múltiples peticiones y manejarlas de forma simultanea..

Los clientes utilizan normalmente como interfaz Navegadores de Internet.

Java Server Pages

Tiene el aspecto de una página HTML donde se pueden incluir scriptlets (scripts) para generar HTML dinámicamente, típicamente los scriptlets se escriben en Java.

Dependiendo del modelo de diseño pueden utilizarse en forma compartida con Servlets. Porque una página con mucho código Java se hace difícil mantener.

El servidor WEB compila el JSP y lo convierte en un Servlet o se puede decir que los JSP son Servlets.

Java Applet

Es un programa escrito en Java que puede ser incluido en una pagina web de la misma forma en que se incluye una imagen. El cliente accede al código (código móvil) y este corre sobre su JVM. El programa normalmente es una interfaz grafica mucho más amigable que las páginas Web. Los programas son confiables dado que Sun lo diseño con un sistema de seguridad muy eficiente para evitar que los programas causen daños a los clientes.

Conclusión

J2EE provee un framework eficiente para desarrollar aplicaciones empresariales. Los servidores de aplicación que cumplen las especificaciones J2EE ahorran mucho trabajo a la hora del desarrollo ya que estos se encargan de la comunicación de los componentes, de la persistencia y de la disponibilidad de estos. Las aplicaciones se pueden actualizar inclusive sin tener que parar el funcionamiento del sistema. Con la ayuda de un diseño débilmente acoplado la escalabilidad del sistema esta asegurada.

Bibliografía

1) Sams Teach Yourself J2EE in 21 Days . Martin Bond Dan Haywood, Debie Law, Andy Longshaw, Peter Roxburgh
2) Thinking y Enterprise Java. Bruce Eckel
3) J2EE Pattern Best Practices and Design Strategies. Deepak Allur, John Crupi, Dan Malks
4) http://www.computerworld.com/developmenttopics/development/story/0,10801,84155,00.html

APÉNDICE

Comparación entre J2EE y .NET

J2EE vs .NET

Propiedades	J2EE	.NET
Tipo de tecnología	Standard	Product
Vendedores de middleware	30+	Microsoft
Interpretador	JRE	CLR
Paginas Web Dinámicas	JSP	ASP.NET
Middle-Tier	EJB	.NET Managed Components
Componentes		
Acceso a base de datos	JDBC SQL/J	ADO.NET
Web	ASP.NET	Java Server Pages (JSP) and Servlets
Middleware implícito	Yes	Yes

Argumentos a favor de .NET

- Microsoft tiene un equipo de desarrollo de el framework .NET
- .NET empezo antes con web service
- Tiene una herramienta potente que es Visual Studio .NET
- Tiene un modelo de programación simple. Apropiado para usuario no tan expertos
- Independencia en el lenguaje, mientras J2EE trata a otros lenguajes como aplicaciones separadas.
- Estrecha comunicación con el sistema operativo

Argumentos a favor de J2EE

- Esta siendo apoyada por toda la industria
- J2EE es una plataforma probada. .NET tiene el riesgo de cualquier tecnología de primera generación.
- J2EE es un modelo mas complicado, apropiado para desarrolladores entrenados que necesitan hacer objetos mas avanzados con mejor performance.
- J2EE es neutral en la plataforma, inclusive se puede utilizar Windows.
- Mejor integración con sistemas de legado a través de JCA.
- J2EE permite usar Java que es mejor lenguaje que C#.
- La mayoría de las consultoras informáticas deciden utilizar J2EE porque no pueden controlar la elección de plataforma de su cliente.

Otras tecnologías n-tier

Mono es una implementación de varias tecnologías:

- Un compilador para el lenguaje C# y Visual Basic.Net
- Un entorno de ejecución virtual: Un compilador JIT (Just-In-Time = justo-a-tiempo, esto es, que compila el código justo antes de ser ejecutado), un compilador AOT (AOT=ahead-of-time, antes-de-tiempo , esto es, que compila a código nativo un archivo y de esta forma no necesita la compilación JIT cada vez que se ejecute el programa), gestión automática de memoria, un interprete (mint), motor multiproceso.
- Una máquina virtual para los bytecodes del Lenguaje Intermedio Común (CLI)
- Una implementación de la librería de clases de .NET: manipulación XML, Remoting, Reflection.Emit, Xslt, etc.
- Librería de clases multiplataforma para el acceso a bases de datos: Postgress, MySQL, DB2, TDS, Sybase, Oracle, ODBC y Gnome-GDA
- Librería de clases UNIX: Mono.Posix
- Librería de clases GNOME: la familia Gtk#

En el mundo Microsoft, a este conjunto se le suele llamar la plataforma .NET en contraposición a .NET, que un término comercial no muy concreto. Cuando me refiero a la plataforma .NET me estoy refiriendo a estas tecnologias.

Existe gente a la que le puede parecer que todo esto es muy parecido a Java y la máquina virtual de Java. Tienen razón, esto es muy parecido a Java.

Pero el CLI (Lenguaje Intermedio Común, el equivalente de los bytecodes de Java) tiene una característica que no se encuentra en Java: la representación de éste es lo sufientemente potente como para servir para varios lenguajes: puedes mezclar lenguajes como C++, C, Fortran, Eiffel, Lisp, Java, C# y Visual Basic, en el mismo programa.