CAPITULO I

1.1. EVOLUCIÓN DEL DESARROLLO DEL SOFTWARE

Según la definición del IEEE, "software es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo"[1]. Según el mismo autor, "un producto de software es un producto diseñado para un usuario". En este contexto, la Ingeniería de Software es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software", es decir, "permite elaborar consistentemente productos correctos, utilizables y costo-efectivos"[2].

El software es un transformador de información, produciendo, gestionando, adquiriendo, modificando, mostrando o transmitiendo información. El rol del software ha sufrido un cambio sustancial a partir de la segunda mitad del siglo 20, unido al gran avance tecnológico que contribuyó al incremento en rendimiento del hardware, profundos cambios de arquitecturas informáticas, grandes aumentos de memoria y capacidad de almacenamiento, una gran variedad de dispositivos de entrada y salida han permitido que se desarrollen sistemas informáticos más sofisticados y más complejos.

El proceso de desarrollo de software "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo". Concretamente "define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo"[3].

El proceso de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se le

[1] Lewis G. 1994. "What is Software Engineering?" DataPro (4015). Feb. 1994. pp. 1-10.
[2] Cota A. 1994 "Ingeniería de Software". Soluciones Avanzadas. Julio de 1994. pp. 5-13.
[3] Jacobson, I. 1998. "Applying UML in The Unified Process" Presentación. Rational Software

llama el ciclo de vida del software que comprende cuatro grandes fases: concepción, elaboración, construcción y transición. La concepción define le alcance del proyecto y desarrolla un caso de negocio. La elaboración define un plan del proyecto, especifica las características y fundamenta la arquitectura. La construcción crea el producto y la transición transfiere el producto a los usuarios.

Los primeros años	La segunda era	La tercera era	La cuarta era
• Orientación por lotes • Distribución limitada • Software a medida	• Multiusuario • Tiempo real • Bases de datos • Producto de Software	• Sistemas distribuidos • Incorporación de inteligencia • Hardware de bajo costo • Impacto en el consumo	• Sistemas personales potentes • Tecnologías orientadas a objetos • Sistemas expertos • Redes neuronales artificiales • Computación en paralelo • Redes de computadoras

Anteriormente los lenguajes de programación se centraban en los caracteres. En estos lenguajes, la mayoría de ellos procedurales u orientados a objetos.

Los requerimientos de programación no eran los mismos que en la actualidad, los tiempos y los sistemas operativos han cambiado significativamente en el lapso indicado.

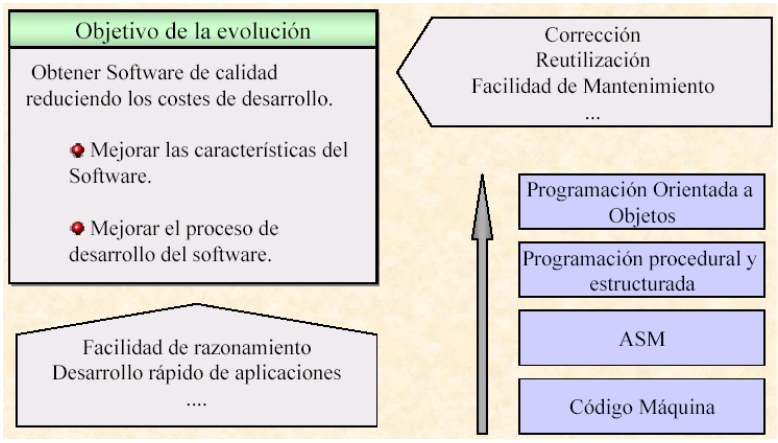


Figura 1.1 (Evolución del desarrollo del software)

Aunque en otro tipo de máquinas ya existían Sistemas Operativos que controlaban en realidad cada uno de estos aspectos, nunca llegaron a la popularidad de DOS, con todo y sus deficiencias.

Con la aparición de Windows 3.X se empezó a tener un mejor control de la máquina y, por ende, redujo significativamente la cantidad de código por escribir al respecto, pero aumentó de forma desproporcionada en cuanto al manejo de la interfaz. Esto es obvio dado que aún se estaban siguiendo las técnicas de programación utilizadas en DOS para programar en un ambiente gráfico.

La solución fue implantar un entorno de desarrollo visual para dominar el entorno gráfico que le rodeaba. Con este entorno también se dominarían las características multitareas que, como usuarios de PC. Y es que la diferencia es tangible: en las aplicaciones de DOS el programa va indicándole al usuario el orden de los pasos a seguir para completar una tarea. Cuando se tenía acceso a un menú, este solía ocupar toda la pantalla y sólo desde allí se podían tener opciones. De allí en fuera la secuencia del programa era inexorablemente seguida en el orden indicado por el programador.

La propuesta de los ambientes multitarea, aunque fuera cooperativa como en Windows 3.1x, es hacer que el programa se adecuara al usuario, es decir, exactamente lo contrario que en las aplicaciones anteriores. Para hacer esto, el programa ya no debía dictar el orden en que el usuario debiese capturar la información, sino que el usuario sería el que lo hiciera. De este modo, el paradigma de la programación tenía que cambiar.

El primero de los lenguajes que se adaptó a esta nueva filosofía fue Visual Basic en Windows. El sólo hecho de que con un cuadro de controles propios de Windows que con tan sólo arrastrarlos y pegarlos en un formulario ya se podía armar una interfaz, redujo considerablemente la cantidad de código por escribir.

Posteriormente, fueron apareciendo nuevos lenguajes de programación, como Delphi y Visual Objects, que aprovecharon esta misma ideología de desarrollo. Con la globalización de Windows y los ambientes gráficos, los lenguajes de

programación visuales han sido el mejor medio para explotar sus capacidades de la mejor manera.

A pesar de todo, los problemas a la hora de desarrollar software han persistido y continúan aumentando a través de la evolución de los sistemas basados en computadora.

1. Los avances del software continúan dejando atrás nuestra habilidad de construir software para alcanzar el potencial del hardware.
2. Nuestra habilidad de construir nuevos programas no pueden ir al ritmo de la demanda de nuevos programas, ni podemos construir programas lo suficientemente rápido como para cumplir las necesidades del mercado y de los negocios.
3. El uso extenso de computadoras ha hecho de la sociedad cada vez más dependiente de la operación fiable del software. Cuando el software falla, pueden ocurrir daños económicos enormes y ocasionar sufrimiento humano.
4. Luchamos por construir software informático que tenga fiabilidad y alta calidad.
5. Nuestra habilidad de soportar y mejorar los programas existentes se ve amenazada por diseños pobres y recursos inadecuados.[4]

1.2. MODELOS PARA EL DESARROLLO DEL SOFTWARE

Un modelo es una abstracción de algo que se extrae de la realidad, enfatiza las características más importantes omitiendo los detalles no esenciales, a este proceso es necesario dedicarle tiempo detenidamente ya que es la base para el desarrollo de un sistema.

[4] Pressman, Roger, S., "Ingeniería del Software. Un enfoque Práctico", McGraw-Hill, Cuarta edición, España, 1998, pp. 6

1.2.1 Modelos Genéricos de Desarrollo de Software

1. Desarrollo Evolutivo
2. Modelo de Cascada
3. Prototipado

Desarrollo Evolutivo

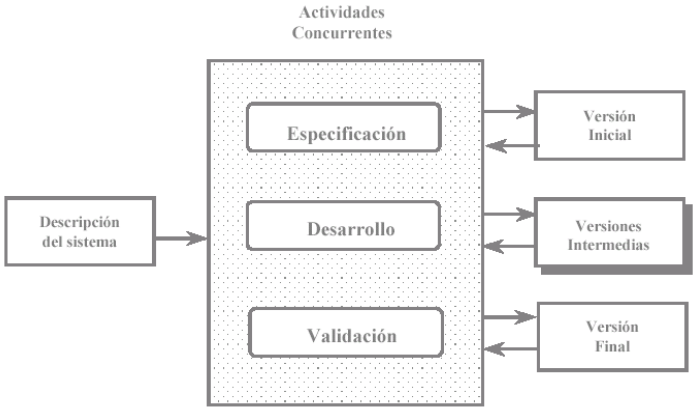


Figura 1.2.1 (Desarrollo Evolutivo)

Modelo de Cascada

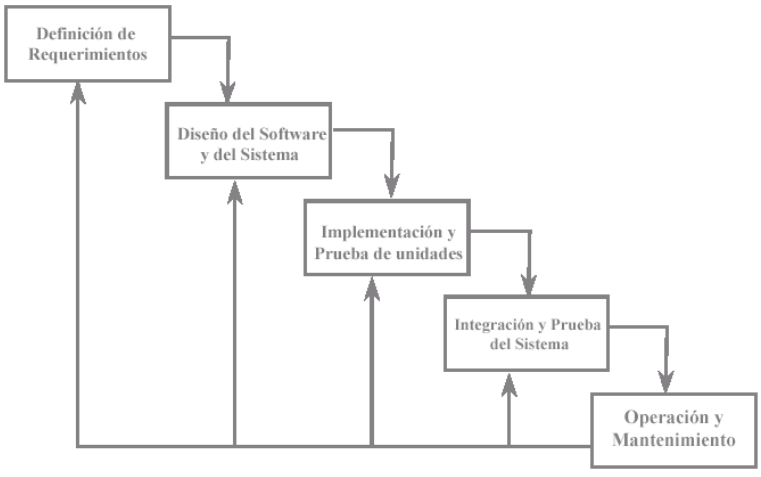


Figura 1.2.2 (Modelo de cascada)

Prototipado

- Se usa un prototipo para dar al usuario una idea concreta de lo que va a hacer el sistema.
- Se aplica cada vez más cuando la rapidez de desarrollo es esencial.
- Prototipado exploratorio: el prototipo inicial se refina progresivamente hasta convertirse en versión final.
- Prototipado de usar y tirar: de cada prototipo se extraen ideas buenas que se usan para hacer el siguiente, pero cada prototipo se crea entero.

1.2.2 PROBLEMAS Y RIESGOS CON LOS MODELOS

Cascada

- Alto riesgo en sistemas nuevos debido a problemas en las especificaciones y en el diseño.
- Bajo riesgo para desarrollos bien comprendidos utilizando tecnología conocida

Prototipado

- Bajo riesgo para nuevas aplicaciones debido a que las especificaciones y el diseño se llevan a cabo paso a paso.
- Alto riesgo debido a falta de visibilidad

Evolutivo

- Alto riesgo debido a la necesidad de tecnología avanzada y habilidades del grupo desarrollador.

1.3. MODELO CLIENTE SERVIDOR

Un proceso puede proporcionar unos servicios a los restantes procesos del sistema. Estos servicios serán operaciones de diverso tipo, por ejemplo imprimir un documento, leer o escribir una información, etc. En el modelo cliente/servidor, cuando un proceso desea un servicio que proporciona cierto proceso, le envía un

mensaje solicitando ese servicio: una petición. El proceso que cumple el servicio se llama servidor y el solicitante se llama cliente.

Este modelo consta de dos partes:

- Una parte que se ejecuta en un servidor
- La otra se ejecuta en el cliente o estaciones de trabajo

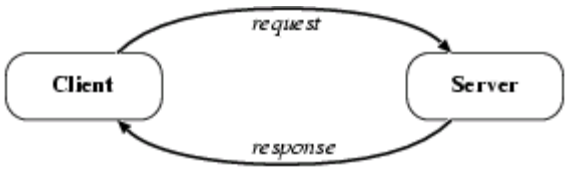


Figura 1.3 (Interacción cliente _ servidor)

El lado del servidor de la aplicación proporciona la seguridad, la tolerancia a fallas, el desempeño, la concurrencia y las copias de seguridad confiables. El lado del cliente proporciona la interfaz de usuario y puede contener informes, consultas y formularios vacíos. La idea consiste en tener lo mejor de ambos mundos aprovechando los dos y juntándolos.

Los procesos clientes y servidores han de seguir un protocolo de comunicaciones que defina: a) cómo se codifican las peticiones; y b) cómo se sincronizan entre sí los procesos.

Los clientes y servidores han de estar de acuerdo en cómo se escriben los mensajes: en qué orden van los posibles parámetros de la petición, cuántos bytes ocupan, etc.
La forma de sincronización nos dice si el cliente puede seguir adelante justo después de enviar la petición (no bloqueante), o por el contrario tiene que esperar a que el servidor le envíe una respuesta (bloqueante). Si la comunicación es no bloqueante, habrá que definir un mecanismo para que el cliente pueda saber si la respuesta del cliente está disponible. En esta práctica se adoptará una comunicación bloqueante: el cliente siempre esperará hasta recibir una respuesta del cliente.

El diálogo cliente/servidor es casi siempre bidireccional. Por un lado, el cliente envía información al servidor (el tipo de servicio solicitado más los parámetros); por otro, el servidor devuelve información al cliente (los resultados del servicio, códigos de error en caso de producirse, etc.)

Ventajas del Cliente:

- Facilidad de uso
- Maneja múltiples plataformas de hardware
- Maneja múltiples aplicaciones de software
- Familiar al usuario

Ventajas del Servidor:

- Confiable
- Concurrente
- Bloqueo sofisticado
- Tolerante a fallas
- Hardware de alto desempeño
- Control centralizado

En la computación cliente/servidor, cuando se ejecuta una consulta, el servidor examina la base de datos y envía al cliente sólo las filas que corresponden. Esto no sólo ahorra tráfico en el ancho de banda de la red, sino que puede ser más rápido que hacer que las estaciones de trabajo realicen la consulta, siempre que el servidor sea una máquina lo suficientemente poderosa.

La arquitectura cliente-servidor permite al usuario en una máquina, llamado el cliente, requerir algún tipo de servicio de una máquina a la que está unida, llamado el servidor, mediante una red como una LAN (Red de Área Local) o una WAN (Red de Área Mundial). Estos servicios pueden ser peticiones de datos de una base de datos, de información contenida en archivos o los archivos en sí mismos o peticiones de imprimir datos en una impresora asociada.

Aunque clientes y servidores suelen verse como máquinas separadas, pueden, de hecho, ser dos áreas separadas en la misma máquina. Por tanto, una única máquina Unix puede ser al mismo tiempo cliente y servidor. Además una máquina cliente unida a un servidor puede ser a su vez servidor de otro cliente y el servidor puede ser un cliente de otro servidor en la red. También es posible tener el cliente corriendo en un sistema operativo y el servidor en otro distinto.

Los clientes en una red cliente-servidor son las máquinas o procesos que piden información, recursos y servicios a un servidor unido. Estas peticiones pueden ser cosas como proporcionar datos de una base de datos, aplicaciones, partes de archivos o archivos completos a la máquina cliente. Los datos, aplicaciones o archivos pueden residir en un servidor y ser simplemente accedidos por el cliente o pueden ser copiados o movidos físicamente a la máquina cliente. Esta disposición permite a la máquina cliente ser relativamente pequeña. Para cada tipo de entorno de cliente, hay habitualmente software específico (y a veces hardware) en el cliente, con algún software y hardware análogo en el servidor.

Los servidores pueden ser sistemas operativos diferentes como Windows Server 2003, Windows 2000 Server, Windows NT, Windows 95, OS/2, Unix. Unix es popular porque como sistema operativo de servidores puede ser utilizado en muchos tipos de configuraciones sobre máquinas servidor además de como servidores de archivos y servidores de impresión.

Los servidores en una red cliente-servidor son los procesos que proporcionan información recursos y servicios a los clientes de la red. Cuando un cliente pide un recurso como, por ejemplo, un archivo, datos de una base de datos, acceso a aplicaciones remotas o impresión centralizada, el servidor proporciona estos recursos al cliente. Como se mencionó antes, los procesos del servidor pueden residir en una máquina que también actúa como cliente de otro servidor. Además de proporcionar este tipo de recursos, un servidor puede dar acceso a otras redes, actuando como un servidor de comunicaciones que conecta a otros servidores o mainframes o minicomputadoras que actúan como hosts de la red.

1.4. APLICACIÓN CLIENTE / SERVIDOR

Las primeras soluciones que incluían servicios de red vinieron con los denominados sistemas centralizados, en los que un solo computador con una o varias unidades centrales de proceso recibía las peticiones de los usuarios que se conectaban a él vía terminales tontas. Sin embargo, debido a razones tales como coste, confianza, falta de escalabilidad, falta de flexibilidad y la diversa naturaleza de las diferentes divisiones que conformaban una organización, hacían que este modelo no fuera especialmente atractivo.

El abaratamiento del hardware, la aparición en escena de los ordenadores personales (PC – Personal Computer), el aumento de la potencia de los PCs y de las estaciones de trabajo y el desarrollo de redes locales, provocaron una evolución en los modelos arquitectónicos de las aplicaciones software, pareciendo razonable repartir el proceso entre algunos sistemas más o menos distribuidos, en los que normalmente seguirían residiendo las bases de datos, y las máquinas desde las que se conectarían los usuarios, PCs o estaciones de trabajo donde se ejecutaría al menos la parte de interfaz de usuario.
Había surgido así el modelo Cliente/Servidor (C/S)[5], caracterizado por dividir la funcionalidad de la aplicación en dos papeles perfectamente definidos: cliente y servidor.

De modo abstracto, el servidor ofrece una serie de servicios que pueden ser utilizados por los clientes para completar la funcionalidad de la aplicación. Una interacción básica implica a un cliente que inicia una petición de algún servicio a un servidor. El servidor entonces realiza la función especificada por el cliente, devolviendo los posibles resultados que el servicio genera. En la práctica, los clientes y servidores se implementan como procesos que se están ejecutando en máquinas conectadas a una red.

Por aplicación Cliente-Servidor básica entendemos que varios clientes pueden acceder a información almacenada en las bases de datos del servidor sin que exista

[5] Bohnhoff, P., Jansen, R. y Martín, R. “Fundamentos Cliente/Servidor”. IBM. 1994.

intención de cooperar entre ellos. Esto es, la idea básica es que varios médicos pudieran acceder a información almacenada en bases de datos únicamente para hacer consultas rutinarias de información. En este caso no existe necesidad de establecer un protocolo de cooperación entre clientes.

La arquitectura C/S se organiza en niveles o capas, existiendo arquitecturas de dos, tres o en general n capas. En las arquitecturas de dos capas, existen diversas configuraciones, que van desde los clientes livianos y servidores gordos (thin clients & fat servers), donde la mayor carga se la llevan los servidores, siendo los clientes sólo responsables de la lógica de presentación, a los clientes gordos (fat clients) donde los servidores se ven relegados a simples repositorios de datos.

La arquitectura C/S más popular es la de tres capas. El cliente se encarga de mantener la interfaz de usuario. En un nivel intermedio se encarga de implementar la lógica de la aplicación. Finalmente, en el último nivel se encuentra la lógica de datos.

Los clientes se construyen sobre la base de unos servicios encapsulados en los procesos que implementan la lógica de la aplicación, siendo más robustos frente a cambios en esta lógica o en la lógica de datos. La funcionalidad que implementan los clientes es muy sencilla, pudiendo varios clientes reutilizar servicios estándares definidos en alguno de los niveles intermedios. La aplicación al estar dividida en partes más pequeñas, hace que el proceso de distribución de funcionalidad en los procesadores más adecuados sea mucho más flexible.

En una aplicación estándar, todos los procesos y operaciones del sistema se llevan cabo en la estación o computador donde se encuentra instalado, generándose una carga de trabajo que, dependiendo de la complejidad de dichos procesos, puede exigir una inversión de hardware excesiva en relación con el beneficio aportado por la aplicación cliente servidor.

En una aplicación Cliente/Servidor, al contrario de una aplicación Estándar, los procesos se reparten entre las estaciones de trabajo y uno o más computadores especiales, denominados Servidores de Aplicaciones quienes normalmente se

encargan del procesamiento y ordenamiento de la información, en beneficio de las estaciones.

Esta organización permite acelerar ciertas operaciones de tipo técnico (transacciones, bloqueos, actualizaciones, etc.) así como garantizar que cualquier inversión en el hardware de los Servidores de Aplicaciones sea de beneficio inmediato para el resto de las estaciones, con la consiguiente disminución de costos.

Recuerde que el hecho de tener una red y compartir recursos mediante uno o más servidores no implica necesariamente la existencia de una estructura Cliente/Servidor, ya que en la mayoría de los casos se tratará simplemente de Servidores de Archivos o Servidores de Impresoras, y esta es una distinción muy importante.

Tomemos por ejemplo una red local con un servidor Windows Server 2003 al cual se encuentran conectadas dos impresoras. El objetivo de este esquema es lograr que las estaciones de trabajo vean los discos e impresoras del servidor como si estuvieran conectados localmente a la estación, por lo que todo el trabajo recae en esta última, tal como si los dispositivos fueran locales. Estamos en presencia de un Servidor de Archivos y un Servidor de Impresoras.

Por otro lado, si se instala Microsoft SQL Server en el servidor Windows Server 2003, entonces sus estaciones podrán pedir al servidor que les suministre ciertos datos, ordenados de cierta manera, a lo que el servidor contestará luego de haber procesado localmente el requerimiento, sin intervención de la estación. Estamos en presencia de un Servidor de Aplicaciones.

1.5. ARQUITECTURAS

La arquitectura de una aplicación es la vista conceptual de la estructura de esta. Toda aplicación contiene código de presentación, código de procesamiento de

datos y código de almacenamiento de datos. La arquitectura de las aplicaciones difiere según como está distribuido este código.

Arquitectura de una Aplicación es un término usado al diseñar aplicaciones, particularmente del tipo Cliente-Servidor. Esta arquitectura se refiere a la manera en la que es diseñada tanto física como lógicamente.

En el diseño físico se especifica exactamente donde se encontrarán las piezas de la aplicación (Como discos, ejecutables, cable de red y computadoras).

En el diseño lógico o conceptual se especifica la estructura de la aplicación y sus componentes sin tomar en cuenta dónde se localizará el software, hardware e infraestructura. Tales conceptos incluyen el orden de procesamiento, mantenimiento y seguimiento comunes en sistemas organizacionales.

Muchas veces se toma demasiado en cuenta el diseño físico de una aplicación. Por añadidura los desarrolladores generalmente asumen, indebidamente, que el diseño lógico corresponde punto a punto con el diseño físico. Contrario a esto, un diseño adecuado debería permitir su implantación en varias plataformas y configuraciones. Como se puede ver, esta característica de portabilidad es un punto deseable para permitir que su aplicación sea flexible y escalable.

Hay muchos programadores para los que programar consiste en estar delante de un teclado escribiendo código, cualquier otra actividad es una pérdida de tiempo. Sin embargo, la experiencia ya viene demostrando como antes de empezar a escribir código es necesario previamente parar a pensar:

- Cuál es la mejor arquitectura para esa aplicación
- Cuál es la mejor herramienta para desarrollar lo que pide el cliente
- Cómo diseñamos la Base de Datos
- Cómo diseñamos las Clases Fundamentales

El análisis de la aplicación no sólo debe satisfacer las necesidades presentes sino que tiene que estar preparada para los posibles cambios que el cliente pueda pedir sin tener que reescribir totalmente la aplicación: la experiencia de aplicaciones similares y el conocimiento del mercado debe hacernos dejar las suficientes puertas abiertas como para poder implementar mejoras a la aplicación; esto es lo que se suele llamar flexibilidad.

Para hacer aplicaciones cliente/servidor que funcionen bien se debe tener en cuenta algunos aspectos como:

- Minimizar el tráfico de la red
- Procesar los datos en el lugar más rápido

La optimización de las velocidades de acceso es el gran problema de estas aplicaciones. En una implementación cliente/servidor, algunos datos se deben guardar en el cliente por razones de optimización. Esto ocurriría con algunas tablas estáticas que cambian poco.

- La posibilidad de integrar aplicaciones que accedan a las mismas bases de datos de una forma sencilla.
- Separar las reglas de negocio de los interfaces especialmente en entornos multiplataforma permite que las reglas se cambien con un mínimo impacto sobre los usuarios de las aplicaciones.
- El uso de modelos tres capas aumenta increíblemente la flexibilidad a la hora de aplicar las posibilidades de la informática para aspectos específicos de la problemática del cliente.
- Es fácil construir nuevas aplicaciones desde los componentes instalados si las reglas del negocio están en unos servidores de aplicaciones más que en cada aplicación.

1.6. ARQUITECTURA MONO-CAPA

Entendemos por aplicaciones mono-capa, aquellas que tanto la propia aplicación como los datos que maneja se encuentran en la misma máquina y son administradas por la misma herramienta: podríamos decir que son una sola entidad.

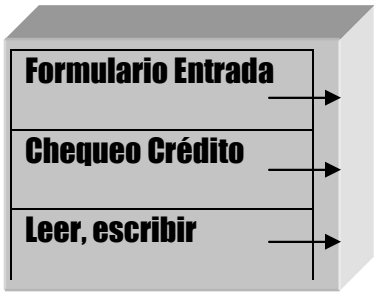


Figura 1.6 (Mono-capa)

Un ejemplo práctico de arquitectura mono-capa son los programas en lotes donde el código de presentación, proceso de datos y código de almacenaje se encuentra en un mismo lugar. Esta arquitectura fue utilizada por los lenguajes de tercera generación.

En vista de las múltiples falencias encontradas en esta arquitectura así como las caídas del sistema se optó por adoptar la arquitectura en dos capas, donde se pensó en separar la capa de aplicación de la capa de almacenaje de datos.

1.7. ARQUITECTURA MULTICAPA

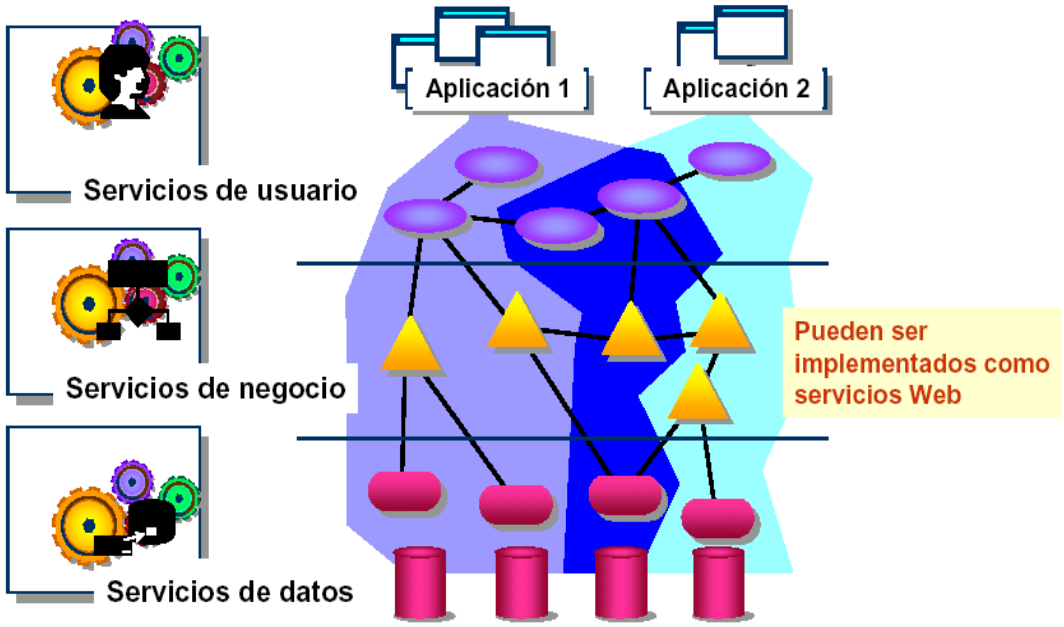


Figura 1.7 (Arquitectura multicapa)

Existen varias propuestas de arquitecturas en capas para el desarrollo de aplicaciones las cuales han venido evolucionando:

1) Arquitectura de dos capas;
2) Arquitectura de tres capas o multicapas

1.8. APLICACIONES CON ARQUITECTURA EN DOS CAPAS

Todavía existen varias aplicaciones que están basados en arquitecturas de dos capas, constituidas por dos niveles:

- Nivel de aplicación;
- Nivel de la base de datos.

Existen herramientas de amplio uso que presuponen esta estructura por ejemplo: Visual Basic, Access, SQL Server. Estas arquitecturas fueron las primeras en aprovecharse de la estructura cliente-servidor. Las desventajas de dos niveles son bien conocidas:

- El nivel de las aplicaciones se recargan, entremezclando aspectos típicos del manejo de la interfaz con las reglas del negocio;

- Las reglas del negocio quedan dispersas entre el nivel de aplicación y los procedimientos de almacenaje de la base de datos;

- La aplicación queda sobrecargada de información de bajo nivel si hay que extraer los datos de varias bases de datos, posiblemente con estructuras diferentes.

- El nivel de aplicación puede ser demasiado pesado para el cliente.

En la raíz de las aplicaciones cliente/servidor está la separación de la aplicación en componentes encapsulados u objetos. La ventaja de romper una aplicación en

partes es que cualquier cambio de uno de esos componentes no tiene un impacto directo sobre los otros o en el resto de la aplicación.

En las arquitecturas dos capas, la aplicación se divide en dos entidades separadas con el interfaz por un lado y las reglas de negocio junto con el Acceso a Bases de Datos:

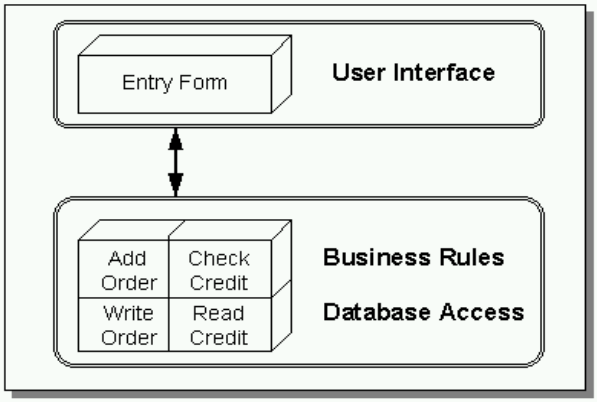


Figura 1.81 (Arquitectura dos capas, con el interfaz y las reglas de negocio encapsuladas juntas)

También se podría ubicar en el mismo lado el interfaz junto con las reglas de negocio y tendríamos lo que se muestra en el siguiente grafico:

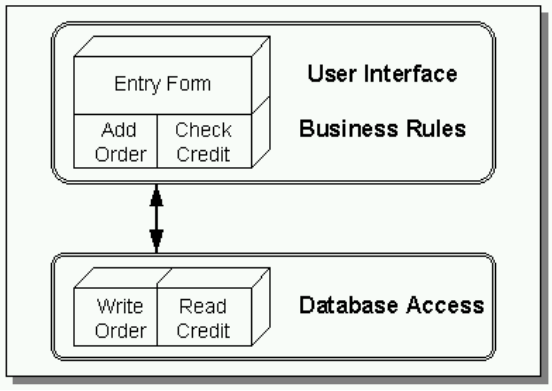


Figura 1.8.2 (Arquitectura en 2 capas, con el acceso a la BD y las reglas de negocio encapsuladas)

1.8.1 ARQUITECTURA CLIENTE/SERVIDOR

Capas en un sistema Cliente/Servidor

En un esquema Cliente/Servidor clásico existen dos capas, el cliente y el servidor: éste está ubicado normalmente en otra máquina, y suele ser un gestor de base de datos, como SQL Server, Oracle, aunque también puede ser una base de datos más pequeña, como Parados, base, etc., que accedemos directamente desde nuestra aplicación cliente.

Los mejores gestores de base de datos relacionales proporcionan soporte para implementar en ellos bastantes reglas de negocio, mediante el uso de claves primarias, integridad referencial, triggers, etc., mientras que sistemas como dBase y otros apenas proporcionan soporte para reglas de negocio.

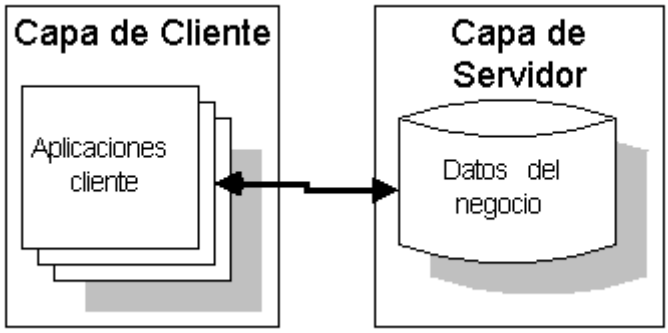


Figura 1.8.3 (Esquema de arquitectura Cliente/Servidor clásica.)

Suponiendo que tengamos la información en un gestor de bases de datos potente, podremos despreocuparnos de llevar a cabo la codificación de numerosas validaciones en nuestras aplicaciones: así, si en la base de datos creamos una regla de integridad referencial que indica que todo pedido pertenece a un cliente, el gestor de base de datos rechazará cualquier intento de almacenar un pedido en el que se nos haya olvidado indicar el mismo. Cualquier aplicación que acceda a esta base de datos se beneficiará de esta y otras validaciones automáticamente, sin tener que añadir ni una línea de código.

1.9. APLICACIONES CON ARQUITECTURA EN TRES CAPAS

Estas aplicaciones están constituidas por tres capas que son:

- **Capa de presentación o** interfaz al usuario, con las entradas de datos y las pantallas de consulta.
- **Capa de lógica de negocios** que sería el procesamiento de la información.
- **Capa de Datos**: El control del almacén de datos.

La necesidad de implementar reglas de negocio dentro de las aplicaciones cliente puede surgir también utilizando gestores de bases de datos más potentes. En primer lugar, las bases de datos relacionales son cada vez más potentes, pero no todas las reglas de negocio pueden reflejarse en ellas: por ejemplo, las reglas de flujo son bastante difíciles de implementar dentro de la base de datos, y suelen ser las aplicaciones cliente las que controlan que la información sigue una ruta válida a través del sistema.

Ya que parece que de cualquier modo seremos nosotros mismos los encargados de obligar a que se cumplan algunas reglas de negocio, puede ser conveniente encontrar la manera de centralizar la gestión de estas reglas en un único lugar, de modo que todo el código necesario no se haya de duplicar en cada una de las aplicaciones. La solución puede ser crear una aplicación que se encargue de llevar a cabo estas tareas, de modo que todos los clientes pidan o envíen información a la misma, no al gestor de base de datos en el servidor: a éste solo accederá la nueva aplicación, que conforma una nueva capa dentro de un sistema Cliente/Servidor, la capa intermedia, con lo que nuestro sistema ha pasado de ser un sistema Cliente/Servidor convencional a ser un sistema con tres capas.

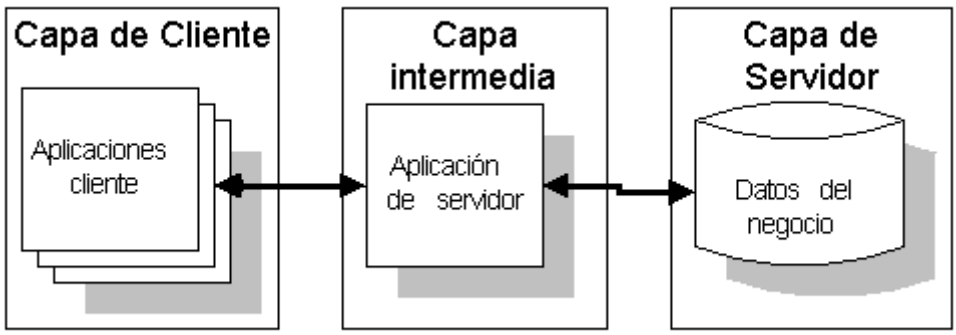


Figura 1.9 (Arquitectura Cliente/Servidor en tres capas)

Hemos hablado hasta ahora de la capa intermedia como si se tratara de una aplicación cualquiera, pero esto no es así. En los esquemas Cliente/Servidor tradicionales, de dos capas, suele ser el gestor de bases de datos el que proporciona la conectividad, así como capacidades tan fundamentales como el soporte de transacciones.

La introducción de una capa intermedia puede romper con esto, al ser necesario un modo de comunicar las aplicaciones cliente con la aplicación que lleva a cabo las labores de la capa intermedia, siendo ahora ésta la que se aprovecha de las capacidades de conectividad, el soporte de transacciones y las distintas capacidades que proporciona el gestor de base de datos. Solucionar todos los problemas de conectividad, etc.

El diseño físico comprende las siguientes tareas:

- Definir los componentes
- Refinar el empaquetamiento y distribución de componentes
- Especificar las interfases de los componentes
- Distribuir los componentes en la red
- Distribuir los repositorios físicos de datos
- Examinar la tolerancia a fallas y la recuperación de errores
- Validar el diseño físico

CAPITULO II

2.1 PLATAFORMAS Y APLICACIONES DISTRIBUIDAS

En la actualidad, dado que la tecnología de gestión de red se encuentra en un estado de madurez suficiente y es utilizada en la mayoría de entornos de red, y debido a su alto grado de flexibilidad, está surgiendo la necesidad de expandir la aplicación de estas probadas tecnologías a otros campos, como la gestión de servicios, aplicaciones y sistemas, existiendo múltiples productos en el mercado que proporcionan soluciones en estos ámbitos.

Desde el punto de vista de gestión, la aparición de entornos distribuidos supone el reto de conseguir una visión global del comportamiento de las aplicaciones a partir de la información de gestión que mantienen los diversos componentes que las forman. Es necesario, por tanto, uniformizar el acceso a dicha información, de forma que sea posible reaprovechar las implementaciones de servicios de gestión para las futuras aplicaciones que vayan apareciendo en estos entornos basados en plataformas de procesamiento distribuido orientadas a objetos.

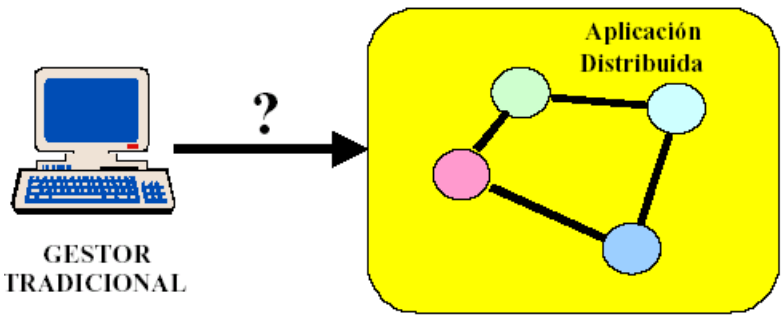


Figura 2.1 (Plataforma de procesamiento distribuido)

Por otro lado, el gran número de plataformas de gestión existentes exige que la gestión de estas aplicaciones se integre con las tecnologías tradicionales de gestión de red, aunque sea de una manera temporal hasta que maduren las nuevas tecnologías de gestión en desarrollo actualmente, tales como la gestión basada en Web.

Los sistemas distribuidos son el último paso en la computación C/S[6]. En vez de diferenciar entre las distintas partes de la aplicación, los sistemas distribuidos ofrecen toda la funcionalidad en forma de "objetos", con un significado similar al término "objeto" de la Programación Orientada a Objetos (POO).

Un sistema de programación distribuida basado en objetos, va a ofrecer, por tanto, las características de un sistema de programación orientado a objetos, además de las características de un sistema descentralizado o distribuido. Dichas características típicamente se resumen en los siguientes tópicos[7]: distribución, transparencia, integridad de datos, tolerancia a fallos, disponibilidad, capacidad de recuperación, autonomía de los objetos, concurrencia de programas, concurrencia de objetos y mejora en el rendimiento.

Normalmente, al ejecutar un programa desarrollado con un lenguaje orientado a objetos, el flujo de ejecución principal se encarga de crear y destruir los objetos dinámicamente. Sin embargo, al distribuir los objetos, estos se ejecutarán en procesos distintos (espacio de memoria disjuntos). El middleware tiene que encargarse de que las invocaciones a objetos remotos sean similares a las que hacen los objetos locales (transparencia de distribución), pasar los mensajes necesarios por la red, y devolver las respuestas de las operaciones al que hizo la llamada.

Los procesos que componen la aplicación, y que se están ejecutando en distintas máquinas de la red, son o pueden llegar a ser a la vez clientes y servidores, los cuales cooperan para conseguir la funcionalidad total de la aplicación. El mundo de los sistemas distribuidos es un mundo de "entidades pares", esto es, elementos de procesamiento con distintas disponibilidades de recursos, distinta capacidad de almacenamiento, que cooperan ofreciendo servicios en forma de objetos y requiriendo servicios de otros objetos implementados en otros nodos de la red.

Aunque la complejidad inherente de los sistemas distribuidos es muy alta, haciendo que los componentes distribuidos sean más difíciles de diseñar, depurar

[6] Lewis, T. G. "Software Cliente/Servidor". IEEE Computer, 28(4): 49-55. Abril, 1995.
[7] Chin, R. S. y Chanson, S. T. "Sistemas Distribuidos de Programación Basados en Objetos". ACM Computing Surveys, 23(1):91-124. Marzo, 1991.

y mantener que otras piezas de software, las ventajas que ofrecen son demasiado prometedoras como para ignorarlas, pudiendo destacar entre éstas[8],[9]:

- **Reducción potencial de costes:** Las redes de computadores incorporan tanto PCs como estaciones de trabajo ofrecen una mejor relación precio/rendimiento que los grandes ordenadores.
- **Capacidad para compartir recursos:** Es la propiedad por la que se permite a los elementos involucrados en el sistema utilizar los recursos de los otros.
- **Mejora en el rendimiento y la escalabilidad:** Cuando el número de clientes que accede a un recurso crece, el tiempo de respuesta comienza a caer. La naturaleza dispersa de los sistemas distribuidos ayuda a evitar esto porque los recursos se hallan físicamente en máquinas diferentes. Las peticiones por los recursos se envían a diferentes máquinas, haciendo que el proceso de la petición sea inherentemente distribuida. Además, el número de computadores en un sistema distribuido beneficia al sistema global en el sentido de que aporta mayor poder de procesamiento.
- **Autonomía local:** Un sistema distribuido es responsable de administrar sus recursos. Ofrece autonomía local a los nodos sobre sus propios recursos. Cada uno de ellos puede aplicar políticas, configuraciones o accesos de control locales sobre los recursos y los servicios.
- **Distribución intrínseca:** Algunas aplicaciones son inherentemente distribuidas.
- **Mejora de la confianza y la disponibilidad:** Una máquina de la red puede fallar sin afectar al resto del sistema.

Independientemente del balance que se pudiera hacer entre las ventajas de los sistemas distribuidos y sus problemas de complejidad de desarrollo, prueba y seguridad, se encuentra el contexto real de la demanda de aplicaciones software que hoy en día se tiene, donde la presencia de los servicios Web es cotidiana, ya sea a través de Internet o intranets, la heterogeneidad resulta evidente como suma de diversos factores (diversidad de plataformas hardware, software y de

[8] Ozsu, T. y Valduriez, P. “Principios de Sistemas de Bases de Datos Distribuidas”. Prentice-Hall. 1991.
[9] Tanenbaum, A. S. “Sistemas Operativos Modernos”. Prentice-Hall. 1992.

comunicaciones, inexistencia de soluciones únicas, sistemas legados...) y se camina cada vez más hacia la idea de sistemas abiertos.

El objetivo principal debe ser conseguir la interoperabilidad, esto es, la capacidad de los objetos de interoperar, independientemente de su ubicación en la red, del lenguaje de implementación y de la plataforma donde se ejecuten.

De forma muy general, para desarrollar aplicaciones para entornos distribuidos heterogéneos habría que tener en cuenta los siguientes aspectos:

- Buscar modelos y abstracciones independientes de plataforma que puedan ser útiles para resolver distintos tipos de problemas.
- Ocultar todo lo posible la complejidad de bajo nivel sin sacrificar demasiado la ejecución.

La utilización de modelos y abstracciones correctos puede proporcionar una capa de aplicación homogénea sobre el sistema distribuido. Esta capa oculta los detalles de bajo nivel y permite a los desarrolladores resolver problemas sin tener que preocuparse de la capa de red para las distintas plataformas.

El diseño de una aplicación distribuida implica la toma de decisiones sobre su arquitectura lógica y física, así como sobre la tecnología e infraestructura que se emplearán para implementar su funcionalidad. Para tomar estas decisiones, se debe tener un conocimiento claro de los procesos empresariales que realizará la aplicación (sus requisitos funcionales), así como los niveles de escalabilidad, disponibilidad, seguridad y mantenimiento necesarios (sus requisitos no funcionales, funcionales u operativos).

El objetivo consiste en diseñar una aplicación que:

- Solucione el problema empresarial para el que se diseña.
- Tenga en consideración la seguridad desde el principio, teniendo en cuenta los mecanismos adecuados de autenticación, la lógica de autorización y la comunicación segura.

- Proporcione un alto rendimiento y esté optimizada para operaciones frecuentes entre patrones de implementación.
- Esté disponible y sea resistente, capaz de implementarse en centros de datos de alta disponibilidad y redundantes.
- Permita la escalabilidad para cumplir las expectativas de la demanda y admita un gran número de actividades y usuarios con el mínimo uso de recursos.
- Se pueda administrar, permitiendo a los operadores implementar, supervisar y resolver los problemas de la aplicación en función del escenario.
- Se pueda mantener. Cada parte de funcionalidad debería tener una ubicación y diseño predecibles teniendo en cuenta distintos tamaños de aplicaciones, equipos con conjuntos de habilidades variadas y requisitos técnicos y cambios empresariales.
- Funcione en los distintos escenarios de aplicaciones y patrones de implementación.[10]

2.2 ARQUITECTURA WINDOWS DNA

La arquitectura Windows para aplicaciones distribuidas sobre Internet (Windows DNA) es un marco de trabajo para construir una nueva generación de soluciones de cómputo que incluyan los mundos de la computación personal e Internet.

Windows DNA es la primera arquitectura de aplicación que contiene e integra totalmente tanto los modelos Web de desarrollo de aplicaciones para cliente como para servidor.

Windows DNA incluye productos y servicios para ayudar a los desarrolladores a implementar los servicios de las aplicaciones de tres capas basadas en componentes. La arquitectura de tres capas es recomendada para construir aplicaciones distribuidas escalables.

[10] Microsoft Corporation. “Arquitectura de Aplicaciones .NET: Diseño de aplicaciones y servicios”. 2003.

WINDOWS DNA (DISTRIBUTED INTERNET APPLICATIONS)

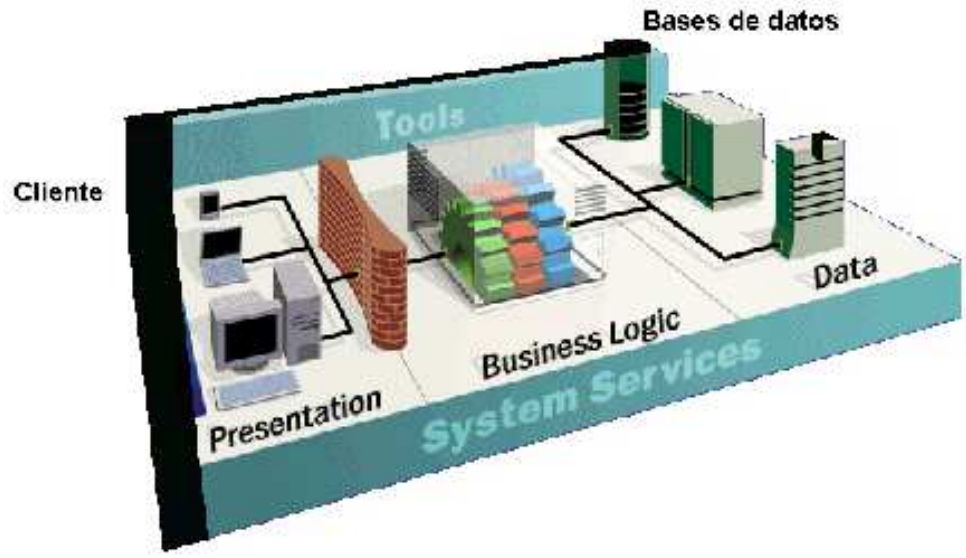


Figura 2.2. (Modelo Windows DNA)

Al utilizar el modelo Windows DNA, se construyen aplicaciones de negocios modernas, escalables, multi-capas, para ser ejecutadas sobre cualquier tipo de red. Las aplicaciones Windows DNA mejoran el flujo de información dentro y fuera de la organización, son dinámicas y flexibles al cambio en la medida en que cambian las necesidades del negocio, se integran fácilmente con los sistemas y datos existentes. Como las aplicaciones Windows DNA impulsan servicios de plataforma Windows profundamente integrados que trabajan juntos, las organizaciones pueden enfocarse a entregar soluciones de negocios en vez de ser integradoras de sistemas.

Una de las principales ventajas de Windows DNA es Internet, que ha cambiado dramáticamente el panorama de la computación. Años atrás, el proceso de desarrollo de aplicaciones ejecutado por una persona en una computadora era relativamente informal. En contraste, algunas de las aplicaciones más poderosas de nuestros días soportan miles de usuarios simultáneos, necesitan estar corriendo las 24 horas del día y deben ser accesibles desde una amplia variedad de dispositivos, desde computadoras portátiles hasta estaciones de trabajo de alto desempeño.

Para satisfacer estos requerimientos imperativos, los desarrolladores de aplicaciones necesitan adecuar herramientas de planificación y guías de cómo incorporar las tecnologías apropiadas.

2.2.1 Principios de Windows DNA

La arquitectura de Windows DNA está diseñada para maximizar en la aplicación lo siguiente:

- La autonomía
- La confiabilidad
- La disponibilidad
- La escalabilidad
- La interoperabilidad

Autonomía

La autonomía de la aplicación se refiere a la habilidad de una aplicación para gobernar sus recursos críticos. Los recursos críticos son los recursos preciosos requeridos por una aplicación para funcionar confiablemente como una entidad independiente. La autonomía de la aplicación es supuestamente uno de los aspectos más importantes del diseño de las aplicaciones para Windows DNA, y es también una de las diferencias más importantes entre los diseños cliente / servidor de dos y de tres niveles.

En un diseño cliente / servidor de dos niveles, los clientes tienen acceso directo a los recursos críticos de la aplicación y son libres de utilizar estos recursos cuando así lo deseen, por este motivo hay poco o nada que la aplicación pueda hacer para protegerse a sí misma del comportamiento inesperado o malintencionado, lo que compromete la estabilidad general de la aplicación. Por ejemplo, un solo cliente que no se comporte adecuadamente puede intencionadamente acabar con los recursos críticos de una aplicación en un intento para prevenir a otros clientes de

hacer su trabajo. Un ataque como éste puede convertir a las aplicaciones indefensas en inútiles.

Las aplicaciones de Windows DNA, por otro lado, al contar con tres niveles, nunca permiten a los clientes tener acceso directo a los recursos críticos. En lugar de ello, los clientes hacen peticiones de componentes especiales de su nivel de confianza llamados ejecutantes que realizan las operaciones de negocios para los que la aplicación fue diseñada.

Antes de que un ejecutante haga alguna operación de negocios en favor de un cliente, debe autentificar la identidad del cliente que hace la petición, validar que el cliente tiene la autorización apropiada para ejecutar la operación de negocios pedida e inspeccionar la petición del cliente para ver si tiene una apropiada sintaxis y para validar los datos.

Confiabilidad

La confiabilidad se refiere a la habilidad de una aplicación para proporcionar resultados exactos. Una aplicación no es confiable si regresa resultados inexactos. Sin embargo, asegurarse que los resultados sean exactos en un ambiente multiusuario es muy difícil. Para asegurar resultados exactos, se deben hacer las operaciones de negocios como parte de una transacción administrada por el Microsoft Transaction Server (MTS). Las transacciones aseguran que el estado de las transformaciones sean Atómicas, Consistentes, Aisladas y Durables (Atomic, Consistent, Isolated, Durable; ACID).

Disponibilidad

La disponibilidad se refiere a la cantidad de tiempo que una aplicación es capaz de dar servicio confiablemente a las peticiones del cliente, y es importante debido a que una aplicación solamente es útil cuando está disponible para dar servicio a las peticiones de los clientes.

Escalabilidad

La escalabilidad es la meta utópica del crecimiento lineal del rendimiento al agregar recursos adicionales, y es lo que le permite a una aplicación soportar desde 10 usuarios, hasta decenas de miles de usuarios, simplemente agregando o quitando recursos como sea necesario para "escalar" la aplicación.

Para incrementar la escalabilidad, los desarrolladores de aplicaciones de Windows DNA deben concentrarse en mantener los tiempos de adquisición de recursos y de uso de recursos tan bajos como sean posibles.

Interoperabilidad

La interoperabilidad se refiere a la habilidad de una aplicación para acceder a aplicaciones, los datos o los recursos en otras plataformas.

Muchos ambientes empresariales soportan diferentes tipos de hardware y sistemas de software que deben trabajar juntos para que la empresa sea exitosa, por lo cual es importante que las aplicaciones de Windows DNA sean interoperables. Para maximizar la interoperabilidad de las aplicaciones, las aplicaciones de Windows DNA deben apoyarse en recursos que proporciona Microsoft.

2.2.2 Modelo de Programación de Windows DNA

Windows DNA está basada en un modelo de programación denominado COM (Component Object Model). El uso del modelo COM se ha difundido ampliamente desde su introducción por Microsoft y por ser parte integral de muchas aplicaciones y tecnologías, incluyendo en Internet Explorer y todo el suite de aplicaciones Office.

Comunicación distribuida en Windows DNA

En Windows DNA, la decisión sobre la comunicación entre componentes ejecutándose en máquinas diferentes es sencilla: DCOM es prácticamente la única

opción. Desde un punto de vista puramente arquitectónico, DCOM es una simple extensión de COM. Sin embargo, en la práctica, el uso de DCOM tiene varias implicaciones. Entre ellas podemos destacar las siguientes:

- Cuando está correctamente configurado, DCOM es un protocolo muy seguro. Sin embargo, conseguir esta configuración no es una tarea fácil, y por ello el protocolo puede ser difícil de utilizar. No obstante, DCOM por sí mismo puede proporcionar buena autenticación distribuida, integridad de datos y privacidad de datos, especialmente en un dominio Windows 2000.
- Debido a que requiere puertos arbitrarios abiertos, DCOM no funciona bien a través de cortafuegos. Por consiguiente, las aplicaciones que deben comunicarse a través de Internet, generalmente no pueden utilizar DCOM para ello.

El Modelo de Diseño de Soluciones

El Modelo de Diseño de Soluciones ayuda al equipo del proyecto a anticiparse a las necesidades del usuario incluyéndolo en el problema. Vale destacar la diferencia entre cliente y usuario, cliente se considera a la persona que paga por el software y usuario es aquella persona que va a utilizar el software. Estos no son necesariamente la misma persona. Es importante conseguir los requerimientos de los usuarios si es que se quiere lograr que la solución este enfocada a la realidad del negocio.

Más allá de involucrar a los usuarios en el diseño, el Modelo de Diseño de Soluciones provee una estrategia para diseñar soluciones orientadas a negocios que deben ser creadas para satisfacer necesidades específicas. Este modelo une el Modelo de Equipo, el Modelo de Aplicación y el Modelo de Procesos, de tal manera que los recursos pueden ser enfocados en las áreas donde tengan mayor rendimiento.

Windows DNA es una arquitectura de tres capas, que algunas veces es llamada n-capas, trabaja sobre o hace énfasis en el uso de Internet.

2.3 ARQUITECTURA .NET FRAMEWORK

.NET es una nueva arquitectura tecnológica, desarrollada por Microsoft para la creación y distribución del software como un servicio. Esto quiere decir, que mediante las herramientas de desarrollo proporcionadas por esta nueva tecnología, los programadores podrán crear aplicaciones basadas en servicios para la Web.

Entorno cerrado y asociado a un sistema operativo, que incluye la infraestructura intermedia orientada a componentes (COM+), un entorno de ejecución común (Common Language Runtime, CLR) , un compilador JIT, y un conjunto de librerías de sistema (ensamblados).

La principal herramienta de desarrollo es Visual Studio .NET (utilizada para implementar la capa de presentación y la capa de negocio mediante una gran variedad de lenguajes y empaquetando en componentes COM+).

ARQUITECTURA DE APLICACIONES .NET

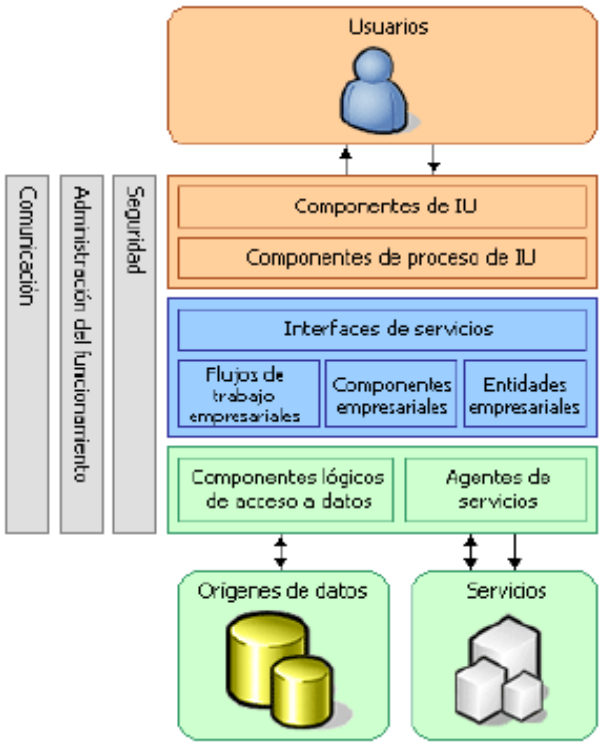


Figura 2.3. (Modelo: Net Framework)

La arquitectura .NET presenta distintos niveles de aplicaciones y componentes ofreciendo distintas capacidades y servicios apoyándose en los distintos sistemas operativos Windows y sistemas compatibles. De esta forma poder presentar a los usuarios incluidos dentro del nivel superior de la arquitectura los siguientes aspectos divididos en 3 capas:

1. La capa de presentación: formada por los Componentes de IU (Interfaz de Usuario), y los componentes de proceso de IU. Los componentes de IU son los cuales interactúan con el usuario (por ejemplo ventanas o páginas Web). Los componentes de proceso de IU distinto tipo de clases del tipo controladora en UML, lo cuales encapsulan la lógica de navegación y control de eventos de la interfase.

2. La capa de negocios: es la cual encapsula la lógica de negocios Las entidades empresariales representan objetos que van a ser manejados por la aplicación (modelo de objetos, xml, datasets con tipo, estructuras de datos), las cuales permiten representar objetos que han sido identificados durante el modelamiento, los componentes empresariales contienen lógica de negocios y en algunos casos pueden ser los objetos raíz que inician las transacciones.

3. La capa de acceso a datos: capa que contiene las clases que interactúan con las base de datos. Éstas surgen con la necesidad de mantener la cohesión o clases altamente especializadas que ayuden a reducir la dependencia entre las clases y capas. Aquí se encuentra la clase de métodos estáticos que permite uniformizar las operaciones de acceso a datos a través de un único conjunto de métodos, esta es la clase SQLHelper.

2.3.1 COMPONENTES DE .NET

Los componentes más importantes son:

- CLR (Common Language Runtime) compila en tiempo de ejecución el código de la aplicación
- El conjunto de clases del .NET Framework (acceso a la funcionalidad del sistema, interfaces)

- ASP.NET (dedicada al desarrollo Web)
- Remoting (invocación de objetos remotos)
- Windows Forms (desarrollo de plantillas graficas bajo un entorno de programación como Visual Basic .Net)

La estrategia .Net es innovadora respecto a los productos anteriores de Microsoft, en el sentido de que no compila aplicaciones en código nativo, es decir, no compila aplicaciones en código específico. La compilación, al igual que sucede con Java, se realiza en dos pasos sucesivos. El código escrito por el programador se compila en MSIL (Microsoft Intermediate Language), del mismo modo que las instrucciones en Java se convierten en bytecodes. Después el CLR (Common Language Runtime) de Microsoft compila en tiempo de ejecución las aplicaciones en código nativo de la plataforma. Esto se realiza mediante el compilador JIT (Just in Time).

El CLR también revisa el código, verificando la seguridad del mismo y recolectando los objetos para los cuales no existe ya ninguna referencia (recolección de basura), además de gestionar las excepciones entre otras tareas. En principio cualquier programa escrito en MSIL puede ejecutarse en cualquier sistema operativo donde funcione un CLR. Todo lo expuesto

Las características principales que conforman .NET son las siguientes:

- Servicios para .NET desarrollados por terceros fabricantes, que podrán ser utilizados por otras aplicaciones que se ejecuten en Internet.
- Esta plataforma utiliza los Servicios Web como un medio para poder comunicar distintas tecnologías.
- Permite conectar distintos sistemas operativos, dispositivos físicos, información y usuarios.
- La idea central de .NET es la de servicio. Más concretamente software como servicio
- y de cómo construir, instalar, consumir, integrar o agregar estos servicios para que puedan ser accedidos mediante Internet.

- La plataforma .NET utiliza Internet y su capacidad de distribución para que los usuarios accedan desde cualquier dispositivo, en cualquier sistema operativo y lugar a la funcionalidad que los servicios Web proveen.

Retos del desarrollo

- Integración de aplicaciones
- Múltiples lenguajes de programación
- Múltiples modelos de programación
- Complejidad del desarrollo y despliegue
- Seguridad no inherente
- Preservar la inversión del desarrollador
- Elevar la productividad del desarrollador

.NET Framework es un software que conecta información, personas, sistemas y dispositivos, a través de servidores, clientes, herramientas y servicios, además proporciona interoperabilidad basada en XML.

.NET Framework permite el desarrollo de aplicaciones a través del uso de un conjunto de herramientas y servicios que proporciona, y que pueden agruparse en tres bloques principales: el Entorno de Ejecución Común o Common Language Runtime (CLR); la jerarquía de clases básicas de la plataforma o .NET Framework Base Classes; y el motor de generación de interfaz de usuario, que permite crear interfaces para la Web o para el tradicional entorno Windows, así como servicios para ambos entornos operativos.

2.3.2 Comunicación distribuida en .NET Framework

El .NET Framework ofrece más posibilidades de elección para la comunicación entre las partes distribuidas de una aplicación que Windows DNA. Estas posibilidades incluyen:

- .NET Remoting, que proporciona un canal TCP y un canal HTTP;
- Soporte de ASP.NET para servicios Web XML invocados mediante SOAP e implementados en páginas .asmx;
- DCOM, para comunicar con objetos COM remotos.
- Un mayor número de opciones implica más posibilidades de elección para la arquitectura; también implica considerar más factores cuando se tome la decisión.

Los aspectos arquitectónicos que deben tenerse en cuenta cuando se crean aplicaciones distribuidas utilizando el .NET Framework incluyen:

- El canal TCP de .NET Remoting no proporciona seguridad de forma nativa. A diferencia de DCOM, no ofrece una sólida autenticación, integridad de datos ni servicios de privacidad. Sin embargo, el canal TCP es mucho más fácil de configurar que DCOM.
- A diferencia de DCOM, que no funciona bien con cortafuegos, el canal HTTP de .NET Remoting se ha diseñado explícitamente para comunicar eficazmente a través de Internet.. En general, el canal TCP es una excelente elección para la comunicación en una intranet, mientras que el canal HTTP o el soporte SOAP de ASP.NET es preferible para la comunicación a través de Internet.
- ASP.NET es más rápido que el canal HTTP de .NET Remoting. Además, el canal HTTP también tiene otras ventajas. Permite pasar parámetros por referencia y el retorno de llamadas, características que no forman parte por naturaleza del soporte SOAP de ASP.NET[11].

Servicios e integración de servicios

A medida que crece Internet y las tecnologías relacionadas, y las organizaciones buscan integrar sus sistemas entre límites de departamentos y de organización, ha evolucionado un enfoque de generación de soluciones basado en servicios. Desde el punto de vista del consumidor, los servicios son conceptualmente similares a los

[11] http://www.ayudadotnet.net/. "Desarrollo de aplicaciones distribuidas con .NET". Abril, 2003.

componentes tradicionales, salvo que los servicios encapsulan sus propios datos y no forman parte de la aplicación.

Finalmente podemos decir que .Net Framework es un entorno cerrado y asociado a un sistema operativo, que incluye la infraestructura intermedia orientada a componentes (COM+), un entorno de ejecución común (Common Language Runtime, CLR) , un compilador JIT, y un conjunto de librerías de sistema (ensamblados). La principal herramienta de desarrollo es Visual Studio .NET y lenguajes que soporta son: (Visual Basic, C++, C#, Visual J#, Fortran, Cobol, etc.)

2.4 ARQUITECTURA J2EE

J2EE (Java 2 Enterprise Edition) creada a inicios de los años 90, diseñadas por Sun Microsystems y secundada por otras empresas como IBM, BEA, ORACLE y muchas más, es un grupo de especificaciones o estándares, que permiten la creación de aplicaciones empresariales, esto sería: acceso a base de datos, utilización de directorios distribuidos, acceso a métodos remotos, funciones de correo electrónico, aplicaciones Web, etc. Aquí es importante notar que J2EE es solo una especificación, esto permite que diversos productos sean diseñados alrededor de estas especificaciones algunos son Tomcat y WebLogic; la especificación más reciente de Sun es J2EE 1.4, la cual está conformada por: JSP 2.0, Servlet 2.4, EJB 2.1 y Conector 1.5 entre otros API's[12].

La especificación de J2EE define su arquitectura basándose en los conceptos de capas, contenedores, componentes, servicios y las características de cada uno de éstos. Las aplicaciones J2EE son divididas en cuatro capas: la capa cliente, la capa Web, la capa negocio y la capa datos.

Por otra parte J2EE es una de las plataformas que cuenta con especial acogida para la construcción de aplicaciones Web multinivel ya que ofrece conceptos básicos de componentes, herramientas y ambientes de desarrollo que permiten

[12] http://java.sun.com/j2ee. “Detalles J2EE”. 2004.

generar programas o páginas dinámicas desplegables en los distintos browsers de la Web.

La una de las características de J2EE es la capacidad de desplazar el control de la interactividad de los servidores hacia las máquinas de los usuarios, permitiendo la utilización de distintos componentes que son interpretados por los navegadores Web y en los casos correspondientes por la máquina virtual de JAVA. Un ejemplo de esto es la carga de "applets", programas compactos y precompilados que generan animaciones y sonidos sobre páginas Web. Otra característica propia de J2EE es el acceso a herramientas de desarrollo de licencia libre que permite a los usuarios un fácil acceso a esta tecnología.

J2EE es una especificación que define una plataforma de desarrollo de aplicaciones distribuidas en n-capas.

J2EE (Java 2 Enterprise Edition)

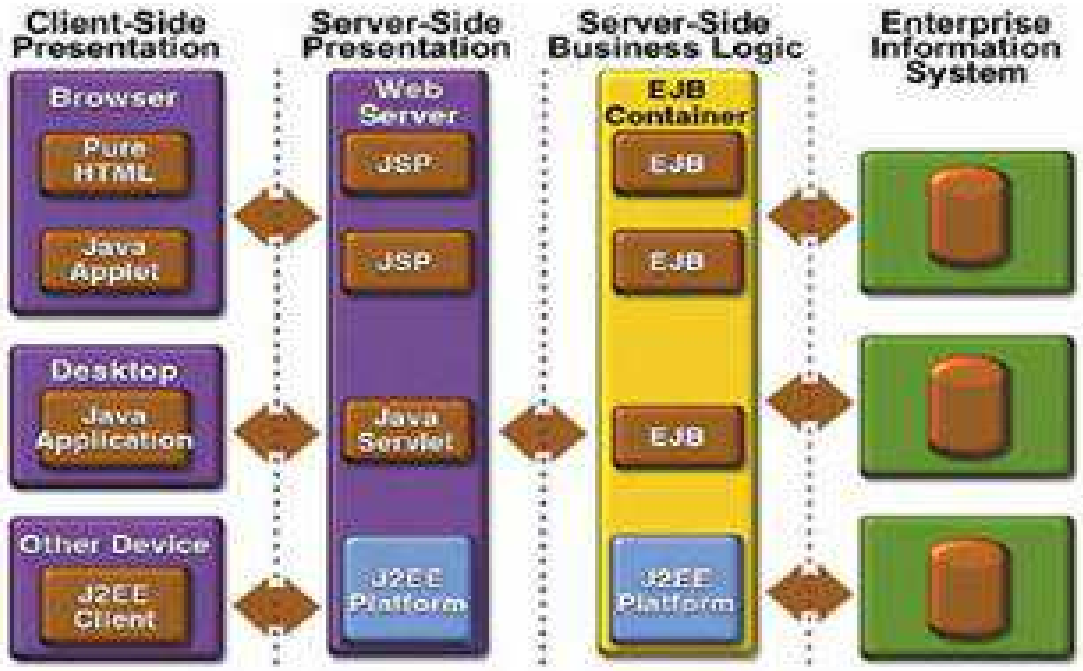


Figura 2.4 (Modelo J2EE)

Capa Cliente

Esta capa corresponde a lo que se encuentra en el computador del cliente. Es la interfaz gráfica del sistema y se encarga de interactuar con el usuario. J2EE tiene soporte para diferentes tipos de clientes incluyendo clientes HTML, applets Java y aplicaciones Java.

Capa Web

Se encuentra en el servidor Web y contiene la lógica de presentación que se utiliza para generar una respuesta al cliente. Recibe los datos del usuario desde la capa cliente y basado en éstos genera una respuesta apropiada a la solicitud. J2EE utiliza en esta capa las componentes *Java Servlets* y *JavaServer Pages* para crear los datos que se enviarán al cliente.

Capa Negocio

Se encuentra en el servidor de aplicaciones y contiene el núcleo de la lógica del negocio de la aplicación. Provee las interfaces necesarias para utilizar el servicio de componentes del negocio. Las componentes del negocio interactúan con la capa de datos y son típicamente implementadas como componentes EJB.

Capa Datos

Esta capa es responsable del sistema de información de la empresa o *Enterprise Information System (EIS)* que incluye bases de datos, sistema de procesamiento datos, sistemas *legados* y sistemas de planificación de recursos. Esta capa es el punto donde las aplicaciones J2EE se integran con otros sistemas no J2EE o con sistemas legados.

2.4.1. COMPONENTES DE J2EE

Cada componente de J2EE es una unidad de software independiente y funcional que cumple con las condiciones de interfaz definidas por la especificación de la componente y sólo tiene dependencias explícitas con su entorno de ejecución o

container. Una componente puede estar compuesta por una única clase, o lo más común, por un conjunto de clases, interfaces y recursos.

Las **componentes** principales en la plataforma J2EE son cuatro:

1. **Aplicaciones cliente**, son programas nativos escritos en Java que en general poseen su propia interfaz gráfica y que se ejecutan en un proceso independiente en un computador personal. Son ejecutados dentro del container de aplicación dado por el JRE y tienen acceso a todas las capacidades de la capa media J2EE.

2. **Applets**, son componentes que se ejecutan típicamente en un browser Web y proporcionan una interfaz web mejorada para aplicaciones J2EE. En general se ejecutan en un container de applets de un browser, pero pueden ejecutarse en una variedad de otras aplicaciones o dispositivos que proporcionen soporte para el container. Son utilizados como alternativa a interfaces más limitadas basadas en HTML.

3. **Java Servlets y JavaServer Pages**, son llamados colectivamente con el nombre de componentes Web. Se ejecutan en un servidor Web para responder a solicitudes HTTP desde clientes y pueden generar páginas HTML, que en general corresponde a la interfaz de usuario de una aplicación, o puede generar XML u otro formato de datos que será utilizado por otras componentes de la aplicación.

4. **Enterprise JavaBeans (EJB)**, son componentes que contienen la lógica del negocio para una aplicación J2EE. Se ejecutan en un ambiente distribuido y que soporta transacciones. Encapsulan el acceso al EIS a través de la utilización de objetos que proveen la funcionalidad de manejo de transacciones y persistencia.

Contenedores J2EE

Un contenedor (container) es un servicio que proporciona la infraestructura necesaria a una componente para ser ejecutada, para proveer sus servicios a un cliente y para dar comunicación con otras componentes.
Las componentes de una aplicación J2EE no interactúan directamente entre ellas, si no que deben utilizar los protocolos y métodos dados por el container para ese fin.

Un producto J2EE típico proveerá un container para cada tipo de componente de la aplicación: container de la aplicación cliente, container de applets, container de componentes Web y container de EJB.

Servicios J2EE

J2EE especifica los siguientes servicios estándares, junto con las APIs necesarias para la utilización por parte de cada componente. Algunos de estos servicios actualmente son provistos por J2EE.

1. HTTP y HTTPS: Protocolos estándares utilizados para comunicaciones Web y para comunicaciones seguras.

2. Una API (Interfaz de programación de aplicaciones) estándar para acceder a los recursos de una base de datos relacional de una forma independiente del proveedor. Esta API consta de dos partes, una interfaz para ser utilizada por las componentes y una interfaz de proveedores para definir drivers específicos. Oficialmente JDBC no es un acrónimo, aunque comúnmente se utiliza el nombre de Java Database Connectivity.[13]

3. JavaMail: Una API que permite crear aplicaciones Java para mensajería y envío de correo electrónico en forma independiente de la plataforma y del protocolo a utilizar.

[13] Marty Hall. “Core Servlets and JavaServer Pages”. Pág. 461. Pretince Hall, 2001.

4. Java Transaction API (JTA): Permite el manejo de transacciones. Las aplicaciones pueden utilizar JTA para iniciar, cerrar o abortar transacciones. Además permite al container comunicarse con monitores transaccionales y administradores de recursos.

5. Java Authentication and Authorization Service (JAAS): Proporciona una forma para la identificación de usuarios y su autorización para acceder a recursos de la aplicación. Implementa una versión en Java del estándar Plugable Authentication Module (PAM).

6. JDBC (Java Data Base Connection): Quizás el API de J2EE más conocida, permite el trabajo con base de datos permitiendo comandos SQL para la programación de métodos de acceso a distintas base de datos. Posee dos partes: una de nivel de aplicación usada por el componente que accede a los datos y además un servicio que provee una interfaz que se agrega al driver JDBC dentro de la plataforma J2EE.

Finalmente se puede decir que la plataforma J2EE es una plataforma conjunta, no exclusiva de Sun, ya que la apoyan grandes empresas del software (IBM, BEA, Oracle, soporta el único lenguaje de programación Java (lenguaje neutral, portable, robusto y estable)

2.5 ASPECTOS GENERALES

2.5.1 CAPA DE PRESENTACIÓN

Llamada también: Interfaz del usuario, Cliente, Front-End es la encargada de:

- Presentación de los datos
- Comunicación con el operador
- Incluye todos los componentes cuya responsabilidad primaria es recibir las entradas del usuario, o presentarle la información al mismo.
- Obtener información del usuario.

- Enviar la información del usuario a los servicios de negocios para su procesamiento.
- Recibir los resultados del procesamiento de los servicios de negocios.
- Presentar estos resultados al usuario.

La capa de presentación contiene la interfaz para entregar, presentar y reunir información. Constituye las pantallas, ventanas de diálogo, browsers o navegadores con los cuales interactúa el usuario.

Asegura los servicios de negocios.

Esto se refiere a los servicios negocios o validaciones al momento de ingreso de los datos dentro de la capa de presentación.

Los servicios de presentación generalmente son identificados con la interfaz de usuario

Normalmente residen en un programa ejecutable localizado en la estación de trabajo del usuario final. Aún así, existen oportunidades para identificar servicios que residen en componentes separados.

El cliente proporciona el contexto de presentación, generalmente un browser como Microsoft Internet Explorer o Netscape Navigator, que permite ver los datos remotos a través de una capa de presentación HTML, O también una aplicación Windows como ser los formularios de Visual Basic.

Mediante el uso de componentes, se separa la programación que da acceso a los datos en las bases de datos y aplicaciones desde el diseño y otros contenidos de la página Web.

Esto permite asegurar que los desarrolladores estén libres para enfocarse en escribir su lógica de negocios en componentes sin preocuparse acerca de cómo se muestra la salida. Recíprocamente, esto da libertad a los diseñadores de usar herramientas para modificar la interfaz

La decisión de rediseñar una interfaz deberá producir un impacto mínimo en la estructura general del sistema.

Esta separación permite elegir entre una variedad de pantallas o interfaz interactivas que hacen más atractivo interactuar al usuario con el computador. Por otro lado se conocerá el requerimiento del tipo de monitor que se empleará para el uso por parte del usuario.

Las ventanas de diálogo, o los presentadores de reportes, son los ejemplos clásicos de esta capa.

2.5.2 CAPA DE APLICACIÓN

La capa del medio trabaja como un intermediario, tomando los requerimientos del cliente para un conjunto particular de información formateado de acuerdo a un conjunto de reglas de negocios. La capa del medio consigue los datos, los transforma de acuerdo a las reglas de negocios y los devuelve al cliente. El cliente no tiene porque preocuparse acerca de las conexiones a la base de datos, tablas o transformación de los datos. El cliente simplemente solicita la información que necesita y la capa del medio se la devuelve.

Llamada también Servidor, Reglas de Negocio, Dominio de aplicación, capa intermedia, business rules es la encargada de:

- Recibir la entrada del nivel de presentación.
- Interactuar con los servicios de datos para ejecutar las operaciones de negocios para los que la aplicación fue diseñada a automatizar (por ejemplo, la preparación de impuestos por ingresos, el procesamiento de ordenes y así sucesivamente).
- Enviar el resultado procesado al nivel de presentación.

Los servicios de negocios son el "puente" entre un usuario y los servicios de datos.

Responden a peticiones del usuario (u otros servicios de negocios) para ejecutar una tarea de este tipo. Cumplen con esto aplicando procedimientos formales y reglas de negocio a los datos relevantes. Cuando los datos necesarios residen en un servidor de bases de datos, garantizan los servicios de datos indispensables para cumplir con la tarea de negocios o aplicar su regla. Esto aísla al usuario de la interacción directa con la base de datos.

La tarea de negocios es una operación definida por los requerimientos de la aplicación.

Como introducir una orden de compra o imprimir una lista de clientes. Las reglas de negocio (business rules) son políticas que controlan el flujo de las tareas.

Como las reglas de negocio tienden a cambiar más frecuentemente que las tareas específicas de negocios a las que dan soporte, son candidatos ideales para encapsularlas en componentes que están lógicamente separados de la lógica de la aplicación en sí.

2.5.3 TIPOS DE REGLAS DE NEGOCIO

Antes de seguir adelante con el estudio de la problemática que presenta la implementación de las reglas de negocio, vamos a establecer una clasificación de las mismas en varios grupos:

Primer grupo.

Engloba todas aquellas reglas que se encargan de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto es válida: no hay precios de artículos negativos, el sexo de una persona solo puede ser masculino o femenino, una fecha siempre debe ser una fecha válida (no existe el 30 de Febrero), estas reglas se llaman reglas del modelo de datos.

Segundo grupo.

Incluye todas aquellas reglas que controlan las relaciones entre los datos. Estas reglas especifican, por ejemplo, que todo pedido debe ser realizado por un cliente, y que el mismo debe estar dado de alta en nuestro sistema: además, una vez que un cliente haya hecho algún pedido, se deberá garantizar que no es posible eliminarlo, a menos que previamente se eliminen todos sus pedidos. Estas reglas constituyen las reglas de relación. Es frecuente que a partir de cierta información se pueda derivar otra: por ejemplo, el total de un pedido se puede calcular a partir de las distintas líneas que lo componen, mientras que el total de cada línea se puede calcular a partir del número de unidades vendidas y el precio por unidad.

Tercer grupo.

Es el compuesto por las reglas de restricción, que restringen los datos que el sistema puede contener. Nótese que este grupo de reglas se solapa en cierto modo con las reglas del modelo de datos, dado que aquellas también impiden la introducción de datos erróneos, como se vio anteriormente. La diferencia estriba en que las reglas de restricción restringen el valor de los atributos o propiedades de una entidad más allá de las restricciones básicas que sobre las mismas existen: por ejemplo, para un saldo existe una regla básica (regla del modelo de datos) que indica que éste debe ser un número, pero además puede haber una regla que indique que el saldo nunca puede ser menor que cierta cantidad tope establecida para cierto tipo de clientes. Esta sería lo que aquí denominamos una regla de restricción, y la diferencia fundamental estriba en el hecho de que este tipo de reglas requiere para su verificación del acceso a otros fragmentos de información, algo que no sucede con las reglas del modelo de datos.

Cuarto grupo.

Incluye aquellas reglas que determinan y limitan cómo fluye la información a través de un sistema. Por ejemplo, un cliente puede hacer una petición de análisis a un laboratorio, que anota un encargado: hecho esto, se genera un parte para uno o más analistas, estos realizan las mediciones correspondientes y devuelven los

partes con la información pertinente, a partir de la cuál se genera un informe de análisis, que será un análisis válido solo cuando sea firmado por los responsables de garantizar su corrección.

2.6 CAPA DE DATOS

Llamada también: repositorio, Base de Datos, Motor de datos es la encargada de:

- Almacenar los datos.
- Recuperar los datos.
- Mantener los datos.
- La integridad de los datos.

Los servicios de datos tienen una variedad de formas y tamaños

Los sistemas de administración de bases de datos relacionales (SABDs) como Microsoft SQL Server, servidores de correo electrónico como Microsoft Exchange Server y sistemas de archivos tales como el Sistema de Archivos NTFS.

La separación de la capa de aplicación del nivel donde se accede físicamente a la información permite una independencia absoluta de la aplicación con respecto a la tecnología o producto que permita el acceso a la información.

Se ha determinado que el uso exclusivo de SQL da un nivel de abstracción

Lo suficientemente alto para poder emplearlo directamente dentro de los servicios de negocios, sin embargo se busca un componente de acceso a los datos que permitirá una "conexión transparente" con cualquier proveedor de datos empleando cualquier tecnología, con el fin de independizar los componentes de los servicios de negocios de cualquier tecnología de acceso en particular.

Esta capa mantiene los datos guardados en un sistema de relacional de base de datos.

La ventaja de esta arquitectura es la independencia de una aplicación cliente. Pues el acceso se realiza a través del navegador.

La presentación de los datos es con HTML y la comunicación entre el cliente y el servidor se realiza a través del protocolo HTTP.

2.7 SERVICIOS WEB

Es un componente de software que puede ser accedido sobre el World Wide Web para el empleo en otras aplicaciones en ambientes distribuidos[14].
Un servicio Web consiste en una función disponible en un servidor conectado al Web que permite:

- Realizar un simple cálculo con unos datos que se le envían como parámetro,
- Acceder a una base de datos para recuperar un conjunto de registros,
- Validar la corrección de una información o contrastarla frente a otros datos, etc.
- El servicio Web podrá ser solicitado desde otro programa informático que se ejecute en un ordenador conectado al Web. Junto a la solicitud de la ejecución, se pueden enviar al ordenador que ofrece el servicio unos parámetros que el servicio Web remoto tomará como base para el cálculo o la función.

La aplicación que actúa como cliente debe conocer:

- La URL del servidor remoto que ofrece el servicio,
- El nombre del servicio que se solicita, y
- Los parámetros que se deben enviar junto con la llamada al servicio.

Estos datos se enviarán mediante HTTP. El servidor que ofrece el servicio Web leerá los parámetros que se le han enviado, llamará a un componente o programa

[14] Se define un sistema distribuido como aquel en el que los componentes están localizados en computadores en red, y comunican y coordinan sus acciones mediante el paso de mensajes. COLOURIS, George “Sistemas Distribuidos, Conceptos y Diseño”, Ed. Prentice Hall.

encargado de implementar el servicio, y los resultados que se obtengan de su ejecución serán devueltos al servidor que solicitó la ejecución del servicio.

Estándares para Servicios Web

Los servicios Web se definen a partir de las siguientes especificaciones:

- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery and Integration)

Figura 2.8 (Tabla básica de servicios Web)

En este modelo se identifican las acciones que afectan el funcionamiento de un servicio Web, así:

1. Un cliente, en este caso una aplicación Web, lanza una solicitud para conseguir una lista de los servicio Web que satisfagan las condiciones de búsqueda.
2. El cliente recibe información a partir de la cual puede determinar la petición que más se ajuste a su necesidad.
3. El cliente lanza la petición al servicio Web y recibe la respuesta.

Existen varias formas distintas de utilizar las tecnologías asociadas a los servicios Web XML (SOAP, Web Services Description Language (WSDL) y otros) para la generación de aplicaciones distribuidas. Algunos ejemplos incluyen:

- Conectar con el cliente Web de una aplicación en 'n' niveles utilizando SOAP en lugar de simplemente HTTP. Una vez realizado, ese cliente puede ser cualquier dispositivo capaz de invocar métodos SOAP. A continuación, el cliente puede proporcionar más funciones para el usuario, puesto que ahora dispone de una forma sencilla de invocar métodos en servidores remotos.
- Conectar una aplicación en 'n' niveles, por ejemplo, una generada sobre una plataforma basada en el .NET Framework, con otra generada sobre otra plataforma, como un servidor de aplicaciones Java.
- Conectar dos aplicaciones de un mainframe, o un sistema Enterprise Resource Planning (ERP) con otro, o cualquier otro tipo de aplicaciones. Como muestran estos ejemplos, los servicios Web XML pueden utilizarse en escenarios mucho más amplios que las aplicaciones en 'n' niveles.

Independientemente de cómo se utilicen, los servicios Web XML presentan muchos aspectos innovadores de arquitectura. Quizás la principal diferencia entre los servicios Web XML y las tecnologías de middleware tradicionales más habitualmente utilizadas por aplicaciones en 'n' niveles, es que los servicios Web XML proporcionan un acoplamiento débil. Sin embargo, esta frase tiene diferentes significados para diferentes personas. En este contexto, hace referencia a la comunicación entre aplicaciones, con las siguientes características:

- Las aplicaciones son en su mayoría independientes unas de otras, y frecuentemente están controladas por diferentes organizaciones.
- La fiabilidad no es absoluta. La disponibilidad de las aplicaciones que se comunican no está garantizada en todo momento.
- Sus interacciones pueden ser síncronas o asíncronas. Un cliente de servicios Web puede bloquearse esperando una respuesta a alguna petición, o puede seguir trabajando tras realizar la petición y comprobar más tarde si hay respuesta.

CAPITULO III

3.1 PARÁMETROS DE COMPARACIÓN

El desarrollo de aplicaciones empresariales, la colaboración tanto entre departamentos como entre empresas, y el desarrollo de aplicaciones Web han sufrido un auge muy importante durante los últimos años. Frente a esta nueva demanda surgen dos plataformas distintas para el desarrollo de este tipo de aplicaciones: J2EE de Sun y .NET de Microsoft sucesora de Windows DNA.

Se han considerado para el análisis como principales los siguientes parámetros:

Arquitectura: En el campo del software, la arquitectura es un término general que se aplica a la estructura de un sistema informático o de una parte del mismo, nos identifica los elementos más importantes de un sistema así como sus relaciones. Es decir nos da una visión global del sistema, es importante porque necesitamos de una arquitectura para entender el sistema, organizar su desarrollo, plantear la reutilización del software y hacerlo evolucionar.

Las arquitecturas software no responden únicamente a requisitos estructurales, sino que están relacionadas con aspectos de rendimiento, reutilización, restricciones económicas y tecnológicas, e incluso cuestiones estéticas.

Dentro de la arquitectura se van a considerar los siguientes elementos como: tipo de tecnología, empresas oferentes, sistemas operativos, librerías, intérprete, páginas dinámicas, componentes, acceso a bases de datos, servicios Web, interfaces gráficas, transacciones distribuidas, servicios de directorios, encolado de mensajes, lenguajes soportados, lenguaje intermedio, acceso a datos relacionales, acceso a datos jerárquicos, motor http, soporte XML.

Interoperabilidad: "La capacidad de comunicación, ejecución de programas, o transferencia de datos entre varias unidades funcionales de modo que el usuario

requiera tener poco o ningún conocimiento de las características individuales de aquellas unidades."—ISO/IEC 2382 Information Technology Vocabulary

Escalabilidad: La escalabilidad es la capacidad de un sistema de incrementar sus prestaciones en función del número de usuarios simultáneos que lo utilizan. Escalar hacia fuera por ejemplo es un medio de aumentar la capacidad añadiendo servidores adicionales.

Plataforma Móvil: También llamada Computación Móvil, se puede definir como la serie de artefactos y equipos portátiles, hardware, que hacen uso de la computación para lograr su funcionamiento, así, se tiene a las computadoras portátiles, los teléfonos celulares, los cuadernos de notas computarizados, las calculadoras de bolsillo, etc.

Herramientas de Desarrollo: Proporcionan la capacidad de "diseñar para las operaciones", es decir que la productividad de un desarrollador depende de las herramientas disponibles.

3.2 ANÁLISIS ARQUITECTURA WINDOWS DNA

La arquitectura Windows Distributed interNet Applications (Windows DNA) representa el enfoque de Microsoft para crear aplicaciones empresariales distribuidas. Las aplicaciones Windows DNA usan una arquitectura lógica de tres niveles basada en componentes. Los servicios del sistema de Microsoft proveen la infraestructura requerida por este tipo de aplicaciones, utilizando tecnologías como COM (Component Object Model) que provee el mecanismo básico para interacción entre componentes y MTS (Microsoft Transaction Server) que provee un ambiente de ejecución para construir aplicaciones escalables en el servidor.

Las aplicaciones Windows DNA (Distributed interNet Application), al basarse en una arquitectura de tres capas, no proporcionan acceso al usuario a las partes críticas de la aplicación, como son conexiones a bases de datos, transacciones, etc.

Es decir, los clientes no tienen acceso directo a los recursos. Para ello están los objetos de negocio, que sí que tienen acceso a estas partes críticas. De este modo, se consigue que la aplicación tenga una mayor estabilidad y autonomía.

Es necesario comprender la división funcional de las aplicaciones, y separar adecuadamente la funcionalidad, para aprovechar todas las características de distribución. De modo que se definirá a continuación los distintos esquemas funcionales.

El desarrollo habitual de aplicaciones se estructura como “monolítico”, esto es todo el acceso a datos, la lógica de ejecución y la presentación de la información están contenidos en un solo ejecutable, el cual debe instalarse en cada equipo cliente. El acceso a datos, si debe ser centralizado, produce un alto flujo en la red, dado que toda operación de selección de información se realiza en la máquina cliente.

Al realizar cambios en la lógica de la aplicación, es obligado reinstalar en cada máquina cliente la misma.

Este es el modelo de “lógica en el cliente” o 2 capas separa los datos en una ubicación única, en la cual se procesen las búsquedas, obteniéndose sólo el conjunto de datos requerido, resulta una solución adecuada para disminuir el flujo de red, permitiendo entonces una mejor respuesta. Esto es, utilizar un Servidor de Base de Datos.

Sin embargo, colocar toda la lógica en el cliente implica, necesariamente, ciertos requerimientos de hardware en cada estación de trabajo, y no soluciona el problema de mantenimiento y actualizaciones de la aplicación.

El modelo tres capas propuesto para el diseño de aplicaciones para Windows DNA, se basa en separar, en componentes bien definidos, las acciones referidas a los distintos servicios posibles.

Esta metodología permite “aislar” las operaciones entre sí, lo cual admite cambios en el tiempo, sin la necesidad de “rearmar” toda la aplicación completa.
Es necesario aislarse de la “forma” concentrándose en la “función” y en las características.

Interoperabilidad

La interoperabilidad se refiere a la habilidad de una aplicación para acceder a aplicaciones, los datos o los recursos en otras plataformas.

Muchos ambientes empresariales soportan diferentes tipos de hardware y sistemas de software que deben trabajar juntos para que la empresa sea exitosa, por lo cual es importante que las aplicaciones de Windows DNA sean interoperables. Para maximizar la interoperabilidad de las aplicaciones, las aplicaciones de Windows DNA deben apoyarse en recursos que proporciona Microsoft.

Las organizaciones quieren que las nuevas aplicaciones que construyan funcionen con sus aplicaciones existentes y extender esas aplicaciones con nueva funcionalidad. Requieren soluciones que trabajen con protocolos y estándares abiertos de tal forma que soluciones de otros proveedores puedan ser integradas también.

La interoperabilidad de las aplicaciones Windows DNA deben sustentarse en:

- Objetos de Datos ActiveX® de Microsoft (ADO) o accesos de datos universales con OLE DB.

- Extensible Markup Language (XML) para compartir datos con otras aplicaciones.

- DCOM para acceder aplicaciones en sistemas UNIX o en sistemas de almacenamiento en Virtual Múltiple (Multiple Virtual Storage, MVS).

Escalabilidad

La escalabilidad es la capacidad de mejorar recursos para ofrecer una mejora (idealmente) lineal en la capacidad de servicio. La característica clave de una aplicación es que la carga adicional sólo requiere recursos adicionales en lugar de una modificación extensiva de la aplicación en sí.

Aunque el rendimiento marca una diferencia a la hora de determinar el número de usuarios que puede admitir una aplicación, la escalabilidad y el rendimiento son dos entidades diferentes. De hecho, las labores de rendimiento pueden ser opuestas a veces a las de escalabilidad.

Para aumentar la escalabilidad, los desarrolladores de aplicaciones de Windows DNA deben centralizarse en conservar los tiempos de adquisición de recursos y de uso de los mismos tan bajos como sean posibles. El Microsoft Transaction Server permite compartir recursos entre usuarios reutilizando los ya existentes.

Plataforma Móvil

Windows DNA también permite su utilización para la computación móvil a través de productos como: CEfusion y ViaDB de Odyssey, software que amplía su infraestructura, haciéndola disponible a partir de aplicaciones para Pocket PC.

Odyssey Software es un proveedor de tecnología en el espacio de integración de sistemas móviles para la empresa. Sus tecnologías se integran de manera transparente con los componentes DNA centrales de Microsoft como fuentes de datos todos optimizados para procesos inalámbricos.

Herramientas de Desarrollo

Microsoft ofrece un conjunto de herramientas de desarrollo integradas, conocido como Visual Studio 6.0. Con ellas, es posible crear componentes que cumplan con la especificación COM (Component Object Model) y ADO, que es la que permite aprovechar toda la plataforma, integrándola con cada uno de los servicios

descritos. En el conjunto, usted puede seleccionar el lenguaje de programación tales como: Visual Basic, VisualFox, Visual C++, Visual J++, son los lenguajes.

3.3 ANÁLISIS ARQUITECTURA .NET FRAMEWORK

Microsoft es el único proveedor que ofrece una plataforma para desarrollo de aplicaciones empresariales que compite directamente con J2EE. Su producto es conocido como *.NET*, anteriormente llamado DNA, que es un modelo de desarrollo para aplicaciones empresariales para la plataforma Windows. Existen analogías directas entre .NET y J2EE, sin embargo, existen diferencias entre .NET y J2EE, siendo la diferencia principal la independencia del lenguaje de .NET v/s independencia del proveedor de J2EE.

Microsoft .NET es una plataforma independiente del lenguaje, así los desarrolladores pueden elegir el lenguaje y las herramientas a utilizar para implementar aplicaciones .NET. Entre los lenguajes a elegir se encuentran Visual Basic, Visual C++, Visual J++ y Visual C.

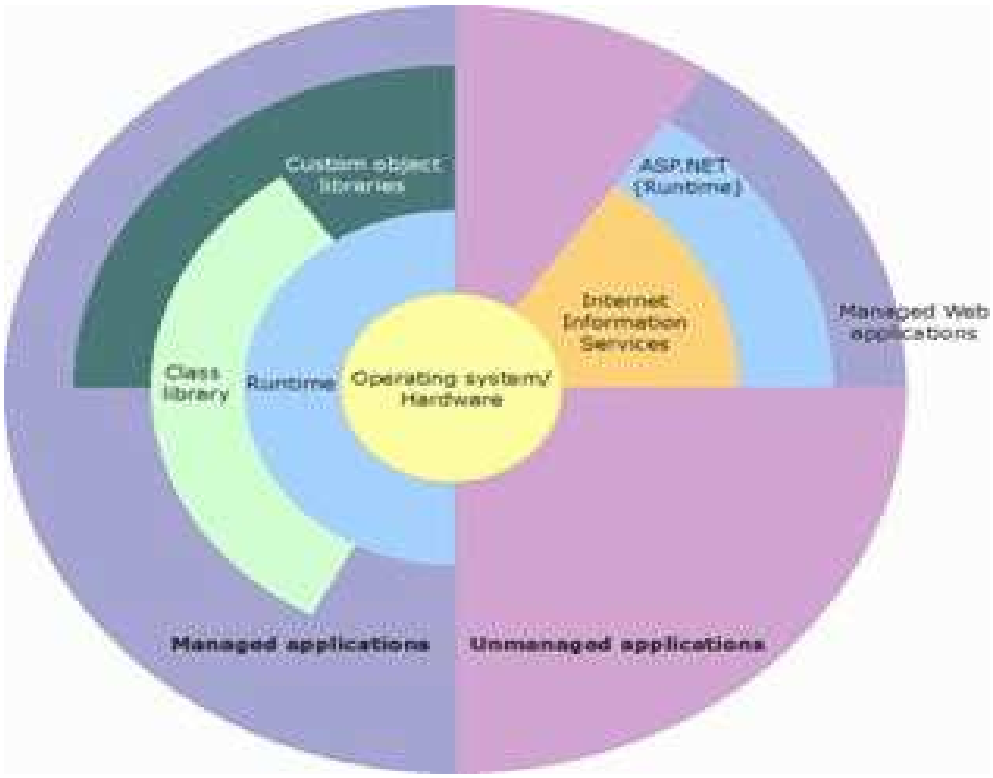


Figura 3.3 (.NET usa ASP.NET para aplicaciones Web)

Interoperabilidad

El Modelo de Objetos Componentes (COM) permite a un objeto exponer su funcionalidad a otros componentes y a aplicaciones host. La mayor parte del software actual incluye objetos COM. Aunque los ensamblados de .NET son la mejor opción para las aplicaciones nuevas, es posible que a veces se tenga que utilizar objetos COM.

.NET Framework es un avance natural desde COM puesto que los dos modelos comparten muchos temas esenciales, como la reutilización de componentes y la neutralidad del lenguaje. En lo que se refiere a compatibilidad con versiones anteriores, la interoperabilidad COM proporciona acceso a los componentes COM existentes sin que sea necesario modificar los componentes originales.

La interoperabilidad COM introduce también compatibilidad con versiones futuras al permitir que los clientes COM tengan acceso a código administrado con la misma facilidad con que tienen acceso a otros objetos

Un ensamblado es la principal unidad de creación de una aplicación de .NET Framework. Es un conjunto de características de funcionalidad que se genera, recibe un número de versión y se implanta como una sola unidad de implementación que contiene uno o más archivos. Cada ensamblado contiene un manifiesto de ensamblado.

Todas las aplicaciones .NET comparten un conjunto de tipos de datos comunes que permiten la interoperabilidad de objetos, independientemente del lenguaje de programación que se utilice. A veces, los parámetros de los objetos COM y los valores devueltos utilizan tipos de datos distintos de los que se utilizan en el código administrado.

Escalabilidad

La escalabilidad es la capacidad de mejorar recursos para ofrecer una mejora (idealmente) lineal en la capacidad de servicio. La característica clave de una

aplicación es que la carga adicional sólo requiere recursos adicionales en lugar de una modificación extensiva de la aplicación en sí.

Aunque el rendimiento marca una diferencia a la hora de determinar el número de usuarios que puede admitir una aplicación, la escalabilidad y el rendimiento son dos entidades diferentes. De hecho, las labores de rendimiento pueden ser opuestas a veces a las de escalabilidad.

La escalabilidad de una aplicación requiere una pertenencia equilibrada entre dos dominios distintos, software y hardware. Puede avanzar grandes pasos que aumenten la escalabilidad de un dominio sólo para sabotearlos cometiendo errores en el otro

Puesto que la escalabilidad no es un problema de diseño de las aplicaciones independientes, aquí se tratan las aplicaciones distribuidas. Las aplicaciones distribuidas están también un paso más allá de las tradicionales aplicaciones de cliente-servidor. Las aplicaciones distribuidas son aplicaciones que están diseñadas como aplicaciones de n niveles. La arquitectura de estas aplicaciones distribuidas favorece el diseño de aplicaciones escalables compartiendo recursos, como bases de datos y componentes empresariales.

Plataforma Móvil

Microsoft .NET es el software que conecta información, personas, sistemas y dispositivos.

.NET conecta una amplia gama de tecnologías personales de negocios (desde teléfonos móviles hasta servidores), permitiéndole acceder y utilizar información importante, donde y cuando quiera que se necesite. Framework .NET ha sido desarrollado con base en los estándares de servicios Web XML, permite que tanto las aplicaciones nuevas como existentes puedan conectarse con software y servicios de todo tipo de plataformas, aplicaciones y lenguajes de programación.

.NET se integra a través de toda la plataforma Microsoft proporcionando la capacidad de desarrollar, implementar, administrar y utilizar rápidamente soluciones conectadas y seguras con servicios Web XML. Estas soluciones permiten una integración más rápida y ágil del negocio, además de que cumplen con la promesa de ofrecer acceso a la información en cualquier momento y en cualquier lugar, a través de cualquier dispositivo.

El .NET Compact Framework es la plataforma tecnológica para aplicaciones móviles de Microsoft que aprovecha la potencia de los servicios Web XML en los dispositivos móviles. Microsoft. NET Compact Framework lleva esta visión a diferentes dispositivos, incluido el Pocket PC y otros basados en Windows CE .NET. Como subconjunto de .NET Framework, .NET Compact Framework comparte el mismo modelo de programación y las herramientas de desarrollo de aplicaciones, permitiendo a los desarrolladores .NET transferir sus conocimientos actuales para construir aplicaciones móviles, lo que sin duda acelerará la adopción de soluciones móviles.

El .NET Compact Framework es una pieza clave en la tecnología de Microsoft para aplicaciones móviles. Aporta las principales características para dispositivos electrónicos como un modelo de programación unificado con el .NET Framework tanto para PC como para servidor, soporte integral de servicios Web XML, acceso a bases de datos corporativas usando ADO.NET y XML, y avanzadas bibliotecas de clases que permitan a los desarrolladores construir potentes aplicaciones en poco tiempo. Las características del .Net Framework como la gestión de código y compilaciones just-in-time aseguran la fiabilidad, aplicaciones de alta calidad de realización para una mejor experiencia del usuario.

.NET Compact Framework se complementa con otras tecnologías como la versión 2.0 de SQL Server 2000 de la Edición CE de Windows de Microsoft, y Mobile Internet de toolkit Microsoft (MIT.). SQL Server para Windows CE es la, base de datos relacional compacta para un desarrollo rápido de aplicaciones que extiendan las posibilidades de datos empresariales a dispositivos inteligentes. El lanzamiento de SQL Server para Windows CE es la única base de datos móvil que se integra con .NET Compact Framework, permitiendo localizar datos, establecerlos y

extraerlos con seguridad para interactuar con aplicaciones back-end o servicios Web. El Mobile Internet Toolkit de Microsoft ofrece un amplio alcance para proyectos de aplicaciones a cualquier tipo de dispositivo móvil con un navegador. La experiencia de desarrollo para estas tecnologías está unificada gracias a Visual Studio .NET de Microsoft, que proporciona un conjunto consistente de herramientas e interfaces para construir aplicaciones usando las tecnologías móviles de Microsoft.

El modelo de programación e IDE a través de servidor, PCs y ahora dispositivos móviles asegura que los desarrolladores .NET pueden reutilizar sus actuales conocimientos a soluciones móviles. Los usuarios de móviles contarán con nuevas posibilidades en sus dispositivos gracias a aplicaciones electrónicas que aprovechan todas las posibilidades de los dispositivos que tienen que agregar datos y servicios tanto dentro como fuera de su empresa.

Herramientas de Desarrollo

Microsoft siempre se ha caracterizado por ofrecer a los desarrolladores algunos de los mejores entornos de desarrollo del mercado. En esta ocasión con Visual Studio.NET lo ha vuelto a conseguir ya que un único interfaz de desarrollo tenemos un editor de código multilenguaje, un compilador, editor de recursos, conexión a base de datos, editor XML, depurador, ayuda en línea. Con este IDE, la productividad del desarrollador se ve claramente aumentada.

Con Microsoft Visual Studio .NET 2003, Microsoft comenzó ya la estrategia DSI ofreciendo una amplia gama de mejoras para desarrolladores empresariales, incluida la selección de objetivos de Microsoft .NET Framework 1.1, compatibilidad mejorada para trabajar con servicios Web XML y la capacidad para implementar fácilmente aplicaciones .NET en tiempo de diseño para una facilidad de uso óptima.

Visual Studio® lanzó también el marco de trabajo de instrumentación de empresas (EIF, del inglés Enterprise Instrumentation Framework), que proporciona un

modelo de seguimiento y control de eventos unificado, así como un conjunto de servicios para mejorar el uso de aplicaciones distribuidas.

La compatibilidad con Microsoft .NET Framework permite el uso de la infraestructura de administración de Windows®, incluidas API WMI (Windows Management and Instrumentation), mantenimiento ampliado de un registro y un seguimiento, así como funcionalidad de medición del rendimiento integrada en Windows. Además, .NET Framework permite el uso de esa infraestructura para eventos personalizados, contadores del rendimiento e instrumentación para mantener un seguimiento de eventos empresariales.

Con Visual Studio .NET 2005 la próxima versión principal de las herramientas de desarrollo Visual Studio, Microsoft ofrecerá un diseñador de aplicaciones orientado a servicios que aprovecha las ventajas del modelo de definición de sistemas (SDM). Estas nuevas herramientas ayudarán a los arquitectos y desarrolladores de aplicaciones a crear aplicaciones orientadas a servicios y basadas en servicios Web, y a prepararlas para implementarlas en el centro de datos de destino.

Para los desarrolladores de aplicaciones, el diseñador ayuda a los arquitectos a convertir visualmente sus requisitos en aplicaciones y sistemas distribuidos basados en servicios. Los diseñadores permitirán la elaboración de modelos para describir estas aplicaciones y sistemas, junto con las correspondientes directivas de seguridad, protocolos y mucho más.

El diseñador de infraestructuras lógicas mitigará estas diferencias permitiendo a los administradores de operaciones especificar el entorno de desarrollo y, a los arquitectos, comprobar que la aplicación funcionará con las limitaciones de implementación especificadas.

3.4 ANÁLISIS ARQUITECTURA J2EE

Existen pocas alternativas que compitan directamente con J2EE, debido a que muchos proveedores en vez de crear productos que similares a J2EE, están escogiendo implementar una plataforma J2EE-compatible.

J2EE es una plataforma independiente de un proveedor, así una aplicación que cumple con los estándares de J2EE puede ser ejecutada en cualquier servidor de aplicaciones J2EE-compatible, sin embargo, tiene una limitación, las aplicaciones J2EE deben ser implementadas en Java.

J2EE es una plataforma más madura, se creó primero y su avance es muy rápido dado los aportes de muchas personas a través del mundo que apoyan el software Java. Por otra parte Microsoft .NET ofrece una solución integrada en la plataforma Windows, donde Microsoft hace el esfuerzo en asegurar que sus productos pueden trabajar juntos sin mayores problemas de configuración.

En resumen J2EE es:

- Conjunto de:
  - Modelo de programación de aplicaciones
  - Especificación de una plataforma – APIs y servicios (1600 páginas)
  - Implementación de referencia

Interoperabilidad

La interoperabilidad J2EE en Aplicaciones Business Siebel, versión 6.0. Los Conectores J2EE proveen una arquitectura estándar para aplicaciones empresariales para interoperar con la plataforma J2EE.

Java 2 Platform, Enterprise Edition (J2EE) versión 1.4. fue desarrollada sobre el legado de la tecnología Java que fue diseñada para ser portable e independiente de plataforma, Sun está trabajando a través del Java Community Process (JCP) para hacer de la tecnología J2EE el estándar para el desarrollo y la interoperabilidad de los servicios Web Java.

La tecnología J2EE proporciona tanto un modelo de programación para los servicios Web como un modelo de datos que permite a los desarrolladores desarrollar aplicaciones y servicios de principio a fin que pueden ser utilizados con cualquier dispositivo de Internet, en cualquier momento, desde cualquier lugar y por cualquiera. Como resultado, los desarrolladores pueden aplicar sus conocimientos de programación ampliamente en los entornos informáticos para desarrollar aplicaciones o servicios Web portables que permitan a los clientes ahorrar costes y aumentar ingresos trayendo sus activos de información a la red.

La interoperabilidad a través de la plataforma IBM WebSphere se lleva a cabo a nivel de las personas, la información, los procesos y las aplicaciones:

Personas -- La plataforma permite desarrollar contenidos de portal sobre su plataforma preferida. Esto protege las inversiones existentes y acelera el desarrollo de los contenidos. Más aún, la plataforma ha sido diseñada para dar soporte al denominado groupware y a los productos de colaboración tales como Lotus.

Información – Siempre se necesita un alto nivel de interoperabilidad para acceder y agregar información de manera consistente.

Procesos -- La interoperabilidad a nivel de procesos minimiza las conexiones de punto a punto, centraliza los conocimientos sobre la integración y arregla los procesos que abarcan las diversas tecnologías disponibles.

Aplicaciones -- El servidor da soporte a los servicios Web independientes de la plataforma utilizada, las aplicaciones de negocios basadas en la Web y los desarrollos de aplicación basados en estándares abiertos. También da soporte a estándares de tecnología tales como XML, SOAP, Web.

Escalabilidad

La escalabilidad es la capacidad de un sistema de incrementar sus prestaciones en función del número de usuarios simultáneos que lo utilizan. Tanto J2EE como

.NET ofrecen métodos de escalabilidad como la carga balanceada que permite a un cluster de servidores colaborar y dar un servicio de forma simultánea.

La ventaja de usar J2EE respecto a .NET en la escalabilidad es debida a que existe hardware disponible más potente en el entorno UNIX que en el entorno Windows, por lo que es necesario un menor número de máquinas para ofrecer el mismo rendimiento en las dos plataformas.

Plataforma Móvil

J2EE a través de J2ME, permite el Desarrollo Java para Dispositivos Móviles. La plataforma J2ME proporciona al desarrollador los medios necesarios para construir aplicaciones Java destinadas a ejecutarse en dispositivos con pocos recursos, principalmente teléfonos móviles.

La plataforma Java 2, edición Micro (J2ME), es la versión más pequeña de la plataforma Java, dirigida al mercado de consumo y dispositivos integrados, incluidos aparatos de tamaño muy reducido como las tarjetas inteligentes o los buscapersonas, pasando por sintonizadores de TV, hasta dispositivos de gran potencia como un equipo de sobremesa.

Las tecnologías J2ME contienen un entorno de tiempo de ejecución Java altamente optimizado especialmente desarrollado para el mercado de gran consumo. Las tecnologías J2ME abarcan una amplia gama de aparatos de tamaño muy reducido y permiten ejecutar programas de seguridad, conectividad y utilidades en tarjetas inteligentes, buscapersonas, sintonizadores de tv y otros pequeños electrodomésticos.

Herramientas de Desarrollo

Varias empresas ofrecen entornos de desarrollo para las aplicaciones de J2EE: Forte de Sun, Visual Café de WebGain, Visual Age for Java de IBM, JBuilder de Borland y muchas otras. Además, muchos de estos entornos de desarrollo están escritos en Java lo que significa que requieren muchos más recursos que la

aplicación equivalente escrita específicamente para un sistema operativo concreto, lo que a su vez, hace que el coste de los equipos necesarios para el desarrollo crezca.

Además, la Compañía también ha presentado nuevas características de desarrollo rápido de aplicaciones Web (RAD), que ayudarán a los desarrolladores a crear aplicaciones de colaboración para la arquitectura J2EE sin programación Java adicional. Con estos anuncios, IBM demuestra su compromiso con la comunidad de desarrolladores, las aplicaciones de clientes y la inversión en conocimientos de Lotus para su reutilización en entornos J2EE.

3.5 COMPARACIÓN DE PARÁMETROS

Como ya se mencionó anteriormente, al ser Windows DNA la arquitectura antecesora a Microsoft. NET el estudio se va a centrar en ésta última plataforma y J2EE.

Arquitectura

En cuanto se refiere a su arquitectura podemos observar que existen ciertas analogías así como también diferencias bien definidas.
Del análisis entre las dos plataformas se deduce que en este contexto J2EE cabe dentro de los siguientes elementos de la plataforma general:

ANALOGIAS

- Compilación Just-in-time; rápido a muy rápido
- Distribución vía Xcopy
- Separación de la lógica
- Página inteligente o cache de fragmentos de página
- Pueden generar y servir XML

.NET FRAMEWORK

- Librería de controles de servidor (Web Forms) con estado
- Servicios Web – muy facil
- Multi-lenguaje: C#, J#, C++, VB, COBOL
- Acceso a los recursos del sistema
- Altamente distribuido

J2EE

- Base bajo nivel, requiere más código
- Orientado a SQL
- Implementaciones varían
- Código más complejo
- No soporta datos desconectados
- Se cuestiona el modelo

Interoperabilidad

En la actualidad ambas plataformas tienen una buena interoperabilidad incluso entre ellas mismas, migración vs. Interoperabilidad.
Se requiere la interoperabilidad cuando la colaboración es entre capas y entre plataformas, Se requiere la migración cuando portamos la misma capa entre plataformas

Escalabilidad

Tanto J2EE como .NET ofrecen métodos de escalabilidad como la carga balanceada que permite a un cluster de servidores colaborar y dar un servicio de forma simultánea.

La ventaja de usar J2EE respecto a .NET en la escalabilidad es debida a que existe hardware disponible más potente en el entorno UNIX que en el entorno Windows, por lo que es necesario un menor número de máquinas para ofrecer el mismo rendimiento en las dos plataformas.

Por otra parte, Roger Sessions, de objectwatch.com, remarca que la plataforma .NET puede escalar desde 16.000 transacciones por minuto a más de 500.000 transacciones por minuto, mientras que IBM WebSphere usando tecnología J2EE/UNIX no puede conseguir nada mejor que pasar de 17.000 a 110.000 transacciones por minuto, con un costo monetario mucho mayor por transacción. Por lo tanto con .NET obtendríamos mayor posibilidad de escalado a un mejor precio.

Plataforma Móvil

En cuanto a plataforma móvil se refiere la más utilizada al momento es la de Microsoft, es decir .NET Compact Framework junto a Windows CE, sin embargo con la incorporación de J2ME se encontrarán cada vez más frecuentes programación móvil en esta plataforma.

Herramientas de Desarrollo

Microsoft soporta las herramientas de desarrollo más sólidas, aún más con la incorporación de Visual Studio .NET 2005.

Aunque muchas herramientas para J2EE ofrecen excelentes productos, ninguno llega al nivel de integración y la facilidad de uso de Visual Studio.NET. Además, muchos de estos entornos de desarrollo están escritos en Java lo que significa que requieren muchos más recursos que la aplicación equivalente escrita específicamente para un sistema operativo concreto, lo que a su vez, hace que el coste de los equipos necesarios para el desarrollo crezca.

Del estudio se deduce que Microsoft ofrece una ventaja con respecto a las herramientas que son de fácil adquisición lo que a su vez reduce los costos sobre J2EE/Linux como una plataforma de desarrollo para aplicaciones de tipo empresarial.

Las principales razones son:

1. El servidor de aplicaciones J2EE y el software de base de datos basado en Unix utilizados en el desarrollo e implantación Linux son mayores a los productos de Microsoft.

2. Las herramientas de Microsoft simplifican el desarrollo de aplicaciones de mejor modo que los productos J2EE/Linux, esta simplificación se traduce en costos menores por labores de desarrollo y administración de aplicaciones personalizadas y además un tiempo más rápido de este desarrollo.

3.7 RESULTADOS

Como conclusión del estudio realizado se pueden encontrar los siguientes resultados:

- Microsoft .NET es un conjunto de productos reales, mientras que J2EE es un conjunto de especificaciones.

- J2EE tiene el soporte de grandes empresas del software que han realizado su propia implementación del estándar: IBM, BEA, Oracle o la misma Sun, ofrecen sus plataformas de desarrollo de aplicaciones, en las que junto a J2EE

ofrecen otros productos como bases datos, caché, firewalls, etc., para dar una solución completa a sus clientes. Por su parte, Microsoft, y su gran equipo de marketing, están decididos a conseguir que .NET sea la plataforma de desarrollo de aplicaciones Business preferida. Microsoft dispone de un mejor soporte de herramientas.

- La arquitectura de lenguaje abierto de Microsoft .NET proporciona una alta flexibilidad.

- La interoperabilidad que expresa la capacidad de que las aplicaciones desarrolladas y ejecutadas sobre sus plataformas, puedan comunicarse con otras corriendo inclusive sobre otras plataformas, es buena para ambas arquitecturas, pero tomando en cuenta que Microsoft únicamente puede ejecutarse bajo Windows.

- El rendimiento es uno de los temas más controvertidos, las únicas referencias válidas que se ha encontrado es un documento publicado por Microsoft sobre la implementación en .Net de una aplicación de referencia de Java para el desarrollo de aplicaciones de comercio electrónico. Por supuesto, al ser una prueba desarrollada por Microsoft, no nos cabía la menor duda de que resultaría favorable a su plataforma, pero ha impresionado ver hasta que punto.

- En este documento también se discuten algunos aspectos de la prueba realizada y por qué debería considerarse como válida. En vista de que ninguno de los vendedores que distribuyen aplicaciones basadas en J2EE ha conseguido ofrecer una réplica a estos resultados creo conveniente considerar que el rendimiento conseguido en la plataforma .Net es mayor que al que se puede obtener en J2EE.

- J2EE nació al mercado tres años antes que .NET. En este tiempo, la plataforma J2EE ha sido puesta en funcionamiento en proyectos reales lo que ha otorgado la suficiente experiencia para mejorar y corregir errores de la misma. Por otra parte, .NET debido a que está menos tiempo disponible en el mercado, es una

tecnología menos probada y más propensa a sufrir errores. La opinión contrapuesta es que gracias a que .NET ha salido tres años después, esta plataforma es más moderna por lo que ha tenido la oportunidad de incluir avances y nuevas tecnologías que J2EE no ha incluido.

- El único lenguaje que soporta J2EE es Java y es el que se tendrá que utilizar para desarrollo de todos los componentes.

- Microsoft .NET ofrece soporte oficial para Visual Basic.NET, C++.NET y un nuevo lenguaje C# que es equivalente (con la excepción de portabilidad) a Java. Otros lenguajes desarrollados por terceros están disponibles como COBOL, Eiffel, Delphi, entre otros.

- Un análisis basado en los requisitos de arquitectura de empresa muestra que la tecnología de Microsoft tiene un costo de implementación cercano a una tercera parte en comparación con los productos IBM WebSphere a la hora de crear un sistema de introducción de pedidos basado en explorador.

- El análisis de proyectos reales de desarrollo de aplicaciones personalizadas señala las áreas en las que las herramientas de Microsoft suponen un ahorro de dinero: costos de producto más bajos, costos de mano de obra inferiores debidos a procesos de desarrollo simplificados y costos de mantenimiento menores. El ahorro por el uso de Windows fue de un 28,2% para grandes empresas.

3.7 RESUMEN GENERAL DE LOS RESULTADOS

Retomando el criterio de la página 64 en cuanto a la comparación de parámetros (DNA es la antecesora de .Net), procedo a generalizar detalles de:

SIMILITUDES ENTRE .NET Y J2EE

1.- El propósito tanto de J2EE como de la plataforma .NET es facilitar y simplificar el desarrollo de aplicaciones empresariales o corporativas. Las JSP (Java Server Pages) son muy similares a ASP (Active Server Pages) o a su descendiente ASP .Net, y los EJB (Enterprise JavaBeans) son muy similares a los COM/COM+ de Microsoft.
2.- Los servidores de aplicaciones J2EE y .Net proporcionan un modelo de acceso de componentes a datos y de lógica del Negocio, separados por una capa intermedia de presentación implementada mediante ASP .Net (.Net) ó Servlets (J2EE).
3.- Desde la perspectiva de los desarrolladores, J2EE y .Net proporcionan las herramientas para crear Servicios Web.
4.- Las dos plataformas hacen uso o trabajan sobre Internet
5.- Tal y como se ha expuesto J2EE y .Net son multiplataforma. Al usar .Net una compilación en dos pasos le permitirá teóricamente proporcionar entornos de ejecución para diferentes plataformas de forma similar a Java y sus JREs y SDKs.
6.- Pueden generar y servir XML
7.- En la actualidad ambas plataformas tienen una buena **interoperabilidad** incluso entre ellas mismas, migración vs. Interoperabilidad. Se requiere la interoperabilidad cuando la colaboración es entre capas y entre plataformas, Se requiere la migración cuando portamos la misma capa entre plataformas
8.- Tanto J2EE como .NET ofrecen métodos de **escalabilidad** como la carga balanceada que permite a un cluster de servidores colaborar y dar un servicio de forma simultánea.
9.- En cuanto a plataforma **móvil** se refiere la más utilizada al momento es la de Microsoft, es decir .NET Compact Framework junto a Windows CE, sin embargo con la incorporación de J2ME se encontrarán cada vez más frecuentes programación móvil en esta plataforma.

DIFERENCIAS ENTRE .NET Y J2EE

CARACTERISTICAS	.NET	J2EE
Tipo de tecnología	Producto	Estándar
Empresas que lo ofrecen	Microsoft	IBM, ORACLE; BEA, etc.
Librerías de desarrollo	Net Framework SDK	Java core API
Intérprete	CLR	JRE
Paginas dinámicas	ASP.NET	Servlet, JSP
Componentes	Net Framework SDK	JRE
Acceso a base de datos	ADO .Net	JDBC, SQL/J
Servicios Web	SOAP, WDSI, UDDI	SOAP, WDSI, UDDI
Herramientas programación	Visual Studio .Net	Depende del fabricante
Librería de mensajes	MSMQ	JMS
Lenguajes utilizados	Visual Basic, C#, C++,Java .Net	JAVA
Año de creación	2000	1990
Licencias	Comprarlas	Libre

REQUERIMIENTOS DE DE HARDWARE

J2EE

Hardware.	Funcionalidad Mínima.	Funcionalidad Recomendada.
Procesador	*Pentium III 500 MHz*	*Cualquiera Superior*
Memoria Ram	*256 MB*	*512 MB*
Espacio en Disco Duro	*350 MB*	*350 MB*

.NET

Hardware.	Funcionalidad Mínima.	Funcionalidad Recomendada.
Procesador	*Pentium II 450 MHz*	*Pentium III 733 MHz*
Memoria Ram	*128 MB*	*256 MB*
Espacio en Disco Duro	*3GB*	*3GB*

VENTAJAS Y DESVENTAJAS DE J2EE

VENTAJAS	DESVENTAJAS
Numero De empresas que apoyan a la plataforma.	Excesivo recursos para algunos software de desarrollo
Capacidad de migración de sistemas desarrollados	Java único lenguaje de programación
Acceso a datos para los distintos motores con la utilización de JDBC	Incompatibilidad J2EE dependiendo de la interpretación del fabricante
Capacidad de acceder a herramientas de licencia libre	Costos de licencia sobre productos de desarrollo de nivel superior
Proyección a futuro de ciertos proyectos de desarrollo de software	Aprendizaje difícil para usuarios iniciales
fundamentó un estricto modelo de seguridad	Propensa a virus por la cantidad de empresas que la secundan

Las implementaciones de J2EE pueden adquirirse a distintas compañías, mientras que .Net solo puede comprarse a Microsoft. El hecho de que haya distintas organizaciones implementando J2EE ofrece mayor variedad para los usuarios finales y permite la existencia de una cierta competencia entre ellas para obtener mejores productos que no existe en el caso de Microsoft y su .Net.
J2EE en base a estudios y entrevistas se ha llegado a la conclusión principal de su utilización por parte de muchos usuarios y desarrolladores, se marco el hecho de: “La disponibilidad de libre licencia”
Las aplicaciones Java pueden correr en una amplia gama de sistemas operativos (desde sistemas empresariales como Windows 2000, OS/390, Solaris, HP-UX, IRIX u otras versiones de Unix hasta en sistemas orientados más a ordenadores personales como Mac OS, Windows 9x ó Linux.
La tecnología Java es una tecnología abierta (en el sentido de que el código de la plataforma completa puede ser obtenido, revisado y estudiado por cualquiera que esté interesado

VENTAJAS Y DESVENTAJAS DE .NET

Capacidad de elección del lenguaje en el que se quiera programar	Costos elevados para adquirir
Proyección a futuro de las distintas herramientas presentadas por .Net	Inexistencia de productos de licencia libre para el desarrollo educacional
Desarrollo de servicios Web	Orientación única sobre sistema operativo Windows
Los requerimientos mínimos y consumo de recursos por las aplicaciones creadas bajo este estándar	Instalación excesiva de la herramienta de visual Studio .Net

Explotación en la red de las paginas dinámicas ASPx
Un lenguaje moderno y totalmente orientado a Objetos
Las herramientas de J2EE provienen de varios proveedores y no interactúan tan bien como las de .Net que son todas de Microsoft
.Net posibilita que programadores iniciales incursionen en esta plataforma reduciendo el tiempo de aprendizaje y entrenamiento.
Las herramientas de desarrollo incluidas por Microsoft en su Visual Studio .Net son mucho más simples, intuitivas y sencillas de manejar que las herramientas de desarrollo equivalentes en J2EE suministradas por otras empresas.
Net va por delante con respecto a J2EE con respecto a servicios Web y estos servicios son propios de la plataforma, aunque J2EE respondió ya con el lanzamiento del Java Web Services Developer Pack. De todos modos, la facilidad, rapidez y sencillez con la que se pueden construir servicios Web con el Asistente de servicios Web de Visual Studio .Net son muy superiores a las de las herramientas para construir servicios Web dentro del entorno de J2EE
.NET presenta menos líneas de código que J2EE. Se debe considerar si es el resultado es el mismo con menos líneas de código, si tiene la misma funcionalidad y simplicidad. Si esto se concretara para ambas plataformas y una presentara menos líneas de código, podría considerarse como un avance.
La información de .NET en el buscador Web más utilizado, Google. aporta mayor numero de paginas en español con relación a J2EE.

NET va más allá de soportar estos lenguajes, sino que también ofrece plena interoperabilidad entre ellos, por lo que es posible construir un componente en un lenguaje, introducirlo en una aplicación escrita en otro distinto e incluso heredarlo y añadir nuevas características en un tercero. Por ejemplo: un componente programado en C++ puede incluirse en una aplicación realizada en C#, y además es posible crear un componente en Cobol que herede del primero (hecho en C++) e incrustarlo también en la aplicación C#.

En conclusión:
.Net en la actualidad tiene una gran ventaja que es su magnifico entorno de desarrollo Visual Studio .Net, Trabajando con J2EE tenemos una variedad mucho mayor de entornos pero no con tanta productividad como la alternativa de Microsoft.

Por otro lado J2EE cuenta con un paquete de instalaciones mucho mayor pero que en breve puede verse seriamente afectado por .Net, finalmente no se puede dejar de lado el cambio de posición de los desarrolladores, ya nadie dice que .Net no funciona o no vale, pero tampoco se dice que J2EE sea mala.
Ahora simplemente las dos plataformas tendrán que luchar por el mercado ya no tanto con armas tecnológicas sino comerciales.

En cuanto a los desarrolladores principiantes y novatos una de las mejores opciones para desarrollar aplicaciones con entorno Web es sin duda .Net por su interfaz muy amigable y entrenamiento más rápido en relación a J2EE que se requiere de conocimientos ya mas elevados.

CAPITULO IV

DISEÑO DE LA APLICACIÓN PROTOTIPO

4.1 ALCANCE Y DELIMITACION

Específicamente este prototipo esta diseñado en tres capas (Interfaz de usuario, reglas de negocios y capa de datos)

Esta aplicación hace referencia a un portal Web que tiene como finalidad mostrar la información comercial o actividad a la que se dedica la empresa Molinos Poultier, permitiendo una comunicación interactiva con el usuario.

4.2 HERRAMIEMTAS A UTILIZAR

Para la interfaz del usuario se requiere:

- Macromedia Flash (animaciones)
- Paint para crear los banner
- Power point para diseñar los botones

Para crearla Base de Datos:

- SQL Server 7.0

En la base de datos se almacena las cuentas de usuario, las características de los productos, las provincias, la información o datos de la encuesta, los mismos que estarán dados de acuerdo a los requerimientos de los departamentos de marketing, ventas, control de calidad y gerencia de la Empresa Molinos Poultier.

- Servidor Web
- Microsoft Deveplopment Environment 2002 version 7.0
- Microsoft .Net Framework
- Microsoft Visual Basic. Net (formularios)

4.3 METAS DEL PROTOTIPO

La creación de esta aplicación prototipo para la empresa Molinos Poultier tiene como objeto:

- Mejorar el servicio y atención al cliente
- Mayor difusión comercial de la empresa
- Captar mayor numero de clientes dentro y fuera de la provincia
- Elevar el nivel empresarial

4.4 REQUERIMIENTOS DEL SISTEMA

Para la creación de esta aplicación se necesita:

Cliente

- Procesador Pentium de 900 GHz
- Memoria RAM 128 MB

Servidor

- Procesador Pentium 4 de 2.4 GHz
- Memoria 256 MB o más

Para cliente y servidor

- Disco duro 20 GB (espacio disponible en la unidad de instalación) 2GB
- Monitor, resolución Super VGA (1024x768) o superior con 256 colores
- Mouse multimedia o compatible

Software

- Sistema operativo Microsoft Windows Me, NT o XP
- Plataforma :net Framework
- Internet Information Server

4.5 MODELADO DE LA BASE DE DATOS

La base de datos constituye la estructura de almacenamiento de cada una de las tablas con sus respectivos privilegios, cuya implementación demanda de una alta eficiencia para que el manejo de los datos sea óptimo y apropiado.

A continuación un recorte del código y modelamiento de la base de datos desde SQL Server:

CIUDADES

```
CREATE TABLE [dbo].[Ciudades] (
  [Codciu] [char] (4) NOT NULL ,
  [Codpro] [char] (4) NOT NULL ,
  [Nomciu] [varchar] (50) NULL
) ON [PRIMARY]
GO
```

PRODUCTOS

```
CREATE TABLE [dbo].[Productos] (
  [Codprodu] [varchar] (10) NOT NULL ,
  [Nombre] [varchar] (50) NULL ,
  [Presentacion] [varchar] (50) NULL ,
  [Unidades] [varchar] (10) NULL ,
  [Peso] [int] NULL ,
  [Existencia] [int] NULL ,
  [Felabora] [datetime] NULL ,
  [Fcaduca] [datetime] NULL ,
  [Precio] [float] NULL
) ON [PRIMARY]
GO
```

PROVINCIAS

```
CREATE TABLE [dbo].[Provincias] (
  [Codpro] [char] (4) NOT NULL ,
  [Nompro] [varchar] (50) NULL
) ON [PRIMARY]
GO
```

REGISTRO DE USUARIOS

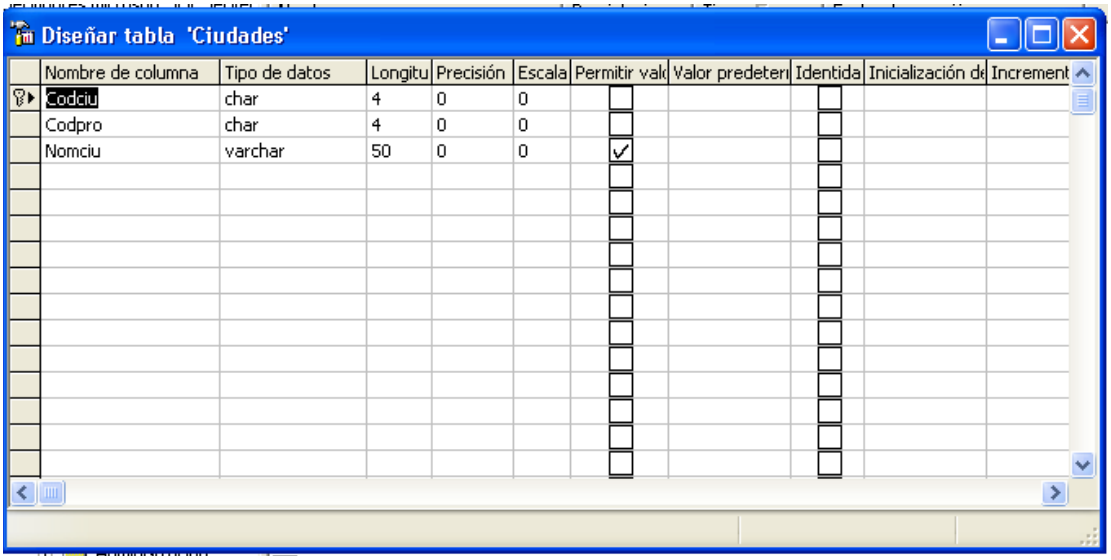
```
CREATE TABLE [dbo].[Usuarios] (
  [Idcli] [varchar] (50) NOT NULL ,
  [Password] [varchar] (50) NULL ,
  [Apellidos] [varchar] (50) NULL ,
  [Nombres] [varchar] (50) NULL ,
  [Mail] [varchar] (50) NULL ,
  [Provincia] [varchar] (50) NULL ,
  [Ciudad] [varchar] (50) NULL
) ON [PRIMARY]
GO
```

ENLACES

```
ALTER TABLE [dbo].[Ciudades] DROP CONSTRAINT
FK_Ciudades_Provincias
GO

if exists (select * from sysobjects where id = object_id(N'[dbo].[Ciudades]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Ciudades]
GO
```

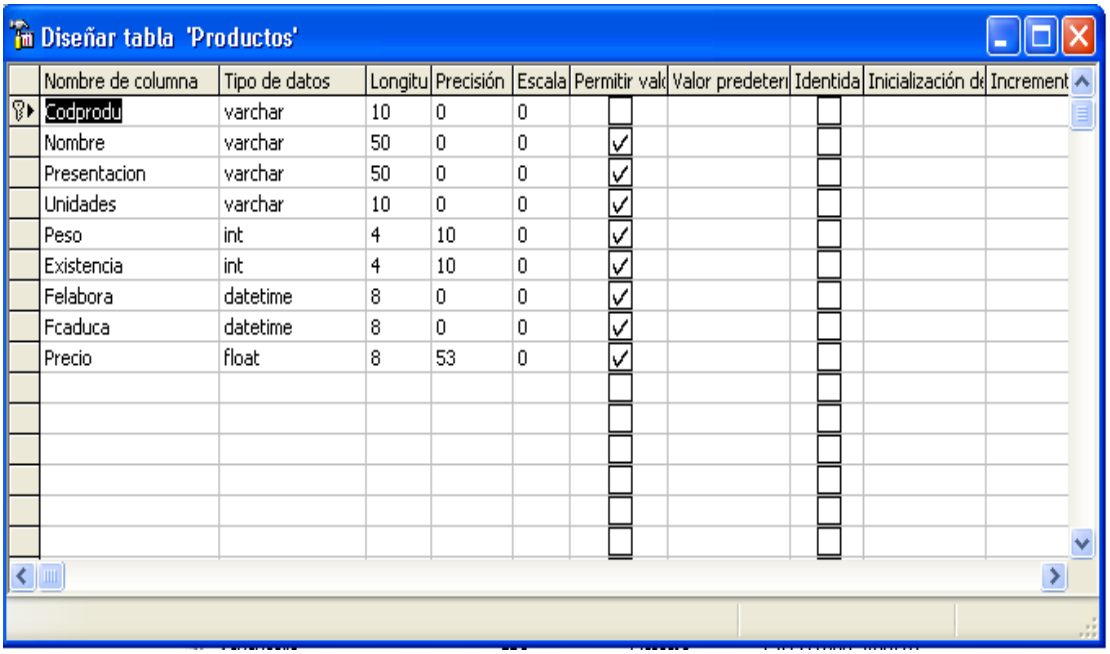
```
if exists (select * from sysobjects where id = object_id(N'[dbo].[Productos]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Productos]
GO

if exists (select * from sysobjects where id = object_id(N'[dbo].[Provincias]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Provincias]
GO

if exists (select * from sysobjects where id = object_id(N'[dbo].[Usuarios]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Usuarios]
GO
```

VISTAS DE LAS TABLAS



Diseñar tabla 'Ciudades'

Nombre de columna	Tipo de datos	Longitu	Precisión	Escala	Permitir val	Valor predeteri	Identida	Inicialización de	Increment
Codciu	char	4	0	0	☐		☐		
Codpro	char	4	0	0	☐		☐		
Nomciu	varchar	50	0	0	☑		☐		



Diseñar tabla 'Productos'

Nombre de columna	Tipo de datos	Longitu	Precisión	Escala	Permitir val	Valor predeteri	Identida	Inicialización de	Increment
Codprodu	varchar	10	0	0	☐		☐		
Nombre	varchar	50	0	0	☑		☐		
Presentacion	varchar	50	0	0	☑		☐		
Unidades	varchar	10	0	0	☑		☐		
Peso	int	4	10	0	☑		☐		
Existencia	int	4	10	0	☑		☐		
Felabora	datetime	8	0	0	☑		☐		
Fcaduca	datetime	8	0	0	☑		☐		
Precio	float	8	53	0	☑		☐		

Diseñar tabla 'Provincias'

	Nombre de columna	Tipo de datos	Longitu	Precisión	Escala	Permitir val	Valor predeter	Identida	Inicialización de	Increment
🔑	Codpro	char	4	0	0	☐		☐		
	Nompro	varchar	50	0	0	☑		☐		

Diseñar tabla 'Usuarios'

	Nombre de columna	Tipo de datos	Longitu	Precisión	Escala	Permitir val	Valor predeter	Identida	Inicialización de	Increment
🔑	Idcli	varchar	50	0	0	☐		☐		
	Password	varchar	50	0	0	☑		☐		
	Apellidos	varchar	50	0	0	☑		☐		
	Nombres	varchar	50	0	0	☑		☐		
	Mail	varchar	50	0	0	☑		☐		
	Provincia	varchar	50	0	0	☑		☐		
	Ciudad	varchar	50	0	0	☑		☐		

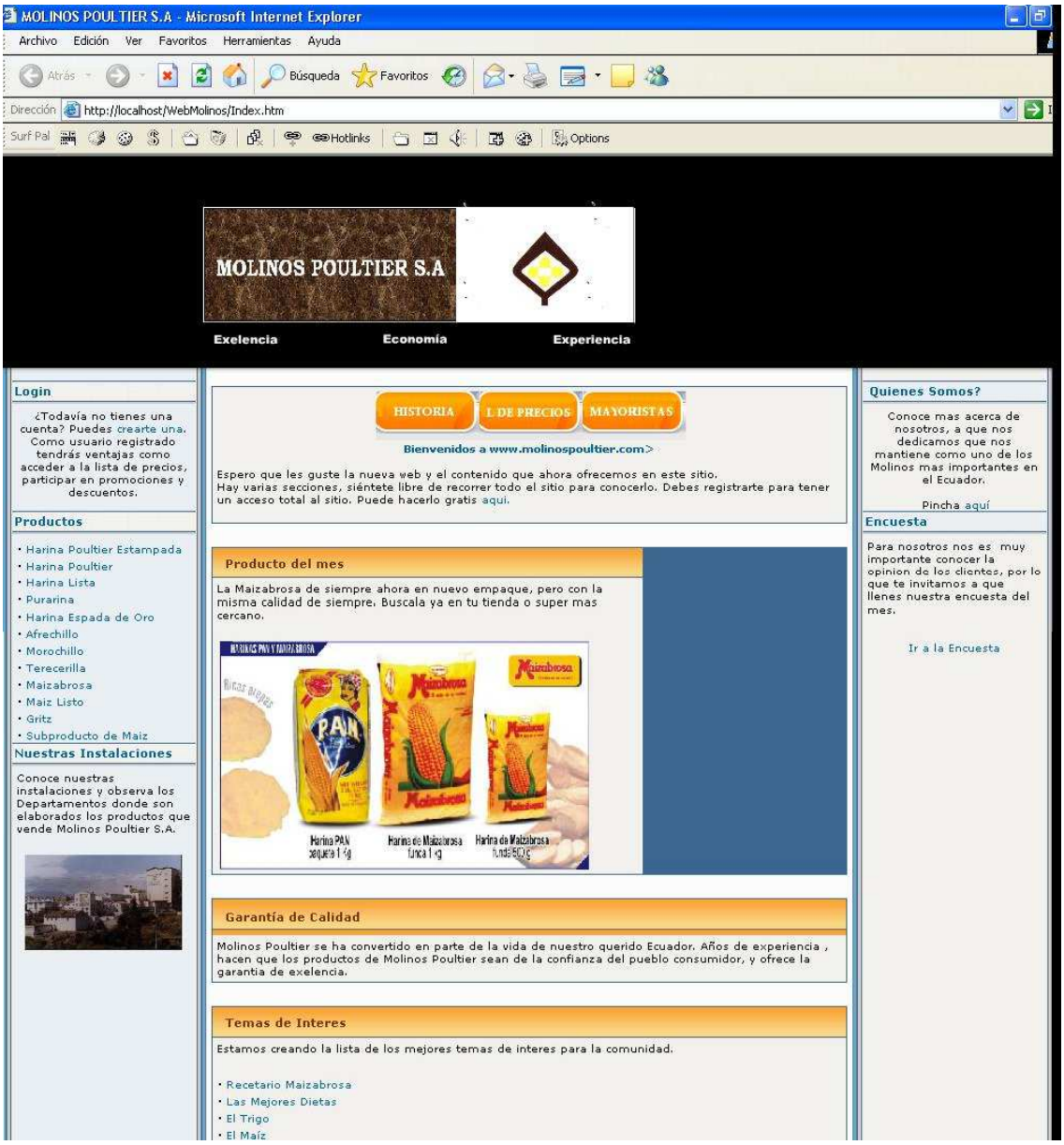
4.6 DISEÑO DE LA INTERFACE

ACTIVIDADES DE LA APLICACIÓN PROTOTIPO

Molinos Poultier es una de las mas grandes empresas dedicada al procesamiento de granos en la provincia de Cotopaxi, el prototipo que se ha diseñado para esta entidad ofrece los siguientes servicios:

- Registra los usuarios
- Logeo de los usuarios (reconoce o valida la existencia de usuarios registrados)
- Muestra la información que la empresa desea compartir con su clientes
- Averigua las existencias y precios de los productos
- Realiza una encuesta para determinar el producto de mayor demanda
- Brinda la posibilidad de enlaces con páginas relacionadas a la actividad de la Empresa.

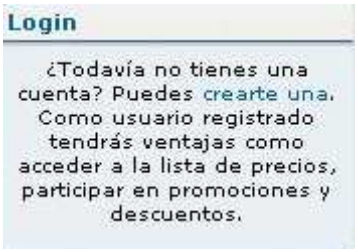
PANTALLA PRINCIPAL



En la Pantalla principal se muestra las opciones en las cuales al dar clic se puede ingresar a las opciones para poder acceder a información o servicios, en la pantalla principal podemos ingresar al modulo login, productos, nuestras instalaciones, quienes somos?, encuesta, precio de los productos y temas de interés.

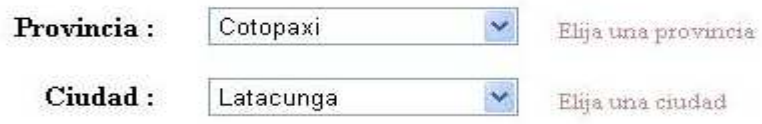
MODULO LOGIN

Permite crear cuentas de usuario nuevas



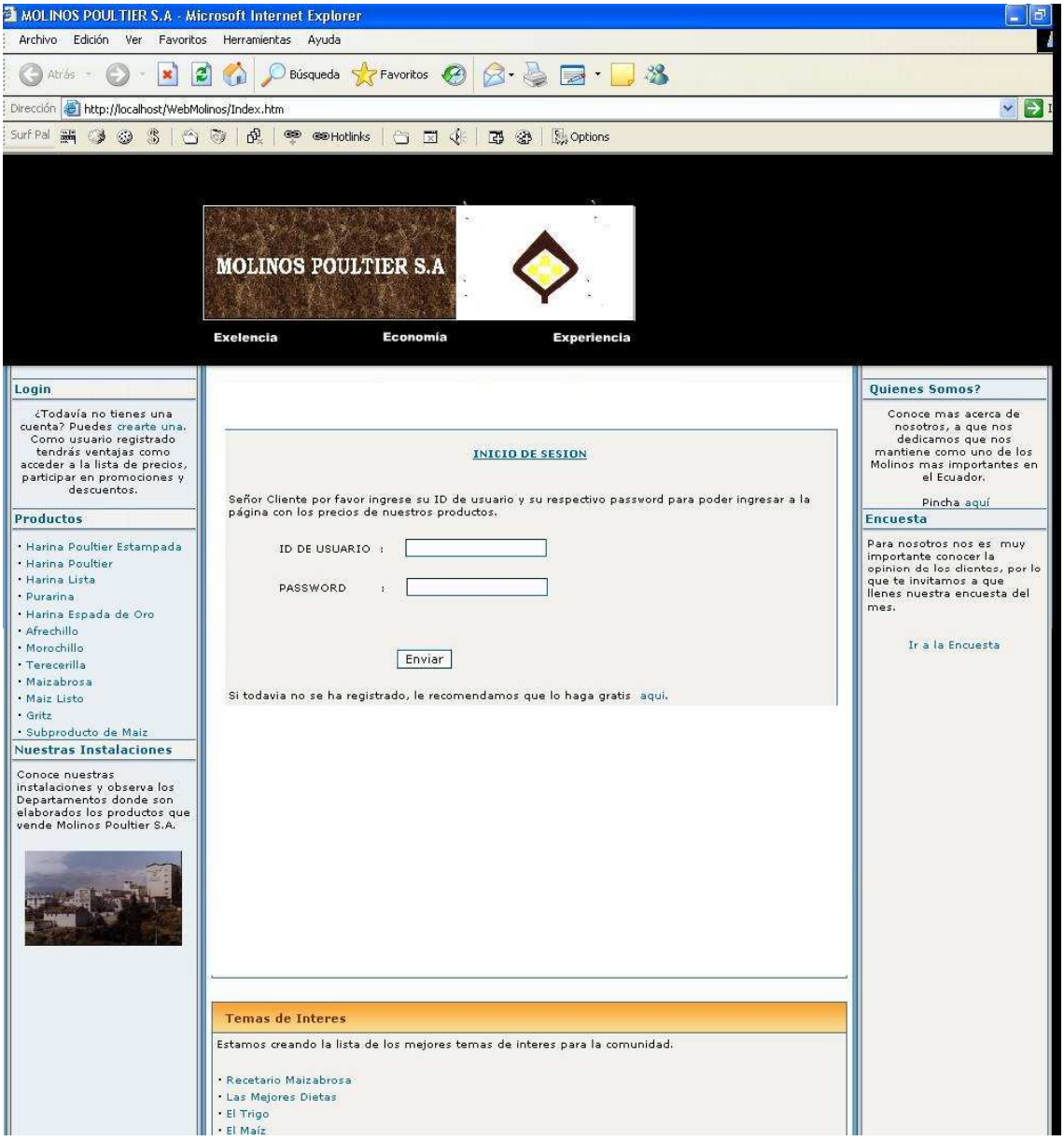
REGISTRAR DATOS DEL CLIENTE

En esta pantalla se registra los datos personales de cada usuario o cliente.



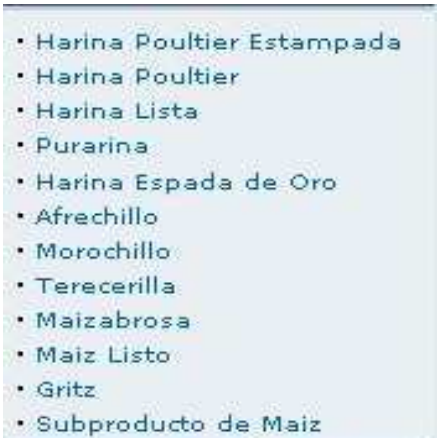
Existen campos que son de tipo dropdownlist que sirven para llenarse con información de la base de datos que sirven para acelerar el ingreso de datos. Ejemplo si elijo la provincia de Cotopaxi, el dropdownlist se llena con las ciudades respectivas.

PANTALLA LOGEO DEL CLIENTE



Esta modulo verifica los datos registrados del usuario y le pide ingresar un nick name o id y un password o contraseña.

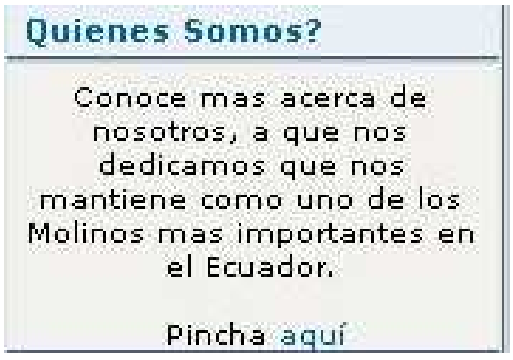
STOCK



Este modulo sirve para visualizar el stock de los productos existentes en bodega.

FOTOGRAFIAS

Desde esta opción se puede conocer los departamentos de la empresa, muestra imágenes e información acerca de las instalaciones.

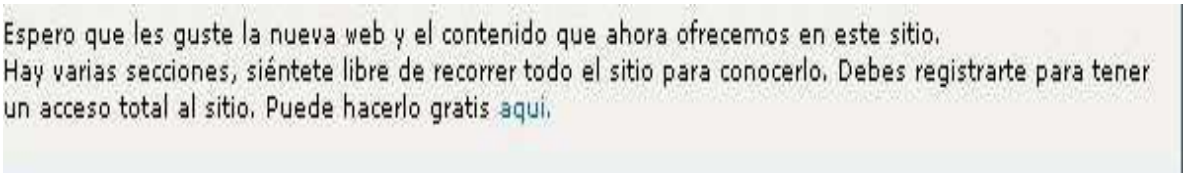


En este modulo se puede conocer la misión y la visión de esta empresa, en general información relacionada con la actividad de la empresa.

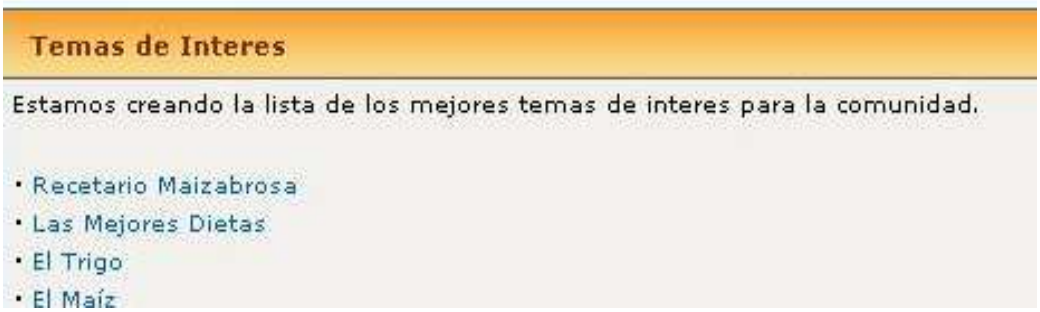
Este botón tiene un carácter informativo, contiene datos de la trascendencia histórica de la empresa.

Este botón sirve para ver información de los productos con un análisis de descuentos de los productos.

CREDENCIALES


Espero que les guste la nueva web y el contenido que ahora ofrecemos en este sitio.
Hay varias secciones, siéntete libre de recorrer todo el sitio para conocerlo. Debes registrarte para tener un acceso total al sitio. Puede hacerlo gratis aqui.

Esta opción permite realizar la validación de las credenciales del usuario.

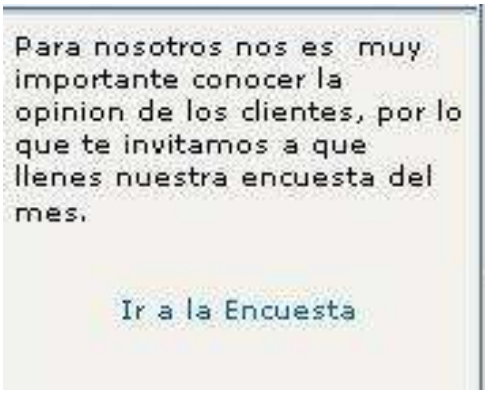

Temas de Interes

Estamos creando la lista de los mejores temas de interes para la comunidad.

- Recetario Maizabrosa
- Las Mejores Dietas
- El Trigo
- El Maíz

Esta opción es usada para establecer enlaces de interés para el usuario con respecto a las actividades de la empresa.

RESULTADOS DE LA ENCUESTA



Esta parte es para ingresar a una encuesta que servirá para la toma de decisiones, respecto a la oferta y demanda de los productos que esta empresa procesa.

4.7 INTEGRACION DE LA APLICACIÓN MULTICAPA

La aplicación maneja una estructura multicapa, en el lado del cliente únicamente hace las peticiones y es el servidor Web quien hace la consulta a la base de datos además de devolver los resultados de la consulta a la pagina HTML, este proceso se denomina modelo de datos desconectados.

El modelo de datos desconectados consiste en que cada vez que el cliente o usuario solicita una consulta la base de datos se abre solo para desplegar los resultados y se cierra después de ello, cabe mencionar que su antecesora permanecía abierta la base de datos en todo el proceso de ingreso al sistema, consulta de datos y despliegue de resultados.

4.8 COMPROBACION DE LA HIPOTESIS

¿Cual de las arquitectura: .NET Framework, Windows DNA y J2EE en tres capas presenta las mejores características para el diseño y desarrollo de aplicaciones distribuidas?

La Arquitectura .Net Framework es la mejor opción para el desarrollo de aplicaciones distribuidas, ya que es una plataforma que ayuda a generar, implementar y ejecutar servicios y aplicaciones Web, además de proporcionar multilenguajes basados en estándares para integrar aplicaciones y servicios con tecnología actual, puede compartir recursos específicos de procesos como subprocesos, memoria y conexiones a bases de datos con un gran número de usuarios, además esta tecnología puede crear un lenguaje intermedio que se ejecuta en cualquier tipo de arquitectura, sistema operativo (ya sea de Microsoft o no), sin desmerecer tampoco a J2EE que es una plataforma muy sólida, con mas experiencia que .Net, que es de código abierto y con la característica principal de software libre y que es una opción muy aceptada en el mercado, con el grado de dificultad claro para los programadores principiantes no deja de ser una gran competencia para .Net.