55

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

2001

EVALUACION DE HERRAMIENTAS JAVA Y XML PARA EL DESARROLLO DE APLICACIONES EN INTERNET.

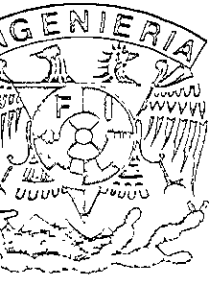
T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

RAYMUNDO SANTANA CARRILLO



DIRECTOR DE TESIS. ING. JUAN JOSE CARREON G

CIUDAD UNIVERSITARIA 2001

EVALUACIÓN DE HERRAMIENTAS JAVA Y XML PARA EL DESARROLLO DE APLICACIONES EN INTERNET

A mis padres y a mi hermana.

INDICE

ANTECEDENTES

Cada etapa en el desarrollo de la humanidad se ha ligado a algún tipo de tecnología dominante en ese momento (la edad de piedra, la edad de hierro, la era de las máquinas de vapor). El siglo XX se caracterizó por del tremendo incremento de conocimiento en muchas ramas de la ciencia. Este incremento dio pie a la posibilidad de un manejo más óptimo de la información a través de diferentes medios, siendo estos por ejemplo: redes telefónicas, la radio, la televisión, redes de computadoras y satélites de comunicación.

Podemos pensar en una sociedad globalizada, donde los medios juegan un papel fundamental en ésta y que debido al incremento anteriormente mencionado, muchas tecnologías están convergiendo a puntos comunes de manera muy rápida y, por tanto, los medios que son basados en estas tecnologías lo hacen de igual forma. La información fluye de manera más continua. El reunir, mover, guardar y procesar información se realiza de forma mas uniforme.

En este punto es donde el desarrollo en la rama de las comunicaciones y la electrónica juegan un papel primordial. Podemos comparar los tiempos que tuvieron que transcurrir para que un determinado medio de comunicación alcanzara una cantidad de personas dada. Por ejemplo, al teléfono le tomo 74 años alcanzar la cantidad de 50 millones de usuarios, al radio le tomo 38 años esta misma cantidad, mientras que el uso de Internet por esta cantidad de personas le tomo tan solo 4 años *1. De aquí que Internet, ha alcanzado a un mayor número de gente en un tiempo mucho más corto que cualquier otro medio (aún tomando en cuenta el incremento total de la población mundial).

"Una red de computadoras es un medio de comunicación entre personas que están muy distantes, ahora la idea de medios de comunicación lleva, de forma inherente, la idea de problemas, éticos, sociales y políticos."*2

Por dar un ejemplo, sabemos que a través de una red de computadoras de cobertura amplia (en los siguientes párrafos definiremos los diferentes tipos de red), la gente puede intercambiar mensajes escritos, archivos de sonido y de video de forma no muy complicada. "Los problemas surgen cuando estos textos tratan temas que a la gente en verdad le importan, como la política, la religión o el sexo ... algunas personas adoptan una postura de vive y deja vivir pero otras sienten que enviar cierto material (por ejemplo pornografía infantil) es simplemente inaceptable"*3.

Otro problema que podríamos mencionar son los mensajes anónimos, estos tiene el fin de poder expresar ideas a través de una red sin miedo a represalias, pero nos lleva también al problema de que si existiera la posibilidad de formular acusaciones, estas no serían tomadas en cuenta legalmente debido a que las acusaciones anónimas no pueden producen efectos jurídicos como pruebas.

No obstante, sabemos que a pesar de los problemas que esta forma de comunicación genere, el uso de una red de computadoras permite que la gente se pueda comunicar y resuelva de forma clara muchas otras situaciones.

Si consideramos lo anterior como verdadero, entonces podemos decir que existe gente que encuentra en las redes de computadoras un buen medio de comunicación.

Así, las redes de computadoras son una forma de comunicación donde se pueden expresar ideas de forma gráfica y textual, esta información fluye de forma muy rápida y está subordinada por el usuario a llegar a lugares tan distantes como este quiera. Desde este punto es donde inicia el desarrollo de este tema. ¿Cómo es posible que esta información fluya de este modo y que infraestructura se ocupa para lograrlo?. Para contestar esta pregunta bastaría con dar un vistazo al mercado de soluciones de comercio electrónico ofrecidos a empresas, dándonos esto una idea mas o menos completa de cómo se logra este tipo de comunicación.

Al revisar, destaca en importancia el mercado las soluciones ofrecidas por Microsoft para un nicho de mercado donde a los clientes se les ofrecen soluciones sencillas en su manejo; estas pueden ser ejecutadas en máquinas donde se tenga instalado software producido por esta misma compañía. En principio, el sistema operativo es la base para que estas soluciones puedan correr.

Si se tratara de migrar algún sistema de información desarrollado en esta plataforma a otra, donde el sistema operativo fuera diferente a éste, implicaría un trabajo considerable.

Existen otras soluciones ofrecidas, las cuales no están sustentadas en una plataforma especifica, y además, el código fuente es ofrecido de manera libre para su análisis y posibles modificaciones. De esta forma, diferentes fabricantes pueden tomar ese código y hacer implementaciones de estas soluciones de comunicación a sus propias máquinas o software para resolver problemas a clientes en lo específico, como también en lo general.

Entre este tipo de soluciones, esta el lenguaje de programación Java y el lenguaje de marcas estándar para el manejo de datos "XML" (eXtendible HyperText Markup Language), de los cuales se hablará en los siguientes capítulos tratando de valorar los alcances y limitaciones de estos lenguajes como herramientas para el desarrollo de aplicaciones en Internet.

Los medios de comunicación juegan un papel preponderante en una sociedad como la nuestra de tipo mediática y, con el advenimiento de nuevas tecnologías como lo es Internet y sus herramientas, es de suma importancia estar conciente del uso que se le pude dar a estas nuevas tecnologías y entenderlas para poder comunicarse de forma muy rápida y a lugares muy distantes con otras personas.

Se comienza en los siguientes capítulos con la teoría necesaria sobre redes, para luego estudiar cada una de estas herramientas.

CAPITULO 1

REDES DE COMPUTADORAS

1.1. DEFINICIONES

La fusión de las computadoras y las comunicaciones han tenido una profunda influencia en la forma en la que los sistemas de computo se organizan. En décadas pasadas existía el modelo de una sola máquina que centralizaba todos los procesos, ahora el modelo de varias máquinas conectadas entre sí, compartiendo recursos e información, es el más usado debido en gran parte al desarrollo de diferentes tecnologías orientadas al manejo individual pero compartido de información entre más personas (las formas en que se desarrollaron estas tecnologías serán explicadas mas adelante).

Existe confusión entre dos términos, redes de computadoras y sistemas distribuidos, será útil definir cada uno de estos términos y mostrar sus diferencias.

Una red de computadoras es una colección interconectada de computadoras autónomas. "Se dice que dos computadoras están interconectadas si son capaces de intercambiar información (...) en una red, el usuario debe ingresar a la red de forma explicita en una máquina, enviar los trabajos remotos explícitamente, mover explícitamente los archivos y, en general, llevar a cabo de manera personal el manejo de la red "*4

Ahora, en un sistema distribuido "la existencia de múltiples computadoras autónomas es transparente al usuario (...) la tarea de seleccionar el mejor procesador, encontrar y transportar todos los archivos de entrada al procesador y poner los resultados en un lugar apropiado, corresponde al sistema operativo"*5, es decir, en un sistema distribuido el usuario no debe preocuparse en cosas como ¿en qué máquina está el archivo que necesito? ¿Dónde se ejecutó mi programa?.

Los recursos del sistema completo son "transparentes" para el usuario, toda la asignación de recursos es realizada por el mismo sistema a través del software.

Teniendo de una manera un poco más clara estas definiciones, podemos tratar de entender cómo se llegó a esta forma de manejar la información a través de computadoras.

Para una compañía, los objetivos de una red son específicos.

Los equipos, los programas y especialmente los datos deben estar disponibles en cualquier momento; es decir, compartir recursos es una necesidad en cualquier empresa. Con una red se puede lograr una alta confiabilidad, los datos importantes pueden copiarse en más de una máquina y también con una red podemos añadir robustez al sistema. Si uno de los CPU's falla, los otros pueden sustituir su trabajo.

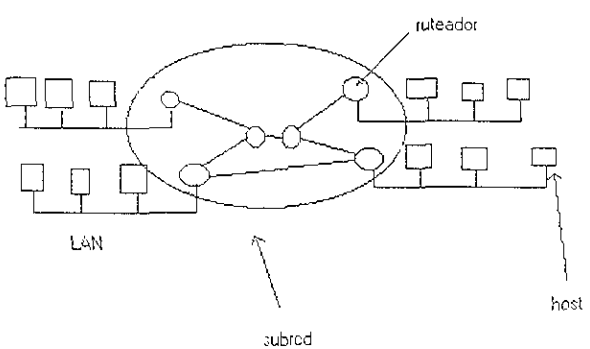
Ahora, las redes punto a punto consisten en muchas comunicaciones individuales entre máquinas. Para que pueda recibir la máquina "d" un mensaje enviado por una máquina "a", tendrá que pasar este por la máquina "b" y la máquina "c". Por esto, los algoritmos de ruteo en este tipo de redes son importantes para encontrar rutas críticas en un determinado sistema.

Una forma de entender este esquema es saber que las redes de difusión se aplican en sistemas geográficamente pequeños y en las redes más grandes se usa el de punto a punto. Esto de pequeñas y grandes puede resultar ambiguo. Otra forma de clasificar las redes es por su escala.

Para los fines que se persiguen, solo consideremos una red WAN para su explicación. Una red de área amplia se extiende sobre una área geográfica amplia, es formada por máquinas destinadas a ejecutar procesos de usuarios individuales (estas máquinas generalmente son llamadas "host", anfitriones, en el idioma inglés). Estas máquinas están conectadas por una subred de transmisión. El trabajo de una subred de transmisión es conducir mensajes de un "host" a otro de forma lógica.

La red se compone de dos elementos, las líneas de transmisión y los elementos de conmutación.

Las líneas de comunicación es el medio físico para transportar bits de un lugar a otro. Los elementos de conmutación son computadoras especializadas que se conectan a las líneas de transmisión. Estas máquinas son llamadas "ruteadores".



Por lo general, todas las redes WAN trabajan de forma tal que si un paquete es enviado, este es recibido por algún ruteador y este a su ves retiene o retransmite el paquete por un canal libre. Así también son llamadas redes de paquete conmutado.

Otro dato importante de este tipo de redes es la topología con la cual los ruteadores están enlazados: estrella, anillo, árbol, anillo con estrella o irregular.

1.3. SOFTWARE DE RED

Para reducir la complejidad en el diseño de una red, esta se organiza por capas superpuestas, cada una de estas capas tiene un objetivo específico prestando servicios a las capas superiores a ellas. Estas capas resuelven etapas necesarias para que la capa siguiente realice su trabajo y soporte a la capa siguiente y así sucesivamente. La forma en que estas capas se comunican es por reglas llamadas protocolos.

Un conjunto de capas y protocolos recibe el nombre de arquitectura de red.

Para el diseño, las reglas de transferencia de datos que por lo general se siguen son los sistemas simplex, semiduplex y duplex, en el primero los datos viajan en una sola dirección, en el siguiente viajan en ambas direcciones pero no simultáneamente y en el ultimo, los datos van en ambas direcciones y simultáneamente.

Los elementos activos de cada capa (que realizan algún proceso) se llaman entidades y pueden ser software o hardware.

Las entidades de la misma capa en sistemas distintos se llaman entidades pares y, en general, cada entidad implementa un servicio para la entidad inmediata superior en su sistema.

Cada uno de estos servicios están disponibles en los SAP (service access points) y cada SAP tiene una dirección que lo identifica.

Ahora las capas pueden ofrecer diferentes tipos de servicio, los orientados a conexión y los que carecen de conexión

En el primero se abre un "tubo", donde el emisor abre una conexión y empuja datos a través del tubo hasta el receptor y después libera la conexión. (ejemplo de esto seria el teléfono).

El servicio sin conexión no sigue el orden de entrega de envíos, puede ser que el paquete "a" llegue después que un paquete "b" a pesar de que fue mandado antes que "b". A este tipo de conexion puede llamársele también servicio de datagramas y es un servicio sin conexión no confiable. En complemento existe el servicio de

datagramas con acuse que es un servicio sin conexión confiable (para ejemplificar esto, el correo electrónico es uno de estos servicios).

Este ultimo servicio esta muy relacionado con el servicio de petición-respuesta que se utiliza para instrumentar la comunicación en un modelo cliente-servidor. El remitente transmite un datagrama que contiene una petición y en la repuesta contendrá la contestación

1.4. CAPAS DE RED, REDES BASADAS EN CAPAS.

1.4.1.EL MODELO DE REFERENCIA OSI Y EL MODELO DE REFERENCIA TCP/IP

El modelo OSI se basa en la propuesta de la organización Internacional de normas como un primer intento de estandarizar los protocolos que se usan en las diversas capas de una red de computadoras.

El modelo OSI tiene siete capas a saber:

a) La capa física

La capa física tiene que ver con la transmisión de bits por un canal de comunicación. Esta capa trata de asegurar la correcta transmisión de "1" y "0", evitando que por ejemplo se transmita un "1" y se reciba erróneamente un "0".

b) La capa de enlace de datos

El objetivo de esta capa es proporcionar a la capa de red un medio físico y libre de errores de transmisión

c) La capa de transporte

Esta capa se encarga de controlar el funcionamiento de la subred. Uno de los principales objetivos de esta capa es encaminar los paquetes de la fuente a su destino.

d) La capa de sesión

Su función es aceptar datos de la capa de sesión, dividirlos en unidades más pequeñas, pasarlos a la capa de red y asegurar que todos los pedazos lleguen correctamente al otro extremo.

e) La capa de presentación

Esta capa permite a los usuarios de máquinas diferentes establecer sesiones entre ellos.

Una sesión permite el transporte ordinario de datos, como lo hace la capa de transporte, pero también proporciona servicios que son útiles en algunas aplicaciones.

Esta capa también se ocupa de la sintaxis y la semántica de la información que se transmite. A diferencia de las otras capas que involucran el transporte y el manejo de bits, esta capa se encarga de resolver problemas frecuentes del usuario.

Esta capa contiene protocolos que se necesitan con frecuencia para presentar los datos al usuario. También esta capa permite la transferencia de archivos entre diferentes sistemas.

f) Capa de aplicación

Contiene los protocolos que se usan con mas frecuencia, por ejemplo una terminal virtual de red que permita la correspondencia entre funciones de la terminal que se esta usando y el software para manejar a ésta de forma remota con otro tipo de terminal.

Otra función de la capa es la transferencia de archivos. La cual resuelve las posibles incompatibilidades con los formatos de los archivos.

1.4.2. EL MODELO DE REFERENCIA TCP/IP

Este modelo surgió de la necesidad de conectar múltiples redes diferentes con una arquitectura flexible para poder resolver requerimientos divergentes como la transferencia de archivos o la transmisión de discursos en tiempo real, entre otras cosas.

Este protocolo presenta las siguientes capas.

a) Capa de interred.

Esta capa carente de conexiones es la que mantiene unida toda la arquitectura a través de conmutación de paquetes. Los nodos envían paquetes a cualquier red y los hacen viajar de forma independiente a su destino. Los paquetes pueden llegar incluso en un orden diferente a aquel en que se enviaron, en cuyo caso corresponde a las capas superiores reacomodarlos, si se desea la entrega ordenada.

La capa de interred define un formato de paquete y protocolo oficial llamado IP. Este protocolo es el que permite entregar, los paquetes a donde se suponen deben de ir.

La capa de interred de TCP/IP es muy parecida en funcionalidad a la capa de red de OSI.

b) Capa de transporte.

Esta capa se diseño para permitir que las entidades pares de los nodos de origen y destino lleven a cabo una conversación, de la misma forma que lo hace la capa de transporte de OSI.

Aquí se definieron dos protocolos de extremo a extremo. El primero, TCP (protocolo de control de transmisión) es un protocolo confiable orientado a la conexión que permite que una corriente de bytes originada en una máquina se entregue sin errores a cualquier otra máquina de la interred.

El segundo, UDP (protocolo de datagrama de usuario), es un protocolo sin conexión, no confiable para aplicaciones que no necesitan la asignación de secuencia ni el control de flujo de TCP y que desean utilizar los suyos. Este protocolo también se usa ampliamente para consultas de petición y respuesta de una sola ocasión, del tipo cliente-servidor, y en aplicaciones en las que la entrega pronta es más importante que la entrega precisa, como las transmisiones de voz y video.

c) Capa de aplicación.

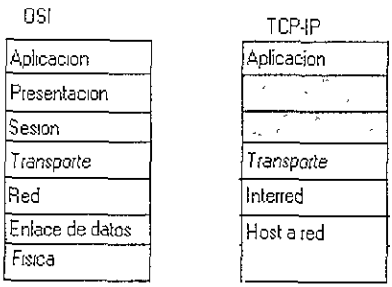
El modelo TCP/IP no tiene capas de sesión ni de presentación. No se pensó que fueran necesarias, así que no se incluyeron. La experiencia con el modelo OSI ha comprobado que esta visión fue correcta, debido a que se utilizan muy poco en la mayoría de las aplicaciones.

Esta capa contiene todos los protocolos de alto nivel. Entre los protocolos más antiguos esta el de terminal virtual (TELNET), el de transferencia de archivos (FTP) y el de correo electrónico (SMTP)

El de terminal virtual permite que un usuario en una máquina ingrese a otra en forma eficiente. El correo electrónico fue en inicios solo una clase de transferencia de archivos, pero más adelante se desarrollo para el un protocolo especializado. Con los años a esta capa se le han añadido muchos otros protocolos, como el servicio de nombres de dominio (DNS) para relacionar los nombres de los nodos con sus direcciones de la red, HTTP el protocolo que se usa para recuperar paginas del WWW (World Wide Web), entre otros.

1.5. LAS REDES, LOS PROTOCOLOS, E INTERNET.

A mediados de la década de los 60's, en la cúspide de la guerra fría el departamento de defensa de los E.U. quería una red de comando y control que pudiera sobrevivir a una guerra nuclear. Este acudió a su rama de investigación, ARPA (Agencia de proyectos de investigación avanzados). ARPA ofreció financiamiento y contratos a universidades y compañías que considero prometedoras para el proyecto.



Así ARPA decidió que la red que necesitaba el departamento de defensa debía de ser una red de paquete conmutado que consistía en una subred y computadoras host.

La subred consistiría de minicomputadoras llamadas IMP (interface message processor) conectadas por líneas de transmisión. Para lograr una alta confiabilidad cada IMP se conectaría cuando menos a otras dos. La subred iba a ser una subred de datagrama, de modo que si algunas líneas e IMP resultaban destruidas, los mensajes se podrían reencaminar de forma automática a través de trayectorias alternas.

Cada nodo de la red consistiría en un IMP y una host en el mismo cuarto, conectados por un cable corto. Una host podría recibir mensajes de hasta 8063 bits a su IMP, que entonces los dividiría en paquetes de 1008 bits a lo sumo y los reenviaría a su destino en forma independiente. Cada paquete se recibía en sus totalidad antes de reenviarse, por lo que la subred fue la primera red electrónica de conmutación de paquetes de almacenar y reenviar. Los IMP se interconectaron con líneas de 56 kbps rentadas a compañías de teléfonos.

El software se dividió en dos partes; subred y host. El software de la subred consistió en el extremo de las IMP de la conexión host-IMP, el protocolo IMP-IMP, y un protocolo de IMP fuente a IMP destino diseñado esto para mejorar la confiabilidad al existir conexiones entre IMP's antes mencionadas.

Para la parte de software del host, se necesito el extremo host-IMP, el protocolo host-host y el software de aplicación. Mas adelante se permitió que las terminales se conectaran de forma directa a un IMP especial llamado TIP (terminal interface processor) y además se agregó el poder tener múltiples hosts por cada IMP, hosts que se comunicaban con múltiples IMP y host e IMP separadas por una gran distancia.

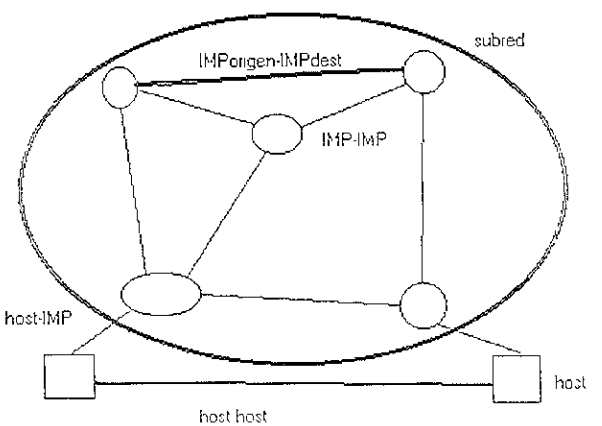
Se necesitaba también una mejor funcionalidad con respecto a los protocolos, lo que culminó con la invención del modelo y los protocolos TCP/IP. TCP/IP se diseño de manera específica para manejar la comunicación en las interredes, algo que se volvió cada vez más importante al conectarse más y más redes a la ARPANET.

Para fomentar la adopción de estos nuevos protocolos, la ARPANET concedió varios contratos a la universidad de Berkeley para integrarlos en el UNIX de Berkeley. Los investigadores de Berkeley desarrollaron una interfaz de programa conveniente para la red (sockets) y escribieron muchos programas de aplicación, utilería y administración para facilitar el trabajo con redes.

Para ese momento no existía software de red para redes LAN, así que el paquete desarrollado por Berkeley fue adoptado de inmediato. Además, con TCP/IP era fácil que las LAN se conectaran a la ARPANET y muchas lo hicieron.

Para 1983, la ARPANET gozaba de estabilidad y éxito, con más de 200 IMP y cientos de hosts. ARPANET cedió el manejo de la red a DCA (Agencia de comunicaciones de la defensa) y ésta lo primero que hizo fue separar la porción militar en una subred independiente llamada MILNET y genero pasarelas estrictas entre MILNET y la red de investigación restante.

La cantidad de redes, máquinas y usuarios conectados a la ARPANET creció con rapidez después de que el protocolo TCP/IP se convirtió en el único protocolo oficial a partir de 1983. Cuando se interconectaron la NSFNET y la ARPANET, el crecimiento se hizo exponencial; se unieron muchas redes regionales y se hicieron conexiones con redes en Canadá, Europa y el Pacífico.



Para 1990, la ARPANET había sido rebasada por redes más nuevas que ella misma había engendrado, de manera que se clausuró y desmanteló a diferencia de MILNET que continúa operando "En este mismo año, la red había crecido a 30,000 redes y

CAPITULO 2

PROGRAMACION ORIENTADA A OBJETOS

2.1. DEFINICIÓN DE PROGRAMACIÓN ORIENTADA A OBJETOS

Se puede definir la programación orientada a objetos como "el método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia."*7

Utiliza objetos, no algoritmos, como sus bloques lógicos de construcción fundamentales.

Cada objeto es una instancia de alguna clase; las clases están relacionadas con otras clases por medio de relaciones de herencia.

Pero ahora, daremos unas definiciones para entender todos los conceptos de la definición anterior, luego ahondaremos un poco mas hasta poder llegar al final de este capítulo entendiendo lo que es un lenguaje para programación orientado a objetos.

2.2. CONCEPTOS

De lo primero que partiremos es que el software es inherentemente complejo. Si tratáramos de estructurar por puntos el problema de la complejidad en el software podemos decir:

- Los problemas a resolver con software pueden resultar en casos sumamente complejos.
- El proceso de desarrollo del software es difícil de gestionar.
- Las aplicaciones informáticas son discretas. Pequeños cambios pueden conducir a situaciones impredecibles.

De manera particular para entender mas la "complejidad" a la cual nos referimos, tenemos que los atributos de la complejidad según Simón y Ando son:

- Los sistemas complejos comprenden subsistemas interrelacionados con algún tipo de relación jerárquica. Cada subsistema es en sí mismo un sistema complejo, hasta llegar a componentes elementales.
- La elección de los componentes elementales o primitivos de un sistema es relativamente arbitraria, dependiendo del observador.

- Uniones o relaciones dentro de estos componentes son, generalmente, más fuertes que las relaciones entre distintos componentes. Se pueden separar efectos de alta frecuencia y baja frecuencia.

- Los sistemas jerárquicos suelen componerse de unos pocos tipos de subsistemas que se unen y combinan de distintas formas.

- Los sistemas complejos que funcionen deben evolucionar desde otros sistemas simples que en principio funcionen correctamente. Un sistema complejo no se crea a partir de uno con fallos, o de la nada.

Teniendo ya esta base sobre el software y su relación con la complejidad, se puede estudiar ahora una solución viable a este problema: Divide et impera. Estudiemos este punto.

Para realizar esta división, se presentan dos formas de hacerlo.

- Descomposición Algorítmica (top-down o estructural): Rompe el sistema en partes, cada una representando un pequeño paso del proceso. Métodos de diseño estructurados conducen a descomposiciones algorítmicas, donde la atención se centra en el flujo del sistema.

- Descomposición orientada a objetos: Trata de identificar semánticamente el dominio del problema. El entorno del problema se estudia como un conjunto de agentes autónomos (objetos) que colaboran para realizar un comportamiento complejo.

Las características de estas descomposiciones se pueden observar en forma de tabla. (ver anexo)

2.3. PRINCIPALES ETAPAS DE LA INGENIERIA DE SOFTWARE CON UNA DESCOMPOSICIÓN ORIENTADA A OBJETOS

En la ingeniería de software dos partes importantes son el análisis y el diseño, que aplicadas en la metodología de orientación a objetos pueden ayudarnos a aclarar mas este tema y llegar al objetivo de este capítulo.

- Las funciones del análisis y diseño son:

- Satisfacer especificaciones funcionales.

- Determina las limitaciones del objetivo.

- Encontrar los requerimientos en la realización y los recursos.

- Cumplir criterios de diseño.
- Satisfacer restricciones, como tamaño o costos.
- Determina herramientas a utilizar.

Los distintos métodos de diseño comparten:

- Notación: un lenguaje para expresar el modelo.
- Proceso: actividades que conducen a la construcción ordenada del modelo.
- Herramientas: artefactos y reglas.

En ese entendido, el análisis orientado a objetos se define como un método de análisis que examina los requerimientos desde la perspectiva de clases y objetos encontrada en el vocabulario original del problema. Por su parte, el diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientado a objetos y una notación para describir modelos lógicos y físicos, dinámicos y estáticos, del sistema bajo diseño.

Podemos seguir las etapas de la ingeniería de software con el desarrollo del sistema, que se lleva en gran medida con la programación del sistema.

La programación orientada a objetos es el método de implementación en el cual los programas se organizan como colecciones cooperantes de objetos, cada uno de los cuales representa un ejemplo de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas por relaciones .
Esta etapa se vale de lenguajes, y en este caso, estos lenguajes son llamados lenguajes orientados a objetos.

Podemos decir que un lenguaje es orientado a objetos, si cumple en general con lo siguiente:

- Soporta objetos que son abstracciones de datos, mediatizados por operaciones, y con estados locales ocultos.
- Los objetos tienen un tipo asociado (clase).
- Los tipos (clases) pueden heredar atributos de supertipos (superclases).

Ya con un leguaje y su definición, las principales características de la programación orientada a objetos son las siguientes:

1 Abstracción
2 Encapsulación
3 Modularidad
4 Jerarquización
5 Tipificado
6 Concurrencia
7 Persistencia

Las cuatro primeras características son esenciales, mientras que las tres últimas son útiles y complementan el modelo.

1 Abstracción

Denota las características esenciales que distinguen a un objeto de otros tipos de objetos, definiendo precisas fronteras conceptuales, relativas al observador.

- Surge del reconocimiento de similitudes entre ciertos objetos, situaciones o procesos en el mundo real.
- Decide concentrarse en estas similitudes e ignorar las diferencias.
- Enfatiza detalles con significado para el usuario, suprimiendo aquellos detalles que, por el momento, son irrelevantes o distraen de lo esencial.
- Deben seguir el "principio de mínimo compromiso", que significa que la interfase de un objeto provee su comportamiento esencial, y nada más que eso. Pero también el "principio de mínimo asombro": capturar el comportamiento sin ofrecer sorpresas o efectos laterales.

2 Encapsulación.

Es la forma de conceptualización, con su entorno, de los elementos de una abstracción (estructura y comportamiento). La encapsulación sirve para separar la interfase de una abstracción y su implementación.

- Es un concepto complementario al de abstracción.
- La encapsulación esconde la implementación del objeto que no contribuye a sus características esenciales.

- La encapsulación da lugar a que las clases se dividan en dos partes:

  a) Interfase: Captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.

  b) Implementación: Comprende la representación de la abstracción, así como los mecanismos que conducen al comportamiento deseado.
  Se conoce también como ocultación o privacidad de la información.

3 Modularidad.

Es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos.

- Cada módulo se puede compilar separadamente, aunque tengan conexiones con otros módulos.

- En un diseño estructural, modularización comprende el agrupamiento significativo de subprogramas. En diseño orientado a objetos, la modularización debe ceñirse a la estructura lógica elegida en el proceso de diseño.

- Dividir un programa en componentes individualizados reduce de alguna manera su complejidad.

4 Jerarquización.

Es una clasificación u ordenación de las abstracciones.

Por jerarquía denotamos el orden de relación que se produce entre abstracciones diferentes.

Los tipos de jerarquía más útiles:

- Herencia: (generalización/especialización, padre/hijo, jerarquía del tipo "es un"...). Una clase (subclase) comparte la estructura o comportamiento definido en otra clase, llamada superclase.

- Herencia múltiple: Una clase comparte la estructura o comportamiento de varias superclases.

- Composición: Comprende relaciones del tipo "es parte de" al realizar una descomposición.

Relaciones entre los conceptos asociados al modelo de objetos.

- Los conceptos de abstracción y encapsulación son conceptos complementarios: abstracción hace referencia al comportamiento observable de un objeto, mientras encapsulación hace referencia a la implementación que la hace alcanzar este comportamiento.

- Existe una tensión entre los conceptos de encapsulación de la información y el concepto de jerarquía de herencia, de tal modo que se requiere de algún mecanismo en el acceso a la información. Algunos lenguajes ofrecen mucha flexibilidad, pudiendo disponer de tres posibilidades en cada clase:

  a) Privado: Declaraciones accesibles sólo a la clase (completamente *encapsulado*)

  b) Protegido: Declaraciones accesibles a la clase y a sus subclases.

  c) Público: Declaraciones accesibles a todos los clientes.

Además de estos tres tipos, soporta la definición de clases cooperativas a las que se les permite acceder a la parte privada de la implementación. Estas clases se denominan friends.

5 Tipificado.

Tipificar es la imposición de una clase a un objeto, de tal modo que objetos de diferentes tipos no se puedan intercambiar, o se puedan intercambiar solo de forma restringida.

- Tipo es una caracterización precisa de las propiedades estructurales y de comportamiento que comparten una colección de entidades.

- Grosso modo, tipo y clase pueden considerarse sinónimos.

- Existen lenguajes fuertemente tipificados (Ada) y débilmente tipificados. Estos últimos soportan polimorfismo, mientras que los fuertemente tipificados no.

6 Concurrencia.

Es la propiedad que distingue un objeto activo de uno no activo. Concurrencia permite que diferentes objetos actúen al mismo tiempo, usando distintos "threads" (hilos) de control.

7 Persistencia.

Es la propiedad por la cual la existencia de un objeto trasciende en el tiempo (esto es, el objeto sigue existiendo después de que su creador deja de existir) o en el espacio (esto es, la localización del objeto cambia respecto a la dirección en la que fue creado).

2.4. CLASES Y OBJETOS.

Ahora, teniendo ya establecido lo general de la metodología, definiremos lo que son objetos y clases, aclarando los elementos específicos.

2.4.1. Objetos.

Se dice de un objeto:

- "Un objeto es una cosa tangible, algo a que se puede aprehender intelectualmente o algo hacia lo que se puede dirigir una acción o pensamiento."[8]

- Un objeto representa un "ítem" entidad individual e identificable, o una entidad real o abstracta, con un papel definido en el dominio del problema

- Un objeto tiene:

    a) Estado
    b) Comportamiento
    c) Identidad

La estructura y el comportamiento de objetos similares se definen en sus clases comunes. El término objeto y ejemplo (instancia) de una clase son intercambiables.

a) Estado de un objeto.

El estado de un objeto abarca todas las propiedades del objeto, y los valores actuales de cada una de esas propiedades. Las propiedades de los objetos suelen ser estáticas, mientras los valores que toman estas propiedades cambian con el tiempo.

- El hecho de que los objetos tengan estado implica que ocupan un espacio, ya en el mundo físico, ya en la memoria de la máquina.

- El estado de un objeto esta influido por la historia del objeto.

- El estado de un objeto representa el efecto acumulado de su comportamiento.

El estado de un objetos es una fotografía de la evolucion del sistema en un tiempo dado.

b) Identidad de un objeto.

Identidad es la propiedad de un objeto que lo lleva a distinguirse de otros.

c) Comportamiento de un objeto.

Comportamiento es la forma como un objeto actúa y reacciona, en términos de sus cambios de estado y de los mensajes que intercambia.

El comportamiento de un objeto representa su actividad externamente visible y con capacidad de prueba. Son las operaciones que una clase realiza (llamadas también mensajes) las que dan cuenta de cómo se comporta la clase. Por operación se denota el servicio que una clase ofrece a sus clientes. Un objeto puede realizar cinco tipos de operaciones sobre otro, con el propósito de provocar una reacción:

- Modificador: altera el estado de un objeto.
- Selector: accede al estado de un objeto, sin alterarlo.
- Iterador: permite a todas las partes de un objeto ser accedidas en un orden.
- Constructor: crea un objeto y/o inicializa su estado.
- Destructor: libera el estado de un objeto y/o destruye el objeto.

Relaciones entre objetos

Las relaciones entre objetos abarcan las operaciones, resultados y suposiciones que unos hacen sobre los otros.

- Ligas. Son conexiones físicas o conceptuales entre objetos. Denota la asociación especifica por la que un objeto (cliente) usa o solicita el servicio de otro objeto (servidor). El paso de mensajes entre objetos los sincroniza.
- Composición. Denota relaciones todo/parte, con capacidad para gobernar desde el todo las partes. Es equivalente a la relación "tener un". El todo puede contener a la parte.

Composición es conveniente en las ocasiones en que el encapsulamiento de las partes es prioritario. Si se requiere que las relaciones entre objetos estén vagamente acopladas, se utilizan ligas.

1.4.2 Clases

"Una clase es un conjunto de objetos que comparten una estructura y comportamiento comunes."*9

- La clase representa una abstracción, la esencia que comparten los objetos.
- Un objeto es un ejemplo de una clase.
- Un objeto no es una clase, y una clase no es un objeto (aunque puede serlo, en otros lenguajes).
- Las clases actúan como intermediarias entre una abstracción y los clientes que pretenden utilizar la abstracción. De esta forma, la clase muestra:

  a) Visión externa de comportamiento (interfase, abstracción), que enfatiza la abstracción escondiendo su estructura y secretos de comportamiento.

  b) Visión interna (implementación, encapsulacion), que abarca el código que se ofrece en la interfase de la clase.

Las relaciones entre clases se representan en diferentes formas, o también como relaciones semánticas.

- Asociación. Indica relaciones de mandato bidireccionales (Punteros ocultos en "C++"). Conlleva dependencia semántica y no establece una dirección de dependencia. Tienen cardinalidad.
- Herencia. Por esta relación, una clase (subclase) comparte la estructura y/o comportamiento definidos en una (herencia simple) o más (herencia múltiple) clases, llamadas superclases.

  a) Representa una relación del tipo "es un" entre clases.

  b) Una subclase aumenta o restringe el comportamiento o estructura de la superclase (o ambas cosas).

  c) Una clase de la que no existen ejemplos se denomina abstracta.

- Composición. Representa una relación del tipo "tener un" entre clases. Cuando la clase contenida no existe independientemente de la clase que la contiene se denomina agregación "por valor" y además implica contenido

físico, mientras que si existe independientemente y se accede a ella indirectamente, es agregación "por referencia".

- Uso. Es un refinamiento de la asociación donde se especifica cual es el cliente y cual el *servidor de ciertos servicios, permitiendo a los clientes acceder sólo* a las interfaces públicas de los servidores, ofreciendo mayor encapsulación de la información.

- Ejemplificación. Se usa en lenguajes que soportan genericidad (declaración de clases parametrizadas y argumentos tipo template). Representa las relaciones entre las clases parametrizadas, que admiten parámetros formales, y las clases obtenidas cuando se concretan estos parámetros formales, ejemplificados o inicializados con un ejemplo.

- Metaclases. Son clases cuyos ejemplos son a su vez clases.

2.4.3 Relaciones entre clases y objetos

Se dice de la s relaciones entre clases y objetos:

- Todo objeto es el ejemplo de una clase, y toda clase tiene cero o más objetos.

- Mientras las clases son estáticas y tienen semántica, relaciones y existencia fijadas previamente a la ejecución de un programa, los objetos se crean y destruyen rápidamente durante la actividad de una aplicación.

El diseño de clases y objetos es un proceso incremental e iterativo. Debe asegurar la optimización en los siguientes parámetros:

- Acoplamiento: Grado de acoplamiento entre módulos.

- Cohesión: Mide el grado de conectividad entre elementos de un módulo, y *entre objetos de una clase.*

- Suficiencia: Indica que las clases capturan suficientes características de la abstracción para conseguir un comportamiento e interacción eficiente y con sentido.

- Completitud: Indica que la interfase de la clase captura todo el significado característico de una abstracción, escrito en el mínimo espacio.

- Primitividad: Las operaciones deben implementarse para dar acceso a una representación fundamental de la abstracción. Cuales son operaciones primitivas y cuales no (se pueden *realizar a partir de otras*) es un asunto subjetivo y afecta a la eficiencia en la implementación.

2.5. CONLUSIONES DE ESTE CAPITULO

Con lo argumentado a lo largo de los párrafos anteriores, se tiene un panorama general de lo que es la programación orientada a objetos y donde se ubica con respecto a la metodología de orientación a objetos.

Este panorama ayudará a entender con mayor claridad el lenguaje de programación Java, que es un lenguaje para programación orientada a objetos que evaluaremos como una herramienta para aplicaciones en Internet.

CAPITULO 3

EL LENGUAJE DE PROGRAMACIÓN JAVA

3.1 EL LENGUAJE DE PROGRAMACIÓN JAVA

Java es un poderoso y completo lenguaje de programación orientado a objetos desarrollado por Sun Soft Inc., una subsidiaria de la empresa fabricante de equipo de cómputo denominada: Sun Microsystems, Inc.

"La plataforma Java es fundamentalmente una nueva forma de computo, basada en el poder de las redes y en la idea de que el mismo software debiera correr en una gran variedad de tipos de computadoras y en otros dispositivos."*10

Técnicamente se puede definir a Java como un lenguaje simple, robusto, totalmente orientado a objetos, distribuido, compilado e interpretado, independiente a la arquitectura, portable, de alto desempeño, "multithread"(multihilo), dinámico, sólido y seguro.

El lenguaje Java es un conjunto de herramientas orientadas a objetos que se utilizan con una sintaxis muy similar a la del lenguaje "C++" que permiten utilizar el mismo contenido fuente de programación distribuyéndolo a través de las redes de cómputo, operando en forma independiente de la plataforma y sistema operativo que se este usando.

3.2. CARACTERÍSTICAS DE JAVA.

Podemos mencionar de manera general varias características del lenguaje Java .

a) Java es Simple

Java ha sido diseñado a modo de eliminar la complejidad de otros lenguajes como "C" y "C++".

Si bien, como se dijo, Java posee una sintaxis similar a "C", con el objeto de facilitar la migración de "C" hacia Java, Java es semánticamente muy distinto a "C".

- Java no posee aritmética de punteros: La aritmética de punteros es el origen de muchos errores de programación que no se manifiestan durante la depuración y que una vez que el usuario los detecta son difíciles de resolver.

- No se necesita hacer "delete": Determinar el momento en que se debe liberar el espacio ocupado por un objeto es un problema difícil de resolver correctamente. Esto tambien es el origen de errores difíciles de detectar y solucionar.

- No hay herencia múltiple: En "C++" esta característica da origen a muchas situaciones de límite en donde es difícil predecir cuál será el resultado. Por esta razón Java opta por herencia simple que es mucho más simple de aprender y dominar.

b) Java posee bibliotecas de Clases Estándares

Toda implementación de Java debe tener las siguientes bibliotecas de clases:

- Manejo de archivos
- Comunicación de datos
- Acceso a la red Internet
- Acceso a bases de datos
- Interfaces gráficas

La interfaz de programación de estas clases es estándar, es decir, en todas ellas las operaciones se invocan con el mismo nombre y los mismos argumentos.

c) Java es Multiplataforma

Los programas en Java pueden ejecutarse en cualquiera de las siguientes plataformas, sin necesidad de hacer cambios:

- Windows/95 y /NT
- Power/Mac
- Unix (Solaris, Silicon Graphics, Linux, etc.)

La compatibilidad de Java es muy grande:

- A nivel de Fuentes: El lenguaje es exactamente el mismo en todas las plataformas.
- A nivel de bibliotecas: En todas las plataformas están presentes las mismas bibliotecas estándares.
- A nivel del código compilado: El código intermedio que genera el compilador es el mismo para todas las plataformas. Lo que cambia es el intérprete del código intermedio.

d) El "LOOK-AND-FEEL"

Lo único que varia de acuerdo a la plataforma es el "look-and-feel". Un programa en Windows/95 tendrá el aspecto característico de esta plataforma (en cuanto a la forma de los botones, barras de deslizamiento, menús, etc.). El mismo programa en Unix tendrá el aspecto característico de Motif. Y en Power/Mac se verá como un programa para Macintosh.
Sin embargo el código escrito por los programadores no tiene que tener presente las características de ninguna de estas plataformas. Es la implementación de la interfaz gráfica estándar de Java la que se encarga de desplegar las ventanas con el "look-and-feel" de la plataforma local.

e) Java es Robusto

Se dice que el lenguaje "C" es un lenguaje poco robusto porque a menudo un error de programación se traduce en un mensaje del estilo "segmentation fault". Este tipo de mensajes se origina de errores clásicos:

- Se accede un elemento de un arreglo con un índice fuera de rango.

  Ejemplo: a[-3]=5;

- Se usa un puntero como si se hiciera referencia a una estructura de tipo A, cuando en realidad en esa área de memoria hay una estructura de tipo B, incompatible con A. En el lenguaje "C" esto ocurre debido al uso de "casts".

Estos errores conducen a que tarde o temprano se use un puntero que direcciona un área de memoria no asignada por el sistema operativo. Esto es lo que detiene la ejecución con el mensaje "segmentation fault".

Lo más desagradable de este tipo de errores, es que es muy difícil determinar en qué línea del código está la verdadera fuente del error. Podría ser en cualquier parte del programa. Encontrar la línea puede llevar varios días y hasta semanas, incluso en el caso de programadores expertos.

Ahora en Java no se pueden cometer los errores mencionados:

- Java siempre revisa los índices al acceder a un arreglo.

- Java realiza chequeo de tipos durante la compilación (al igual que "C"). En una asignación entre punteros el compilador verifica que los tipos sean compatibles.Además, Java realiza chequeo de tipos durante la ejecución (cosa que "C" y "C++" no hacen) Cuando un programa usa un "cast" para acceder a

un objeto como si fuese de un tipo específico, se verifica durante la ejecución que el objeto en cuestión sea compatible con el "cast" que se le aplica. Si el objeto no es compatible, entonces se levanta una excepción que informa al programador la línea exacta en donde está la fuente del error.

- Java posee un recolector de basura que administra automáticamente la memoria. Es el recolector el que determina cuando se puede liberar el espacio ocupado por un objeto. El programador no puede liberar explícitamente el espacio ocupado por un objeto.

- Java no posee aritmética de punteros, porque es una propiedad que no se necesita para programar aplicaciones. En C sólo se necesita la aritmética de punteros para programa malloc/free o para programar el núcleo del sistema operativo.

Por lo tanto, Java no es un lenguaje para hacer sistemas operativos o administradores de memoria, pero sí es un excelente lenguaje para programar aplicaciones.

f) Java es Flexible

Pascal también es un lenguaje robusto, pero logra su robustez prohibiendo tener punteros a objetos de tipo desconocido. Lamentablemente esta prohibición es demasiado rígida. Aunque son pocos los casos en que se necesita tener punteros a objetos de tipo desconocido, las contorsiones que están obligados a realizar los programadores cuando necesitan estos punteros dan origen a programas ilegibles.

Lisp por su parte, es un lenguaje flexible y robusto. Todas las variables son punteros a objetos de cualquier tipo (un arreglo, un elemento de lista, etc.). El tipo del objeto se encuentra almacenado en el mismo objeto. Durante la ejecución, en cada operación se revisa que el tipo del objeto manipulado sea del tipo apropiado. Esto da flexibilidad a los programadores sin sacrificar la robustez. "... lamentablemente, los programas en Lisp son poco legibles debido a que al estudiar su código es difícil determinar cuál es el tipo del objeto que referencia una variable. "[11]

Java combina flexibilidad, robustez y legibilidad gracias a una mezcla de chequeo de tipos durante la compilación y durante la ejecución. En Java se pueden tener punteros a objetos de un tipo específico y también se pueden tener punteros a objetos de cualquier tipo. Estos punteros se pueden convertir a punteros de un tipo específico aplicando un cast, en cuyo caso se revisa en tiempo de ejecución que el objeto sea de un tipo compatible.

El programador usa entonces punteros de tipo específico en la mayoría de los casos con el fin de ganar legibilidad y en unos pocos casos, usa punteros a tipos desconocidos cuando necesita tener flexibilidad. Por lo tanto, Java combina la robustez de Pascal con la flexibilidad de Lisp, sin que los programas pierdan legibilidad en ningún caso.

g) Java administra automáticamente la memoria

En Java, los programadores no necesitan preocuparse de liberar un trozo de memoria cuando ya no lo necesitan. Es el recolector de basura el que determina cuando se puede liberar la memoria ocupada por un objeto.

Un recolector de basura es un gran aporte a la productividad. El porcentaje de tiempo que se gasta en determinar en el momento en que se puede liberar la memoria aumenta a medida que aumenta la complejidad del software en desarrollo. Es relativamente sencillo liberar correctamente la memoria en un programa de 1,000 líneas. Sin embargo, es difícil hacerlo en un programa de 10,000 líneas, y se puede postular que es imposible liberar correctamente la memoria en un programa de 100,000 líneas.

Para entender mejor esta afirmación, supongamos que hicimos un programa de 1,000 líneas hace dos meses y ahora necesitamos hacer algunas modificaciones. Ahora hemos olvidado gran parte de los detalles de la lógica de este programa y ya no es sencillo determinar si un puntero hace referencia a un objeto que todavía existe, o si ya fue liberado.

Peor aún, supongamos que el programa fue hecho por otra persona y evaluemos cuan probable es cometer errores de memoria al tratar de modificar ese programa.

Ahora volvamos al caso de un programa de 100,000 líneas. Este tipo de programas los desarrolla un grupo de programadores que pueden tomar años en terminarlos. Cada programador desarrolla un módulo que eventualmente utiliza objetos de otros módulos desarrollados por otros programadores. ¿Quién libera la memoria de estos objetos? ¿Cómo se ponen de acuerdo los programadores sobre cuándo y quién libera un objeto compartido?

Sería inevitable que la fase de prueba dejará pasar errores en el manejo de memoria que sólo serán detectados más tarde por el usuario final.

Se puede concluir:

- Todo programa de 100,000 líneas que libera explícitamente la memoria tiene errores latentes.
- Sin un recolector de basuras no hay verdadera modularidad.
- Un recolector de basura resuelve todos los problemas de manejo de memoria en forma trivial.

Aquí cabría la siguiente pregunta: ¿Cuál es el impacto de un recolector de basura en el desempeño de un programa?

Para contestarla, podríamos afirmar lo siguiente: existe un sobrecosto para el sistema al tener que hacer llamadas a un recolector de basura a diferencia de que el mismo programa implementara en su mismo código un liberador de memoria, pero aún así ésta sobrecarga no es de gran impacto en la eficiencia del sistema.

Este sobrecosto no es importante si se considera el incremento periódico en la velocidad de los procesadores. El impacto de un recolector de basura en el tiempo de desarrollo y en la confiabilidad del software resultante es mucho más importante que la pérdida en eficiencia.

Tratemos ahora un poco la historia de este lenguaje, sabiendo de donde vino y así comprender mejor sus alcances y limitaciones.

3.3. HISTORIA DEL LENGUAJE JAVA.

En Diciembre de 1990, Patrick Naughton, un empleado de la empresa Sun, reclutó a sus colegas James Gosling y Mike Sheridan para trabajar sobre un nuevo tema conocido como "El proyecto verde". Este a su vez estaba auspiciado por la compañía "Sun founder Bill Joy" y tenía como objetivo principal crear un lenguaje de programación accesible, fácil de aprender y de usar, que fuera universal, y que estuviera basado en un ambiente C++ ya que habia mucha frustración por la complejidad y las limitaciones de los lenguajes de programación existentes.

En abril de 1991, el equipo decidió introducir sistemas de software con aplicaciones para consumidores smart como plataforma de lanzamiento para su proyecto. James Gosling escribió el compilador original y lo denominó "Oak", y con la ayuda de los otros miembros del equipo desarrollaron un decodificador que más tarde se convertiría en lenguaje Java.

Para 1992, el equipo ya había desarrollado un sistema prototipo conocido como "7", que era una especie de cruce entre un asistente digital personalizado y un mecanismo inteligente de control remoto.

Por su parte el presidente de la compañía Sun, Scott McNealy, se dio cuenta en forma muy oportuna y estableció el Proyecto Verde como una subsidiaria de Sun. De 1993 a 1994, el equipo de Naughton se lanzó en busca de nuevas oportunidades en el mercado, mismas que se fueron dando mediante el sistema operativo base. La incipiente subsidiaria fracasó en sus intentos de ganar una oferta con Time-Warner; sin embargo, el equipo concluyó que el mercado para consumidores electrónicos smart y las cajas Set-Up en particular, no eran del todo eficaces. La subsidiaria Proyecto Verde fue amortizada por la compañía Sun a mediados del año de 1994.

Afortunadamente, el cese del Proyecto Verde coincidio con el nacimiento del fenómeno mundial Web. Al examinar las dinámicas de Internet, lo realizado por el ex equipo verde se adecuaba a este nuevo ambiente ya que cumplia con los mismos

requerimientos de las set-top box OS que estaban diseñadas con un código de plataforma independiente pero sin dejar de ser pequeñas y confiables.

Patrick Naugthon procedió a la construcción del lenguaje de programación Java que se accionaba con un browser prototipo, más tarde se le fueron incorporando algunas mejoras y el browser Hot Java fue dado a conocer al mundo en 1995.

Con el paso del tiempo el Hot Java se convirtió en un concepto práctico dentro del leguaje Java y demostró que podría proporcionar una forma segura multiplataforma para que el código pueda ser bajado y corrido desde un Host remoto del World Wide Web.

Una de las características más atractivas del Hot Java fue su soporte para los "applets", que son las partes del código Java que pueden ser cargadas mediante una red de trabajo para después ejecutarlo localmente y así lograr alcanzar soluciones dinámicas en computación, acordes al rápido crecimiento del ambiente Web.

Para dedicarse al desarrollo de productos basados en la tecnología Java, Sun formó la empresa Java Soft en enero de 1996, de esta forma se dio continuidad al fortalecimiento del lenguaje y el trabajar con terceras partes para crear aplicaciones, herramientas, sistemas de plataforma y servicios para aumentar las capacidades del lenguaje.

Durante ese mismo mes, Java Soft dio a conocer el Java Developmet Kit (JDK) 1.0, una rudimentaria colección de componentes básicos para ayudar a los usuarios de software a construir aplicaciones de Java. Dicha colección incluía el compilador Java, un visualizador de applets, un debugger prototipo y una máquina virtual Java (JVM), necesaria para correr programas basados en Java, también incluía paquetería básica de gráficos, sonido, animación y trabajo en red.

Asimismo, el Netscape Comunications Inc, mostró las ventajas de Java y rápidamente se asoció con Java Soft para explotar su nueva tecnología. No pasó mucho tiempo antes de que Netscape Communications decidiera apoyar a los Java applets en Netscape Navigator 2.0. Este fue el factor clave que lanzó a Java a ser reconocido y famoso, y que a su vez forzó a otros vendedores para apoyar el soporte de applets en Java.

Como parte de su estrategia de crecimiento mundial y para favorecer la promoción de su nueva tecnología, Java Soft otorgó permisos a otras compañías para que pudieran acceder al código fuente de Java y al mismo tiempo mejorar sus navegadores, dicha licencia también les permitía crear herramientas de desarrollo para programación Java y los facultaba para acondicionar Máquinas Virtuales Java (JVM), a varios sistemas operativos.

Muy pronto las licencias o permisos contemplaban a prestigiadas firmas como IBM, Microsoft, Symantec, Silicon Graphics, Oracle, Toshiba y Novell, entre otras.

Desde su aparición, Java ha ganado una impresionante cantidad de apoyo. Virtualmente cada vendedor importante de software ha obtenido autorización de Java y ahora ha sido incorporado en los principales sistemas operativos base de "PC's" de escritorio hasta estaciones de trabajo UNIX.

Los applets Java (basados en JDK 1.02) son apoyados por los dos más populares navegadores web (Netscape Navigator y Microsoft Internet Explorer , I.B.M./Lotus, Computer Asociates, Symantec, Informix, Oracle, Sybase y otras poderosas empresas de software están construyendo Software 100% puro JAVA.

Los nuevos proyectos de Java son co-patrocinados por cientos de millones de dólares en capital disponible de recursos tales como la Fundación Java, un fondo común de capital formado por varias compañías, incluyendo Cisco Systems, IBM, Netscape y Oracle.

Las escuelas y Universidades alrededor del mundo están adoptando Java como un lenguaje universal y de enseñanza indispensable.

Para darse una idea de la rápida aceptación que tiene Java en el mundo, tenemos el ejemplo de las conferencias "Java Soft Java One" en San Francisco, el primer Java One fue celebrado en abril de 1996 y atrajo a 5000 usuarios, un año después, en la segunda conferencia Java One albergó a 10,000 usuarios, asistentes. Java Soft estima que el número actual de usuarios Java llega a 400 mil y sigue creciendo. Java también está ganando aceptación en el área empresarial, se ha estimado que actualmente las compañías de hoy cuentan con más de 5000 usuarios.

A manera de tabla cronológica, la historia de Java puede verse en el anexo

3.4. CONCLUSIONES DE ESTE CAPITULO.

Java es un lenguaje que ha sido diseñado para producir software:

- Confiable: Minimiza los errores que se escapan a la fase de prueba.
- Multiplataforma: Los mismos códigos binarios funcionan correctamente en Windows/95 y /NT, Unix/Motif y Power/Mac.
- Seguro: Applets recuperados por medio de la red no pueden causar daño a los usuarios.
- Orientado a objetos: Beneficioso tanto para el proveedor de bibliotecas de clases como para el programador de aplicaciones.

- Robusto: Los errores se detectan en el momento de producirse, lo que facilita la depuración.

Java es un lenguaje con características útiles para el manejo de información a través de Internet debido a su condición de multiplataforma, orientación a objetos y soporte de protocolos de comunicación para este tipo de red. Aun así, podría pensarse que Java genera inherentemente sobrecargas de tiempo debido a las capas de software del mismo lenguaje que se tiene que pasar antes de que un programa sea cargado y corrido en una máquina cliente.

CAPITULO 4

XML, EL LENGUAJE UNIVERSAL

4.1. LENGUAJES DE MARCAS

En los años sesentas, IBM intentó resolver sus problemas con relación al tratamiento de documentos en diferentes plataformas a través de lo que denominó GML ("Generalized Markup Language", lenguaje de marcas generalizado).

El problema consistía en que cada aplicación utiliza sus propias marcas para describir las entidades de los documentos. Las "marcas" son los códigos que indican al programa cómo debe tratar su contenido, así, si se desea que un texto aparezca en cursiva, cada aplicación introduce al principio y al final del texto correspondiente una marca que le permita mostrarlo en pantalla e imprimirlo de esta forma. Lo mismo ocurre con las tablas, los márgenes, las imágenes, los tipos de letra, los enlaces, etc.

Los usuarios de WordPerfect conocen este sistema de marcas, ya que las aprovechan desde las primeras versiones del programa para controlar a la perfección sus documentos. Word también permite tratar algunas de las marcas que utiliza, aunque son pocas las que en realidad utiliza internamente. No obstante, las marcas que utiliza cada sistema propietario son intratables directamente desde el código interno del archivo que describe el documento.

Conocer las marcas que utiliza cada programa de tratamiento de documentos hace posible diseñar filtros que permiten pasar la información de unos formatos de marcas a otros sin perder el diseño. La forma que IBM ideó para solventar este problema se basaba en tratar documentos con marcas desde documentos TXT, o sea, en código ASCII, haciendo así posible su tratamiento desde cualquier sistema.

Más tarde, GML cayó en manos de ISO, que lo convirtió en un estándar oficial en los años 80's (ISO 8879), denominándose SGML (Standard Generalized Markup Language, lenguaje de marcas generalizado estándar). Esta norma de carácter general se aplica desde entonces para diseñar lenguajes de marcas específicos, cuyos ejemplos más conocidos son el HTML y el RTF (Rich Text Format, formato de texto enriquecido).

A finales de los años 80's, cuando Tim Berners-Lee aplicó las normas del SGML para diseñar el HTML como solución para publicar las investigaciones de muy diversas fuentes y autores que se producían en el CERN.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación, aunque se definan igualmente como "lenguajes". Son sistemas complejos de descripción de información, normalmente documentos, que si se ajustan a SGML, se pueden controlar desde cualquier editor ASCII. Las marcas más utilizadas suelen describirse por textos descriptivos encerrados entre signos de "menor" (<) y "mayor" (>), siendo lo mas usual que existan una marca de principio y otra de final. así por ejemplo, la siguiente marcación:

<NOMBRE>Duna</NOMBRE>

Deja bien claro que "Duna" no es la descripción de una acumulación de arena, sino que corresponde a un nombre, y de paso, nos indica con claridad el principio <NOMBRE> y el final </NOMBRE> de la marcación.
Existe la posibilidad de aprovechar la etiqueta para incluir atributos internos, ahorrando así otras etiquetas:

<LIBRO Nº="1">El Quijote</LIBRO>

La utilización del atributo "Nº" nos permite evitar que sea más complejo:

<LIBRO>
<Nº>1</Nº>
<TITULO>El Quijote</TITULO>
</LIBRO>

La importancia de "marcar" adecuadamente la información se puede ver con un simple ejemplo. Si se nos presenta la siguiente información:

Kishons beste Familiengeschichten
Satiren
Das wunderkind
Ich liebe es, auf parkbanken zu sitzen, aber nur in winter

Seguramente tendremos problemas para entender de qué se trata, aunque sepamos alemán.

Ahora bien, si se nos presenta la información de la siguiente forma:

<LIBRO>
<TITULO> Kishons beste Familiengeschichten </TITULO>
<SERIE>Satiren</SERIE>
<CAPITULO ORDEN="24"> Das wunderkind </CAPITULO>
<PARRAFO ORDEN="1"> Ich liebe es, auf parkbanken zu sitzen, aber nur in winter
</PARRAFO>
</LIBRO>

Es muy fácil comprender de qué se trata la información, aunque no sepamos alemán y no entendamos los detalles.

Se puede decir que existen tres utilizaciones básicas de los lenguajes de marcas

- Los que sirven principalmente para describir su contenido
- Los que sirven más que nada para definir su formato

- Los que realizan las dos funciones indistintamente.

Las aplicaciones de bases de datos son buenas referencias del primer sistema, los programas de tratamiento de textos son ejemplos típicos del segundo tipo, y aunque no lo parezca, el HTML es la muestra más conocida del tercer modelo.

A mayor abundamiento, HTML es originariamente un lenguaje subconjunto de SGML, especializado en la descripción de documentos en pantalla a través de marcas "tags" (etiquetas).

El proyecto inicial se basaba en una colección de etiquetas que permitían describir documentos de texto y vínculos de hipertexto que permitían desplazarse entre diferentes documentos, siempre con independencia de la máquina. Conociendo las normas de actuación de estas etiquetas y disponiendo de un sencillo editor ASCII de textos, se pueden confeccionar fácilmente documentos HTML.

La facilidad de uso y la particularidad de que no es propiedad de nadie, hizo al HTML el sistema idóneo para compartir información en Internet. La expansión de Internet le ha dado una posición de privilegio y ha hecho que la idea inicial se halla modificado considerablemente.

En principio, la intención de HTML era que las etiquetas fueran capaces de marcar la información de acuerdo con su significado, sin importar cómo se mostraban en la pantalla. Lo importante era el contenido y no la forma, o sea, que era un lenguaje de marcas orientado a describir los contenidos. En otras palabras: el título del documento, los títulos de los apartados, el autor del documento, los textos resaltados, etc., eran marcados por las etiquetas TITLE, ADDRESS, STRONG, etc., dejando a cada visualizador (browser) la tarea de formatear el documento según su criterio.

Esto daba lugar a que una aplicación podía presentar una etiqueta H2 como texto centrado, con tamaño de 20 y color rojo, mientras otra lo podía mostrar alineado a la izquierda, con tipo de letra Arial de 16 y color Azul, por ejemplo. Una etiqueta STRONG podría tomarse como cursiva, negrita o color verde, por ejemplo, según la interpretase el visualizador empleado.

Esto producía presentaciones diferentes, pero permitía controlar fácilmente su contenido. Si una persona o un motor de búsqueda quería conocer el título del documento, el autor de la página o las cabeceras de los capítulos, siempre buscaba en el código las etiquetas TITLE, ADDRESS. Además, si a alguien no le gustaba la idea de dejar a cada aplicación la decisión de cómo mostrar el contenido de las etiquetas, siempre le quedaba la posibilidad de controlar el formato del documento con descripciones particulares, como es el caso de las hojas de estilo en cascada (CSS).

Por diversos motivos, los creadores de los navegadores fueron añadiendo más etiquetas HTML dirigidas a controlar la presentación, como FONT !, CENTER, xCOLOR, etc. y los usuarios las utilizaron para que sus documentos estuviesen perfectamente

formateados, sin permitir diferencias importantes entre visualizadores distintos, por lo que HTML pasó a ser un lenguaje de marcas más dirigido al control de la presentación. Ahora es más difícil encontrar al autor o las cabeceras de los capítulos de un documento, pues todos los textos se describen con P y FONT, sobre todo si se utilizan los editores WYSIWYG ("What You See What You Get") -Microsoft FrontPage, Netscape Composer- que proliferan por donde quiera.

Si a esto le añadimos que para facilitar la vida a los usuarios, los analizadores sintácticos de las marcas que incluyen los navegadores permitieron saltarse algunas normas sin que el propio usuario lo notase (por ejemplo, permiten trabajar solo con la etiqueta <P>, cuando lo correcto es que se necesite las etiquetas de principio y de final: <P> y </P>), dio como resultado que HTML ya no es un lenguaje que sigue las normas estrictas del SGML.

"... al llegar al un punto en el que HTML dejó de servir para su función inicial, no le quedo más remedio al Consorcio World Wide Web (W3C) que la descripción de un nuevo subconjunto del SGML que sirva para describir contenidos de documentos, al que ha denominado XML, publicando las especificaciones de la versión 1.0 en 1998. Se puede decir por tanto que XML es más un "SGML light" que un "HTML plus"."*12

4.2. ¿QUÉ ES XML?

En primer lugar hay que olvidarse un poco de HTML para entender mejor XML. En la situación actual, en teoría, HTML es un subconjunto de XML especializado en presentación de documentos para la Web, mientras que XML es un subconjunto de SGML especializado en gestión de información para la Web. En la práctica, HTML está un poco dentro de XML (que a su vez es parte de SGML) y otro poco fuera de SGML. Para reconducir esta situación, el grupo W3C ha dictado reglas expresas para distinguir el HTML que sigue estrictamente las normas de XML, denominándolo XHTML (eXtendible HyperText Markup Language), que no es más que una reformulación de HTML 4 dentro de las normas de XML.

La particularidad más importante del XML es que no posee etiquetas prefijadas con anterioridad, ya que es el propio diseñador el que las crea a su antojo, dependiendo del contenido del documento. De esta forma, los documentos XML con información sobre libros tienen etiquetas como <AUTOR>, <EDITORIAL>, <Nº_DE_PÁGINAS>, <PRECIO>, <ISBN>, etc., mientras que los documentos XML relacionados con educación incluyen etiquetas del tipo de <ASIGNATURA>, <ALUMNO>, <CURSO>, <NOTA>, etc.

Esta característica proporciona una gran facilidad de control de cualquier documento, así por ejemplo, la información incluida por un código típico HTML como el siguiente:

Otra cuestión que hay que aclarar es que XML es más un sistema complejo de tratamiento de información que un simple lenguaje de descripción. XML es más una familia de lenguajes, y al igual que podemos decir que HTML es un "lenguaje", para ser exactos, hay que definir al XML como un "metalenguaje", o sea, un lenguaje capaz de definir otros lenguajes. Veamos esto con más detalle.

4.3. DTD/ESQUEMA.

Cualquier diseñador de HTML está totalmente despreocupado de las interioridades del sistema que hacen posible la creación de los archivos HTML (o HTM) cuando utiliza algún editor de los que genera el código de forma automática. Es suficiente con realizar el documento de forma similar a como se hace con un programa de tratamiento de textos, activar la opción de "Guardar como HTML" y ya está. A partir de entonces, cualquiera que abra el archivo desde un navegador, verá dicho documento en su pantalla.

Esto que parece tan sencillo (gracias al esfuerzo realizado por las empresas creadoras de los navegadores), exige que se estén manejando varias herramientas internas al sistema.

Cuando se genera el documento en un editor HTML, el código HTML que se crea está basado en un DTD (Document Type Definition, definición de tipo de documento) interno, que es una definición de las normas que regulan la formación de las etiquetas de un lenguaje de marcas determinado, en este caso, el HTML. Por lo tanto, en el DTD que integra cada editor de HTML se especifican todas las etiquetas existentes, sus atributos, sus valores, etc., haciendo posible que se vayan integrando en el código del documento únicamente de acuerdo con dichas normas. Se puede decir que un DTD es una definición exacta de la gramática de un documento, con la misión de que se genere el código adecuado sin errores.

El fichero HTML creado con el editor, al cargarse en un navegador, es vuelto a analizar por su DTD interno para descubrir las etiquetas correctas y las equivocadas, siempre de acuerdo a sus normas internas. Por este motivo, si se crea una página web con el editor de una empresa y se visualiza con el navegador de otra, puede ocurrir que algunas entidades no se vean correctamente (o incluso puede pasar que no se puedan controlar), bien porque no existan dichas etiquetas (casos de BLINK o MARQUEE), o bien porque no se gestionen con las mismas normas (caso de TABLE).

En XML no existen DTD's predefinidas, por lo que es labor del diseñador especificar su propia DTD para cada tipo de documento XML. En la especificación de XML se describe la forma de definir DTD's particularizados para documentos XML, que pueden ser internos (cuando van incluidas junto al código XML) o externos (si se encuentran en un archivo propio).

Para el ejemplo anterior del documento XML, se podría crear un archivo conteniendo el siguiente código:

```
<!ELEMENT LIBROS (LIBRO)+>
<!ELEMENT LIBRO (TITULO,AUTOR,PRECIO)>
<!ELEMENT TITULO (#"PC"DATA)>
<!ELEMENT AUTOR (#"PC"DATA)>
<!ELEMENT PRECIO (#"PC"DATA)>
```

En el DTD del ejemplo se definen los elementos (!ELEMENT) que integran el documento XML. En la primera línea se indica que el elemento principal es LIBROS, del que dependen uno o más (+) elementos LIBRO. La segunda línea sirve para especificar que el elemento LIBRO contiene tres elementos (TITULO,AUTOR,PRECIO) que se deben marcar en dicho orden. Las restantes líneas aclaran que los elementos TITULO, AUTOR y PRECIO contienen valores de cadenas de texto.

Este sencillo DTD se guarda como "libros.dtd" (también se puede integrar dentro del código del propio documento XML) y es llamado desde el documento XML con la siguiente línea:

```
<!DOCTYPE LIBROS SYSTEM "libros.dtd">
```

Que cada usuario pueda crear su propia DTD es una gran ventaja, ya que proporciona total libertad de adecuación a cada documento, pero también puede suponer un grave inconveniente, ya que es muy fácil que para documentos de un mismo sector (arquitectura, edición, educación, etc.), existan muy variadas DTD's, haciendo muy difícil su manejo por usuarios distintos a los que hayan diseñado la información.

"... En la actualidad se están definiendo DTD's por grupos sectoriales con intereses similares, de forma que existirán DTD's estándares avalados por asociaciones de empresas y organismos que garanticen que cualquier usuario que las adopte como suyas, trabaje con las mismas etiquetas (de forma similar al actual HTML)... como ejemplos de estas DTD's estándares tenemos: CDF - Channel Definition Format (define canales para enviar información periódica a los clientes), CML - Chemical Markup Language (define información del sector químico), MathML - Mathematical Markup Language (define datos matemáticos), SMIL - Synchronized Multimedia Integration Language (define presentaciones de recursos multimedia)..."[13]

Últimamente, se está imponiendo otra forma más eficaz de definición de elementos conocida como "esquema" (schema), que de forma sencilla se puede definir como un DTD que permite su ampliación mediante un lenguaje de definición de esquemas. Se pueden ver ejemplos de esquemas en el código XML que añade Office 2000 al principio de sus documentos.

El navegador, después de revisar sintácticamente el código del archivo, debe presentar la información del documento con un formato determinado. Los documentos HTML utilizan las descripciones de formatos internas del propio navegador, o si existen descripciones CSS (que son opcionales), utilizan la

información de la hoja de estilo para ajustar la presentación en la pantalla. Los documentos XML siempre necesitan normas que describan su presentación, por lo que el paso siguiente obliga a que hablemos de este tema.

4.4. CSS/XSL.

Para describir cómo se deben presentar los documentos XML podemos optar por dos soluciones: las mismas descripciones CSS que se utilizan con HTML y/o las descripciones que se basan en XSL (Extensible Stylesheet Language, lenguaje de hojas de estilo extensible).

CSS (Cascading Style Sheets, hojas de estilo en cascada) es muy bien conocido por todos los diseñadores profesionales de HTML. Las posibilidades de las especificaciones CSS, se deben entender como una descripción del formato en el que se desea que aparezcan las entidades definidas en un documento.

Por ejemplo, si se define una hoja de estilo ligada con un archivo HTML con el siguiente código:

```
P {font-family:Verdana; font-size:10 pt}
TABLE {border:2; font-family:Tahoma; font-size:9 pt}
H3 {font-family:Comic Sans MS; font-size:12 pt; color:blue}
```

Se indica al navegador que presente los textos incluidos entre <P> y </P> con un tipo de letra Verdana de tamaño 9, las tablas con una fuente de letra Tahoma de 9 y un ancho de 2 en los bordes, y los titulares incluidos entre <H3> y </H3> con una letra Comic Sans MS de tamaño 12 y color azul.

Utilizar CSS con XML es similar, con la excepción de que las etiquetas son diferentes a las de HTML. Un código como el siguiente:

```
AUTOR {display:block; font-family:Arial; font-size:small; width:30em}
PRECIO {display:block; padding:1.2em; font-size:x-small}
TITULO {display:block; font-size:x-large; text-align:center; color:#888833}
```

Sería perfectamente válido para que los datos de las etiquetas <AUTOR>, <PRECIO> y <TITULO> se presentasen según su descripción.

Aquí cabría la pregunta, si ya existe una forma de definir las presentaciones de los documentos web, ¿por qué se ha sacado a la luz otra (XSL), específica para XML?

La respuesta es que CSS es eficaz para describir formatos y presentaciones, pero no sirve para decidir qué tipos de datos deben ser mostrados y cuáles no deben salir en la pantalla. Esto es, CSS se utiliza con documentos XML en los casos en los que todo su contenido debe mostrarse sin mayor problema.

XSL no solo permite especificar cómo queremos presentar los datos de un documento XML, sino que también sirve para filtrar los datos de acuerdo a ciertas condiciones. Se parece un poco más a un lenguaje de programación.

No trataremos las posibilidades del XSL, siendo suficiente con entender que además de permitir la descripción de la presentación física, también posibilita la ejecución de bucles, sentencias del tipo IF... THEN, selecciones por comparación, operaciones lógicas, ordenaciones de datos, utilización de plantillas,..., y otras cuestiones similares.

Para tener una pequeña idea de cómo es un código XSL, a continuación se muestra un sencillo ejemplo que permitiría mostrar todos los contenidos de las etiquetas <TITULO>, <AUTOR> y <PRECIO> de un documento XML, mediante un bucle sin condiciones:

```
<xsl:template match="/">
<HTML>
<BODY>
<xsl:for-each select="/LIBROS/LIBRO">

Título:
<xsl:value-of select="TITULO"/><BR/>


Autor:
<xsl:value-of select="AUTOR"/><BR/>

Precio:
<xsl:value-of select="PRECIO"/> [illegible]<BR/>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
```

Podríamos comentar que un estándar basado en SGML que regula las normas de presentación de documentos de marcas para la Web se denomina DSSSL (Document Style Semantics & Specification Language, lenguaje de especificación y semántica de estilo de documentos), del que se puede decir que XSL es un subconjunto, siempre cumpliendo las normas del XML.

Una de las características de todo documento web es la inclusión de enlaces de todo tipo: a imágenes, a sonidos, a videos, a otros párrafos, a otros documentos, etc. En HTML la cuestión está resuelta, ya que existen etiquetas para cada caso, pero ya sabemos que en XML todo está por hacer, así que vamos a ver cómo se diseñan los enlaces de forma particular.

4.5. XLINK/XPOINTER.

La cuestión de los enlaces e hipervínculos es tan importante para los documentos XML, que el W3C ha sacado las especificaciones que las controlan fuera de las descripciones del DTD, no conformándose con crear una sola norma, ya que existen dos: XLink y XPointer.

XLink (anteriormente conocido como XLL, Extensible Linking Language) define la forma en la que los documentos XML deben conectarse entre sí. XPointer describe cómo se puede apuntar a un lugar específico de un determinado documento XML. Resumiendo, XLink determina el documento al que se desea acceder y XPointer marca el lugar exacto de dicho documento.

A los diseñadores de HTML les puede parecer que esto es una complicación sin sentido, ya que están acostumbrados a las pocas posibilidades que brinda la etiqueta <A>, pero si pensamos en las muchas variaciones que pueden tener los vínculos, se comprende la solución adoptada.

Las especificaciones de los hipervínculos para XML permiten cosas como: adherirse a cualquier etiqueta, hacer referencia a un lugar concreto de un documento determinado a través de su nombre o localización, ser descritos en documentos externos, ser procesados de formas distintas (automáticamente, activándolo, etc.) multifuncionales (permitir varios saltos), etc.

Al contrario de lo que ocurre con HTML, en XML existen dos tipos básicos de hipervínculos: simples y extendidos.

Un ejemplo de hipervínculo simple sería:

```
<AUTOR xlink:href="autores.xml#juan" xlink:show="new">
<NOMBRE>Juan Primero Segundo</NOMBRE>
</AUTOR>
```

Un ejemplo de hipervínculo extendido podría ser:

```
<EDITOR_AUTOR xlink:extended>
<xlink:locator href="#ana" id="editor"/>
<xlink:locator href="autores.xml#juan" id="autor"/>
<xlink:arc from="editor" to="autor" show="replace"/>
</EDITOR_AUTOR xlink:extended>
```

En el primero se puede observar la definición de un hipervínculo simple que se abre en una nueva ventana (show="new"), mientras que en el segundo se definen un hipervínculo con tres posibilidades diferentes: a una sección determinada del

documento (#ana), o a un determinado lugar de otro documento (autores.xml#juan), o a una zona delimitada por dos marcadores
(editor y autor). El funcionamiento de un hipervínculo simple no tiene secretos, pues es similar al que se utiliza en HTML.

Aunque no se han explicado todas las posibilidades de los hipervínculos XML, sí debe quedar claro que los enlaces XML son más variados que los que nos proporciona la sencilla, útil y conocida etiqueta <A> del HTML.

4.6. PARSER/DOM.

Siguiendo con el proceso que se desarrolla en el interior del navegador, después de recoger la información de todos los documentos que definen la información XML, se genera internamente una estructura que organiza a los elementos que describen las etiquetas en forma de árbol jerárquico, lo que facilita el control de dichos elementos.

Estos árboles presentan ramas expansibles (+) y contraíbles (-) a través de pulsaciones del ratón.
En caso que se detecte algún error incompatible con las estrictas normas XML, el navegador interrumpe el proceso y muestra dicho error en la pantalla
Todo este proceso se puede realizar gracias al analizador (parser) interno que incluye cada navegador, que en la mayoría de los casos se relaciona directamente con el estándar DOM (Document Object Model, modelo de objeto de documento), que entre otras cosas, permite acceder a cada nodo del árbol a través de scripts.

Las reglas mínimas que hay que cumplir para no ser rechazados por un analizador XML son:

- Solo se permite un elemento raíz. Es imprescindible cumplir esta norma para que el parser pueda saber que el documento está completo. Es el equivalente a la etiqueta .

  <HTML> del HTML.

  Ejemplo correcto:

  <DISCOS> <DISCO> ... </DISCO> <DISCO> ... </DISCO>
  <DISCO>... </DISCO> </DISCOS>

  Ejemplo incorrecto:

  <DISCO> ... </DISCO> <DISCO> . . </DISCO>
  <DISCO> ... </DISCO>

- Incluir etiquetas de inicio (sin la "barra") y final (con la "barra") para todos los elementos. Siempre debe existir una marca <ETIQUETA> y otra </ETIQUETA> antes y despues de cada elemento. No se permiten casos de

etiquetas"sueltas" como las <P> o <LI> del HTML.

Ejemplo correcto:

<CIUDAD>Atenas</CIUDAD>

Ejemplo incorrecto:

Atenas<CIUDAD>

- En caso de trabajar con etiquetas "vacías", hay que incluir la "barra" (/) antes del signo "mayor" (>). Cuando se trata con elementos que no se desean marcar con etiquetas de principio y de final (normalmente llevan atributos), se pueden utilizar marcas únicas del tipo <ETIQUETA />. Serían los casos típicos de <HR> o
 del HTML.

Ejemplo correcto:

<ALUMNO NÚMERO="1" />

Ejemplo incorrecto:

<ALUMNO NÚMERO="1">

- Hay que anidar las etiquetas correctamente. Las etiquetas anidadas deben situarse en el mismo orden de apertura que de cierre: <ETIQUETA1> ... <ETIQUETA2> ..<ETIQUETA3> ... </ETIQUETA3> ... </ETIQUETA2>...</ETIQUETA1>.

No se permiten códigos como:
<B><I>Texto de prueba</B></I>, permitidos en HTML .

Ejemplo correcto:

<ROJO>Color rojo <CLARO> claro
</CLARO>normal</ROJO>

Ejemplo incorrecto:

<ROJO>Color rojo <CLARO> claro </ROJO></CLARO>
normal

- Es obligatorio que los valores de los atributos vayan entre comillas. Las etiquetas con atributos deben marcar sus valores entre comillas: <ETIQUETA ATRIBUTO="x"/>. En HTML suele ser optativo.

  Ejemplo correcto:

  <LIBRO ISBN="84-415-0845-3" />

  Ejemplo incorrecto:

  <LIBRO ISBN=84-415-0845-3 />

- XML distingue entre mayúsculas y minúsculas. Puede utilizar los dos tipos de letra, pero no las considera iguales. Al contrario de HTML, que no se preocupa de que sus etiquetas estén en mayúsculas, minúsculas o mezcladas, aceptando sin problemas etiquetas como <tAbLe>, <table> o <TABLE>, en XML se produciría un error.

  Ejemplo correcto:

  <AMIGOS> <amigo> Pepe </amigo> </AMIGOS>

  Ejemplo incorrecto:

  <AMIGOS> <amigo> Pepe </AMIGO> </AMIGOS>

Cumpliendo estrictamente las reglas anteriores (existen otras reglas menores) se consigue lo que se denomina un documento XML "bien formado" (well-formed), pero si el documento XML dispone de un DTD de referencia (puede no tenerlo), y cumple también sus descripciones, se dice que es un documento XML "válido".

Una vez vistos los estándares principales, las normas más importantes que se precisan para trabajar con los documentos XML, y dejado claro que XML es un sistema de archivos de descripción de información, rigurosa y libre, se tratarán algunas aplicaciones.

4.7. APLICACIONES DE XML.

XML se puede trabajar ya en el WEB, aunque dado que es una normativa nueva, no todas las herramientas relacionadas con Internet son capaces de trabajar con XML, pero según van actualizándose (Office, Oracle, Explorer), adoptan sus recomendaciones.

Si pensamos que a través del XML se puede crear cualquier tipo de documento y manejar datos del lado del cliente, sin la necesidad de motores especiales en el lado

del servidor, tendremos la respuesta a cualquier duda que tenga respecto al futuro del XML.

En realidad, con un editor de textos ASCII (el bloc de notas, por ejemplo) y con un visualizador que incluya un parser adaptado a XML (Microsoft Internet Explorer 5, por ejemplo), se pueden crear y ver documentos XML.

No obstante, existen muchas nuevas herramientas que facilitan un poco la labor del diseñador, algunas de ellas gratuitas.

Existen otros varios visualizadores / navegadores que admiten XML, como Amaya, HotMetal o HyBrick, aunque casi cada día están saliendo nuevos productos (analizadores, visualizadores, motores, editores, DTD's, etc.).

Aunque la versión 1.0 de XML es ya definitiva, no pasa lo mismo con las demás normativas que le acompañan, que poco a poco van pasando del estado de "borrador de trabajo" al de "recomendación". Este es otro de los motivos por los que XML no termina de generalizarse, ya que las empresas analizan demasiado en invertir en productos que no están soportados por estándares definitivos.

4.8. CONCLUSIONES DE ESTE CAPITULO.

Hay varias cuestiones sobre el XML que deben quedar claras:

- La existencia de XML no implica que desaparecerá el HTML. Dado que la mayor parte de las páginas web son documentos de texto con algunas imágenes, HTML seguirá siendo el medio más eficaz para crearlas y publicarlas, aunque muy probablemente se tenderá a utilizar la versión XHTML.

- XML puede convivir con los restantes lenguajes, tales como HTML, JAVA, JavaScript, VBScript, Visual Basic, etc. En muchos de estos casos, XML hará las veces de "base de datos", donde los scripts servirán para realizar búsquedas selectivas y el HTML permitirá que se muestre la información resultante en la pantalla.

- Hasta que no se publiquen todas las recomendaciones oficiales de los estándares que complementan a la de XML, no "explotará" la publicación de páginas XML en Internet, ya que todavía se modificarán algunas cuestiones importantes.

- XML se utiliza principalmente por los profesionales, debido a la complejidad de las especificaciones, pero según vayan saliendo al mercado editores WYSIWYG que permitan conseguir documentos XML, se podrá utilizar por

cualquier persona, de forma similar a como se trabaja con los editores de textos actuales.

- Para evitar que no existan innumerables formas de describir documentos de un determinado sector, ya se está trabajando en la definición de DTD's sectoriales de carácter público que estén arropados por el máximo de empresas y organismos posibles. Según se vayan publicando, se crearán nuevas herramientas para su tratamiento.

- No cabría mucha duda que dentro de poco, XML será el lenguaje que nos garantizará el intercambio de cualquier información, sin que ocasione problemas del tipo "contenido" o de tipo "presentación". Al saber lo que es XML y las características que ofrece para el manejo de datos, ahora regresaremos a Java y evaluaremos el uso del lenguaje para peticiones cliente servidor a través de Internet y se comparara con otras soluciones.

CAPITULO 5

SERVLETS Y CGI´S

5.1. INTRODUCCIÓN.

Los CGIs y los servlets de Java son actualmente dos de las formas más populares para desarrollar proyectos relacionados con bases de datos y el Web. Aunque estas dos formas funcionan de forma muy similar, las soluciones a través de los servlets de Java proveen su propia forma de mantener conexiones a bases de datos usando un almacenamiento en memoria rápida llamada "connection pool" y procesando múltiples solicitudes de servicio a través del manejo de múltiples hilos.

Estas dos características son los puntos más importantes de Java sobre los CGI y marca una gran diferencia, en el desempeño de aplicaciones Web y bases de datos. Estas características son parte de las soluciones de Java y dan la opción de remplazar a los CGI en un futuro muy cercano. En este capítulo se investiga el diferente desempeño entre los CGI y servlets. Tomaremos los resultados de Benchmarks realizados por Amanda Wanhong Wu realizados en la universidad de Mississippi y que puede ser encontrado en la red en la forma de un papper.

Los CGI´s han sido la forma más tradicional de manejar aplicaciones entre bases de datos y Web. El desempeño de programas CGI es estable y estos ha sido una buena opción para desarrollar proyectos Web.

El principal problema con la técnica de CGI es la ineficiencia en el manejo de la sobrecarga de peticiones concurrentes por varios usuarios. Y esto ha limitado el uso de estos en varias aplicaciones. Ahora, un número de soluciones alternativas son posibles con otras aplicaciones como: los servlets de Java, Perl (Practical extration and report languages) y PHP .

Entre todas estas soluciones los servlets de Java han llegado a ser los más usados para aplicaciones de comercio electrónico. Los servlets tiene características que proveen una fuerte ayuda para afinar el desempeño de aplicaciones Web. Como se explicó en el capítulo anterior, conexiones permanentes a la base de datos y multihilos son las características más importantes que ofrece Java.

Así, la combinación de PERL, Apache y otras aplicaciones han sido cada vez más y más atractiva en aplicaciones Web pequeñas y medianas, y si las comparamos con las soluciones Java estas son más fáciles de desarrollar. Trataremos de considerar los detalles de estas soluciones cuando se maneja una determinada cantidad de usuarios concurrentes, mostrando los resultados con algunas gráficas.

5.2. ANÁLISIS Y DESARROLLO DE PRUEBAS

a) Teoría necesaria.

Los Benchmarks presentados fueron implementados en dos de las soluciones posibles: soluciones con CGI y soluciones con Java Servlets.

Debido al hecho de que la mayoría de los CGI son implementados en PERL, los CGI's de los benchmarks fueron implementados con este lenguaje.

El significado de PERL es "Practical Extraction and Report Languge". Este lenguaje fue originalmente desarrollado por Larry Wall para manejar conjuntos muy grandes de datos y crear reportes. Con estas raíces encaminadas hacia el proceso de datos, el manejo de archivos con PERL es muy útil y puede organizar el manejo de grandes cantidades de datos como una tarea trivial.

b) El ciclo de vida de un CGI.

El significado de "CGI" es "Commun Gateway Interface". Muchos lenguajes pueden ser usados para desarrollar programas CGI, tales como C, C++, Perl, visualbasic, PHP, etc. Cuando estos lenguajes son utilizados para implementar archivos CGI, estos archivos tiene extensión ".cgi", y deberán ser puestos bajo un subdirectorio especial llamado "cgi-bin" en la máquina usada como servidor WEB. Estos programas cargan librerías de archivos especiales con lo cual se realiza el análisis necesario para traducir el código y hecho esto se puedan ejecutar las acciones requeridas en el programa CGI.

c) Forma de operación de un programa CGI.

La parte conflictiva de un CGI es debida a su mismo ciclo de vida. Un programa CGI necesita crear un proceso separado para cada una de las peticiones de los usuarios, y este proceso será terminado tan pronto como la transferencia de datos sea completada.

Utilizar un programa separado por cada petición de usuario toma tiempo. El sistema operativo tiene que cargar el programa, localizar memoria para el programa y después, al terminar localizar de nuevo el programa para liberar la memoria.

Estos pesados cambios de contexto durante la creación y descarga de programas, son los que afectan el desempeño de los programas CGI; así de esto los programas CGI no son muy viables en aplicaciones con alto tráfico, donde se necesita procesar un gran número de peticiones para los clientes.

d) Los SERVLETS de Java

Los Servlets de Java son código del lado del servidor, corriendo en una aplicación para contestar peticiones de los clientes. Los servlets no siguen un protocolo específico, pero por lo regular son usados con el protocolo HTTP la mayoría de las veces, y la palabra Servlet usualmente tiene referencia a "HTTP Servlet".

Los Servlets hacen uso de las clases de Java en el paquete "Javax.servlet" y "Javax.servlet.http". Así, los servlets escritos en lenguaje Java son altamente portables y pueden ser ejecutados sin importar el sistema operativo de la máquina en la que estén corriendo.

e) El ciclo de vida de un SERVELT y un CGI

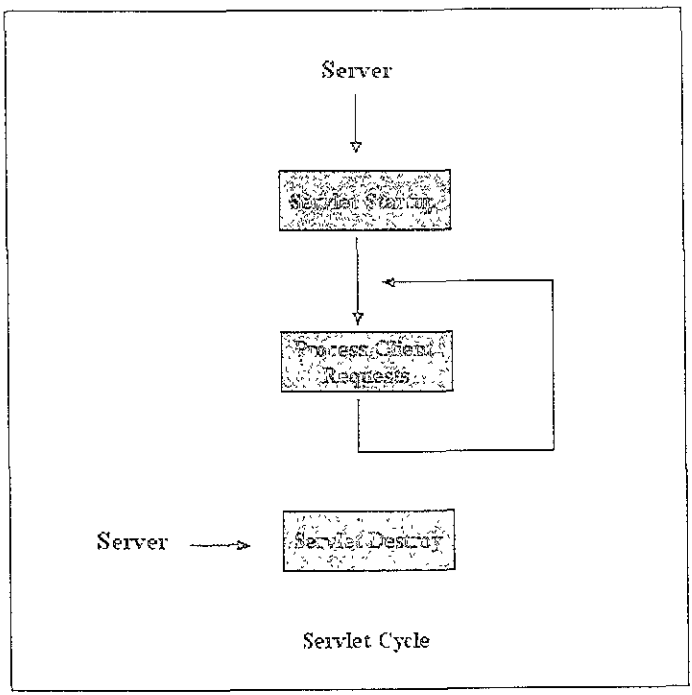
Cuando un servlet es llamado por primera vez, este es cargado en la memoria. Después de que la petición es procesada, el servlet se mantiene en la memoria y no será descargado hasta que este sea desmontado del servidor WEB.

Normalmente un servidor WEB estará corriendo todo el tiempo. Por lo tanto, después de que el servlet es activado, este se mantiene corriendo en segundo plano y vive tanto como el servidor WEB viva.

Usaremos el termino "servlet desmontado" para referirnos a un servlet que no ha sido cargado en memoria y "servlet montado" al que ya ha sido cargado en la memoria. La primera ves que el servlet es llamado y cargado en la memoria , el método init() del servlet es llamado.

Este método no será ejecutado en futuras peticiones, por lo tanto será un servlet montado. Así, peticiones subsecuentes para el servlet pueden simplemente resultar en llamadas a métodos definidos por el programador en el servlet sin necesidad de recargar el programa. Al implementar servlets como clases de Java se tiene la ventaja que el servlet puede fácilmente ser mantenido en la memoria y ser eficientemente llamado una y otra vez.

La mayoría de los servidores WEB dan la opción de cargar algunos servlets que son usados frecuentemente cuando el servidor web es levantado, esto con el fin de obtener ventaja en desempeño. De esta forma, los servlets pueden estar ya en memoria para la petición del primer cliente y evitar la sobrecarga de los primeros accesos a estos servlets. Cargando los servlets cuando el sistema se levanta, se asegura que el tiempo de respuesta para todas las peticiones sea mantenida como mínima, aunque esto implica que la carga en memoria del servidor WEB sea mas tardada.



f) Desempeño de SERVLETS y programas CGI.

No todos los servlets tienen un mejor desempeño que un CGI, para toda regla existen excepciones. Un servlet desmontado se desempeña peor que un CGI.

Estos serán descargados de la memoria inmediatamente después de completar la petición del cliente, en este caso un servlet que es cargado y descargado por demanda, se reduce de un servlet a un CGI.

Cuando un servlet es activado, éste llama al método init() para preparar todos los parámetros que este necesitará, también necesita cargar la máquina virtual de Java y todo esto es obviamente una gran trabajo.

Los servlet's desmontados siguen todo este procedimiento cada vez que son llamados. A diferencia un CGI, solo necesita cargar un interprete externo y no ejecuta nada sino hasta la ejecución del CGI, así no tienen mucha sobrecarga por la cual preocuparse

g) Los SERVLETS "desmontados"

Usualmente un servlet desmontado no es practico y no es recomendable debido a la descarga y carga del servlet (lo cual lo hace no tan diferente de utilizar un CGI). El problema está en que una vez cargado el servlet dentro de la memoria, este corre en segundo plano sin que nadie lo note. Esto toma recursos del sistema. Cuando más y más servlets son cargados en la memoria, mas recursos son tomados y el servidor WEB eventualmente podría caerse.

En este caso y para mantener un desempeño óptimo, podríamos considerar mantener cargados solo aquellos servlets que son más usados. Esto mantendría un límite en memoria compartida y prevendría un uso ineficiente de los recursos del sistema. Algunos servlets solo necesitarían correr en peticiones especificas, para utilizar mejor los recursos del sistema.

h) Los SERVLETS y las aplicaciones con alto tráfico.

Después de considerar las diferencias entre los ciclos de vida de un servlet y un CGI, no es difícil entender por qué los servlets pueden manejar grandes cantidades de peticiones concurrentes.

Aplicaciones de comercio electrónico, tales como compras en línea, logran una notable ganancia con el uso de servlets.

La programación multihilo es la principal ventaja de los servlets sobre los CGI. Un servlet no necesita usar un proceso separado para manejar cada petición de un cliente, éste simplemente utiliza otro hilo con el mismo proceso y esto hace posible que cientos de usuarios tengan acceso al mismo tiempo a un sevlet sin causar gran problema en el servidor.

i) Conexiones persistentes a una base de datos con Servlets.

Por la misma forma del ciclo de vida de un servlet, éste permite un buen acceso a bases de datos.
La mayoría de las aplicaciones WEB manejan conexiones a bases de datos.

Una base de datos es una excelente forma de manejar datos almacenados. Desgraciadamente las operaciones en bases de datos son los métodos más caros en una aplicación WEB. El llamado cuello de botella aparece cuando una conexión a base de datos es abierta. Un CGI no tiene muchas opciones excepto la de abrir una conexión cada vez que se procese una petición. Los Servlets son un poco más listos, estos reciclan las conexiones a la base de datos por medio de "connection pool" para evitar sobrecargas. Cuando un Servlet es activado, este puede hacer varias peticiones de conexión a la base de datos a través de llamadas por el método init() y guarda estas conexiones en el "connection pool". Cuando otra petición aparece, el servlet checa para ver si alguna conexión está disponible, si existe alguna disponible éste asignará la conexión a la petición, de lo contrario, crea una nueva y se la asignará a esta.

Cuando la petición ha sido procesada, ésta regresará la petición al "connection pool" y esta petición estará disponible para la siguiente llamada. De esta forma, los servlets proveen una forma propia de mantener conexiones a bases de datos persistentes.

Una conexión a una base de datos puede ser reciclada y mantenida persistentemente, mientras el servlet este corriendo y la conexión a la base de datos no se halla terminado en tiempo. Una conexión a base de datos puede ser limitada en tiempo, dependiendo de la configuración de la misma base de datos.

Para los datos obtenidos en éste capítulo, los tiempos de límite de conexión a la base de datos fueron de 8 horas usando MySQL, siendo este el valor de configuración por default.

Para sistemas de alto tráfico, es de gran ventaja crear un gran número de conexiones objeto durante el arranque del servlet y así poder realizar asignaciones dinámicas a peticiones de conexiones a la base de datos. Aunque las conexiones persistentes no proveen ninguna transferencia rápida, si se obtiene una rápida conexión a la base de datos y evita sobrecargas en el sistema.

j) Análisis y desarrollo de pruebas.

Las aplicaciones WEB son diseñadas con el propósito de compartir recursos y se tiene en mente, al diseñarlas, que puedan manejar múltiples peticiones de usuarios concurrentemente. Cuando una aplicación se está diseñando, es importante escoger la solución mas apropiada conociendo los requerimientos de ésta misma. El desempeño es uno de los más importantes problemas que necesitan ser considerados.

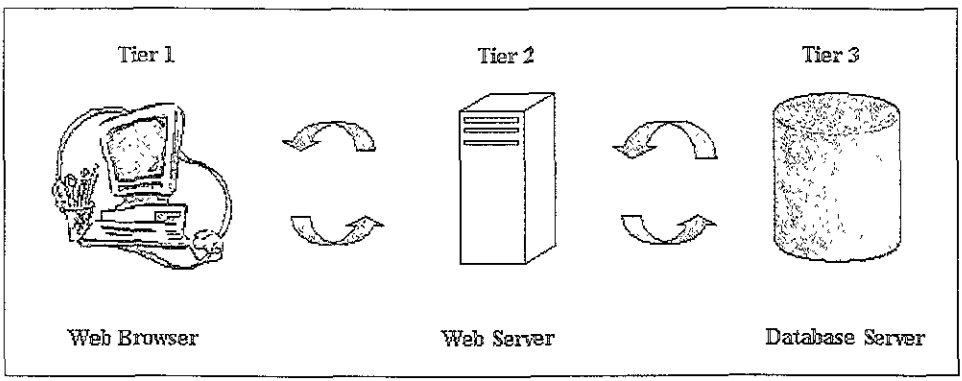
k) Soluciones posibles para una aplicación WEB y diferencias entre éstas.

En los siguientes párrafos, haremos uso de la serie de benchmarks tomados del trabajo de Amanda Wanhong Wu y analizaremos los resultados para obtener las diferencias de desempeño.

Fue seleccionado un ejemplo usando un CGI, y este mismo ejemplo fue nuevamente desarrollado usando Java servlets. El programa ApacheBench (ab) fue utilizado para simular a múltiples usuarios concurrentes haciendo peticiones en las diferentes versiones del programa de ejemplo.

El tiempo de respuesta para procesar la petición fue obtenido para comparar el desempeño de las diferentes alternativas que tomamos (Servlets y CGI).

El sistema tiene una arquitectura de tres capas: WEB browser (cliente), WEB server y un servidor de base de datos, este servidor de base de datos fue implementado en una maquina Sun Enterprise 450.



La aplicación es similar a los requerimientos de muchas aplicaciones de WEB con el servidor de base de datos separado del servidor WEB. Esta separación es con el fin de asegurar que los usuarios no tengan un acceso directo a los datos de la base. Las peticiones del cliente fueron a través de la red y alcanzando al servidor WEB. El servidor WEB se comunica con el servidor de la base de datos y procesa la petición y envía la información de regreso al cliente.

l) Estructura de las Tablas en la Base de Datos

Cinco tablas de la base de datos están involucradas en el programa de ejemplo usado: evals, deptinfo, schoolinfo, courses y terminfo. La figura muestra las relaciones de estas cinco tablas.

tabla "evals": guarda 27307 entradas "cid" es una llave foránea.
tabla "courses": guarda 1358 grabaciones "cid" es la llave primaria.
tabla "terminfo": guarda 13 registros de duracion ,año y semestre.
tabla "deptinfo": guarda 92 ID de nombres de departamentos.
tabla "schoolinfo":guarda 9 ID de escuelas y nombres de escuelas

m) Pruebas en Plataformas.

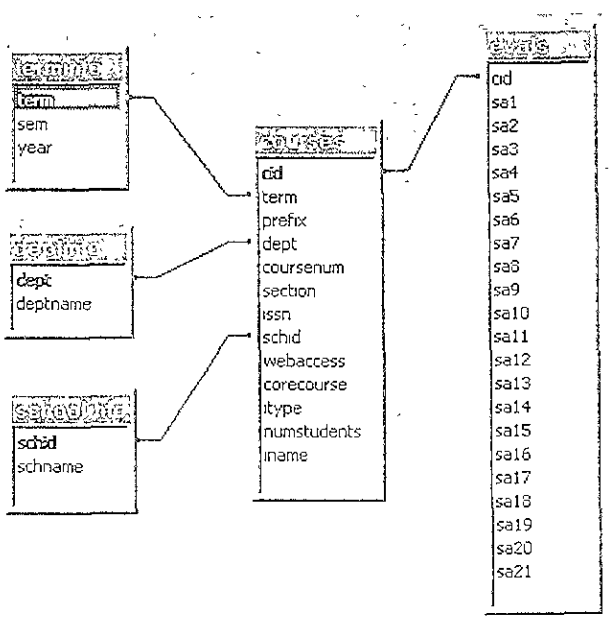
Los benchmark's fueron realizados en dos diferentes plataformas:

- Sistema Solaris: Servidor Sun 450 con cuatro procesadores. Se le nombra "cedar". En este servidor hay una gran cantidad de aplicaciones corriendo, puede considerarse que tiene las caracteristicas de trabajo que se aplicarían en la realidad en cualquier servidor WEB

- Sistema IRIX: Estación de trabajo SGI O2 con un procesador. Se le nombra "rain". En esta estación de trabajo hay solo algunas aplicaciones corriendo. Su desempeño puede ser considerado como ideal.

El ejemplo de prueba usado en los experimentos de benchmark, se muestra a continuación:

Dentro de este programa, dos queries son usados para obtener información de la base de datos: el primer query selecciona aleatoriamente un ID curso de la tabla de cursos:




```
$rows = int (rand 1000);
$sql_statement = "SELECT cid FROM courses where term='001' AND webaccess<>'1'
LIMIT $rows, 1";
```

El segundo query es mucho más complejo, el cual une las cinco tablas para seleccionar toda la información que es necesitada para llenar la parte izquierda del cuadro de salida.

```
$sql_statement = "SELECT courses.prefix, courses.coursenum, courses.section, courses.iname, courses.itype, courses.numstudents, count(evals.cid), deptinfo.deptname, terminfo.sem, terminfo.year, schoolinfo.schname FROM courses, deptinfo, terminfo, schoolinfo, evals where courses.cid=\'$vals[0]\' AND deptinfo.dept = courses.dept AND terminfo.term =courses.term AND schoolinfo.schid = courses.schid AND evals.cid = courses.cid GROUP BY courses.cid";
```

Para hacer la comparación de desempeño de las diferentes soluciones, este cuadro de salida fue implementado en tres diferentes versiones.

Versión 1: CGI con PERL, probado en "cedar" y en "rain".

Version 2: Servlet de Java sin una conexión persistente a la base de datos, probada en "cedar" y en "rain".

(En esta versión, el codigo para abrir la conexión a la base de datos y el cierre a esta misma, fueron puestos en el bloque doGet(). Así la conexión a la base de datos es abierta y cerrada en cada petición. Esta opera de la misma manera que un CGI.)

Versión 3: Servlet de Java con una conexión persistente a la base de datos, probada en "cedar" y en "rain".

En esta versión, el código para abrir la conexión a la base de datos fue puesta en el bloque init() y esta conexión solo se cerrará cuando el servidor sea apagado. Así la conexión puede persistir para todas las peticiones entrantes.

n) Herramientas de "Benchmark"

Estas cinco versiones del programa de prueba fueron ejecutadas con la simulación de múltiples usuarios concurrentes usando ApacheBench (ab).

ApacheBench (ab) es parte de la aplicación Apache WEB server. Esta es una herramienta de bechmark que puede ser usada para comparar el desempeño de las diferentes versiones de programas.

Esta fue diseñada para probar como un servidor de WEB se desempeña cuando los usuarios están haciendo peticiones al programa propuesto.

ApacheBench (ab) es muy fácil de usar. En particular, éste muestra como muchas peticiones por segundo en el servidor Apache son capaces de ser servidas con un programa en particular. Si versiones diferentes del programa de ejemplo están corriendo en el mismo servidor WEB, entonces la diferencia de procesamiento en estas dos versiones muestra la diferencia de desempeño.

Entre los resultados que arrojó Ab, el único número que se considero fue "request per second".

El uso de Ab es:

ab -n (número de peticiones hechas por cada usuario –c (número de usuarios actuales) –w (URL completo del programa)

Para los datos obtenidos se usaron múltiples usuarios (de 1 a 50) los cuales fueron simulados como peticiones concurrentes para cada una de las diferentes versiones del programa de ejemplo.

Cada uno de los usuarios simulados hace una petición al programa en cada prueba.

o) Comparaciones

La figura muestra la gráfica de desempeño entre un sevlet y un servlet con una conexión persistente, corriendo en una situación muy parecida a la realidad con varios usuarios simulados haciendo peticiones en la estación de trabajo.

La distancia entre las dos curvas muestra la diferencia en desempeño de las dos versiones de servlets que se están probando.

Una de las versiones mantiene una conexión persistente a la base de datos y comparte ésta con las peticiones entrantes. La velocidad de procesamiento es relativamente estable.

En respuesta a un número mayor de clientes, el número de peticiones que pueden ser procesadas decrece lentamente.

Esta ganancia de desempeño con la velocidad de procesamiento es favorable, y esto es debido a que existe una ganancia en tiempo de conexión a la base de datos, la cual siempre consume una mayor cantidad de tiempo.

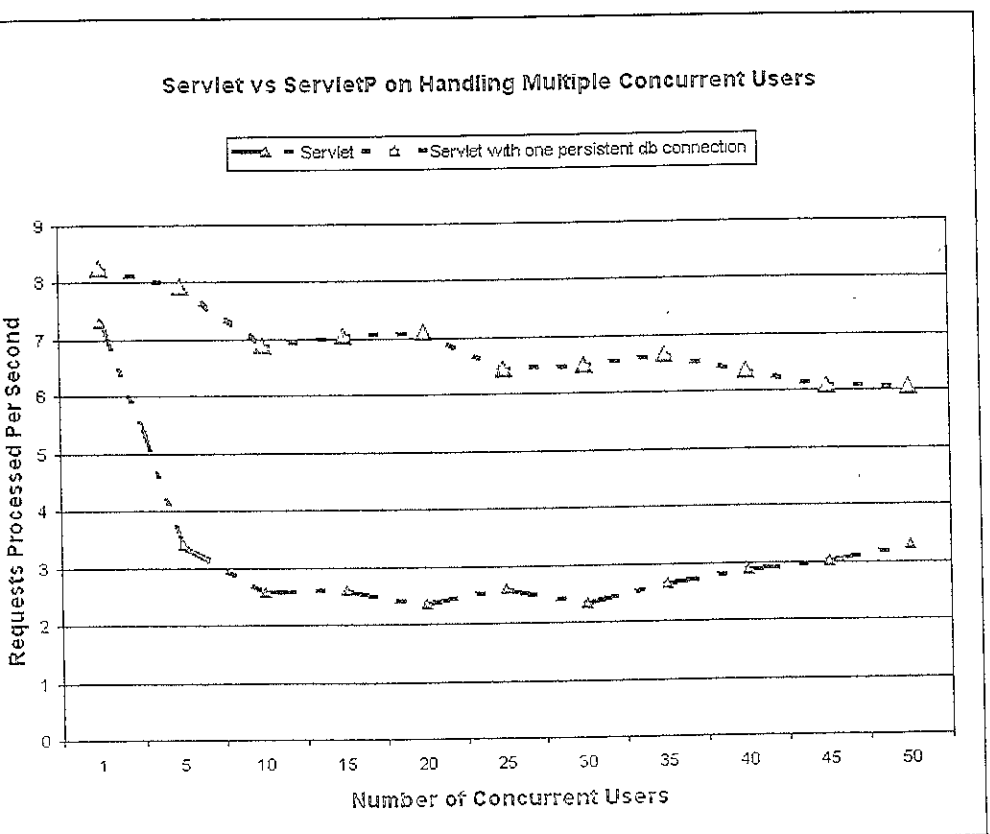
La otra versión abre y cierra las transferencias a la base de datos. Así, un pequeño incremento de tráfico resulta en un alto decremento de velocidad de procesamiento. La velocidad de respuesta a peticiones parece ser estable en el rango de 5 a 30 peticiones concurrentes y después de este número se incrementa un poco.

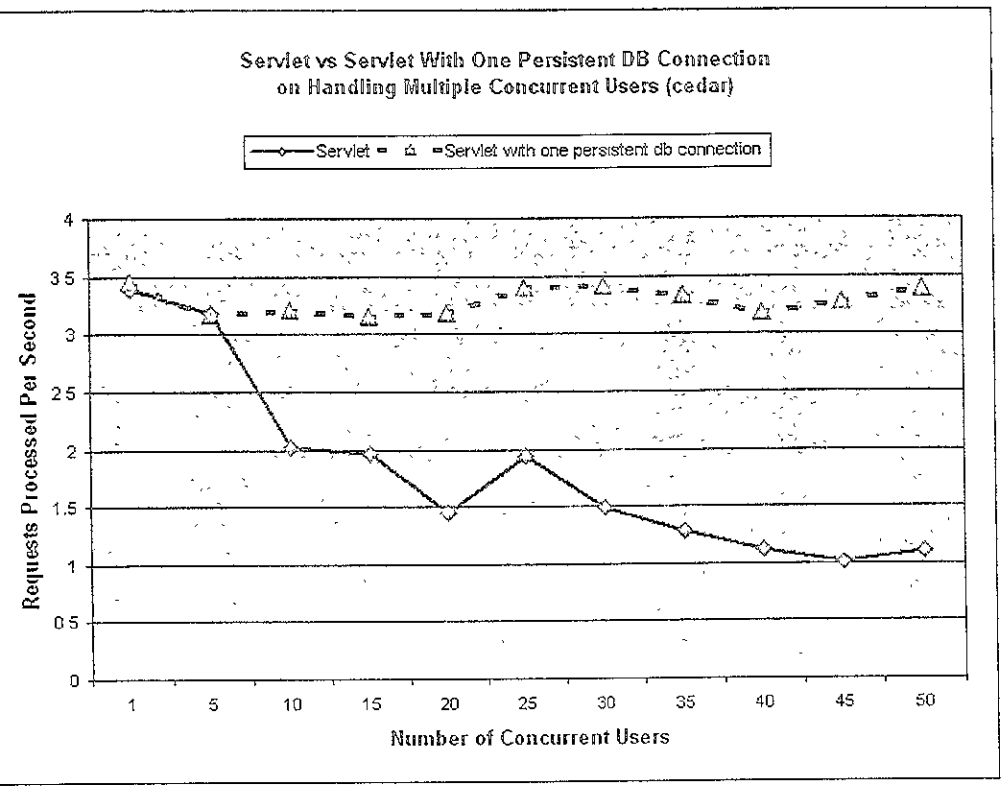
Este resultado puede ser causado por las características del cache de los Servlets. Esta característica pude ser usada para poner a punto el desempeño usando el cache en el servidor WEB.

p) Comparación en el servidor de las dos versiones de Servlets

Las estaciones de trabajo por lo regular no tienen una gran cantidad de aplicacionescargadas a diferencia de, por ejemplo, el servidor de la red de una universidad.

Las versiones fueron pasadas a la máquina de Sun y el benchmark fue nuevamente ejecutado.
Los resultados son mostrados abajo. La velocidad absoluta de procesamiento es más pequeña que en la estación de trabajo, y esto es lógico debido a que en ésta máquina existe un número mayor de peticiones (estas circunstancias se acercan más a una situación real).





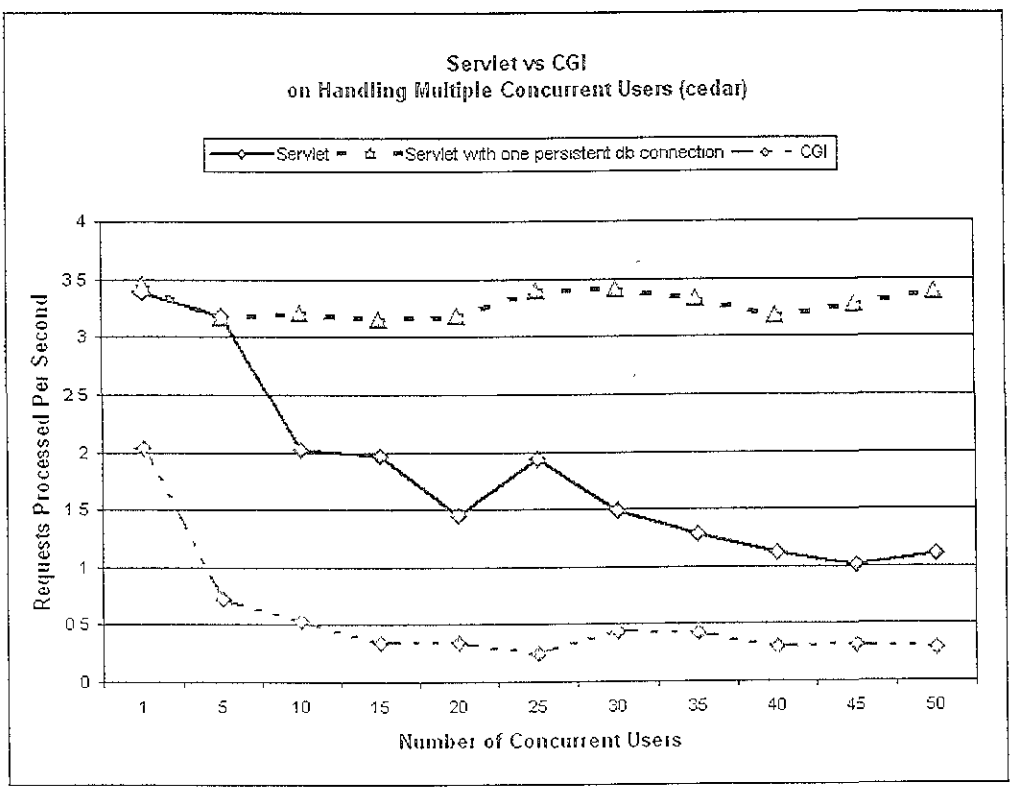
La versión de conexión y desconexión sigue pagando por su propia naturaleza, pero en esta prueba lo hace aun más. La velocidad de procesamiento decrece notablemente hasta que el tráfico llega a ser bastante pesado (de 40 usuarios en adelante). Por la gráfica parece que el aumento de usuarios no beneficia en nada a la versión, a diferencia de lo que pasó en la estación de trabajo. Una posible explicación puede estar en la forma en la que el cache trabaja. El cache trabaja siguiendo la política de "el menos usado". Si una aplicación es frecuentemente usada, su copia será guardada en el cache sustituyendo a la copia de la aplicación menos usada y de la misma forma probablemente esta aplicación será desplazada por otra copia que en determinado momento es usada con mas frecuencia, todo esto porque la capacidad del buffer es limitada. Así, en sistemas muy cargados como en la máquina Sun, la copia en el cache tiene que competir con otros procesos que son accedidos frecuentemente y en determinados momentos la copia no esta disponible en el cache y esto tomara más tiempo para nuestra versión de ejemplo.

Para la versión con conexión persistente, se obtiene una respuesta estable para en el incremento de usuarios concurrentes. La diferencia de desempeño es notable entre

las dos versiones. Así, teniendo una conexión persistente en un sistema con alto tráfico se logra el objetivo de mejorar el desempeño del sistema.

g) Comparación entre programas CGI y SERVLETS.

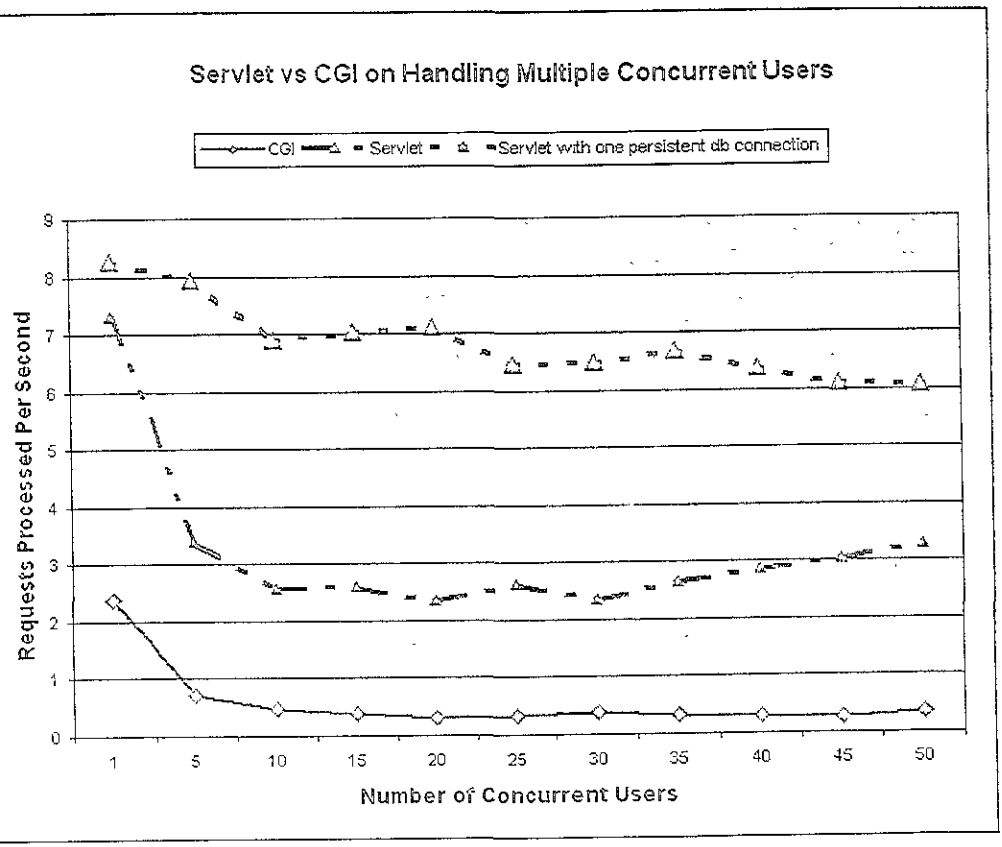
Se puede observar que las dos versiones tienen un mejor desempeño cuando manejan un número pequeño de usuarios concurrentes haciendo peticiones, el servidor WEB tiene la suficiente habilidad para manejar un número pequeño de peticiones y procesar éstas de forma muy rápida.



Cuando el número de usuarios se incrementa, el número de peticiones también lo hace y el tiempo de procesamiento se incrementa significativamente. Un CGI se desempeña mejor solo cuando maneja

menos de cinco usuarios concurrentes, cuando el número de usuarios se incrementa el desempeño decrece según se muestra en las siguientes figuras.

En ambas pruebas, se puede observar que hay una gran diferencia en el desempeño de las versiones de CGI y Servlets. Así fácilmente podemos concluir que los servlet's de Java son una mejor solución que los programas CGI.



Poniendo todos los resultados de los bencmarks, se pueden apreciar mejor las diferencias.

La solucion con los Servlets y conexiones persistentes fueron la mejor solución en desempeño

Los modelos usando Java Servlets, con conexiones no persistentes a la base de datos tuvieron un menor desempeño que aquellos con conexiones permanentes, pero tambien mostraron un mejor desempeño que los programas CGI, y también quedo claro que la solución con CGI fue poco viable con alta carga de tráfico en la aplicación.

5.3. CONCLUSIONES DE ESTE CAPÍTULO

Los Servlets de Java tienen la ventaja de contar con la metodología de orientación a objetos y con la característica de programación multihilo; los sevlet's de Java son una excelente opción para los requerimientos de comercio electrónico. Estas aplicaciones necesitan manejar peticiones del cliente en un modo interactivo de alto desempeño y deben procesar estas peticiones múltiples de la mejor forma posible.

Las soluciones con CGI's son apropiadas para pequeñas aplicaciones con una cantidad límite para acceso de clientes. En aplicaciones con un bajo tráfico pueden ser desarrolladas usando programas CGI de forma rápida, teniendo en cuenta que se tendrá que pagar por esta rapidez una baja cantidad de clientes que puedan acceder a la aplicación.

En el caso de procesar una gran cantidad de datos binarios o de texto, muchas llamadas directas al sistema pueden ser hechas, y es aquí donde PERL o C pueden mostrar su poder.

LIMITACIONES Y ALCANCES.

Como se pude observar, el lenguaje Java tiene características que lo hacen parecer un lenguaje útil para desarrollar aplicaciones, pero si se quisiera desarrollar algún proyecto enfocado hacia la estructura y el manejo de la organización de un sistema computacional, presentaría serias dificultades.

Java, a través de su máquina virtual, lleva a cabo muchas tareas que ahorran problemas con relación a la arquitectura y organización del sistema de computo que está siendo usado. Pero esta misma situación lleva a considerar el uso de otros lenguajes de programación para el desarrollo de sistemas donde se tenga que ver con el sistema de administración de archivos, configuración de bits de entrada y salida de registros específicos del procesador, administración de puertos para red , etc.

Si bien es claro que Java presenta ya paquetes de clases que implementan los métodos necesarios para el manejo de comunicaciones, también es cierto que el programador está de algún modo amarrado a estos paquetes para poder hacer un correcto uso de los dispositivos del sistema que se estan usando. Aunque Java fue en principio diseñado para el manejo directo de dispositivos, en su misma evolución a ser un lenguaje multiplataforma, su enfoque se dirigió a ser un lenguaje para aplicaciones y no un lenguaje para el desarrollo de sistemas de organización y arquitectura de computadoras.

Si se toma en consideración que existen paquetes que están siendo escritos para aplicaciones especificas (Java Communications, por dar un ejemplo de paquetes que se involucran un poco más con los dispositivos del sistema), entonces estos paquetes serán los que le irán dando más fuerza al mismo lenguaje.

Un programa CGI escrito ya sea en el shell o con C, puede acceder a las variables de entorno de un servidor WEB; para lograr esto con Java, se debe escribir algún JSP o un Servlet para hacer la petición de la variable al servidor. Ahora bien, si el servidor fuera un servidor de seguridad que usa un protocolo https, los mismos programas CGI con algunas pequeñas modificaciones nos darían la misma utilidad que cuando se usa un servidor sin el protocolo de seguridad.

Para poder hacer que el Servlet o el JSP antes mencionado funcione, se tendrán que hacer algunas modificaciones un poco más a fondo, como instalar y configurar el paquete que contiene las clases que lograrán que la comunicación con el protocolo https sea llevada a buen fin. Aun así, se podría tener algún problema con las variables de entorno del servidor de seguridad que se esta usando.

Java es dependiente de las capas que se le implementen para poder darle nuevas funcionalidades. A diferencia de otros lenguajes que por su mismo diseño tienen en si mismos el manejo de desarrollos específicos.

Cabe mencionar también que con Java se tienen muchas características que son sumamente útiles como la multiplataforma, pero con esta característica se pierde inherentemente la velocidad de repuesta. Cuando, por ejemplo, se desarrollan proyectos en los que se tiene que hacer accesos a bases de datos, estos accesos tienen que pasar, si estuviéramos programando con Java, por varias capas hasta llegar a obtener el dato deseado, de lo que es claro que no puede ser comparada la eficiencia de la respuesta de alguna petición a la base por algún programa escrito en Java, que esta misma petición por un programa escrito por ejemplo con PL/SQL que pasa por muy pocas capas al hacer la misma petición. (aun teniendo que, las clases de Java, implementan un eficiente manejo de datos).

Ahora, si consideramos a XML, se tienen muchas características que permiten un mejor manejo de los datos, pero algunos grupos han promocionado a XML más allá de sus capacidades reales. XML es un estándar flexible para el formateo de datos, pero también no es el mejor formato de datos para todos los que lo usan.

W3C ha decidido no soportar la definición de ningún documento DTD en especifico, y es dejado a las industrias y compañías definir y usar sus propios documentos DTD. Esto desalienta a los procesos de estandarización de documentos DTD, en si mismos largos en tiempo y en trabajo, alentando la proliferación de documentos DTD diferentes, los cuales se sobrepondrán unos a otros y crearan conflictos.

Otra desventaja de XML, es el tamaño en el que se incrementan los documentos XML. En general XML no es un lenguaje optimizado para algún uso en particular, y no ofrece un gran desempeño en velocidad y tamaño. Así, las etiquetas del mismo documento pueden incrementar el tamaño de este sustancialmente, especialmente cuando la legibilidad como los detalles de los datos son importantes. Este incremento podría ser cuando menos la mitad del mismo documento en principio y si se está haciendo uso de datos binarios este incremento es aún mucho mayor. Obviamente, esta es una condición inapropiada para aplicaciones donde la velocidad de transmisión y el tamaño de los archivos son importantes.

Ahora bien, dentro de los alcances, Java y XML ofrecen una gran utilidad en el desarrollo de aplicaciones para Internet, debido esto en gran parte al mismo enfoque con el que fueron creados (o modificados para ubicarse mejor en el mercado, en el caso de Java). Esta funcionalidad la tienen también por su status de código abierto y multiplataforma, que es precisamente hacia donde a tendido el desarrollo de las comunicaciones con Internet, y esta misma a su ves ha seguido la tendencia de una comunicación global a través de diferentes medios, máquinas, interfaces, plataformas, y demás componentes que puedan dan un servicio de comunicación y/o transferencia de datos.

Java, además, cuenta con enfoque de orientación a objetos, esta orientación es una de las metodologías que ofrecen una mayor funcionalidad para los objetivos deseados por las compañías o grupos de investigación al compartir información en aplicaciones de Internet.

Provee también de una programación multihilo, lo que da un mejor desempeño en el momento en que muchos procesos tengan que ser cargados a un mismo tiempo (esto da gran utilidad para el comercio electrónico, por ejemplo).

"... Es asombroso todo lo que se puede manejar mediante dicha plataforma, desde dispositivos de información inalámbricos, hasta tarjetas inteligentes, computación "peer-to-peer", dispositivos inteligentes, dispositivos "embedded" y en tiempo real, así como temas como los de seguridad, dispositivos de baja potencia, desempeño y arquitecturas, sin dejar de considerar servicios web, J2EE, JSPs, XML, aplicaciones inalámbricas, multimedia, sonido, Swing, applets, procesamiento de imágenes, procesamiento de transacciones, mensajería, e-Business, patrones, manejo de memoria, criptografía, administración de "layout" y desempeño de redes, entre otros (...) Es curioso que las empresas presentes en el Moscone Center se caracterizan por desarrollar servicios web mediante XML, SOAP y UDDI, las piezas centrales de la estrategia NET de Microsoft. Todas admiten estar en una ruta de desarrollo de productos parecida a la de MS en lo que se refiere a construir, integrar y liberar servicios web, pero distinta en el sentido de que la apertura de Java es radicalmente diferente a los planes centrados en lo que se denomina el plan basado en software propietario de MS, lo que les concede una inmensa ventaja competitiva en el corto y largo plazos."*[14]

Se puede observar como, el encaminamiento de las nuevas características del lenguaje y sus aplicaciones, tratan de seguir el mismo camino que llevan las demás compañías que desarrollan los mismos productos (en este caso Microsoft), siguiendo ambas el mercado al cual se enfrentan, pero remarcando la característica de código abierto como una gran ventaja sobre su competidor.

XML da una forma muy útil de manejo de datos. Al poder establecer un formato para compartir información se logra una mejor comunicación entre grupos de trabajo que manejan el mismo tipo de información. Con XML se eliminarán problemas de presentación y contenido al intercambiar información por la red. Y si se logra un formato común entre los grupos de trabajo que manejan información semejante, XML podría ser el formato estándar para transferir datos a través de Internet y otros medios a no muy largo plazo .

CONCLUSIONES

Internet, en su comportamiento, es en alguna medida parte de la misma globalización que se está desarrollando, es un medio de intercambio de información entre grupos de personas ubicados en puntos muy distantes unos de otros. Ahora, este medio de comunicación puede traer múltiples usos como vender algún productos a personas, poder llegar a regiones geográficamente difíciles y distantes a través de conexiones inalámbricas para aportar algún tipo de ayuda, tener el intercambio de información necesaria para grupos de trabajo en investigaciones para el desarrollo de proyectos, llegar a un grupo relativamente grande de personas y exponer ideas sobre principios y valores válidos para el grupo que expone, etc, etc.

Las consecuencias de los usos que se le den a la red dependen del conocimiento y accesibilidad que se tenga a ésta. En el desarrollo de este trabajo, se expuso una valoración de algunas herramientas que son útiles para la presentación e intercambio de estos datos a través de Internet. La valoración expuesta, presentó limitaciones y alcances de estas herramientas, de las cuales se encontró que estas herramientas presentan una gran ayuda en el desarrollo de aplicaciones para Internet, debido principalmente a la estandarización y al enfoque de la programación de objetos que estas presentan. Pero que, el crecimiento de programas relativos a Internet, (y por tanto las herramientas) dependerán y evolucionaran con el mismo movimiento del mercado económico y/o del uso que se le de a la red para alcanzar objetivos específicos, siendo estos últimos tan divergentes como los mismos grupos humanos alrededor del mundo que tengan acceso y puedan manejar la red.

ANEXO 1

TABLA CRONOLÓGICA DEL MODELADO ORIENTADO A OBJETOS

Evolución los lenguajes de programación orientados a objetos y su inclusión con los demás lenguajes.

Primera generación (1954 - 1958)

FORTRAN I: Cálculos matemáticos.
Algol 58: Idem.
Flowmatic: Idem.
IPL V: Idem.

Segunda generación (1959 - 1961)

FORTRAN II: Subrutinas, compilación separada.
Algol 60: Estructura de bloques, tipos de datos.
COBOL: Descripción de datos, manejo de ficheros.
Lisp: Procesamiento de listas, punteros, recolección de basura.

Tercera generación (1962 - 1970)

PL/1 FORTRAN + Algol + COBOL.
Algol 68 Sucesor riguroso de Algol 60.
Pascal Sucesor sencillo de Algol 60.
Simula: Clases, abstracciones de datos.

Posteriormente aparecen lenguajes que marcan el camino de la OO:
SmallTalk: Sucesor de Simula.
Ada: Algol 68 + Pascal + contribuciones de Simula y otros.
C: Derivado del Algol 60.
C++: Unión de C y Simula.
Eiffel: Simula + Ada.

Características de las descomposiciones algorítmica y orientada a objetos

algorítmica	orientada a objetos
diagramas tipo árbol	varias posibilidades
desmenuza el problema	identifica semánticamente el problema
se programa en detalle	se programa a lo grande
lenguajes imperativos	lenguajes declarativos

Tabla cronológica del desarrollo del lenguaje Java

Año	Descripción
1991	(15/ene) Inicio del desarrollo en Sun Microsystems por James Gosling, et. al.
	(01/feb) Se inician trabajos sobre el sistema de gráficos "Aspen", programación de lenguajes y desarrollo de negocios.
	(junio) Gosling inicia trabajos sobre el intérprete "Oak". (19/ago) El equipo verde presenta y demuestra el uso de ideas básicas de UI.
1992	(verano) Se tienen hackeos masivos sobre Oak y sus componentes.
	(01/oct) Wayne Rosing se une y toma el liderato del equipo El primer producto Oak: "*7", un control remoto inteligente (hand-held). Tenía una imagen animada llamada "Duke".
1993	(15/mar) El equipo "verde" cambia de nombre a "Primera Persona" y se enfoca a la televisión interactiva. (abril) Se libera NCSA Mosaic 1 0.
	(14/jun) Time Warner y SGI se adelantan con pruebas de televisión interactiva. (verano) Oak consigue más de 300,000 licencias en la industria de electrónicos y televisión interactiva. (agosto) 3DO ofrece comprar la tecnología, lo cual es rechazado. (septiembre) Arthur Van Hoff se une al equipo para apoyar el desarrollo de aplicaciones de televisión interactiva, pero termina diseñando más sobre el lenguaje. La WWW crecía rápidamente.

1994	(17/feb) Se planea una plataforma multimedia en CD-ROM basada en Oak. (25/abr) Se crea Sun Interactive. (junio) Se inicia el proyecto "Oak vive", para un "gran sistema operativo pequeño". (julio) Se reduce el alcance del proyecto para Internet, después de escribir una implementación de un navegador Web. (16/sep) Se inicia la escritura de "WebRunner", posterioremente "HotJava". (29/sep) El prototipo de HotJava se presenta a los ejecutivos de Sun. (otoño) Se implementa el compilador Java en Java (antes se hizo en C). Se renombra Oak por Java.
1995	(23/may) Se anuncia formalmente Java y HotJava. (23/may) Netscape anuncia su intención de utilizar la licencia de Java para uso en el navegador. (21/sep) Sun lleva a cabo la conferencia de Java en New York. (25/sep) Alianza con Toshiba para el desarrollo de productos en recuperación remota de información, los cuales incorporan Java. (26/sep) Sunsoft anuncia productos para desarrollar con Java. (30/oct) Oracle anuncia WebSystem para la Web, que incluye Java (04/dic) Sun y Netscape anuncian JavaScript, un lenguaje basado en Java y diseñado para los no programadores. (04/dic) Su, Netscape y SGI anuncian alianza para el desarrollo de herramientas Internet. (06/dic) IBM y Adobe piden uso de licencia para Java (07/dic) Microsoft planea hacer uso de la licencia Java e incluye VBScript.

CONFIGURACION DEL ARCHIVO ".bash_profile" , PARA VARIABLES DE AMBIENTE
Y LIBRERIAS DE SERVLETS (TOMCAT, JDK,JSDK)

```
[root@tropanosoma /root]# more .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=/usr/java/jdk1.3.0_02/bin:/usr/local/sbin:/usr/sbin:/sbin:$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"
TOMCAT_HOME=/tomcat/jakarta-tomcat-3.2 ; export TOMCAT_HOME
JAVA_HOME=/usr/java/jdk1.3.0_02/bin
export USERNAME BASH_ENV PATH
CLASSPATH=.:/usr/java/jdk1.3.0_02/lib/jsdk.jar ; export CLASSPATH
[root@tropanosoma /root]#
```

.
.
.

UBICACION DE SUBDIRECTORIOS PARA SERVLETS Y JSP

```
[root@tropanosoma WEB-INF]# pwd
/tomcat/jakarta-tomcat-3.2/webapps/examples/WEB-INF
[root@tropanosoma WEB-INF]# ls
classes  jsp  web.xml
[root@tropanosoma WEB-INF]# cd classes | ls
classes  jsp  web.xml
[root@tropanosoma WEB-INF]# cd classes/
[root@tropanosoma classes]# ls
CookieExample.class           RequestParamExample.class   dates
CookieExample.java            RequestParamExample.java    error
HelloWorldExample.class       SessionExample.class        examples
HelloWorldExample.java        SessionExample.java         num
LocalStrings.properties       SimpleServlet.class         servletToJsp.class
LocalStrings_en.properties    SimpleServlet.java          servletToJsp.java
LocalStrings_es.properties    SnoopServlet.class          sessions
RequestHeaderExample.class    SnoopServlet.java           simple.class
RequestHeaderExample.java     cal                         simple.java
RequestInfoExample.class      checkbox
RequestInfoExample.java       colors
[root@tropanosoma classes]#
```

ARCHIVO "web-app.xml" (DEFINICION DE URL PARA SERVLETS, Y UBICACION DE CLASES)

```
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
   <servlet>
      <servlet-name>
         snoop
      </servlet-name>
      <servlet-class>
         SnoopServlet
      </servlet-class>
<!--
      <init-param>
         <param-name>foo</param-name>
         <param-value>bar</param-value>
      </init-param>
-->
   </servlet>

   <servlet>
     <servlet-name>
       primero
     </servlet-name>
     <servlet-class>
       SimpleServlet
     </servlet-class>
   </servlet>

   <servlet>
     <servlet-name>
       segundo
     </servlet-name>
     <servlet-class>
       simple
     </servlet-class>
   </servlet>

   <servlet>
     <servlet-name>
       servletToJsp
     </servlet-name>
     <servlet-class>
       servletToJsp
```

```
<taglib>
  <taglib-uri>
    http://java.apache.org/tomcat/examples-taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/example-taglib.tld
  </taglib-location>
</taglib>
<security-constraint>
 <web-resource-collection>
  <web-resource-name>Protected Area</web-resource-name>
  <!-- Define the context-relative URL(s) to be protected -->
  <url-pattern>/jsp/security/protected/*</url-pattern>
  <!-- If you list http methods, only those methods are protected -->
  <http-method>DELETE</http-method>
  <http-method>GET</http-method>
  <http-method>POST</http-method>
  <http-method>PUT</http-method>
 </web-resource-collection>
 <auth-constraint>
  <!-- Anyone with one of the listed roles may access this area -->
  <role-name>tomcat</role-name>
  <role-name>role1</role-name>
 </auth-constraint>
</security-constraint>
<!-- Default login configuration uses BASIC authentication -->
<!--
<login-config>
 <auth-method>BASIC</auth-method>
 <realm-name>Example Basic Authentication Area</realm-name>
</login-config>
-->
<!-- Form-based login is enabled by default.  If you wish to
    try Basic authentication, comment out the <login-config>
    section below and uncomment the one above. -->
<login-config>
 <auth-method>FORM</auth-method>
 <realm-name>Example Form-Based Authentication Area</realm-name>
 <form-login-config>
  <form-login-page>/jsp/security/login/login.jsp</form-login-page>
  <form-error-page>/jsp/security/login/error.jsp</form-error-page>
 </form-login-config>
</login-config>

</web-app>
```

*1 www,cofetel.gob.mx
*2 Tanenbaum. Redes de computadoras. Pág. 6
*3 Ibidem
*4 Tanenbaum. Redes de computadoras Pág. 2
*5 Ibidem
*6 Tanenbaum. Redes de computadoras. Op. Cit. Paxon[1994]
*7 http://www-lifia.info.unlp.edu.ar/~luciod/objcour/2.htm
*8 http://www.uaq.mx/ingenieria/eureka/n06/en0606.htm
*9 http://www.uaq.mx/ingenieria/eureka/n06/en0606.htm
*10 http://java.sun.com/nav/whatis/
*11 http://www.elwoodcorp.com/alu/table/history.htm#family
*12 http://www.xml.com/pub/a/98/10/guide0.html
*13 http://www.sun.com/xml/cover/2001-0205/
*14 http://www.excelsior.com.mx/ Articulo Ing. Juan José Carreón G. 14 Mayo 2001