Capítulo II
Marco Teórico

Dentro de este capítulo analizaremos la tecnología de sistemas distribuidos así como también se hará un análisis de sistemas existentes tomando en cuenta sus ventajas y desventajas.

Además de analizar los operadores que se implementarán en este proyecto de tesis. También se realizará la selección de herramientas para el diseño del sistema SCII creación de imágenes irreales así como la justificación de lenguajes de programación que utilizaremos. Y para finalizar este capítulo explicaremos sobre la instalación del servidor Apache Tomcat y Java, los cuales, fueron necesarios para la realización de este proyecto de tesis.

2.1 Concepto de Sistema Distribuido

Los sistemas distribuidos son aquellos en el que dos o más máquinas colaboran para la obtención de un resultado y están basados en las características de transparencia, eficiencia, flexibilidad, escalabilidad y fiabilidad. El objetivo principal de los Sistemas Distribuidos es el mejor desempeño, mayor fiabilidad y disponibilidad, compartición de recursos e información, además del mejoramiento de la comunicación, mayor adaptabilidad (más natural, distribución de carga, etc.) La importancia del desarrollo de los Sistemas Distribuidos es debido a las necesidades de los usuarios que requieren de aplicaciones más sofisticadas así como también de las funciones administrativas y económicas.

A continuación tenemos algunas definiciones de Sistemas Distribuidos :

"Sistema en el cual múltiples procesadores autónomos, posiblemente de diferente tipo, están interconectados por una subred de comunicación para interactuar de una manera cooperativa en el logro de un objetivo global." [Lelann, 1981]

"Sistema en el cual componentes de hardware y software, localizadas en computadores en red, se comunican y coordinan sus acciones sólo por paso de mensajes." [Coulouris, 2002]

"Conjunto de computadores independientes que se muestran al usuario como un sistema único coherente." [Tanenbaum, 2001]

Ahora si analizaremos cada una de las características mencionados anteriormente de los sistemas distribuidos.

1. **Transparencia.** El concepto de transparencia de un sistema distribuido va ligado de que todo el sistema funcione de forma similar en todos los puntos de la red, independientemente de la posición del usuario. La labor que el sistema operativo tiene es la de establecer los mecanismos que oculten la naturaleza distribuida del sistema y que permitan trabajar a los usuarios como si se tratara de un único equipo. En un sistema transparente, las diferentes copias de un archivo deben aparecer al usuario como un archivo. Y la tarea del sistema operativo es la de controlar las copias, actualizarlas en caso de modificación y en general, la unicidad de los recursos y el control de la concurrencia. El que el sistema disponga de varios procesadores debe lograr un mayor rendimiento del sistema, pero el sistema operativo debe controlar que tanto los usuarios como los programadores vean el núcleo del sistema distribuido como un único procesador. Otro punto clave que debe controlar el sistema operativo es el paralelismo, debe distribuir las tareas entre los distintos procesadores como en un sistema multiprocesador, pero con la dificultad añadida de que ésta tarea hay que realizarla a través de varios ordenadores.

Existen diferentes tipos de transparencia como son:

- **Transparencia de acceso:** acceso a objetos locales o remotos de la misma manera.
- **Tranparencia de lugar:** acceso a objetos sin conocer dónde están.
- **Transparencia de concurrencia:** varios procesos pueden operar concurrentemente usando objetos de información compartidos sin estorbarse.

- **Tranparencia de replicación:** diferentes réplicas de un mismo objeto de información sin enterarse a cuál se accede, ni diferencias entre ellos.
- **Transparencia de fallo:** aislamiento de fallos, de forma que las aplicaciones puedan completar sus tareas.
- **Transparencia de migración:** permite mover los objetos de información sin afectar a las aplicaciones.
- **Tranparencia de rendimiento:** redistribución de cargas en el sistema sin modificación en las aplicaciones.
- **Tranparencia de escalabilidad:** permite asumir cambios de tamaño del sistema y aplicaciones sin modificar la estructura del sistema ni los algoritmos de los programas.

2. **Eficiencia.** La idea base de los sistemas distribuidos es la de obtener sistemas mucho más rápidos que los ordenadores actuales. Con esto de nuevo nos encontramos con el paralelismo. Para lograr un sistema eficiente hay que descartar la idea de ejecutar un programa en un único procesador de todo el sistema, y pensar en distribuir las tareas a los procesadores libres más rápidos en cada momento. La idea de que un procesador vaya a realizar una tarea de forma rápida es bastante compleja, y depende de muchos aspectos concretos, como la propia velocidad del procesador, la localidad, los datos, los dispositivos, etc.
3. **Flexibilidad.** Un proyecto en desarrollo como el diseño de un sistema operativo distribuido debe estar abierto a cambios y actualizaciones que mejoren el funcionamiento del sistema. Esta necesidad ha provocado una diferenciación entre las dos diferentes arquitecturas del núcleo del sistema operativo: el **núcleo monolítico y el micronúcleo**. Las diferencias entre ambos son los servicios que ofrece el núcleo del sistema operativo. El **núcleo monolítico** ofrece todas las funciones básicas del sistema integradas en el núcleo, como ejemplo de este núcleo esta UNÍX. Estos sistemas tienen un núcleo grande y complejo, que engloba todos los servicios del sistema. Mientras tanto, el **micronúcleo** incorpora solamente las funciones fundamentales, que incluyen únicamente el control de los procesos y la comunicación entre ellos y la memoria. El resto de los servicios se cargan dinámicamente a partir de servidores en el nivel de usuario. En la actualidad la

mayoría de sistemas operativos distribuidos en desarrollo tienden a un diseño de micronúcleo. Los núcleos tienden a contener menores errores y a ser más fáciles de implementar y de corregir.

4. **Escalabilidad.** Un sistema operativo distribuido debería funcionar tanto para una docena de ordenadores como para varios millares. Al igual que no debería de ser determinante el tipo de red utilizada (LAN o WAN) ni las distancias entre los equipos, etc. La escalabilidad propone que cualquier ordenador individual debe ser capaz de trabajar independientemente como un sistema distribuido, pero también debe poder hacerlo conectado a muchas otras máquinas.
5. **Fiabilidad.** Una de las ventajas claras que nos ofrece la idea de sistema distribuido es que el funcionamiento de todo el sistema no debe estar ligado a ciertas máquinas de la red, sino que cualquier equipo pueda suplir a otro en caso de que uno se estropee o falle. La forma más evidente de lograr la fiabilidad de todo el sistema se encuentra en la redundancia. La información no debe estar almacenada en un solo servidor de archivos, sino por lo menos en dos máquinas. Mediante la redundancia de los principales archivos o de todos evitamos el caso de que el fallo de un servidor bloquee todo el sistema, al tener una copia idéntica de los archivos en otro equipo. Otro tipo de redundancia más compleja se refiere a los procesos. Las tareas críticas podrían enviarse a varios procesadores independientes, de forma que el primer procesador realizaría la tarea normalmente, pero ésta pasaría a ejecutarse en otro procesador si el primero hubiera fallado [Dueñas, R. Francisco, 2004]

En cuanto a su desarrollo tecnológico en los sistemas operativos distribuidos tenemos:

- Ingeniería de Software.
- Disminución de costos.
- Redes de comunicación de alta velocidad.
- Microelectrónica.

Además en todo sistema distribuido se establecen una o varias comunicaciones siguiendo un protocolo prefijado mediante un esquema Cliente-Servidor.

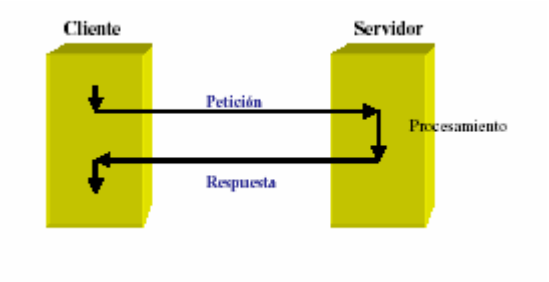


Figura 2.1. Arquitectura Cliente-Servidor de Software Distribuido

En un esquema Cliente-Servidor se denomina Cliente a la máquina que solicita un determinado servicio y Servidor a la máquina que lo proporciona. El servicio puede ser la ejecución de un determinado algoritmo, el acceso a determinado banco de información o el acceso a un dispositivo hardware. [Monge, Raúl, 2004]

Por extensión, se puede aplicar el esquema Cliente-Servidor dentro de una máquina, donde el proceso servidor y el proceso cliente son dos procesos independientes que corren dentro de la misma instancia del sistema operativo. Es por tanto un elemento primordial para que haya un sistema distribuido, la presencia de un medio físico de comunicación entre ambas máquinas. A continuación se muestra en forma gráfica esta explicación para entender mejor.

Figura 2.2. Arquitectura Cliente-Servidor con Bases de Datos

Una arquitectura es un conjunto de reglas, definiciones, términos y modelos que se emplean para producir un producto. La arquitectura Cliente-Servidor agrupa conjuntos de elementos que efectúan procesos distribuidos y computo cooperativo.

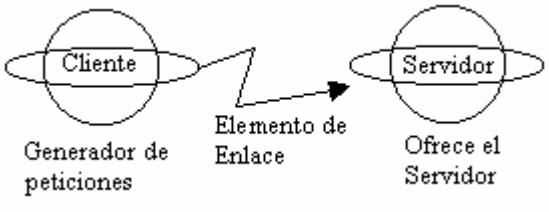


Figura 2.3. Arquitectura Cliente-Servidor

Los beneficios que ofrece esta arquitectura es un mejor aprovechamiento de la potencia de computo (reparte el trabajo), reduce el tráfico en la red (viajan requerimientos), opera bajo sistemas abiertos y permite el uso de interfaces gráficas variadas y versátiles.

El **cliente** es un conjunto de software y hardware que invoca los servicios de uno o varios servidores y algunas de sus características son:

- El cliente oculta al Servidor y la Red.
- Detecta e intercepta peticiones de otras aplicaciones y puede redireccionarlas
- Dedicado a la cesión del usuario (Inicia ...Termina)
- El método más común por el que se solicitan los servicios es a través de RPC (Remote Procedure Calls)

Y las funciones más comunes del cliente son la de mantener y procesar todo el diálogo con el usuario, el manejo de pantallas, menús e interpretación de comandos, entrada de datos y validación, procesamiento de ayudas y la recuperación de los errores.

En cuanto al Servidor, éste es un conjunto de hardware y software que responde a los requerimientos de un cliente, dentro de los tipos más comunes de Servidores se encuentran: Servidor de Archivos (FTP, Novell), Servidor de Bases de Datos (SQL, CBASE, ORACLE, INFORMIX), Servidor de Comunicaciones, Servidor de Impresión, Servidor de Terminal, Servidor de Aplicaciones (Windows NT, Novell).

Algunas de las funciones del servidor son:

- Acceso, almacenamiento y organización de datos.
- Actualización de datos almacenados.
- Administración de recursos compartidos.
- Ejecución de toda la lógica para procesar una transacción.
- Procesamiento común de elementos del servidor (Datos, capacidad de CPU, almacenamiento en disco, capacidad de impresión, manejo de memoria y comunicación).

2.1.1 Características de Sistemas Distribuidos

Algunas de las características más importantes en los sistemas distribuidos son que tienen varios procesadores autónomos, una subred de comunicación compartida que permite el paso de mensajes entre componentes de hardware y software, un estado compartido distribuido o replicado entre los participantes, concurrencia y paralelismo, cada elemento de computo tiene su propia memoria y su propio sistema operativo, control de recursos locales y remotos, sistemas abiertos (facilidades de cambio y crecimiento), plataforma no estándar (UNIX, NT, RISC, etc), dispersión y parcialidad.

- Compuesto por múltiples ordenadores. Un sistema distribuido esta compuesto por mas de un sistema independiente, cada uno, con uno o mas CPU´s, memoria local, memoria secundaria (discos) y, en general, conexiones con periféricos de acceso inmediato (on line).
- Hay interconexión entre ellos. Parece claro que si varios ordenadores distintos van a colaborar en la realización de tareas, deben comunicarse y sincronizarse entre ellos, por lo que debe haber alguna línea o red de interconexión.
- Tienen un estado compartido. Si los ordenadores realizan un trabajo conjuntamente, deben mantener un estado compartido, es decir, todo los ordenadores tienen la misma visión del estado del sistema distribuido (tablas, bases de datos del sistema, servidores, etc.)

Para entender mejor veamos la siguiente figura.

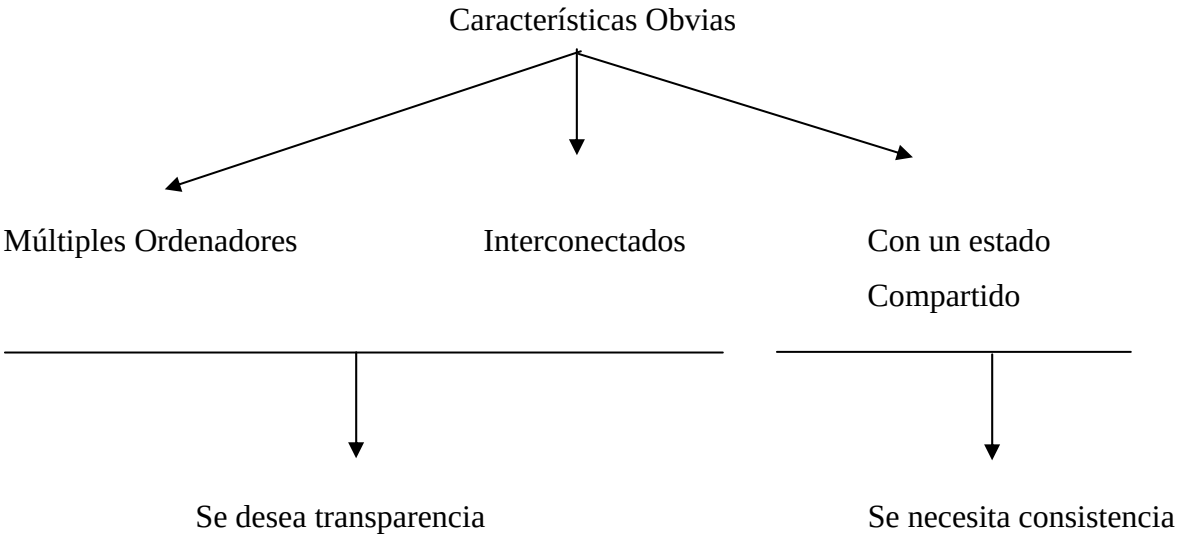


Figura 2.4 Características Sistemas Distribuidos.

2.1.2 Aplicaciones de Sistemas Distribuidos

Aplicaciones comerciales. Son aplicaciones típicas históricamente construidas con hardware dedicado y alrededor de sistemas centralizados. [Sistemas Distribuidos, 2004]

- Reservas de líneas aéreas
- Aplicaciones bancarias
- Cajeros de grandes almacenes
- Cajeros y Almacén de cadenas de supermercados

Aplicaciones para redes WAN. Dado al espectacular crecimiento de las redes de área extensa (WAN), en concreto internet, una de las áreas que más auge ha tomado ha sido la dedicada al intercambio de información a través de la red.

- Correo electrónico
- Servicio de noticias (NEWS)
- Servicio de transferencia de ficheros (FTP)
- Búsqueda de ficheros (Archie)
- Servicio de consulta textual (Gopher)
- World Wide Web (WWW)

Aplicaciones Multimedia. Son las últimas en incorporarse a los sistemas distribuidos, pues por ser isócronas imponen ciertas necesidades del hardware, especialmente en lo referente a la velocidad y regularidad de transmisión de una gran cantidad de datos.

- Videoconferencia
- Televigilancia
- Juegos multiusuario
- Enseñanza asistida por ordenador

2.1.3 Ventajas y Desventajas de los Sistemas Distribuidos

2.1.3.1 Ventajas

En general, los sistemas distribuidos (no solamente los sistemas operativos) exhiben algunas ventajas, una ventaja potencial de un sistema distribuido es una mayor confiabilidad esto es:

- Al distribuir la carga de trabajo en muchas máquinas, la falla de una de ellas no afectará a las demás, ya que la carga de trabajo podría distribuirse.
- Si una máquina se descompone, el sistema sobrevive como un todo.

Otra ventaja importante es la posibilidad del crecimiento incremental o por incrementos:

- Podrían añadirse procesadores al sistema, permitiendo un desarrollo gradual según las necesidades.
- No son necesarios grandes incrementos de potencia en breves lapsos del tiempo.
- Se puede añadir poder de cómputo en pequeños incrementos.

Algunas otras ventajas de los sistemas distribuidos son que satisfacen la necesidad de muchos usuarios de compartir ciertos datos. Por ejemplo, sistema de reservas de líneas aéreas. También se pueden compartir otros recursos como programas y periféricos costosos. Por ejemplo, impresoras láser color, equipos de fotocomposición, dispositivos de almacenamiento masivo (cajas ópticas), etc.

Otra importante razón es lograr una mejor comunicación entre las personas. Por ejemplo, mediante el correo electrónico, ya que posee importantes ventajas sobre el correo

por cartas y el fax, además, de la velocidad, disponibilidad, generación de documentos editables por procesadores de texto, etc.

La mayor flexibilidad es también otra ventaja importante en los sistemas distribuidos ya que la carga de trabajo se puede difundir (distribuir) entre las máquinas disponibles en la forma más eficaz según el criterio adoptado (por ejemplo, costos). Además de que los equipos distribuidos pueden no ser siempre PC, se pueden estructurar sistemas con grupos de PC y de computadoras compartidas, de distinta capacidad.

2.1.3.2 Desventajas

Así como los sistemas distribuidos exhiben grandes ventajas, también se pueden identificar algunas desventajas, algunas de ellas tan serias que han frenado la producción comercial de sistemas operativos en la actualidad. El problema más importante en la creación de sistemas distribuidos es el software: los problemas de compartición de datos y recursos es demasiado complejo que los mecanismos de solución generan mucha sobrecarga al sistema haciéndolo ineficiente. El checar, por ejemplo, quiénes tienen acceso a algunos recursos y quiénes no, el aplicar los mecanismos de protección y registro de permisos consume demasiados recursos.

Otros problemas de los sistemas operativos distribuidos surgen debido a la concurrencia y al paralelismo. Tradicionalmente las aplicaciones son creadas para computadoras que ejecutan secuencialmente, de manera que el identificar secciones de código “paralelizable” es un trabajo arduo, pero necesario para dividir un proceso grande en sub-procesos y enviarlos a diferentes unidades de procesamiento para lograr la distribución. Con la concurrencia se deben implantar mecanismos para evitar las condiciones de competencia, las postergaciones indefinidas, el ocupar un recurso y estar esperando otro, las condiciones de espera circulares y, finalmente, los "abrazos mortales" (deadlocks). Estos problemas de por sí se presentan en los sistemas operativos multiusuarios o multitareas, y su tratamiento en los sistemas distribuidos es aún más complejo, y por lo tanto, necesitará de algoritmos más complejos con la inherente sobrecarga esperada. [Introducción a los Sistemas Distribuidos, 2004]

Además de estos problemas también esta la administración más compleja, así como también el costo.

2.2 Arquitectura Model-View-Controller

La arquitectura MVC (*Model/View/Controller*) fue introducida como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk a la ingenieria de Software. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales son que el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas.

En la figura siguiente, vemos la arquitectura MVC en su forma más general. Hay un Modelo, múltiples Controladores que manipulan ese Modelo, y hay varias Vistas de los datos del Modelo, que cambian cuando cambia el estado de ese Modelo.

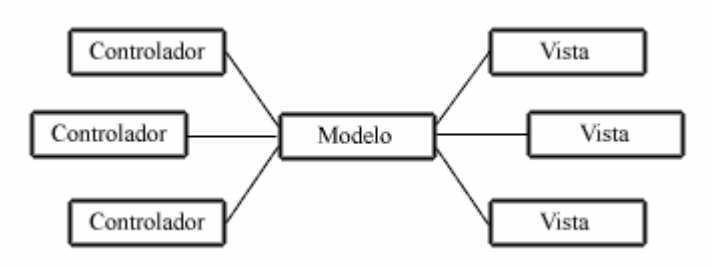


Figura 2.5. Arquitectura Model-View-Controller.

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado
- Hay un API muy bien definido; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.
- La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los Componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas. Este escenario contrasta con la aproximación monolítica típica de muchos programas Java. Todos tienen un *Frame* que contiene todos los elementos, un controlador de eventos, un montón de cálculos y la presentación del resultado. Ante esta perspectiva, hacer un cambio aquí no es nada trivial.

Algunas de las ventajas en la utilización de esta arquitectura son,

- Independencia del dispositivo que visualiza nuestra aplicación, html, sHtml, Pda's,…
- Automatización y reutilización del código en todas las aplicaciones que desarrollemos.
- Posibilidad de crear equipos de trabajo altamente especializados.

2.2.1 Definición de las partes del Model-View-Controller

El ***Modelo*** es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

La ***Vista*** es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo.

El ***Controlador*** es el objeto que proporciona significado a las ordenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo.

La siguiente figura muestra como funciona esta Arquitectura en web.

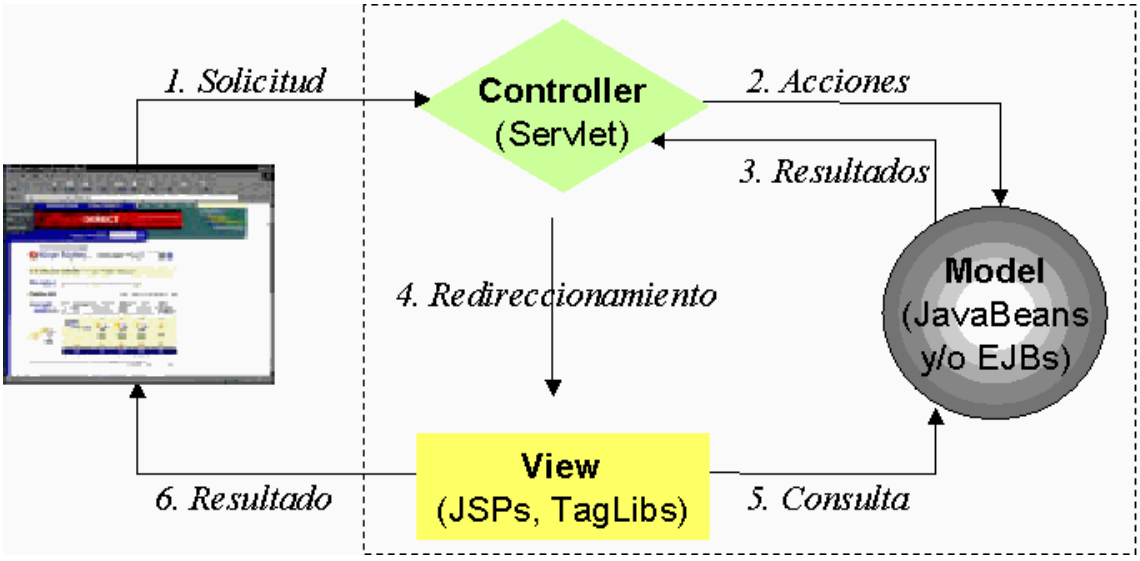


Figura 2.6. Arquitectura Model-View-Controller en web.

2.3 Análisis de los Operadores

Los operadores a implementar en este proyecto de tesis serán mencionados a continuación. Aunque existe un gran número de operadores para la realización de imágenes irreales solo utilizaremos algunos de ellos debido al tiempo y a la complejidad que algunos de estos presentan.

Antes de analizar cada uno de los operadores que vamos a implementar en este proyecto de tesis, veremos lo que es una imagen digital y sus características así como el procesamiento básico que se lleva a cabo a la hora de procesarla.

Una **Imagen** como vimos en el capitulo anterior, es una función en dos dimensiones que asigna un valor correspondiente a un nivel de intensidad a cada par de coordenadas x,y (píxel).

$$f(x,y) = i(x,y)\, r(x,y)$$

Donde i(x,y) es la función de iluminación y r(x,y) la función de reflectancia.

Píxel es la abreviatura de la expresión inglesa Picture Element (Elemento de Imagen), y es la unidad más pequeña que encontraremos en las imágenes compuestas por mapa de bits. El píxel es la unidad mínima en que se divide la retícula de la pantalla del

monitor y cada uno de ellos tiene diferente color. Su tono de color se obtiene combinando los tres colores básicos (rojo, verde y azul). Un píxel tiene tres características distinguibles:

- forma cuadrada
- posición relativa al resto de píxeles de un mapa de bits
- profundidad de color (capacidad para almacenar color), que se expresa en bits

Todas las imágenes se representan, procesan y guardan utilizando diferentes técnicas de codificación. Existen dos tipos básicos de imágenes en dos dimensiones (2D) generadas por un ordenador: **Imágenes de Mapa de Bits** e **Imágenes Vectoriales**.

En este proyecto de tesis solo analizaremos y trabajaremos con las **Imágenes de Mapa de Bits o Bitmap,** estas imágenes están compuestas por pequeños puntos o píxeles con unos valores de color y luminancia propios. El conjunto de esos píxeles, componen la imagen total. Técnicamente una Imagen Digital puede definirse como una larga cadena de código binario y sus características es que alguna de la información que se encuentra almacenada en un archivo de imagen digital, al ser interpretada mediante los dispositivos y el software adecuado permite visualizar su contenido en ordenadores tales como cámaras digitales, tv, etc. Así como también la realización de tareas de edición y composición electrónica con texto, audio, video u otras imágenes.

Las operaciones que se utilizan para la aplicación de imágenes digitales para la transformación de una imagen de entrada a[m,n] en una imagen de salida b[m,n], se clasifican en: Operaciones Individuales y Operaciones de Múltiples Puntos.

2.3.1 Operaciones Individuales.

Estas operaciones implican la generación de una nueva imagen modificando el valor del píxel en una simple localización basándose en una regla global aplicada a cada localización de la imagen original. El proceso de estas operaciones consiste en obtener el valor del píxel de una localización dada en la imagen, modificándolo por una operación lineal o no lineal y colocando el valor del nuevo píxel en la correspondiente localización de la nueva imagen. Este proceso se repite para todas y cada una de las localizaciones de los

píxeles en la imagen original. [Pajares y de la Cruz, 2002]. Dicho de otra manera, son aquellas en que el valor de cada píxel G[x,y] de la imagen resultante se obtiene a partir del valor del píxel F[x,y] de la imagen original, sin involucrar ningún otro píxel. Esto lo vemos representado en la siguiente figura.

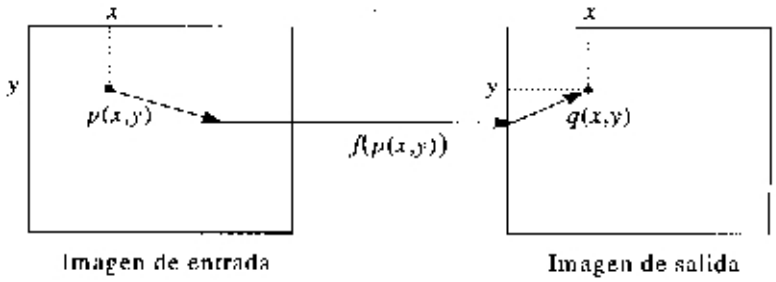


Figura 2.7. Operación Individual

Como podemos ver en la figura, el operador individual es una transformación uno a uno. La función transforma el valor del nivel de gris de cada píxel en la imagen y el nuevo valor se obtiene a través de la siguiente ecuación:

$$q\ (x,y) = f\ (p(x,y))$$

Los operadores que se encuentran dentro de este grupo son: el operador inverso (o negativo), operador umbral, umbral inverso, umbral escala de grises, umbral escala de grises inverso, umbral binario, umbral binario inverso.

2.3.1.1 Operador Inverso

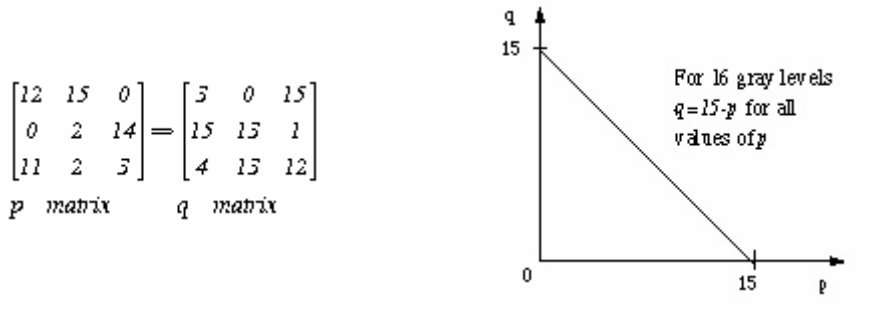
Es una operación que transforma cada uno de los píxeles de la imagen en escala de grises de forma independiente, obteniendo una imagen con sus niveles de intensidad invertidos. [Dueñas R., Francisco, 2004]

Solo es necesario aplicar la siguiente función de transformación de la imagen:

$$q = 255 - p$$

Este operador es útil en diversas aplicaciones tales como imágenes médicas.

La imagen resultado de esta transformación tiene sus niveles de intensidad invertidos, es decir, aquellos píxeles cuyos valores eran máximos ahora son mínimos y viceversa. Para entender mejor este operador analizamos la figura 2.8 en la cual tenemos nuestra matriz de entrada figura 2.8(a), a esta matriz le aplicamos la función de transformación *q=15-p* figura 2.8(c) y nos da la matriz de salida de la figura 2.8(b).



(a) (b) (c)

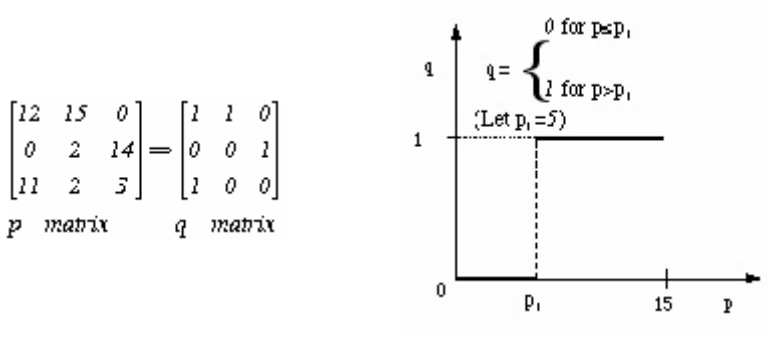
Figura 2.8. Operador Inverso.(a)matriz de entrada(b)matriz de salida(c)función

2.3.1.2 Operador Umbral

El operador umbral se aplica a una imagen cuando se requiere que una imagen tenga dos tonalidades, las cuales regularmente son blanco(255 o 1) y negro(0). Este operador utiliza un valor entero que determina que a partir de esa cantidad los valores de los píxeles menores a ella se van a convertir a valores en tono negro y todos los valores de los píxeles mayores o iguales a ella se van a convertir a valores en tono blanco. El nivel de transición esta dado por el parámetro de entrada p_1. La función de transformación es la siguiente: [Starostenko, 2004]

$$q = \begin{cases} 0 & \text{para } p \le p_1 \\ 255 & \text{para } p > p_1 \end{cases}$$

Analizando la figura 2.9 con nuestra matriz de entrada (a) y con nuestra función de transformación (c) con valor de entrada (p_1=5) tenemos como resultado la matriz (b), en donde podemos observar que nuestra matriz de salida a obtenido solo valores de ceros y unos (negro y blanco absoluto dentro ese intervalo de p_1.



(a) (b) (c)

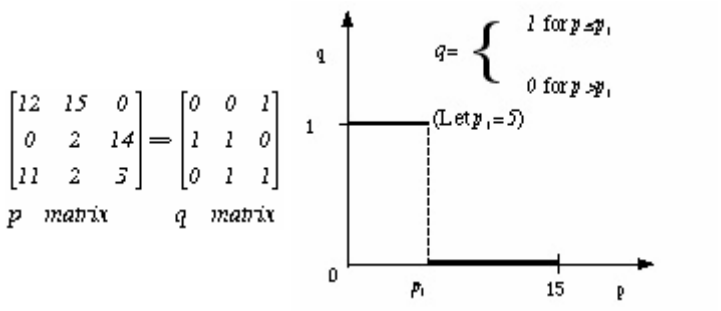
Figura 2.9. Operador Umbral.(a)matriz de entrada(b)matriz de salida(c)función.

2.3.1.3 Operador Umbral Inverso

Este operador funciona como el anterior solo que a manera inversa como su nombre lo indica, los valores que fueron mínimos serán máximos y viceversa, es decir, Al aplicar este operador los valores de los píxeles menores a p_1 se convierten a 255 o 1 y los valores de los píxeles mayores o iguales que p_1 son convertidos a cero. Para entenderlo mejor veamos la función de transformación. [Pajares y de la Cruz, 2002]

$$\mathbf{q} = \begin{cases} \mathbf{255} & \mathbf{para\ p \le p_1} \\ \mathbf{0} & \mathbf{para\ p > p_1} \end{cases}$$

Ahora analicemos nuestra figura 2.10 que es la misma matriz de entrada que la del operador umbral pero ahora aplicaremos la función de transformación de manera inversa, la cual, nos dará como resultado que cada píxel de la imagen original que era obscura ahora será blanco y los píxeles que eran blancos ahora serán obscuros.

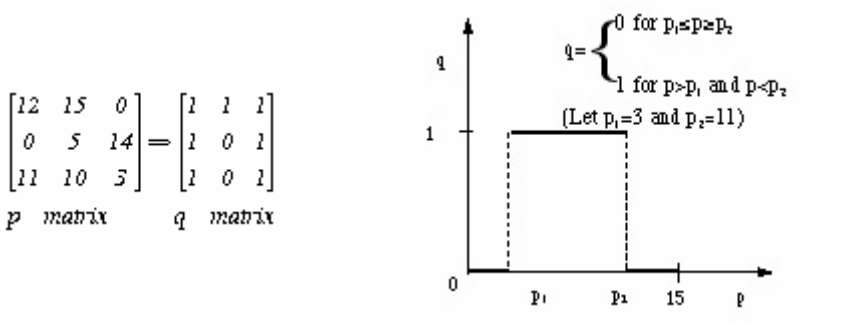


(a) (b) (c)

Figura 2.10. Operador Umbral Inverso.(a)matriz de entrada(b)matriz de salida(c)función.

2.3.1.4 Operador Umbral Binario

El operador del umbral binario crea una imagen de salida binaria a partir de una imagen de grises, donde todos los valores de gris cuyo nivel está en el intervalo definido por p_1 y p_2 son transformados a cero y todos los valores fuera de ese intervalo a 255. [Pajares y de la Cruz, 2002]



(a) (b) (c)

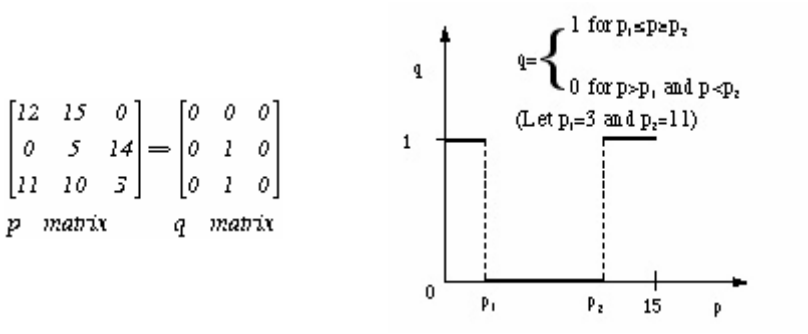
Figura 2.11. Operador Umbral Binario.(a)matriz de entrada(b)matriz de salida(c)función.

La ecuación para aplicar este operador es la siguiente:

$$q = \begin{cases} 255 & \text{para} \quad p \le p_1 \quad o \quad p \ge p_2 \\ 0 & \text{para} \quad p_1 < p < p_2 \end{cases}$$

2.3.1.5 Operador Umbral Binario Inverso

Crea una imagen de salida binaria a partir de una imagen de grises, donde todos los valores de gris cuyos niveles están en el intervalo definido por p_1 y p_2 son transformados a 255 y todos los valores fuera de ese intervalo son transformados a cero. [Dueñas R., Francisco, 2004]



(a) (b) (c)

Figura 2.12. Operador Umbral Binario Inverso.(a)matriz de entrada(b)matriz de salida(c)función.

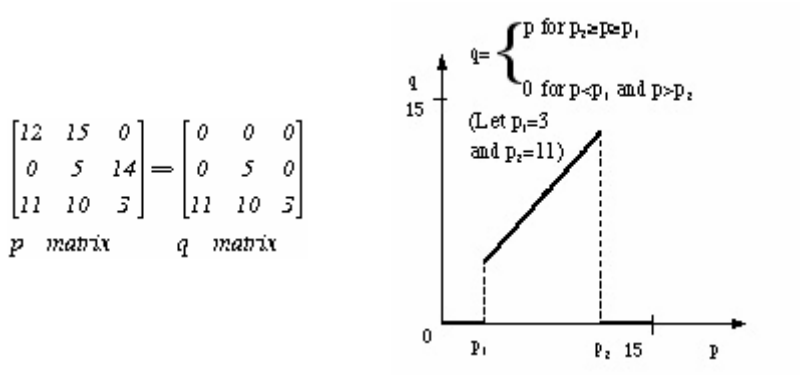
Esta función se representa con la siguiente ecuación:

$$q = \begin{cases} 0 & \text{para} \quad p \le p_1 \quad o \quad p \ge p_2 \\ 255 & \text{para} \quad p_1 < p < p_2 \end{cases}$$

2.3.1.6 Operador Umbral Escala de Grises

Crea una imagen de salida con los únicos valores de nivel de gris comprendidos en el intervalo definido por p_1 y p_2 y el resto a 255. Su función es la siguiente: [Starostenko, 2004]

$$q = \begin{cases} 255 & \text{para} \quad p \le p_1 \quad o \quad p \ge p_2 \\ p & \text{para} \quad p_1 < p < p_2 \end{cases}$$



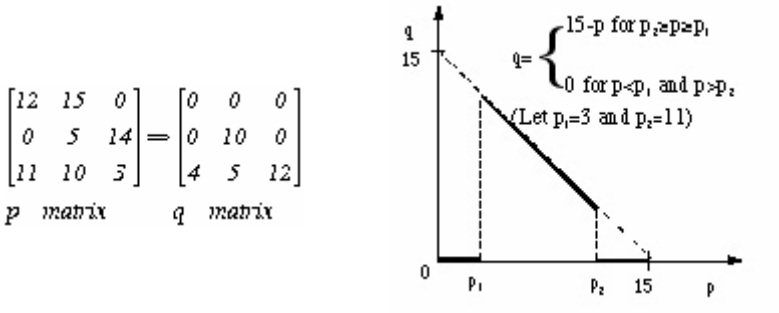
(a) (b) (c)

Figura 2.13. Operador Umbral Escala de Grises.(a)matriz de entrada(b)matriz de salida(c)función.

2.3.1.7 Operador Umbral Escala de Grises Inverso

Crea una imagen de salida con los únicos valores de nivel de gris invertidos comprendidos en el intervalo definido por p_1 y p_2 y el resto a 255. La ecuación para este operador es: [Starostenko, 2004]

$$\mathbf{q} = \begin{cases} \mathbf{255} & \mathbf{para} \quad \mathbf{p \le p_1} \quad \mathbf{o} \quad \mathbf{p \ge p_2} \\ \mathbf{255 - p} & \mathbf{para} \quad \mathbf{p_1 < p < p_2} \end{cases}$$



(a) (b) (c)

Figura 2.14. Operador Umbral Escala de Grises Inverso.(a)matriz de entrada(b)matriz de salida(c)función.

2.3.2 Operaciones Múltiples Puntos

Vistas las imágenes como matrices, cabe la posibilidad de llevar a cabo sobre ellas diferentes y diversas operaciones de naturaleza aritmética. Estas operaciones aritméticas pueden considerarse como un determinado tipo de transformación. En efecto, se trata de transformaciones que utilizan información contenida en la misma localización (posición de los píxeles) de dos imágenes de entrada A y B para crear una nueva imagen C. La función de la transformación f_D puede ser lineal o no. Esta función se aplica a todos los pares de píxeles en las imágenes de entrada, esto es, la información en una localización de píxel de una imagen se combina con la información de la correspondiente localización de la segunda imagen para obtener el valor, también en la misma localización de píxel de la imagen resultante. Dicho de otra manera, son aquellas en las que el valor de cada píxel G[x,y] de la imagen resultante se obtiene a partir de los valores del píxel F[x,y] y de sus vecinos en la imagen original. Para entender como funciona este tipo de operación veamos la siguiente figura.

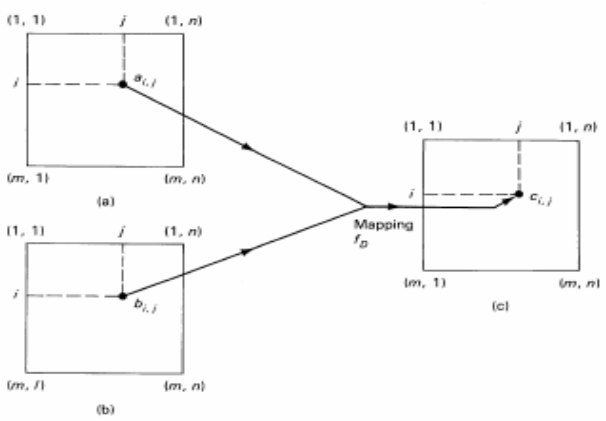


Figura 2.15. Representación de la transformación de dos imágenes en una.

La función está dada por la siguiente ecuación.

$$C_{x,y} = f_D\,(a_{x,y}\,,\, b_{x,y})$$

Donde f_D es una función de dos variables y los subíndices x e y varían de 0 a M y N, respectivamente (que son las dimensiones de las imágenes). La función f_D puede ser suma, resta, multiplicación, o cualquier otra función que opere con dos valores numéricos. La

función deberá tener un factor de escala apropiado k para mantener los valores de salida dentro del rango de niveles de gris o de intensidad adecuado, así como para evitar desbordamientos y valores negativos.

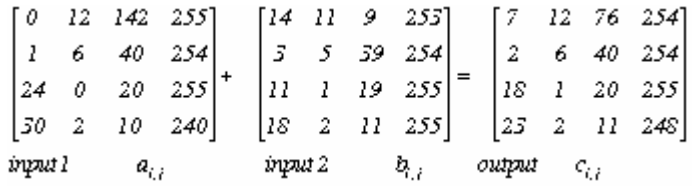
2.3.2.1 Operador Suma

Utiliza la información contenida en la misma localización de dos imágenes distintas, A y B, para crear una nueva imagen C. La dimensión de las imágenes ha de ser la misma para obtener un mejor resultado. Este operador suele utilizarse para reducir el efecto del ruido y en este proyecto de tesis lo utilizaremos además para la creación de imágenes irreales. [Pajares y de la Cruz, 2002]

El valor de salida esta dado por:

$$\mathbf{c_{(i,j)} = (a_{(i,j)} + b_{(i,j)})/k}$$

donde *K* es dos para el caso de las dos imágenes de entrada. Si la suma llegara a ser mas de dos imágenes entonces *K* tomaría el valor del número de imágenes utilizadas. Como podemos ver en la figura 2.16 tenemos nuestras matrices de entrada (a) y (b) lo que hace nuestra función de transformación es sumar cada uno de los píxeles de $a_{i,j}$ y $b_{i,j}$ para después dividirlo entre dos en este caso es dos porque son dos imágenes de entrada.



(a) (b) (c)

Figura 2.16.Operador Suma o Adición.(a)matriz de entrada $a_{i,j}$;(b)matriz de entrada $b_{i,j}$ y (c)matriz de salida $c_{i,j}$.

2.3.2.2 Operador Resta

La sustracción de imágenes es una técnica útil para detectar el cambio producido en dos imágenes que han sido captadas en dos instantes de tiempo diferentes. La relación esta dada por: [Dueñas R., Francisco, 2004]

$$c_{(x,y)} = k(a_{(x,y)} - b_{(x,y)})$$

donde *k* es una función no lineal para que el valor mínimo de *c(x,y)* sea cero y el máximo 255 y no haya valores negativos.

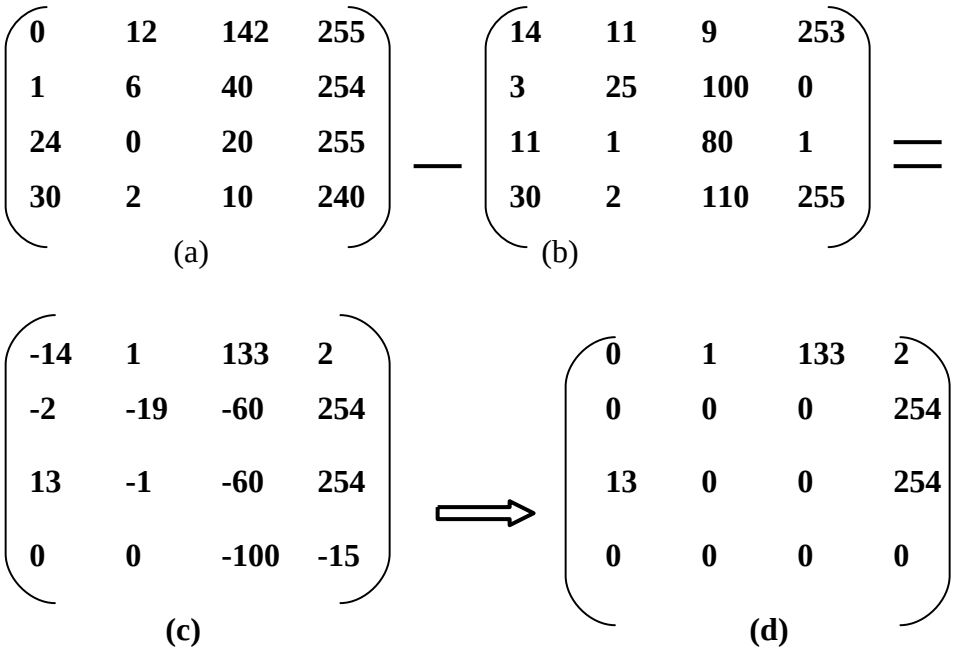


Figura 2.17.Operador Resta.(a) y (b) matrices de entrada, en donde se restan cada uno de los valores de los píxeles y que da como resultado (c), pero como da valores negativos lo convertimos en la matriz de salida (d), donde los valores negativos toman el valor de cero y los positivos de 255 o 1 (en este caso les dejamos los valores positivos que resulto de la resta de las matrices de entrada.

2.3.2.3 Convolución

La convolución, consiste en permitir que el valor asignado a un determinado píxel sea una función de su propio valor y de los valores de sus vecinos o la vecindad de un píxel que se usa para la convolución esta determinada por una máscara, ventana o kernel, que es una matriz cuyo tamaño determina el tamaño de la vecindad. Los tamaños usuales del kernel son 3x3, 5x5 y 7x7 y el centro de este píxel es el píxel que se va a modificar.

3	5	6	8	126	230
12	7	8	9	170	129
9	14	56	47	17	230
24	67	100	87	34	93
155	4	0	0	1	255
87	34	66	27	34	134

Ahora supóngase que se va a usar un kernel de tamaño 3x3:

Donde w_{ij} es un entero cualquiera. El píxel que se va a modificar es el centro de la región que en este caso es del 3x3, entonces la primera duda que aparece es que cual debe ser la región para los píxeles de la primera y última columna y de la primera y última fila de la imagen, ya que estos no tienen una región adyacente cuadrada, por ejemplo el píxel p(0,0) no tiene píxeles adyacentes a la izquierda y arriba y el píxel p(5,4) no tiene píxeles adyacentes abajo. La respuesta es que sobre esos píxeles no se realiza ninguna operación, entonces la convolución debe empezar desde el primer píxel que tenga una vecindad completa, por ser de 3x3 debe empezar en el píxel p(1,1) y debe terminar en el último píxel que tenga región completa, que es p(4,4).

w_{00}	w_{01}	w_{02}
w_{10}	w_{11}	w_{12}
w_{20}	w_{21}	w_{22}

Ahora si el píxel a modificar es p, el resultado p' es:

$$p'[x][y] = \sum_{j=0}^{M-1}\sum_{i=0}^{M-1} W[i][j] \times p\left[x-\left(i-\frac{M-1}{2}\right)\right]\left[y-\left(j-\frac{M-1}{2}\right)\right]$$

Donde W es el kernel y MxM es su tamaño. Entones para el píxel p (1,1) = 7 el valor nuevo

es:

$$p'(1,1) - 3*w_{00} + 5*w_{01} + 6*w_{02} + 12*w_{10} + 7*w_{11} + 8*w_{12} + 9*w_{20} + 14*w_{21} + 56*w_{22}$$

Luego de asignar el nuevo píxel se avanza una posición en la imagen en dirección x (p(2,1)=8) y el nuevo valor es:

$$p'(2,1) - 5*w_{00} + 6*w_{01} + 8*w_{02} + p(1,1)*w_{10} + 8*w_{11} + 9*_{w12} + 14*w_{20} + 56*w_{21} + 47*w_{22}$$

Donde p(1,1) es el valor antes de cambiar a p'(1,1). Entonces se avanza hasta el ultimo valor en la fila que tiene region adyacente (p(4,1)) y luego se avanza una fila y se hace la misma operación desde el primer píxel en la nueva fila con región adyacente (p(1,2)) y se sigue así hasta el píxel en la posición p(4,4).

2.4 Detectores de Bordes

En la actualidad se emplean diferentes tipos de detectores de bordes cada uno de ellos tiene sus ventajas y desventajas de acuerdo con el modelo matemático de borde que utilice.

La detección de bordes considera el cambio de intensidad que se produce en los píxeles en el contorno o bordes de un objeto como lo muestra la figura 2.18. Dado que se ha encontrado una región con atributos similares, pero se desconoce la forma del contorno, este último se puede determinar mediante un simple procedimiento de seguimiento del borde. Los detectores de bordes realizan otro tipo de operaciones con los datos, pero siempre con el resultado de enfatizar los bordes que rodean a un objeto en una imagen, para hacerlo más fácil de analizar. Estos filtros típicamente crean una imagen con fondo gris y líneas blancas y negras rodeando los bordes de los objetos y características de la imagen.

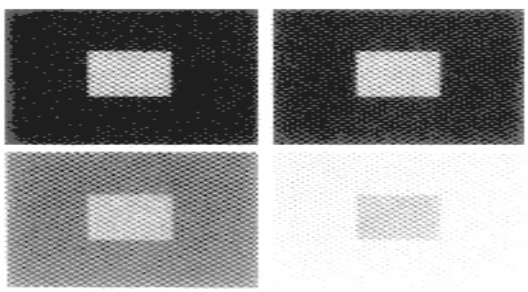


Figura 2.18. Detección de bordes en intensidad de iluminación(blanco-negro).

2.4.1 Operador Sobel

El operador gradiente de **Sobel** tiene la propiedad de realzar líneas verticales y horizontales más oscuras que el fondo, sin realzar puntos aislados. Este operador como el resto de los operadores de vecindad tiene la propiedad del suavizado de la imagen, eliminando parte del ruido subyacente y por tanto minimizando la aparición de falsos bordes.

Las mascaras utilizadas para obtener las componentes del gradiente son:

$$\begin{pmatrix} Z_1 & Z_2 & Z_3 \\ Z_4 & Z_5 & Z_6 \\ Z_7 & Z_8 & Z_9 \end{pmatrix} \quad \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

(a) (b) (c)

Figura 2.19.Mascaras Operador Sobel.(a)Región de la imagen de dimensión 3x3(b)Mascara usada para obtener G_x en el punto central de la región 3x3(c)Mascara usada para obtener G_y en el mismo punto.

Cada píxel de la imagen original se sustituye por el producto de sus vecinos por las mascaras correspondientes y de ahí:

$$G_x = (Z_3 + 2Z_6 + Z_a) - (Z_1 + 2Z_4 + Z_7) \quad \text{y}$$

$$G_y = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3)$$

Donde los distintos valores de z en la región de la figura 16(a). Son los niveles de gris de los píxeles solapados por la mascara en cualquier localización de la imagen. Para obtener los valores de las componentes del vector gradiente en el punto definido por el píxel central de la región, se utilizan las expresiones mencionadas arriba.

2.4.2 Operador Prewitt

Se diferencia del de Sobel en el valor de los coeficientes de las mascaras. [Pajares y de la Cruz, 2002]

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

(a) (b)

Figura 2.20.Mascaras Operador Prewitt.(a)Mascara usada para obtener G_x en el punto central de una región de dimensión 3x3(b)Mascara usada para obtener G_y en el mismo punto.

2.4.3 Operador Roberts

Emplea la diferenciación como método para calcular el grado de separación entre niveles de grises vecinos. Marca solamente los puntos de borde, sin informarnos sobre la orientación de estos. Es un operador muy simple que trabaja muy bien con las imágenes binarias. Opera según las dos diagonales perpendiculares mostradas en la siguiente figura. [Dueñas R., Francisco, 2004]

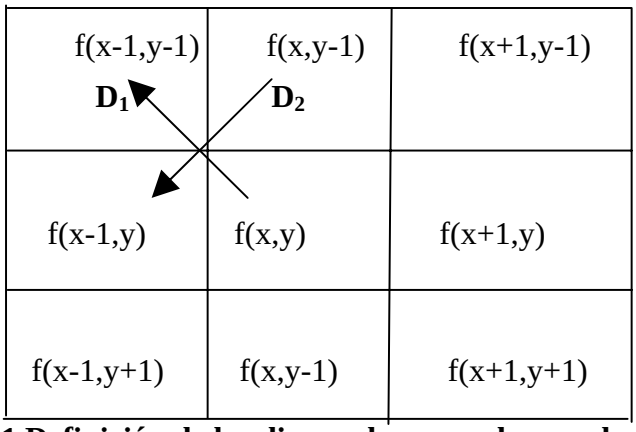


Figura 2.21.Definición de las diagonales para el operador de Roberts.

Concretamente, y para realizar una diferenciación bidimensional, se efectúa la operación:

$$g(x,y) = |f(x,y) - f(x+1,y+1)| + |f(x,y+1) - f(x+1,y)|$$

2.5 Brillo

Modificar el brillo de una imagen es añadir una cantidad constante al valor del tono de cada uno de los píxeles de la misma. El valor constante se le sumara para cada componente del color en este caso RGB. [Pajares y de la Cruz, 2002]

2.6 Zoom

Al aplicarle zoom a la imagen nos permitirá verla más grande. El cual consiste en seleccionar una pequeña porción de la imagen (subimagen) separarla del resto de la imagen original y realizar un zoom mediante una expansión. [Pajares y de la Cruz, 2002] Este proceso de expansión puede hacerse de muchas formas, pero generalmente se utilizan dos y que serán también utilizados en este proyecto de tesis el del vecino más cercano y el zoom bilineal los cuales explicaremos a continuación.

2.6.1 Zoom Vecino más cercano

Este tipo de zoom consiste en repetir los valores de los píxeles previos, para crear un efecto de bloques. La matriz de entrada es N x M y la matriz de salida será 2N x 2M. Para entender mejor este operador veamos la siguiente figura.

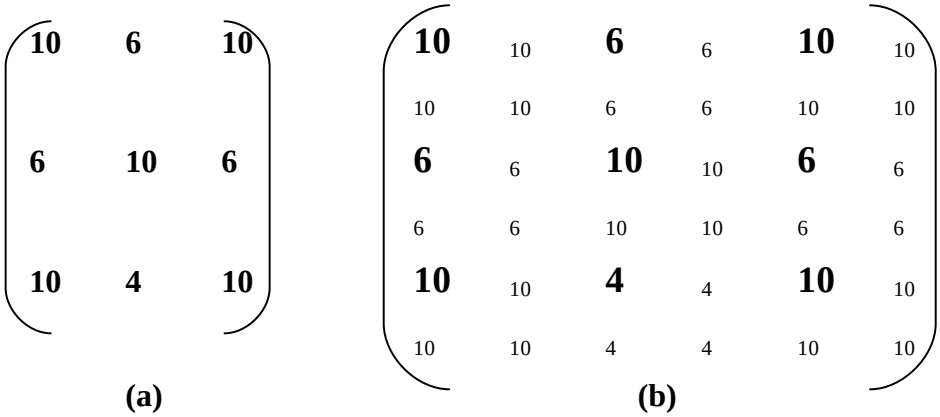


Figura 2.22.Operador Zoom Vecino mas Cercano.(a)matriz de entrada,(b)matriz de salida con nuevos valores del píxel los valores en negritas son de la matriz original y los demás son los valores de los píxeles mas cercanos.

2.6.2 Zoom Bilineal

En el zoom bilineal se utiliza interpolación lineal. Se encuentra el valor medio entre dos píxeles y se usa dicho valor como el valor del píxel entre los dos. Analicemos la siguiente figura para entender mejor este operador.

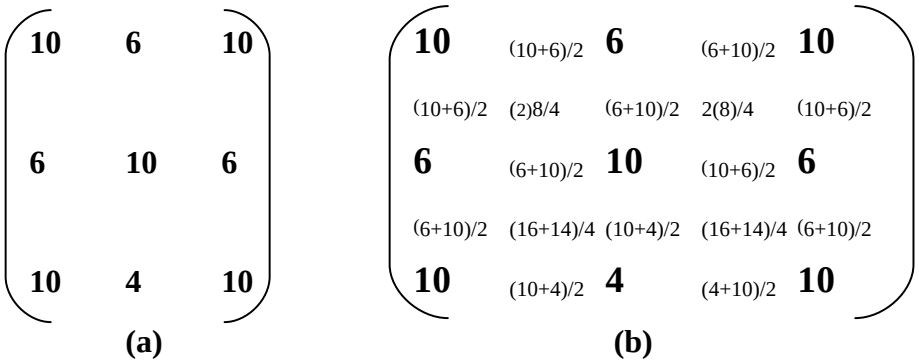


Figura 2.23.Operador Zoom Bilineal.(a)matriz de entrada,(b)matriz de salida, los valores en negritas es la matriz original y los demás son los valores medios que se obtienen de el valor medio entre dos píxeles.

2.7 Operador Flip

Este operador funciona como un espejo, como su nombre lo indica voltea la imagen y hay dos tipos de Flip, el Flip Horizontal y el Vertical. Este tipo de operador puede ser útil en el área de serigrafía o cerámica.

2.7.1 Operador Flip Horizontal

El funcionamiento de este operador consiste en cambiar los números de la matriz de posición, es decir los números que estaban primeros serán ahora los últimos y estos serán los primeros. Para entender mejor este operador veamos las siguientes matrices.

$$\begin{pmatrix} 10 & 5 & 20 & 35 \\ 25 & 15 & 12 & 10 \\ 15 & 0 & 20 & 30 \end{pmatrix} \qquad \begin{pmatrix} 35 & 20 & 5 & 10 \\ 10 & 12 & 15 & 25 \\ 30 & 20 & 0 & 15 \end{pmatrix}$$

(a) (b)

Figura 2.24.Operador Flip Horizontal.(a)matriz de entrada,(b)matriz de salida.

2.7.2 Operador Flip Vertical

Este operador cambia los valores de la matriz verticalmente como su nombre lo dice, es decir, los números que se encuentren en la última fila serán los primeros y viceversa. Veamos las siguientes matrices para entender mejor este operador.

$$\begin{pmatrix} 10 & 5 & 20 & 35 \\ 25 & 15 & 12 & 10 \\ 15 & 0 & 20 & 30 \\ 25 & 85 & 41 & 21 \end{pmatrix} \qquad \begin{pmatrix} 25 & 81 & 41 & 21 \\ 15 & 0 & 20 & 30 \\ 25 & 15 & 12 & 10 \\ 10 & 5 & 20 & 35 \end{pmatrix}$$

(a) (b)

Figura 2.25.Operador Flip Vertical.(a)matriz de entrada,(b)matriz de salida.

2.8 Rotación

Los giros de una imagen suelen utilizarse para lograr efectos estéticos, pero su uso más importante es para similar la rotación de la cámara de captura o la del propio objeto. Los parámetros necesarios para simular la rotación son el ángulo de giro y las coordenadas del centro de rotación.

2.9 Selección del Lenguaje de Programación

Utilizaremos Java como lenguaje de programación debido a las muchas características que este posee para la creación de algoritmos en el procesamiento de imágenes los cuales no permitirán la creación de las imágenes irreales.

Desde el punto de vista tecnológico Java es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable, también proporciona un conjunto de clases potente y flexible, así como también pone al alcance de cualquiera la utilización de aplicaciones que se pueden incluir directamente en páginas Web (sistema distribuido). Además de aportar a la Web una interactividad entre el usuario y aplicación.

A continuación mencionaremos algunas características que este lenguaje de programación tiene:

Lenguaje simple: posee una curva de aprendizaje muy rápida

Orientado a Objetos: Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos.

Distribuido: proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Interpretado y compilado a la vez: es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).

Robusto: Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo los punteros, y la recolección de basura elimina la necesidad de liberación explícita de

memoria.

Seguridad: se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

Indiferente a la arquitectura: Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.

Portable: La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la *Máquina Virtual Java* (JVM).

Alto rendimiento multithread: soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas.

Dinámico: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas.

Produce Applets: las applets son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

2.9.1 Servlet´s

Los servlet´s son componentes del servidor. Estos componentes pueden ser ejecutados en cualquier plataforma o en cualquier servidor debido a la tecnología Java que se usa para implementarlos. Los servlet´s incrementan la funcionalidad de una aplicación Web. Se cargan de forma dinámica por el entorno de ejecución Java del servidor cuando se necesitan. Cuando se recibe una petición del cliente, el contenedor/servidor Web inicia el

servlet requerido. El servlet procesa la petición del cliente y envía la respuesta de vuelta al contenedor/servidor, que es enrutada al cliente. Es por todas estas características que utilizamos Servlet´s para la creación de la herramienta SCII.

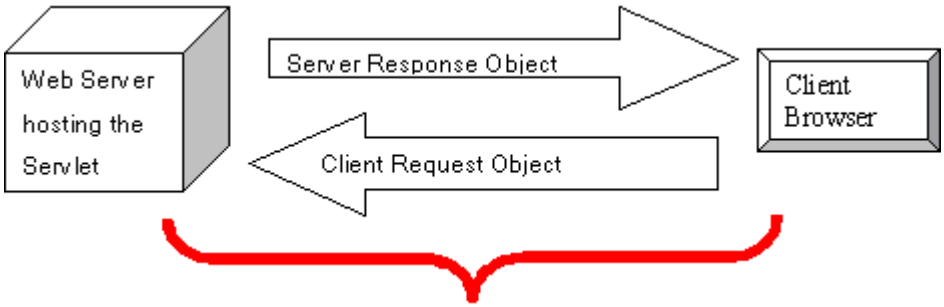


Figura 2.26. Modelo de Respuesta a Peticiones HTTP

2.9.1.1 Estructura del Servlet y Ciclo de Vida

Antes de ver el ciclo de vida de los servlet´s, necesitamos comprender las clases básicas y las interfaces usadas en la implementación del servlet.

Public Interface Servlet. Todo servlet debe directa o indirectamente implementar esta interface. Como cualquier otra interface de Java, este es también una colección de declaraciones vacías de métodos. Los siguientes métodos están declarados en la interface Servlet.

1.public abstract void init (ServletConfig config) throws ServletException.

El método init se usa para inicializar los parámetros proporcionados por el objeto ServletConfig. Se invoca sólo una vez, a menos que el servlet sea reiniciado si se destruye y se vuelve a cargar. El método init es el lugar en el que inicializar los parámetros de configuración como la conexión con una base de datos, la inicialización de archivos y otros valores de entorno. Ningún método del servlet puede ser invocado a no ser que el servlet esté inicializado mediante el uso del método init().

2.public abstract ServletConfig getServletConfig ()

Este método proporciona el objeto ServletConfig para la inicialización de los parámetros del servlet. Se pueden crear parámetros adicionales especificándolos en el archivo servlet.properties. Una vez que hayan sido especificados en este archivo, se puede acceder a ellos usando el objeto ServletConfig.

3.public abstract void service (ServletRequest req, ServletResponse res) throws ServletException, IOException.

El método service es el punto esencial del modelo petición respuesta del protocolo HTTP. Recibe una petición del cliente en forma de objeto ServletRequest. Los parámetros del cliente son pasados junto al objeto de petición (aunque existen otras formas de enviar parámetros desde el cliente al servlet, por ejemplo, usando cookies o por medio de una reescritura del URL). La respuesta resultante se envía al cliente usando el objeto ServletResponse.

4.public abstract String getServletInfo()

Este método se usa para la extracción de metadatos del servlet, como por ejemplo el autor, la versión del servlet, y otra información concerniente al copyright. El método tendrá que ser redefinido dentro del servlet para que devuelva esta información.

5.public abstract void destroy ()

El método destroy se invoca para liberar todos los recursos solicitados como la base de datos, y otros recursos del servidor. También se encarga de la sincronización de cualquier hilo pendiente. Este método se llama una sola vez, automáticamente, como el método init.

La clase GenericServlet proporciona una implementación básica de la interface Servlet. Para escribir un servlet especificamente para el protocolo HTTP, se usa la clase HTTPServlet, que extiende a Generic Servlet (Figura 2.27).

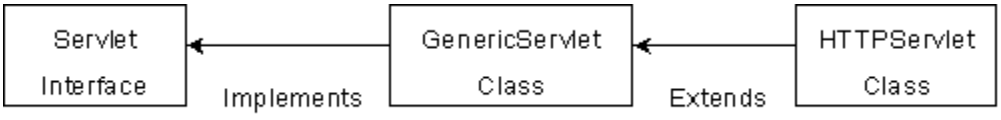


Figura 2.27. Clases Relevantes de los Servlet´s

Como se ha tratado previamente, el ciclo de vida de los eventos para un servlet (Figura 2.28) se especifica en la interface javax.servlet.Servlet. Todos los servlets siguen el modelo del ciclo de vida. El contenedor web tiene la responsabilidad de crear una instancia del servlet y de invocar al método init (1). Si un cliente ha enviado una petición al contenedor web, entonces, esa petición se pasa al método servicio del servlet (2), y se envia una respuesta de vuelta al contendor web (3). Finalmente, cuando el servlet haya finalizado su propósito, el contenedor web invoca al método destroy (4).

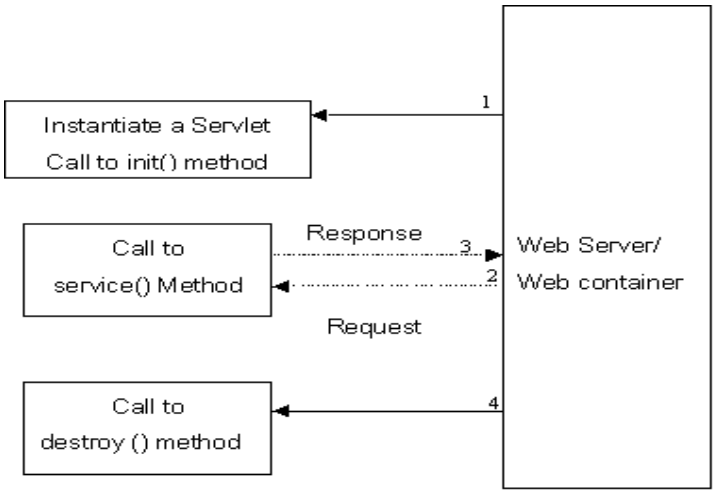


Figura 2.28. Ciclo de Vida de un Servlet.

2.10 Configuración del Sistema

Para el desarrollo de este proyecto de tesis fue necesario la instalación del compilador de Java así como también la instalación de un compilador Web en este caso es el Apache Tomcat, el cual servirá de soporte para el servidor ya que de otra manera no podremos ejecutar nuestro sistema SCII.

2.10.1 Instalación del Java Development Kit

El JDK (Kid de Desarrollo de Java), es un conjunto de utilerías basadas en texto para programas de línea de comandos, que no utilizan una interface gráfica de usuario. Los programadores ejecutan cada una de las utilerías del JDK al escribir un comando en una terminal.

La versión 1.4.1_01 del JDK se encuentra disponible para las siguientes plataformas: Windows 95, 98, 2000, NT 4.0 y XP, Solaris SPARC, x86 y Linux. Las cuales, puede descargar en la página de Sun http://java.sun.com .

Los requerimientos mínimos para la instalación del JDK son:

- Procesador Pentium a 166 MHz
- Memoria Ram 32Mb
- Espacio en Disco Duro 80Mb

Antes de instalar el JDK en el sistema, debemos asegurarnos que no haya otras herramientas de desarrollo de Java instaladas, ya que tener más de una herramienta de programación Java puede causar problemas de configuración cuando tratemos de usar el JDK. Para instalarlo damos doble clic en el archivo de instalación, después de ver un cuadro de diálogo que nos pregunta si deseamos instalar el JDK 1.4.1_01, se desplegará en asistente de configuración del JDK, que nos pregunta sobre el directorio predeterminado en el que se desempaquetará el contenido del archivo, nuestro archivo está ubicado en el siguiente directorio, C:\j2sdk1.4.1_01
El asistente instalará tres componentes del JDK:

1. Archivos de programa. Los programas ejecutables para crear, compilar y verificar el funcionamiento de las clases de Java.
2. Archivos de biblioteca y encabezados. Archivos usados únicamente por los programadores que hacen llamadas a código nativo desde sus programas de Java.
3. Archivos de demostración. Son programas de Java 2, con versiones que puede ejecutar y archivos fuente que puede examinar para aprender más de cerca del lenguaje.

Es necesario que el sistema sepa en que directorio esta ubicado nuestro JDK, lo cual lo hacemos mediante las variables de entorno, con esto podremos compilar las clases en Java, visualizar applets, etc. En la siguiente figura vemos el directorio en el que se encuentra instalado nuestro JDK.

Figura 2.29. Variable de entorno del sistema JAVA_HOME.

Ya que hicimos la instalación es importante verificar que el JDK se haya instalado correctamente, para esto, abrimos una consola o terminal y tecleamos lo siguiente: java –versión, con esto nos desplegara la versión actual del JDK como se muestra en la siguiente figura.

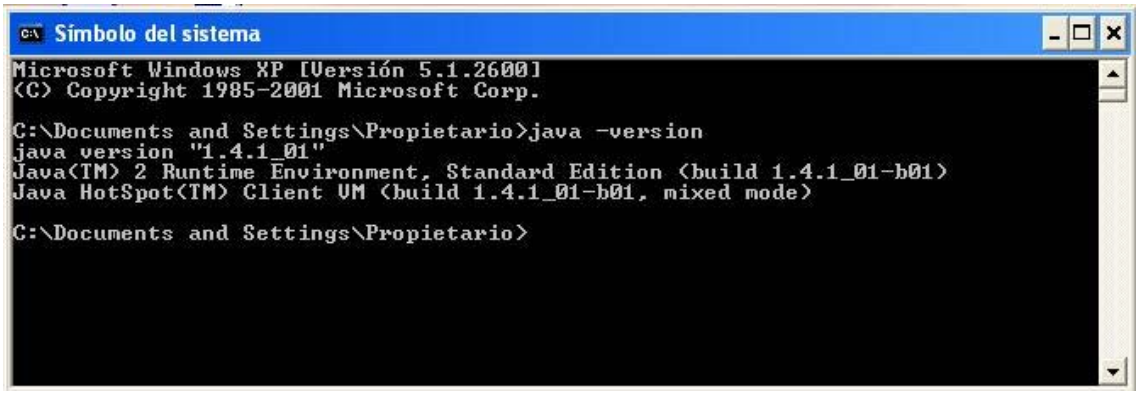


Figura 2.30. Imagen que muestra la verificación del JDK.

El JDK en Solaris se puede instalar en las siguientes plataformas:

1. Sistemas SPARC que ejecuten Solaris 2.4 o superior
2. Sistemas x86 que ejecuten Solaris 2.5 o superior

El archivo de instalación del JDK deberá ser desempacado en un directorio que no tenga previamente un subdirectorio llamado jsdk 1.4.1_01; si lo hay podría sobrescribir algunos archivos existentes en el sistema. Para comprobar que se descargó el archivo de instalación correctamente hágalo desde el comando chmod a+x con el nombre del archivo. [Lemay, L. y Cadenhead R. 1999] Los usuarios de SPARC usarían el comando siguiente:

% chmod a+x jsdk1.4.1_01-solaris2-sparc.bin

Para instalar el JDK después de hacer el cambio con chmod, solo se teclea lo siguiente:

% ./jdk1.4.2-solaris2-sparc.bin

2.10.2 Instalación del Apache Tomcat

Tomcat es un contenedor de Servlets con un entorno JSP, lo cual significa que dicho contenedor es un shell de ejecución que manejará e invocará a los servlets por cuenta del usuario y que además es compatible con las especificaciones Servlets 2.2 y JSP 1.1. El contenedor Tomcat para Servlets y JSP es un paquete gratuito desarrollado como parte del proyecto Jakarta de la Fundación de Software Apache.

Apache Tomcat se encuentra y puede descargarse en la siguiente dirección http://jakarta.apache.org.

Al igual que con el compilador Java, es necesario establecer las variables de entorno del sistema. Para el perfecto funcionamiento del sistema, primero creamos la variable CATALINA_HOME, la cual va apuntar al directorio raíz de Tomcat , en este caso es C:\tomcat

Figura 2.31. Variable de entorno del sistema CATALINA_HOME

Al igual que con el JDK, es necesario crear una segunda variable de entorno del sistema que permitirá hacer visibles las clases contenidas en los paquetes javax.servlet.*; y javax.servlet.http.*;. Esto lo hacemos agregando al CLASSPATH del sistema el archivo servlet.jar;. Como se muestra en la siguiente figura.

Figura 2.32. Modificación del PATH para hacer accesible el paquete javax.servlet.*;

Una vez realizado lo anterior podremos levantar y bajar el servidor Web Tomcat como se muestra en las siguientes figuras.

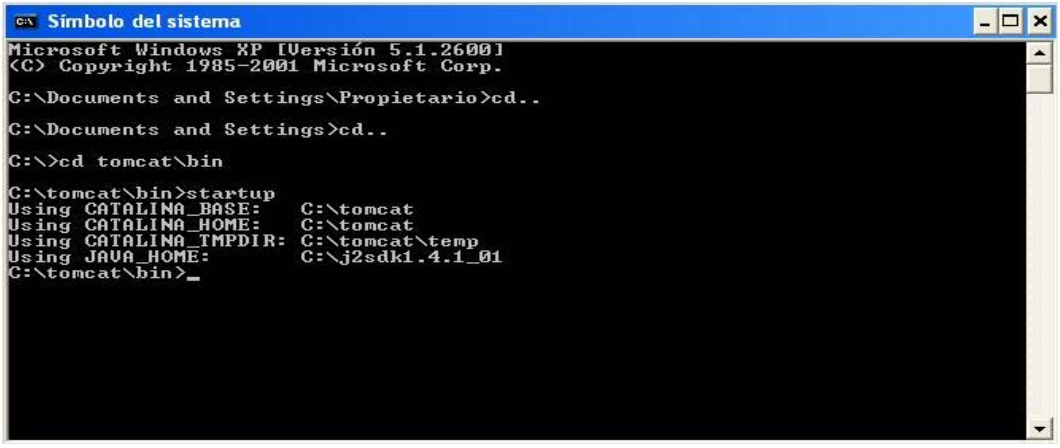


Figura 2.33. Imagen que muestra cuando se inicia el servidor Tomcat.

Ahora vemos la siguiente imagen, la cual, nos muestra que ya se levanto el servidor Tomcat.

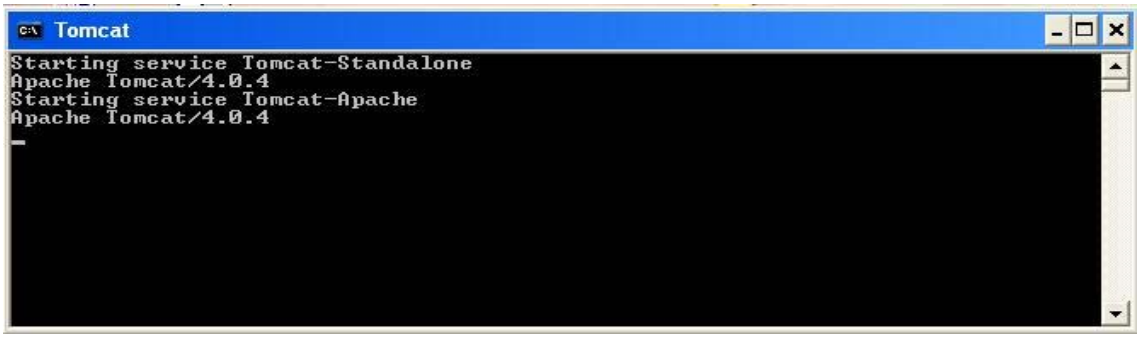


Figura 2.34. Imagen que muestra que ya esta arriba el servidor Tomcat.

Para comprobar la instalación del Tomcat y asegurarnos que si esta arriba dicho servidor solo abrimos un navegador y hacemos referencia a la siguiente dirección http://localhost:8080 y nos mostrara la siguiente imagen. Este puerto lo tiene por default, pero el usuario lo puede cambiar si así lo requiere, mas adelante mencionamos como se puede cambiar este puerto. Para acceder al sistema SCII desde la red de la UDLA-P, se hace por medio de la siguiente dirección http://localhost:9589/Tesis

Figura 2.35. Imagen que muestra la página de bienvenida del servidor Tomcat.

Ahora para bajar el servidor escribimos en la consola shutdown como se muestra en la siguiente figura y se bajara el servidor(véase figura 2.36).

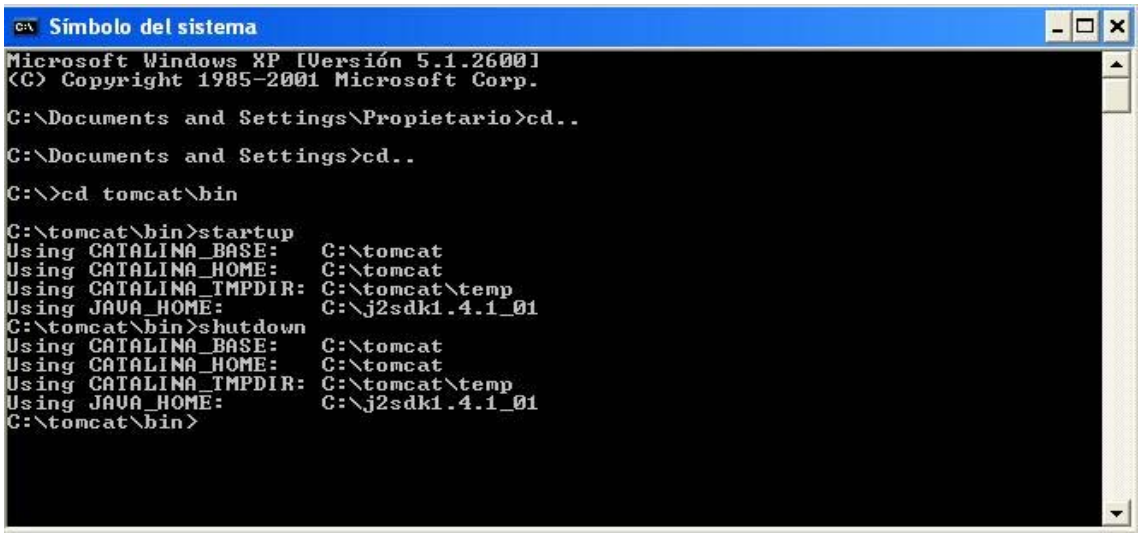


Figura 2.36. Imagen que muestra cuando se baja el servidor Tomcat

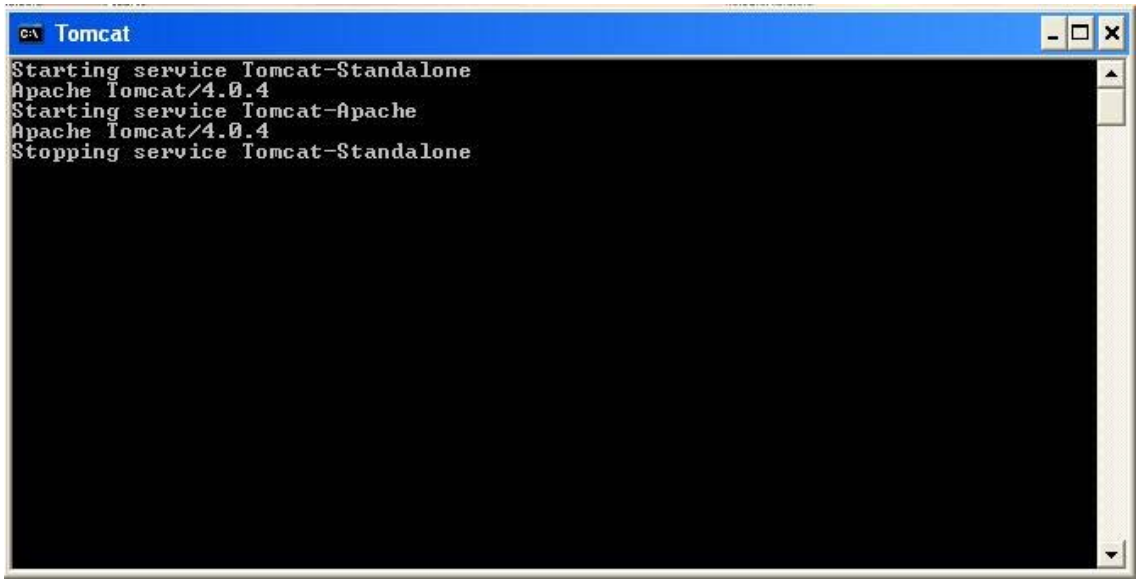


Figura 2.37. Imagen que muestra que se esta bajando el servidor Tomcat.

El puerto que tiene por default el servidor es el 8080, pero este valor se puede modificar si el usuario así lo desea, la modificación se hace en el archivo server.xml, el cual se encuentra en el directorio $TOMCAT_HOME\conf\.

2.11 Conclusiones

Analizando los operadores aquí mencionados es como los elegimos para su implementación analizados a fondo los encontraremos en el capítulo 4, el motivo de haberlos elegido es por su velocidad, complejidad y eficiencia que estos operadores presentan. Además de mostrar la configuración del sistema.

En el siguiente capítulo hablaremos del diseño del sistema, en el cual, mostraremos los diagramas de clase, de casos de uso y de secuencia para entender mejor a nuestro sistema.