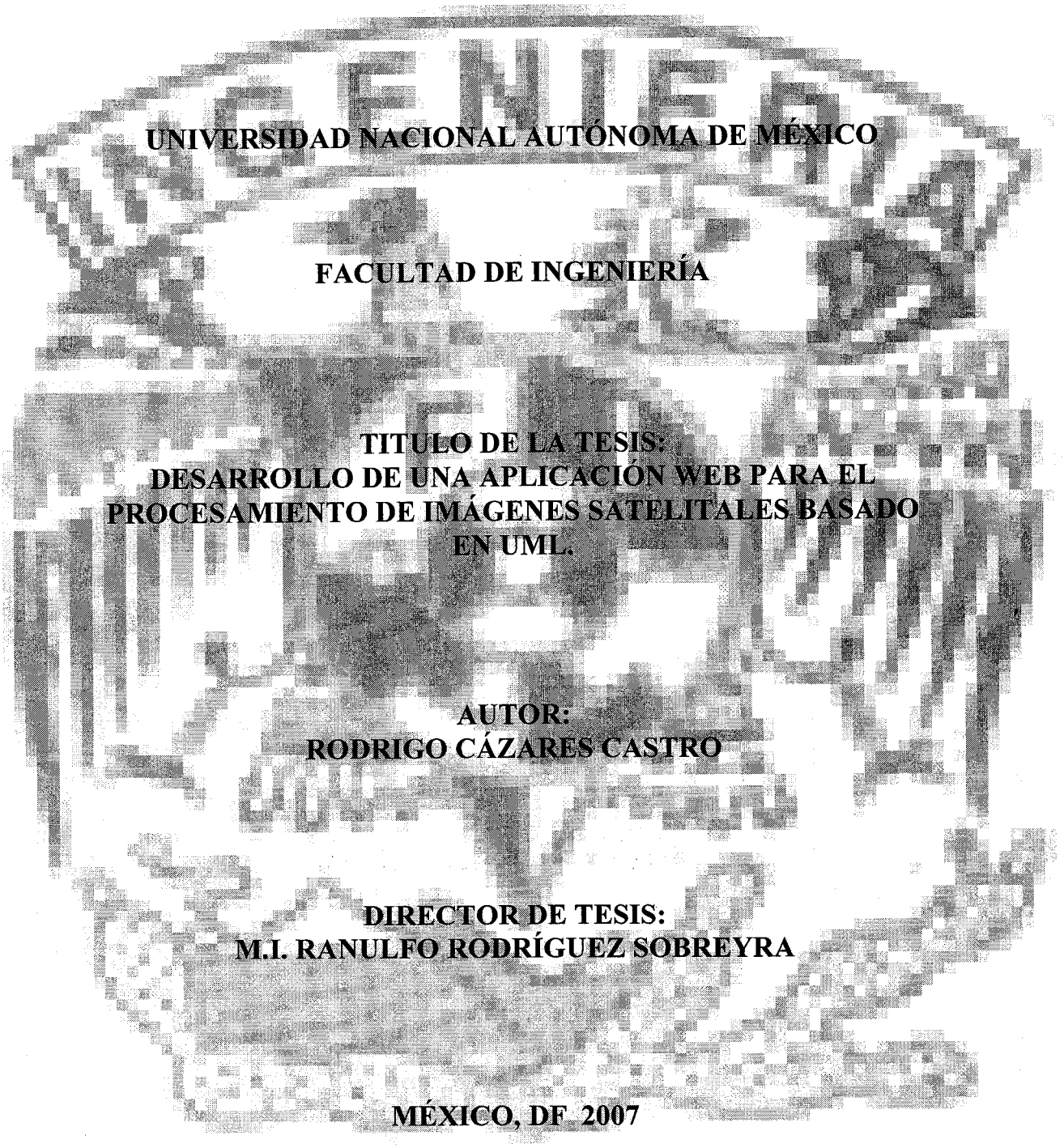


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TITULO DE LA TESIS:
DESARROLLO DE UNA APLICACIÓN WEB PARA EL PROCESAMIENTO DE IMÁGENES SATELITALES BASADO EN UML.

AUTOR:
RODRIGO CÁZARES CASTRO

DIRECTOR DE TESIS:
M.I. RANULFO RODRÍGUEZ SOBREYRA

MÉXICO, DF. 2007

CONTENIDO

INTRODUCCIÓN

CAPÍTULO 1
Conceptos básicos de programación Orientada a Objetos y Metodologías de diseño.

CAPÍTULO 2
Aplicaciones en la Web.

CAPÍTULO 3
Metodología UML.

Objetivo General

INTRODUCCIÓN

Unified Modeling Language "UML", se ha convertido en la notación estándar para definir, organizar y visualizar los elementos que configuran la arquitectura de un sistema. Un sistema es algo "compuesto", una construcción realizada por manos y herramientas siguiendo las directrices de un propósito. La palabra se aplica casi exclusivamente a abstracciones con el fin de captar la totalidad de una realidad.
A través de la notación UML podemos comunicar y compartir el conocimiento de una arquitectura gracias a la combinación simultánea de cinco perspectivas:

1. Definir.- Fijar, determinar, decidir, explicar un concepto a través de sus atributos distintivos. Señalar sus límites y dar una idea exacta de lo que es esencial y de lo que es circunstancial.

2. Organizar.- Establecer unos recursos, disponer un orden de responsabilidades y formalizar unas reglas de relación y actuación; todo ello orientado a conseguir un propósito.

3. Visualizar.- Representar mediante imágenes y/o símbolos el contenido y la organización de los conceptos que configuran un sistema. Hacer visible su naturaleza y su complejidad.

4. Actuar.- Pensar y tomar decisiones de manera ágil y sistemática, siguiendo un método; éste a su vez, define el modo de actuar en base a la relación de un conjunto de actores, actividades, entregables y certificaciones posibles en un escenario concreto.

5. Certificar.- Comprobar de manera fehaciente que un código es completo, coherente y utilizable para el propósito que ha sido creado.

Una de las características más relevantes de la notación UML es su capacidad para absorber nueva semántica sin romper su lógica interna.
La necesidad de implementar servicios Web a través de complejas arquitecturas con múltiples capas de componentes y una gran dispersión geográfica de nodos, ha supuesto todo un reto al abordar su modelado y especificación.
No hay que confundir la implementación de un Web site con el desarrollo de una aplicación Web. Un "Web site" es relativamente estático. En cambio, la aplicación Web es mucho más dinámica, dispone de una lógica de negocio que puede reaccionar y alterar su estado a partir de la interacción con un usuario.
Su contrapartida es la complejidad, ya que requiere implementar una arquitectura que se adapte a los cambios constantes, que facilite su ágil integración con otros sistemas y que resuelva picos variables de interacción con un buen rendimiento.

Objetivo General

Desarrollar una aplicación Web que permita procesar la información contenida en imágenes satelitales; aplicando conocimientos básicos de la programación orientada a objetos, procesamiento digital de imágenes y mediante el estudio de la metodología UML, se pretende realizar el sistema de procesamiento de imágenes satelitales vía Internet. Los requerimientos permitirán definir rutinas para procesar la información de las imágenes satelitales de la temperatura superficial del mar de los mares mexicanos y aguas internacionales adyacentes.

Objetivos Particulares

La identificación de las metodologías existentes para el diseño de sistemas.

La identificación clara de lo que son las aplicaciones Web y sus alcances; así como también el conocimiento de las herramientas de desarrollo de estas.

El estudio de la Metodología UML como herramienta de diseño para el desarrollo de software.

Para cumplir con los objetivos se presenta este trabajo, el que comprende los siguientes capítulos:

Capítulo1: Conceptos básicos de programación Orientada a Objetos y Metodologías de diseño.

Este capítulo es una descripción de lo que es la Programación Orientada a Objetos, la definición de algunos de sus elementos principales, y una descripción breve de las metodologías para el diseño de software existentes y su estructura básica.

Capítulo2: Aplicaciones en la Web

Se presenta una explicación de lo que son las aplicaciones Web, sus ventajas y desventajas, herramientas de diseño y desarrollo de estas así como los factores que deben de tomarse en cuanta al desarrollar una aplicación Web.

Capítulo3: Metodología UML

Es un tratado de la Metodología UML que incluye definiciones de elementos que forman parte de UML, el estudio de lo diagramas básicos para el diseño de cualquier sistema bajo la metodología UML, así como los tipos de modelado y el proceso de desarrollo.

Capítulo4: Desarrollo de la Aplicación

El diseño de la aplicación Web para la cual se desarrolla esta tesis, se hace presente en este capitulo. Se presentan los requerimientos con que deberá contar el sistema, los casos de uso conforme a la metodología UML, el diagrama de clases, los diagramas de colaboración correspondientes a los casos de uso formulados además del diagrama de secuencia general diseñado para el funcionamiento general del sistema.

Capítulo5: Resultados

Se hace un breve descripción del sistema final obtenido, después de realizar la programación correspondiente al diseño UML formulado en el capitulo anterior.

Anexo

Este anexo presenta el código java completo que se ha desarrollado, el cual corresponde a la aplicación Web que se describe como resultado en el capítulo 5 y que se desprende del diseño descrito en el capítulo 4.

CAPÍTULO 1

Conceptos básicos de programación Orientada a Objetos y Metodologías de diseño.

1.1 Programación Orientada a Objetos

1.1.1 ¿Qué es la programación orientada a objetos?

Grady Booch, autor del método de diseño orientado a objetos, define la programación orientada a objetos (POO) como:

Un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representan una instancia de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de herencia.

Existen tres importantes partes de la definición: la programación orientada a objetos; 1) utiliza *objetos*, no algorítmicos, como bloques de construcción lógicos, 2) cada objeto es una instancia de una *clase*, y 3) las clases se relacionan unas con otras por medio de relaciones de herencia. Un programa puede parecer orientado a objetos pero si cualquiera de estos elementos, no es un programa orientado a objetos [Joyanes, 1998].

Los conceptos fundamentales de programación son: o*bjetos, clases, herencia, mensajes y polimorfismo.*

Los conceptos de la programación orientada a objetos tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard del Centro de Cómputo Noruego en Oslo. Según se informa, la historia es que trabajaban en simulaciones de naves, y fueron confundidos por la explosión combinatoria de cómo las diversas cualidades de diversas naves podían afectar unas a las otras. La idea ocurrió para agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamiento. Fueron refinados más tarde en Smalltalk, que fue desarrollado en Simula en Xerox PARC pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "en marcha" en lugar de tener un sistema basado en programas estáticos.
La programación orientada a objetos tomó posición como la metodología de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de C++, una extensión del lenguaje de programación C. Su dominación fue consolidada gracias al auge de las Interfaces gráficas de usuario, para los cuales la programación orientada a objetos está particularmente bien adaptada.

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp, Pascal, y otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y a la capacidad de mantenimiento del código. Los lenguajes orientados a objetos "puros", por otra parte, carecían de las características de las cuales muchos programadores habían venido depender. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos. El Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos pero ahora ha sido esencialmente reemplazado por Java, en gran parte debido a la aparición de Internet, para la cual Java tiene características especiales.
Apenas como programación procedual conducida a los refinamientos de la técnica, tales como la programación estructurada, los métodos modernos de diseño de software orientado a objetos incluyen refinamientos tales como el uso de los patrones de diseño, diseño por contrato, y lenguajes de modelado (tales como UML) [Web_1].

1.1.2 El objeto

La idea fundamental de los lenguajes orientados a objetos es combinar en una sola unidad datos y funciones que operan sobre esos datos. Tal unidad se denomina objeto. Por consiguiente, dentro de los objetos residen los datos de los lenguajes de programación tradicionales, tales como números, arreglos (*arrays*), cadenas y registros, así como funciones o subrutinas que operan sobre ellos.
Un objeto es una entidad que contiene los atributos que describen el estado de un objeto del mundo real y las condiciones que se asocian con el objeto del mundo real. Se designa por un nombre o identificador del objeto.

1.1.3 Clases

Una clase es la descripción de un conjunto de objetos; consta de métodos y datos que resumen características comunes de un conjunto de objetos. Se pueden definir muchos objetos de la misma clase. Dicho de otro modo, una clase es una declaración de un tipo de objeto. Las clases son similares a los tipos de datos y equivalen a modelos o plantillas que describen como se construyen ciertos tipos de objetos. Cada vez que se construye un objeto a partir de una clase estamos creando lo que se llama una *instancia* de clase. Por consiguiente, los objetos no son más que una instancia de una clase.

1.1.4 Herencia

El encapsulamiento es una característica muy potente, y junto con la ocultación de la información, representan el concepto avanzado de objeto, que adquiere su mayor relevancia cuando encapsula e integra datos, más las operaciones que los datos de dicha entidad. Sin embargo, la orientación a objetos se caracteriza, además de por las propiedades anteriores, por incorporar características de herencia, propiedad que permite a los objetos ser construidos a partir de otros objetos. Dicho de otro modo, la capacidad de un objeto para utilizar estructuras de datos y métodos previstos en antepasados ascendientes. El objetivo final es la reutilización, es decir reutilizar código anteriormente ya desarrollado.

La herencia supone una clase base y una jerarquía de clases que contienen las clases derivadas de la clase base. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos a ellas, incluso cambiar aquellos elementos de la clase base que necesita sean diferentes.
Existen dos mecanismos de herencia utilizados comúnmente en programación orientada a objetos: herencia simple y herencia múltiple.
En la herencia simple, un objeto (clase) puede tener sólo un ascendente, o dicho de otro modo, una sub-base puede heredar datos y métodos de una única clase, así como añadir o quitar comportamientos de una clase base. La herencia múltiple es la propiedad de una clase de poder tener más de un ascendente inmediato, o lo que es igual, adquirir datos y métodos de más de una clase.

1.1.5 Mensajes

Los procedimientos y funciones, denominados métodos o función miembro, residen en el objeto y determinan como actúan los objetos cuando reciben un mensaje. Un mensaje es la acción que hace un objeto. Un método es el procedimiento o función que se invoca para actuar sobre un objeto. Un método especifica como se ejecuta un mensaje.
El conjunto de mensajes a los cuales puede responder un objeto se denomina protocolo del objeto. Cuando de ejecuta un programa orientado a objetos ocurren tres sucesos. Primero los objetos se crean a medida que se necesitan. Segundo, los mensajes se mueven de un objeto a otro (o desde un usuario a un objeto) a medida que el programa procesa información internamente o responde a la entrada del usuario. Tercero, cuando los objetos ya no son necesarios, se borran y se libera la memoria.

1.1.6 Polimorfismo

Otra propiedad importante de la programación orientada a objetos es el polimorfismo. Esta propiedad, en su expresión más simple, es el uso de un nombre o símbolo para representar o significar más de una acción. Así, en C, Pascal y Fortran, los operadores aritméticos representan un ejemplo de esta característica. La utilización de operadores o funciones de formas diversas se denomina polimorfismo (múltiples formas). Cuando un operador existente en un lenguaje tal como +, = ó * se le asigna la posibilidad de operar sobre un nuevo tipo de dato, se dice que está sobrecargado. La sobrecarga es una clase de polimorfismo, que también es una característica importante de POO. Un uso típico de los operadores aritméticos es la sobrecarga de los mismos para actuar sobre tipos de datos definidos por el usuario (objetos), además de sobre los tipos de datos definidos por el que se tienen tipos de datos que representan las posiciones de puntos en la pantalla de la computadora.

Las tecnologías orientadas a objetos son uno de los motores clave de la industria del software. El desarrollo de programas orientados a objetos es un enfoque diferente del mundo informático. Implica la creación de modelos del mundo real y la construcción de programas informáticos basados en modelos. El proceso completo de programación comienza con la construcción de modelos del mundo real y la construcción de programas informáticos basados en esos modelos. El proceso completo de programación comienza con la construcción de un modelo suceso (evento) real y la construcción de programas. El

resultado final del proceso es un programa de computadora que contiene características que representan algunos de los objetivos del mundo real que son parte del suceso.
El principio básico de la programación orientada a objetos es que un esquema de software se ve como secuencia de transformaciones en un conjunto de objetos. El termino objeto tiene el mismo significado que un nombre o una frase nominal. Es una persona, un lugar o una cosa.

1.1.7 Abstracción

Por medio de la abstracción conseguimos no detenernos en los detalles concretos de las cosas que no interesen en cada momento, sino generalizar y centrarse en los aspectos que permitan tener una visión global del tema. La abstracción consiste entonces en la generalización conceptual de los atributos y propiedades de un determinado conjunto de objetos. Precisamente la clave de la programación orientada a objetos esta en abstraer los métodos y los datos comunes a un conjunto de objetos y almacenarlos en una clase. Una abstracción que describa un conjunto de objetos en términos de una estructura de datos encapsulada u oculta y las operaciones sobre esa estructura, la denominamos *tipo abstracto de datos.*

1.1.8 Encapsulamiento

El *encapsulamiento* u *ocultamiento de la información* se refiere a la práctica de incluir dentro de un objeto todo lo que necesita, de tal forma que ningún otro objeto necesite conocer nunca su estructura interna. Esta característica permite ver un objeto como una caja negra, en la que se ha metido de alguna manera toda la información relacionada con dicho objeto. Esto nos permitirá manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.
[Ceballos, 1998]

1.2 Metodologías de diseño Orientadas a Objetos

Las metodologías de desarrollo de sistemas tratan las siguientes fases del ciclo de vida del desarrollo de sistemas: planeación, análisis, diseño, protección y transición. La planeación de la estrategia produce modelos de alto nivel de un negocio y, con éstos, define un plan para desarrollar un conjunto de proyectos de sistemas interrelacionados. Al análisis del sistema modela un área de sistemas basados en ideas y conceptos de los expertos de dominio proponiendo cualquier decisión relacionada con la instrumentación. El diseño del sistema desarrolla un modelo de instrumentación o implementación basado en los modelos conceptuales desarrollados durante el análisis del sistema. La construcción del sistema implica la elaboración y la prueba de programas, base de datos y redes de acuerdo con el que haya quedado definido durante el diseño del sistema. La transición del sistema instala los sistemas construidos.

Cuando analizamos sistemas, creamos modelos del área de aplicación que nos interesa.
Un modelo puede incorporar un sistema, centrarse en el área de la empresa o abarcar toda la empresa. El modelo representa un aspecto de la realidad y se construye de modo que nos ayude a comprender a esta. Con el análisis orientado a objetos, la forma de modelar la realidad difiere del análisis convencional. Modelamos el mundo en términos de tipos

de objetos y lo que le ocurre a éstos. Los modelos que construimos en el análisis OO reflejan la realidad de modo más natural que los del análisis tradicional de sistemas. Mediante las técnicas OO, construimos software que modela más fielmente el mundo real. Cuando el mundo real cambia, nuestro software es más fácil de cambiar, lo que es una ventaja real.

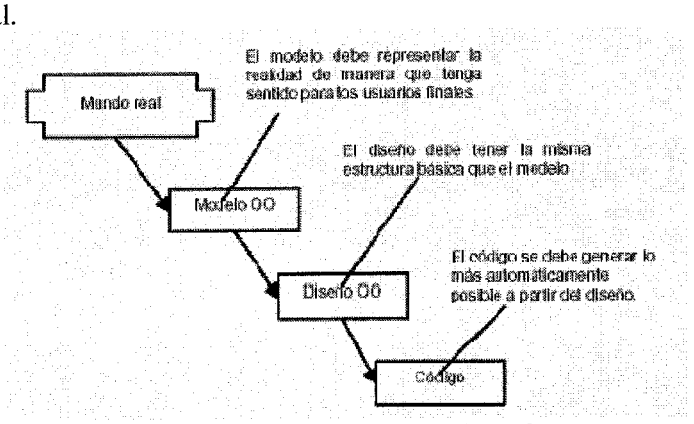


Fig1. Análisis y diseño orientado a objetos

Para el presente trabajo nos enfocamos al estudio y comprensión de las metodologías OMT (Object Modeling Technique) desarrollada por James Rumbaugh, la metodología propuesta por Booch (y que lleva su mismo nombre) y finalmente la metodología OOSE desarrollada por Ivar Jacobson.

1.2.1 Metodología OMT (RUMBAUGH)

La metodología OMT es una técnica de modelado de objetos, desarrollada por James Rumbaugh, que es uno de los precursores del Lenguaje Unificado de Modelado (UML).
El significado de las siglas de esta metodología es Técnica de Modelado en Objetos (Object Modeling Technique), la definen como una de las metodologías de la Ingeniería de Software aplicable al desarrollo orientado a objetos en las fases de análisis y diseño.
La fase de análisis comienza con la declaración del problema que incluye, una lista de objetivos (metas) y conceptos claves definitivos definidos para el dominio del problema a resolver. La declaración del problema se "expande" después en tres modelos:

- Modelo de objetos
- Modelo dinámico
- Modelo funcional

El modelo de objetos representa los objetos del sistema. El modelo dinámico representa la interacción entre esos objetos representados como eventos, estados y transiciones. El modelo funcional representa los métodos del sistema desde la perspectiva de flujo de datos. La fase de análisis genera diagramas del modelo de objetos, diagramas de estado, diagramas de eventos de flujo y diagramas de flujos de datos. Es entonces cuando se tiene completa la fase de análisis.
Después de la fase de análisis, se sigue con la fase de diseño de sistema. Aquí se define la arquitectura completa del sistema. Primero el sistema se organiza en subsistemas que

están asignando a ciertos procesos y tareas, tomando en cuenta la colaboración y concurrencia entre ellos. Luego, el almacenamiento persistente de datos se establece por medio de una estrategia de manejo de información global compartida. Después, se examinan las situaciones límite para ayudar a establecer las prioridades de negociación. La fase de diseño de objetos viene después de la fase de diseño del sistema. Aquí se establece el plan de implementación. Se definen las clases de objetos, así como sus algoritmos, poniendo especial atención con la optimización y persistencia de datos. Se definen cuestiones de herencia, asociaciones, agregaciones y valores por omisión.

La metodología OMT es secuencial en el sentido de que la primera fase es la de análisis, seguida por el diseño. En cada fase, se hacen aproximaciones iterativas entre los pequeños pasos a seguir. La metodología OMT es muy similar a la metodología Booch, cuyo principal criterio es hacer énfasis en las fases de diseño y análisis para una primera entrega del producto. Ambas OMT y Booch no hacen prioritarias las fases de implementación, evaluación u otras del ciclo de vida. OMT pone especial atención en el modelo y uso de modelos para lograr una abstracción, en el cual el análisis está enfocado en el mundo real a nivel de diseño, también pone detalles particulares para modelado de recursos físicos. Esta tecnología es aplicable en varios aspectos de implementación incluyendo archivos, bases de datos relacionales y orientadas a objetos. OMT se construye alrededor de descripciones de estructuras de datos, constantes, sistemas de procesos de transacciones.
OMT hace énfasis en especificaciones declarativas de la información, captura de manera transparente los requerimientos, especificaciones imperativas para poder descender prematuramente en el diseño y declaraciones que permiten optimizar los estados.

1.2.1.1 Conceptualización

El desarrollo comienza con el análisis de la empresa o negocio, la visión que tienen los usuarios del sistema y la formulación de requerimientos.
La conceptualización se hace frecuentemente por la re-ingeniería de procesos del negocio, es decir, se pretende tener una observación crítica de los procesos de la empresa y su impacto económico.

En esta etapa se plantean preguntas como:

- ¿Cuál es la aplicación?
- ¿Qué problemas tendrán que ser resueltos?
- ¿Dónde será usado el sistema?
- ¿Cuándo será requerido el sistema?
- ¿Para qué es necesario el sistema?

1.2.1.2 Análisis

Los requerimientos establecidos durante la conceptualización son revisados y analizados para la construcción del modelo real. La meta del análisis es especificar las necesidades que deben ser satisfechas.
Pueden existir diversas fuentes de información que pueden ser útiles en esta etapa.

Puede existir algún lenguaje formal para describir el problema. Algunas veces los expertos del dominio pueden proveer escenarios, *storyboards* y casos de uso para un nuevo sistema.
Aquí es donde se determina el modelo del objeto, se hace una tentativa de clases, se eliminan las clases irrelevantes, se definen las posibles asociaciones entre las clases y se hace la refinación de ellas eliminando las redundantes o las que no tienen relevancia, posteriormente se hace una tentativa de atributos de objetos y enlaces.
Una vez determinados los objetos del sistema, se hace la depuración del modelo, posteriormente se busca un nivel de abstracción para modelar subsistemas, para buscar un sistema tangible y sólido.
Luego de tener definido el modelo, se introduce la noción de transacción, que es una forma de modelar procesos o describir cambio de datos y flujo de datos, una vez definido el flujo de datos se define el diccionario de datos de todas las entidades modeladas.

1.2.1.3 Diseño del sistema

El diseño tiene un alto nivel estratégico y decisivo para la resolución de problemas. Los problemas grandes se deben ver desde el punto de vista de análisis y diseño, subdividir el sistema en subsistemas, poder modular los subsistemas de tal manera que cada modulo sea manejable y comprensible.
En esta etapa se deben crear estrategias, formular una arquitectura para el sistema y las políticas que deben guiarla, además del detalle de diseño. Se deben considerar los siguientes aspectos:

- Visualización de la arquitectura
- Elegir un tipo de implementación para control extremo
- Si se usa una base de datos, definir el paradigma a utilizar
- Determinar oportunidades para el re-uso
- Elegir estrategia para interacción de datos
- Elegir una forma de identificar objetos
- Detalle del diseño

Durante el diseño del sistema se debe hacer un "cuadro" de estrategias y decisiones en cuestiones de arquitectura, tener ideas precisas de clases y métodos individuales.
Adicionalmente se puede mejorar el modelo de diseño para mejorar la implementación. Se deben considerar los siguientes puntos:

- Uso de transformaciones para simplificar y optimizar el modelo de objetos desde el análisis.
- Elaborar un modelo de objeto.
- Elaborar un modelo funcional.
- Evaluar la calidad del diseño del modelo.
- Implementación.

Luego, el diseño se traduce a un lenguaje de programación, es decir, se codifica. Este paso puede considerarse para las etapas de análisis y diseño para elevar el desempeño del sistema.

1.2.1.4 Mantenimiento

La documentación del desarrollo y seguimiento de los modelos a través de código facilita el posterior mantenimiento.
La metodología OMT soporta múltiples estilos de desarrollo. Se puede usar OMT para conseguir un alto desempeño en las fases de análisis, diseño e implementación. Primero se desarrolla el núcleo del sistema, se analiza, diseña, implementa y se genera el código fuente.
La idea principal de OMT es la conceptualización de una entidad que permita el manejo de atributos y asociaciones, sus transformaciones y transacciones para el modelado orientado a objetos.

1.2.2 Metodología BOOCH

La metodología Booch se enfoca principalmente al diseño de estado de un proyecto. Booch describe una serie de propiedades generales de los sistemas complejos bien estructurados. Los sistemas construidos con una metodología de análisis y diseño orientado a objetos deben satisfacer estas propiedades.
En el análisis y diseño orientado a objetos, el dominio del problema se modela a partir de dos perspectivas distintas. La estructura lógica del sistema y la estructura física. Para cada perspectiva (dinámica y estática) se modela la semántica.
La metodología Booch define diferentes modelos para la descripción de un sistema. El modelo lógico (dominio del problema) se representa en la estructura clase-objeto. En el diagrama de clase, se construye la arquitectura y el modelo estático. El diagrama de objeto, representa la interacción de clases entre sí, captura algunos momentos de la vida del sistema y ayuda en la descripción del modelo dinámico.
La arquitectura del modelo y del proceso describe la ubicación física de las clases en módulos y procesos.

1.2.2.1 El proceso del desarrollo O.O.

Booch soporta el desarrollo iterativo e incremental de un sistema.

Procesos "macro":
- Establecer los requerimientos del núcleo (conceptualización)
- Desarrollar un modelo del comportamiento deseado (análisis),
- Crear la arquitectura (diseño),
- Evolucionar la implementación (evolución),
- Administrar la evolución posterior a la entrega (mantenimiento).

Procesos "micro":
- Identificar las clases y objetos a cierto nivel de abstracción.
- Identificar la semántica de estas clases y objetos.
- Identificar las relaciones entre las clases y objetos.
- Especificar la interfaz y después la implementación de las clases y objetos.

La metodología Booch cubre las fases de análisis y diseño de un sistema O.O. Esta metodología en ocasiones es criticada por el uso de muchos símbolos diferentes. Es cierto que Booch define muchos símbolos que documentar casi para cada decisión de diseño. Si se trabaja con esta metodología, uno se percata de que nunca se utilizan todos estos símbolos y diagramas. Se comienza con diagramas clase-objeto en la fase de análisis y se depuran en varios pasos. Únicamente cuando se está listo para generar código, se agregan algunos símbolos de diseño. Y está es la parte en que la metodología Booch es fuerte, realmente es posible documentar el código orientado a objetos.

Representación de clases (el comportamiento dinámico de clases se representa mediante diagramas de estados), la utilidad de clases que denota clases con miembros estáticos, clase parametrizada, categoría de clase. Booch recomienda que sólo se muestren los atributos y/o las operaciones que son importantes en el contexto del diagrama, con ello estos diagramas son más fáciles de leer o agregar algunas propiedades de diseño. Sintaxis para atributos:

- A: nombre del atributo únicamente,
- C: atributo de clase únicamente
- AC: nombre de atributo y clase
- A:C=E nombre de atributo, clase y expresión por omisión

Sintaxis para métodos:
- N() , nombre de operación únicamente,
- RN(argumentos), función de regreso de la clase, nombre y argumentos formales.
- {} , engloba clase

1.2.3 Metodología OOSE (IVAR JACOBSON)

Este método proporciona un soporte para el diseño creativo de productos de software, las actividades consisten en la transformación de un conjunto de requerimientos en un plan estructurado de construcción y un plan de acción. El diseño creativo tomando como referencia una base arquitectónica es seguir paso a paso los métodos y procesos con la asistencia de herramientas, para convertir los requerimientos dentro de una arquitectura viable para la construcción de un proyecto incluyendo la creación de prototipos.
A grandes rasgos el modelo consta de cuatro pasos generales.

1.2.3.1 Modelo de análisis
Especifica el comportamiento funcional del sistema bajo prácticamente circunstancias ideales y sin hacer alusión a un ambiente particular de implementación.

- *Construcción*

La primera actividad en la construcción consiste en la implementación de los detalles que conciernen a la arquitectura y construcción del plan, que es ir de una mayor abstracción a concretizar más el plan.

• *Diseño*
Formaliza el modelo de análisis en términos del ambiente de implementación y especifica la identidad de los bloques de construcción

• *Prueba del sistema*
Consiste en la verificación del trabajo de cada uno de los paquetes de servicio definidos en el modelo de análisis Esta fase tiene lugar en varios niveles, desde funciones específicas, hasta el sistema completo.

Fig2. Metodología OOSE

Alguno más específico se trabaja con 5 modelos:

- El modelo de requerimientos: El objetivo es la captura de requerimientos funcionales.
- El modelo de análisis: El objetivo es dar al sistema una estructura de objetos robusta y flexible a los cambios.
- Modelo de diseño: Tiene como objetivo adoptar y refinar la estructura de objetos en el ambiente actual de implementación.
- El modelo de implementación: Tiene como objetivo implementar el sistema.
- El modelo de prueba: Su objetivo es verificar el sistema.

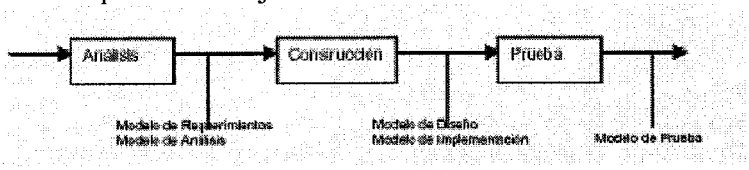


Fig3. Modelado Metodología OOSE

El proceso de análisis produce dos modelos, a partir de la especificación de requerimientos, un modelo de requerimientos es creado para especificar toda la funcionalidad del sistema. Esto es principalmente hecho por: casos de uso en el modelo de casos de uso, el cual forma parte del modelo de requerimientos.
El modelo de requerimientos es la base de otro modelo creado por el proceso de análisis, llamado modelo de análisis. El modelo de análisis es la base de la estructura del sistema.
En este modelo se especifican todos los objetos lógicos que serán incluidos en el sistema y como están relacionados y agrupados.

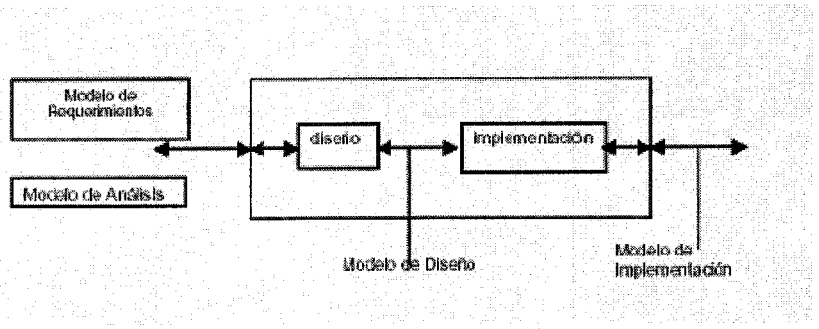


Fig4. Relaciones entre los modelos

1.2.3.2 Modelo de requerimientos

La primera transformación hecha de la especificación de requerimientos para el modelo de requerimientos consiste en:

- Un modelo de caso de uso
- Descripción de la interfaz
- Un modelo en el dominio del problema

El modelo de caso de uso controla la formulación de otros modelos. Esto es desarrollado en cooperación con el modelo de dominio de objeto.

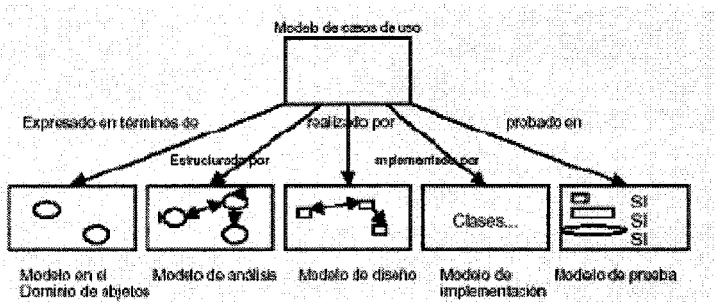


Fig5. Modelo de requerimientos

Se ha visto que el modelo de requerimientos tiene como objetivo definir las limitaciones del sistema y especificar su comportamiento. Cuando el modelo de requerimientos ha sido desarrollado y aprobado por los usuarios se puede iniciar el desarrollo del sistema.
La información para este sistema se enfoca en la captura de:

Información: Especifica la información de ayuda en el sistema. Así como describe el estado interno del sistema.
Comportamiento: Especifica el comportamiento que adopta el sistema. Especifica cuando y como el sistema cambia de estado.
Presentación: detalla la presentación del sistema al mundo exterior.
Existen varios tipos de objetos usados para la estructura del sistema en el modelo de análisis. Cada objeto al menos captura dos de las tres dimensiones del modelo de análisis, sin embargo cada uno de ellos tiene cierta inclinación hacia una de las dimensiones.

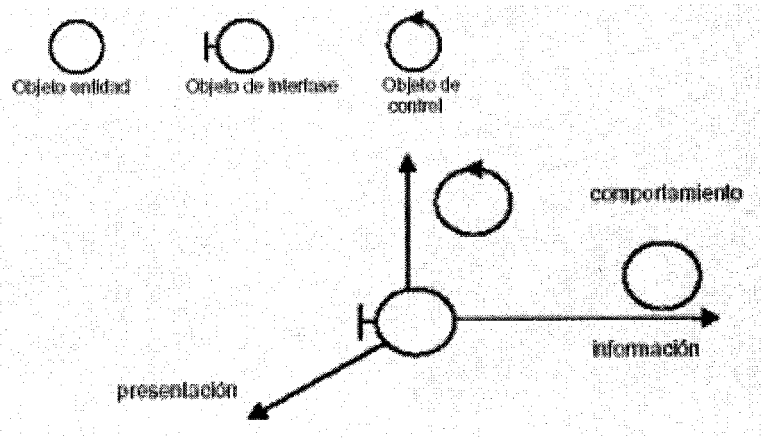


Fig6. Modelo de análisis

1.2.3.3 Modelo de diseño de objetos

El proceso de construcción edifica el sistema usando tanto el modelo de análisis como el modelo de requerimientos. Primero se crea el modelo de diseño que es un refinamiento y formalización del modelo de análisis. Al inicio del trabajo cuando se desarrolla el modelo de diseño es para adaptarlo a la implementación del ambiente actual.

Una diferencia entre el modelo de análisis y el modelo de diseño es que el modelo de análisis debe ser visto como un modelo conceptual o lógico del sistema, y el modelo de diseño contiene el código, por lo cual el modelo de diseño deberá ser una representación de la manera como el código fuente es estructurado, manejado y escrito.

Durante la construcción del proyecto, se procede a la edificación del modelo de diseño. Para cada objeto en el modelo de análisis, se asigna un bloque en el modelo de diseño. El concepto de bloque describe la intención de cómo el código debe ser producido. Los bloques son el diseño de objetos y ellos se dibujan como rectángulos. Un bloque normalmente apunta para implementar un objeto de la etapa de análisis, aquí puede ser posible usar los diferentes tipos de bloques:

- Bloque de interfaz.
- Bloque de entidad.
- Bloque de control.

Cuando se tiene que crear la estructura del bloque, se dibuja un diagrama de interacción para mostrar como los bloques se comunican. Normalmente se dibuja un diagrama de interacción para cada caso de uso.

Diagrama de interacción

Para describir una secuencia de estímulos se usan los diagramas de interacción. Se puede describir como varios bloques se comunican mediante envió de estímulos de uno a otro. Como una base para esa interacción se usa otra vez el modelo de caso de uso.

Describe en detalle para cada caso de uso, una secuencia de estimulo la cual es enviada entre los bloques.

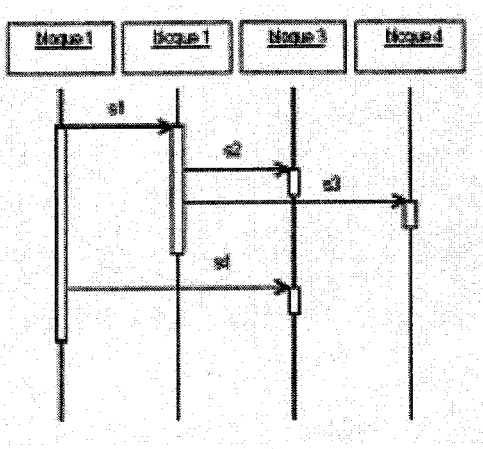


Fig7. Diagrama de secuencia

Antes de iniciar la implementación se puede usar una gráfica de transición de estado, su propósito es proporcionar una descripción simplificada, que mejore la comprensión del bloque, sin tener que bajar a nivel de código fuente, proporcionando una descripción que es menos dependiente del lenguaje de programación seleccionado. En este tipo de gráficas se describe cual estímulo puede ser recibido y que va a suceder cuando el estímulo es recibido. La grafica utiliza los símbolos mostrados.

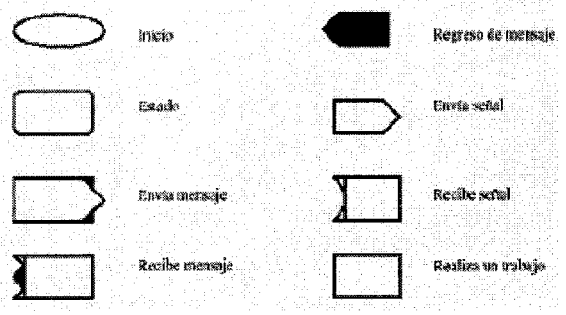


Fig8. Símbolos para gráfica de transición de estados

1.2.3.4 Modelo de implementación

La implementación del modelo consiste de la notación del código. La información de espacio es la opción del lenguaje de programación que se usa. La base para la implementación es el modelo de diseño. Aquí se especifica la interfase de cada bloque.

1.2.3.5 Modelo de prueba

El modelo de prueba es el último modelo a construir. Describe simplemente el estado de resultados de la prueba. El modelo de requerimientos de nuevo representa una herramienta potente de prueba, al probar cada caso de uso, se verifica que los objetos se comuniquen correctamente en dicho caso de uso. De manera simular se verifica la interfase de usuario, descrita en el modelo de requerimientos, con todo lo anterior, el modelo de requerimientos es la base de verificado para el modelo de prueba [Web_2].

CAPÍTULO 2

Aplicaciones en la Web.

2.1. ¿Qué es una aplicación Web?

En ingeniería del software una aplicación Web es aquella que los usuarios usan accediendo a un servidor Web a través de Internet o de una intranet. Las aplicaciones Web son populares debido a la practicidad del navegador Web como cliente ligero. La habilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad. Aplicaciones como los webmails, wikis, weblogs, MMORPGs, tiendas en línea son ejemplos bien conocidos de aplicaciones Web.

2.1.1 Historia

Inicialmente la Web era simplemente una colección de páginas estáticas documentos, etc., que podían consultarse o descargarse. Cada aplicación tenía su propio programa cliente y su interfaz de usuario, estos tenían que ser instalados separadamente en cada estación de trabajo de los usuarios. Una mejora al servidor, como parte de la aplicación, requería típicamente una mejora de los clientes instalados en cada una de las estaciones de trabajo, añadiendo un costo de soporte técnico y disminuyendo la eficiencia del personal.

El siguiente paso en su evolución fue la inclusión de un método para confeccionar páginas dinámicas que permitiesen que lo mostrado fuese dinámico (generado o calculado a partir de los datos de la petición). Dicho método fue conocido como CGI (common gateway interface) y definía un mecanismo mediante el cual podíamos pasar información entre el servidor HTTP y programas externos. Los CGI siguen siendo muy utilizados, puesto que la mayoría de los servidores Web los soportan debido a su sencillez. Además, nos proporcionan total libertad a la hora de escoger el lenguaje de programación para desarrollarlos.

El esquema de funcionamiento de los CGI tenía un punto débil: cada vez que recibíamos una petición, el servidor Web lanzaba un proceso que ejecutaba el programa CGI. Como, por otro lado, la mayoría de CGI estaban escritos en algún lenguaje interpretado (Perl, Python, etc.) o en algún lenguaje que requería run-time environment (VisualBasic, Java, etc.), esto implicaba una gran carga para la máquina del servidor.

Además, si la Web tenía muchos accesos al CGI, esto suponía problemas graves.

Por ello se empiezan a desarrollar alternativas a los CGI para solucionar este grave problema de rendimiento. Las soluciones vienen principalmente por dos vías. Por un lado se diseñan sistemas de ejecución de módulos más integrados con el servidor, que evitan que éste tenga que instanciar y ejecutar multitud de programas. La otra vía consiste en dotar al servidor de un intérprete de algún lenguaje de programación (RXML, PHP,

VBScript, etc.) que nos permita incluir las páginas en el código de manera que el servidor sea quien lo ejecute, reduciendo así el tiempo de respuesta.
A partir de este momento, se vive una explosión del número de arquitecturas y lenguajes de programación que nos permiten desarrollar aplicaciones Web. Todas ellas siguen alguna de las dos vías ya mencionadas. De ellas, las más útiles y las que más se utilizan son aquellas que permiten mezclar los dos sistemas, es decir, un lenguaje de programación integrado que permita al servidor interpretar comandos que "incrustemos" en las páginas HTML y un sistema de ejecución de programas más enlazado con el servidor que no presente los problemas de rendimiento de los CGI.
La que quizás sea la más exitosa y potente de estas aproximaciones, la seguida por Sun Microsystems con su sistema Java, que está integrada por dos componentes; a saber, un lenguaje que permite incrustar código interpretable en las páginas HTML y que el servidor traduce a programas ejecutables, JSP (Java Server pages) y un mecanismo de programación estrechamente ligado al servidor, con un rendimiento muy superior a los CGI convencionales, llamado Java Servlet.
Otra de las tecnologías que más éxito ha obtenido y una de las que más se utiliza en Internet es el lenguaje de programación interpretado por el servidor PHP. Se trata de un lenguaje que permite incrustar HTML en los programas, con una sintaxis que proviene de C y Perl. Además, habida cuenta de su facilidad de aprendizaje, su sencillez y potencia, se está convirtiendo en una herramienta muy utilizada para algunos desarrollos.
Otros métodos de programación de aplicaciones Web también tienen su mercado. Así sucede con mod_perl para Apache, RXML para Roxen, etc., pero muchos de ellos están vinculados a un servidor Web concreto.

2.1.2 Desarrollo de aplicaciones Web.

El desarrollo de aplicaciones *Web* involucra decisiones no triviales de diseño e implementación que inevitablemente influyen en todo el proceso de desarrollo, afectando la división de tareas. Los problemas involucrados, como el diseño del modelo del dominio y la construcción de la interfaz de usuario, tienen requerimientos disjuntos que deben ser tratados por separado.
El alcance de la aplicación y el tipo de usuarios a los que estará dirigida son consideraciones tan importantes como las tecnologías elegidas para realizar la implementación. Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también pueden reducir su capacidad de extensión y reusabilidad. Es por ello que el uso de una metodología de diseño y de tecnologías que se adapten naturalmente a ésta, son de vital importancia para el desarrollo de aplicaciones complejas.

Existen en la actualidad tecnologías ampliamente usadas para el desarrollo de aplicaciones *Web*, pero muchas de ellas obligan al desarrollador a mezclar aspectos conceptuales y de presentación. Esto sucede principalmente con aquellas tecnologías no basadas en objetos. La elección de tecnologías complejas demora el proceso e incrementa los costos, pero en ocasiones permite adecuarse a metodologías de diseño más fácilmente. Tal es el caso de las tecnologías orientadas a objetos, las cuales tienden a demorar el desarrollo en etapas tempranas. El tiempo de desarrollo en la actualidad es crítico, tanto

por razones de marketing como por límites en el presupuesto y los recursos [1], pero la adopción de estas tecnologías hace que el mantenimiento se transforme en una actividad más simple, la división en capas sea tarea natural del desarrollo y el tiempo invertido en el diseño facilite el trabajo necesario para el resto de las actividades.
En la siguiente sección se presenta una introducción a las aplicaciones *Web* en general, haciendo hincapié en la importancia del diseño en capas y en la programación orientada a objetos como herramienta de desarrollo.
En la sección 3 se describe cómo realizar el diseño de las aplicaciones *Web*, qué capas involucra y en qué consisten cada una de ellas. Se introduce para ello a OOHDM (Método de Diseño Hipermedia Orientado a Objetos1) [2], una metodología de diseño de aplicaciones hipermedia, y en particular de aplicaciones *Web*.
En la sección 4 se describen las características sobresalientes de las tecnologías sugeridas para la implementación de una aplicación concreta construida con la metodología.
En la sección 5 se presenta una idea de implementación basada en un ejemplo simple, utilizando las tecnologías sugeridas, en función de los requerimientos de cada capa de diseño.

2.1.2.1 Arquitectura Web

Ante tal aluvión de posibilidades, conviene repasar algunos aspectos básicos de la arquitectura Web.
Para abrir una página Web en un navegador, normalmente se teclea el correspondiente URL o se pica en el hiperenlace oportuno. Una vez que se solicita esta petición mediante el protocolo HTTP y la recibe el servidor Web, éste localiza la página Web en su sistema de ficheros y la envía de vuelta al navegador que la solicitó, según se muestra en la Figura 1

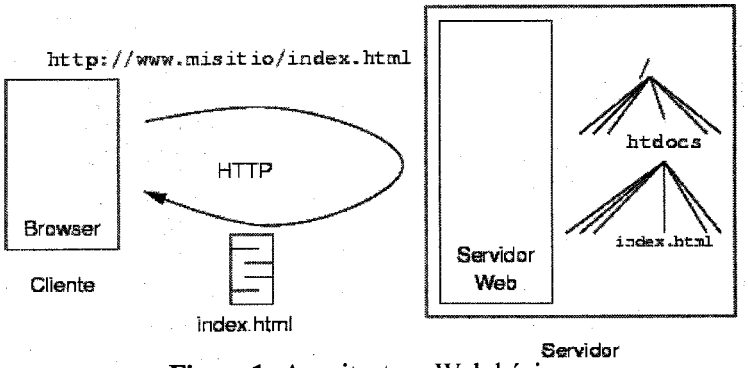


Figura1: Arquitectura Web básica.

2.1.2.2 El Navegador Web, *Browser*

El navegador puede considerarse como una interfaz de usuario universal. Dentro de sus funciones están la petición de las páginas Web, la representación adecuada de sus contenidos y la gestión de los posibles errores que se puedan producir.
Para todo esto, los fabricantes de navegadores les han dotado de posibilidades de ejecución de programas de tipo *script*, con modelos de objetos que permiten manipular los contenidos de los documentos. Estos lenguajes de programación son VBScript, JScript (ambas de Microsoft) y JavaScript (de Netscape), y proporcionan las soluciones llamadas del lado del cliente, *client side* y permiten realizar validaciones de datos

recogidos en las páginas antes de enviarlos al servidor y proporcionan un alto grado de interacción con el usuario dentro del documento.
Otras de las posibilidades de los navegadores es la gestión del llamado HTML dinámico (*Dinamic HTML*, DHTML). Éste está compuesto de HTML, hojas de estilo en cascada, (*Cascade Style Sheets*, CSS), modelo de objetos y *scripts* de programación que permiten formatear y posicionar correctamente los distintos elementos HTML de las páginas Web, permitiendo un mayor control sobre la visualización de las páginas.
En esta línea, los navegadores han ido un poco más allá y permiten la visualizaciones de documentos XML (*eXtensible Markup Language*) después de haber sido transformado adecuadamente a HTML por las hojas de estilo extensibles (*eXtensible Style Sheets*, XSL). De esta manera se puede elegir visualizar ciertos elementos y otros no, dependiendo de las circunstancias.
Además, los navegadores permiten la ejecución de aplicaciones dentro de los documentos mostrados. Las dos posibilidades más populares son la tecnología ActiveX y los *applets* Java. Los applets Java son pequeños programas que se descargan del servidor Web y se ejecutan en la JVM del navegador.

2.1.2.3 El Servidor Web

El servidor Web es un programa que corre sobre el servidor que escucha las peticiones HTTP que le llegan y las satisface. Dependiendo del tipo de la petición, el servidor Web buscará una página Web o bien ejecutará un programa en el servidor. De cualquier modo, siempre devolverá algún tipo de resultado HTML al cliente o navegador que realizó la petición.
El servidor Web va a ser fundamental en el desarrollo de las aplicaciones del lado del servidor, *server side applications*, que vayamos a construir, ya que se ejecutarán en él.

2.1.3 Ventajas e inconvenientes de las aplicaciones Web

En estos días que tan de moda están las aplicaciones Web, y más que lo van a estar, hay que considerar sus ventajas e inconvenientes.

Las ventajas:

- Desarrollo barato, sencillo y rápido.
- Acceso ubicuo, sin necesidad de distribución e, idealmente, con pocos requerimientos técnicos.
- Datos centralizados y fácil integración de datos de múltiples fuentes.
- Permiten el desarrollo de comunidades que dan valor a las aplicaciones (software social).

Todas estas obvias ventajas dejan claro el potencial de las aplicaciones Web.

Los inconvenientes:

- Acceso limitado, la necesidad de conexión permanente y rápida a Internet hacen que el acceso a estas aplicaciones no esté al alcance de todos.

- La interactividad no se produce en tiempo real, en las aplicaciones Web cada acción del usuario conlleva un tiempo de espera excesivo hasta que se obtiene la reacción del sistema.
- Elementos de interacción muy limitados. En comparación con el software de escritorio, las posibilidades de interacción con el usuario que ofrecen las aplicaciones Web (mediante formularios principalmente) son muy escasas.
- Diferencias de presentación entre plataformas y navegadores. La falta de estándares ampliamente soportados dificulta el desarrollo de las aplicaciones.

Por suerte, casi todas estas limitaciones están en la actualidad camino de ser superadas. Nuevas tecnologías y estándares hacen pensar que en poco tiempo muchas de estas dificultades serán simples recuerdos.

2.2 Aplicaciones *Web* y la importancia del desarrollo en capas

Las aplicaciones hipermedia han evolucionado en los últimos años y se han concentrado mayormente en la *Web*. Las antiguas aplicaciones distribuidas en cd's dieron lugar a aplicaciones dinámicas, de constante actualización e incluso personalizables, capaces de adaptarse a los tipos de usuarios y en casos avanzados, a cada usuario en particular. Estas características encuentran el medio ideal en la *Web*, ya que de otra forma sería costoso su mantenimiento y evolución.

La complejidad del desarrollo [3] ocurre a diferentes niveles: dominios de aplicación sofisticados (financieros, médicos, geográficos, etc.); la necesidad de proveer acceso de navegación simple a grandes cantidades de datos multimediales, y por último la aparición de nuevos dispositivos para los cuales se deben construir interfaces *Web* fáciles de usar. Esta complejidad en los desarrollos de software sólo puede ser alcanzada mediante la separación de los asuntos de modelización en forma clara y modular.

La metodología OOHDM [4], presentada en la próxima sección, ha sido utilizada para diseñar diferentes tipos de aplicaciones hipermedia como galerías interactivas, presentaciones multimedia y aplicaciones *Web*. El éxito de esta metodología es la clara identificación de los tres diferentes niveles de diseño en forma independiente de la implementación.

La justificación de tanto trabajo puede encontrarse en cualquier aplicación que requiera navegación: en términos de programación orientada a objetos, si los elementos por los que se navega son los del diseño conceptual se estaría mezclando la funcionalidad hipermedia con el comportamiento propio del objeto. Por otro lado, si los nodos de la red de navegación tienen la capacidad de definir su apariencia, se estaría limitando la extensión de la aplicación para ofrecer nuevas presentaciones del mismo elemento y eventualmente se estaría dificultando la personalización de la interfaz.

Es necesario, entonces, mantener separadas las distintas decisiones de diseño según su naturaleza (conceptual, navegacional, de interfaz) y aplicar las tecnologías adecuadas a cada capa en el proceso de implementación.

2.2.1 Aplicaciones Multinivel

Al hablar del desarrollo de aplicaciones Web resulta adecuado presentarlas dentro de las aplicaciones multinivel. Los sistemas típicos cliente/servidor pertenecen a la categoría de las aplicaciones de dos niveles. La aplicación reside en el cliente mientras que la base de

datos se encuentra en el servidor. En este tipo de aplicaciones el peso del cálculo recae en el cliente, mientras que el servidor hace la parte menos pesada, y eso que los clientes suelen ser máquinas menos potentes que los servidores. Además, está el problema de la actualización y el mantenimiento de las aplicaciones, ya que las modificaciones a la misma han de ser trasladada a todos los clientes.
Para solucionar estos problemas se ha desarrollado el concepto de arquitecturas de tres niveles: interfaz de presentación, lógica de la aplicación y los datos.
La capa intermedia es el código que el usuario invoca para recuperar los datos deseados. La capa de presentación recibe los datos y los formatea para mostrarlos adecuadamente. Esta división entre la capa de presentación y la de la lógica permite una gran flexibilidad a la hora de construir aplicaciones, ya que se pueden tener múltiples interfaces sin cambiar la lógica de la aplicación.
La tercera capa consiste en los datos que gestiona la aplicación. Estos datos pueden ser cualquier fuente de información como una base de datos o documentos XML.
Convertir un sistema de tres niveles a otro multinivel es fácil ya que consiste en extender la capa intermedia permitiendo que convivan múltiples aplicaciones en lugar de una sola (véase la Figura 2)

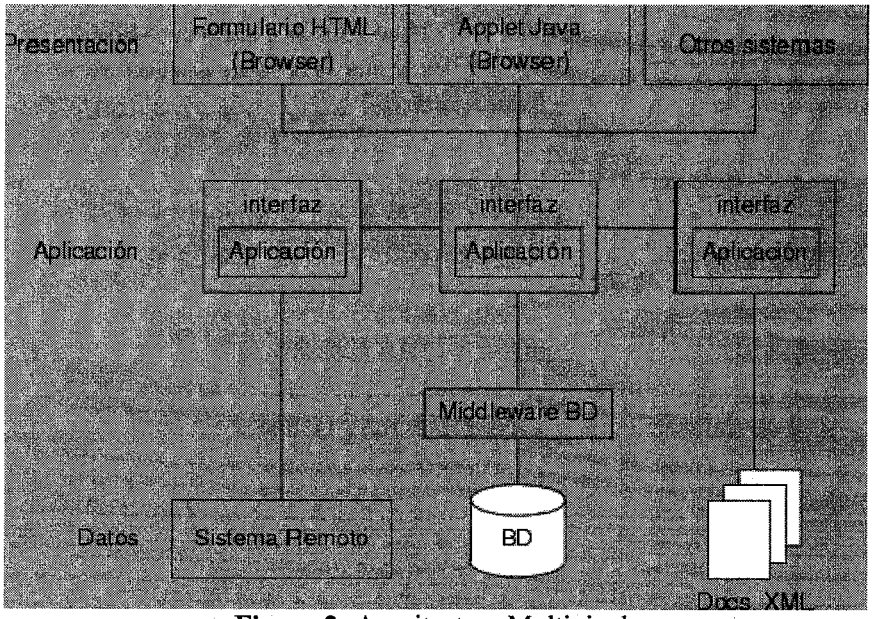


Figura 2: Arquitectura Multinivel.

La arquitectura de las aplicaciones Web suelen presentar un esquema de tres niveles (véase la Figura 3). El primer nivel consiste en la capa de presentación que incluye no sólo el navegador, sino también el servidor Web que es el responsable de dar a los datos un formato adecuado. El segundo nivel está referido habitualmente a algún tipo de programa o *script*. Finalmente, el tercer nivel proporciona al segundo los datos necesarios para su ejecución.

Una aplicación Web típica recogerá datos del usuario (primer nivel), los enviará al servidor, que ejecutará un programa (segundo y tercer nivel) y cuyo resultado será formateado y presentado al usuario en el navegador (primer nivel otra vez).

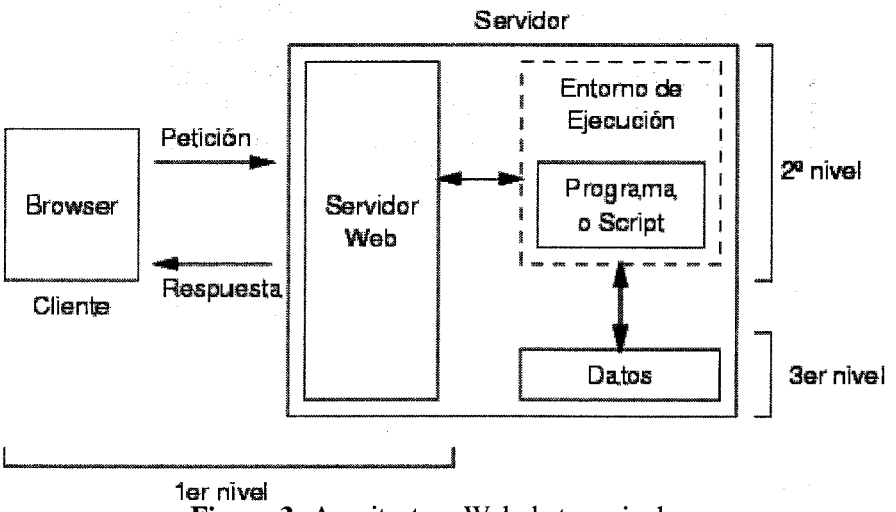


Figura 3: Arquitectura Web de tres niveles.

2.3 Introducción a OOHDM (Método de Diseño Hipermedia Orientado a Objetos)

Las metodologías tradicionales de ingeniería de software, o las metodologías para sistemas de desarrollo de información, no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones hipermedia. El tamaño, la complejidad y el número de aplicaciones crecen en forma acelerada en la actualidad, por lo cual una metodología de diseño sistemática es necesaria para disminuir la complejidad y admitir evolución y reusabilidad.

Producir aplicaciones en las cuales el usuario pueda aprovechar el potencial del paradigma de la navegación de sitios *Web*, mientras ejecuta transacciones sobre bases de información, es una tarea muy difícil de lograr.

En primer lugar, la navegación posee algunos problemas. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada.

Construir la interfaz de una aplicación *Web* es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían ser implementados, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación.

En hipermedia existen requerimientos que deben ser satisfechos en un entorno de desarrollo unificado. Por un lado, la navegación y el comportamiento funcional de la aplicación deberían ser integrados. Por otro lado, durante el proceso de diseño se debería poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio.

OOHDM propone el desarrollo de aplicaciones hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación.

2.3.1 Diseño Conceptual

Durante esta actividad se construye un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos. En las aplicaciones hipermedia convencionales, cuyos componentes de hipermedia no son modificados durante la ejecución, se podría usar un modelo de datos semántico estructural (como el modelo de entidades y relaciones). De este modo, en los casos en que la información base pueda cambiar dinámicamente o se intenten ejecutar cálculos complejos, se necesitará enriquecer el comportamiento del modelo de objetos.

En OOHDM, el esquema conceptual está construido por clases, relaciones y subsistemas. Las clases son descritas como en los modelos orientados a objetos tradicionales. Sin embargo, los atributos pueden ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real.
Se usa notación similar a UML (Lenguaje de Modelado Unificado) y tarjetas de clases y relaciones similares a las tarjetas CRC (Clase Responsabilidad Colaboración). El esquema de las clases consiste en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces.

2.3.2 Diseño Navegacional

La primera generación de aplicaciones Web fue pensada para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de hipermedia. En OOHDM, la navegación es considerada un paso crítico en el diseño aplicaciones. Un modelo navegacional es construido como una vista sobre un diseño conceptual, admitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional provee una vista subjetiva del diseño conceptual.
El diseño de navegación es expresado en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. En OOHDM existe un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. La semántica de los nodos y los enlaces son las tradicionales de las aplicaciones hipermedia, y las estructuras de acceso, tales como índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.
La principal estructura primitiva del espacio navegacional es la noción de contexto navegacional. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden ser definidos por comprensión o extensión, o por enumeración de sus miembros.
Los contextos navegacionales juegan un papel similar a las colecciones y fueron inspirados sobre el concepto de contextos anidados. Organizan el espacio navegacional en conjuntos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para ayudar al usuario a lograr la tarea deseada.

Los nodos son enriquecidos con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos (incluidos las anclas), así como métodos (comportamiento) cuando se navega en un particular contexto.

2.3.3 Diseño de Interfaz Abstracta

Una vez que las estructuras navegacionales son definidas, se deben especificar los aspectos de interfaz. Esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación, qué transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas.

Una clara separación entre diseño navegacional y diseño de interfaz abstracta permite construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario.

El aspecto de la interfaz de usuario de aplicaciones interactivas (en particular las aplicaciones *Web*) es un punto crítico en el desarrollo que las modernas metodologías tienden a descuidar. En OOHDM se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de hipermedia.

El modelo de interfaz ADVs (Vista de Datos Abstracta) [5] especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos, y la disposición de las propiedades de las ADVs en la pantalla real son hechas en la fase de implementación.

2.3.4 Implementación

En esta fase, el diseñador debe implementar el diseño. Hasta ahora, todos los modelos fueron construidos en forma independiente de la plataforma de implementación; en esta fase es tenido en cuenta el entorno particular en el cual se va a correr la aplicación.

Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería comportarse. A fin de implementar todo en un entorno *Web*, el diseñador debe decidir además qué información debe ser almacenada.

2.4 Comparación de OOHDM con otras metodologías

La comparación de métodos de desarrollo de sistemas de software es una tarea difícil. El foco de cada metodología puede ser diferente, algunas tratan de concentrarse en varios aspectos del proceso de desarrollo, otras tratan de detallar en profundidad algún aspecto en particular. En la Tabla 1 se presenta una comparación de distintas metodologías extraídas de [6], teniendo en cuenta los pasos que componen el proceso, la técnica de modelado, la representación gráfica, la notación elegida para los modelos y la herramienta *CASE* de soporte proporcionada para el desarrollo.

Las metodologías comparadas son: HDM (Método de Diseño Hipermedia) [7], RMM (Metodología de Administración de Relaciones) [8], EORM (Metodología de Relaciones de Objetos Mejorada) [9], OOHDM, SOHDM (Metodología de Diseño Hipermedia orientada a objetos y basada en escenarios) [10], WSDM (Método de Diseño de Sitios *Web*) [11], y WAE-Proceso *Conallen* (Extensión de Aplicación *Web* para UML) [12].

Tabla 1. Comparación de OOHDM con otras metodologías

Tabla 1. Comparación de OOHDM con otras metodologías

	Proceso	Técnica de modelado	Representación gráfica	Notación	Herramienta de soporte
HDM	1.Desarrollo a largo plazo 2.Desarrollo a corto plazo	E-R[12]	1.-2.Diagrama E-R	1.E-R	
RMM	1.Diseño E-R 2.Diseño *Slice*[13] 3.Diseño de navegación 4.Diseño de protocolo de conversión 5.Diseño de UI[14] 6.Diseño de comportamiento en tiempo de ejecución 7.Prueba y construcción	E-R	1.Diagrama E-R 2.Diagrama *Slice* 3.Diagrama RMDM[15]	1.E-R 2.3.Propio	*RMCase*
EORM	1.Clases del entorno de desarrollo 2.Composición del entorno de desarrollo 3.Entorno de desarrollo de UI	OO[16]	1.Diagrama de clases 2.Diseño GUI[17]	1.OMT[18]	*ONTOS Studio*
OOHDM	1.Diseño conceptual 2.Diseño navegacional 3.Diseño abstracto de la UI 4.Implementación	OO	1.Diagrama de clases 2.Diagrama navegacional, clase + contexto 3.Diagrama de configuración de ADV + Diagrama ADV	1.OMT/ UML[19] 2.Propio 3.ADVs	*OOHDM-Web*
SOHDM	1.Análisis del dominio 2.Modelo en OO 3.Diseño de la vista 4.Diseño navegacional 5.Diseño implementación 6.Construcción	Escenarios Vistas-OO	1.Diagramas de escenarios de actividad 2.Diagrama de estructura de clase 3.Vista OO 4.Esquema de enlace navegacional 5.Esquema de páginas	1.-5.Propio	
WSDM	1.Modelado del usuario 2.Diseño conceptual 2.1.Modelo objetos 2.2.Diseño navegacional 3.Diseño implementación 4.Implementación	E-R/ OO	1.Diagrama de E-R o clase 2.Capas de navegación	1.E-R/ OMT 2.Propio	
WAE-Proceso *Conallen*	1.Manejo de proyecto 2.Captura de requerimientos 3.Análisis 4.Diseño 5.Implementación 6.Prueba 7.Desarrollo 8.Configuración y manejo de cambios	OO	2.-5.Diagramas UML	UML	*Rational Rose*

En la Tabla 2 se presenta un segundo estudio comparativo de la misma fuente [6], que relaciona los conceptos de diseño de los tres niveles típicos de diseño *Web*: conceptual, estructural y visible. La mayoría de estos métodos realizan una clara separación entre el análisis del dominio, la especificación de la estructura navegacional y el diseño de la interfaz de usuario.

Tabla 2. Comparación de conceptos de diseño de las metodologías de desarrollo *Web*

	HDM	RMM	EORM	OOHDM	SOHDM	WSDM	WAE
Nivel Conceptual	Entidad Colección Perspectiva Relaciones	entidad relación	clases relación-OO -generalizada -definida por el usuario	clases perspectiva relación-OO	escenarios: -evento -actividad flujo de actividad	ojeto perspectiva relación	*case* relación-OO
Nivel Estructural	Enlace: -estructural -aplicación -perspectiva componente nodo Estructuras de acceso: -enlace colección -enlace índice -visita guiada	enlace: -unidireccional -bidireccional *Slices* primitivas de acceso: -agrupar (menú) -índice -visita guiada -visita guiada indexada	enlace: -simple -navegacional -nodo a nodo -tramo a nodo -estructural -conjunto -lista	enlace clase navegacional contexto navegacional estructuras de acceso: -índice -visita guiada	enlace navegacional vista-OO: -base -asociación -colaboración ASN[20]: -agrupar -índice -visita guiada	enlace componente -navegación -información -externo camino navegacional	enlace enlace dirigido redirigir construir enviar página *web* -página del cliente -página del servidor
Nivel Visible	Ranura Marco	*Slices*		ADV en contexto	componente UI: -elección -texto de entrada de búsqueda -botón -imagen -barra de desplazamiento - ancla HTML[21] -otros		conjunto de marcos formulario objetivo elemento de selección elemento de entrada elemento de área de texto

2.5 Revisión de tecnologías para el desarrollo de aplicaciones Web

Cada capa de diseño OOHDM constituye un sector de la aplicación con objetivos específicos; las tecnologías aplicadas en cada uno de ellos deben ser capaces de satisfacer sus requerimientos y de acoplarse fácilmente a las tecnologías asociadas a las capas restantes.

2.5.1 Lenguajes de programación

Existen numerosos lenguajes de programación utilizados para el desarrollo de Aplicaciones Web, entre los que destacan:

- PHP
- ASP/ASP.NET
- JSP
- Perl
- Ruby
- Python

Aunque ciertamente ASP no es un lenguaje de programación, sino una arquitectura de desarrollo Web en la que se pueden usar por debajo distintos lenguajes (por ejemplo VB.NET o C# para ASP.NET, o VBScript/JScript para ASP).

2.5.2 Lenguaje Java

Java es un lenguaje de programación orientado a objetos desarrollado por la compañía Sun Microsystems [13]. Está construido a partir de lenguajes orientados a objetos anteriores, como C++, pero no pretende ser compatible con ellos sino ir mucho más lejos, añadiendo nuevas características como recolección de basura, programación multi-hilos y manejo de memoria a cargo del lenguaje.

Java fue diseñado para que la ejecución de código a través de la red fuera segura, para lo cual fue necesario deshacerse de herramientas de C tales como los punteros. También se han eliminado aspectos que demostraron ser mejores en la teoría que en la práctica, tales como sobrecarga de operadores y herencia múltiple.

La portabilidad fue otra de las claves para el desarrollo de *Java*, para lograr que las aplicaciones se escriban una sola vez sin la necesidad de modificarlas para que corran en diferentes plataformas. Esta independencia se alcanza tanto a nivel de código fuente (similar a C++) como a nivel de código binario. La solución adoptada fue compilar el código fuente para generar un código intermedio (*bytecodes*) igual para cualquier plataforma.

La JVM (Máquina Virtual de *Java*), donde reside el intérprete *Java*, sólo tiene que interpretarlos.

2.5.2.1 Java DataBase Connectivity

JDBC (Conectividad de Base de Datos) es una interfaz que provee comunicación con bases de datos. Consiste de un conjunto de clases e interfaces escritas en *Java* [14], que proveen una API (Interfaz de Programación de Aplicación) estándar para desarrolladores

de herramientas de base de datos, permitiendo independizar la aplicación de la base de datos que utiliza.
La API JDBC es la interfaz natural a las abstracciones y conceptos básicos de SQL (Lenguaje de Consultas Simple): permite crear conexiones, ejecutar sentencias SQL y manipular los resultados obtenidos.
Conceptualmente es similar a ODBC (Conectividad de Base de Datos Abierta), pero ésta no es apropiada para usar directamente desde *Java* porque usa una interfaz en C y una traducción literal de C a *Java* no es deseable.
JDBC soporta dos modelos de acceso a base de datos: modelo de dos capas y modelo de tres capas [14]. En el primer caso, la aplicación *Java* se comunica directamente con la base de datos mediante un controlador JDBC específico para cada DBMS (Sistema de Administración de Base de Datos) que se desee manipular.

En el segundo caso, los comandos son enviados a un capa intermedia de servicios, encargado de reenviar la sentencias SQL a la base de datos.
Existe un controlador, llamado puente JDBC-ODBC, que implementa las operaciones de JDBC traduciéndolas en operaciones ODBC, con lo cual se provee acceso a cualquier base de datos cuyo controlador ODBC se encuentre disponible.

2.5.2.2 Servlets

Un *servlet* es una clase *Java* [15], embebida dentro del *Web server,* y utilizada para extender la capacidad de servidor. La API de *servlets* provee clases e interfaces para responder a cualquier tipo de requerimientos; e particular, para las aplicaciones que corren en servidores *Web*, la API define clases de *servlet* específicas par requerimientos HTTP.
No necesitan ser ejecutados como nuevos procesos dado que corren directamente en el *Web server.* Vive entre sesiones y se puede decir que son persistentes: no es necesario crear un *servlet* por cada requerimiento realizado desde el cliente, sino que corren dentro de éste múltiples hilos.
Los *servlets* [16] son programas *Java* que proveen la funcionalidad de generar dinámicamente contenidos *Web*. Pueden ser ejecutados a través de una línea de comando. A diferencia de los *applets*, no poseen restricciones en cuanto a seguridad. Tienen las propiedades de cualquier aplicación *Java* y pueden acceder a los archivos del servidor para escribir y leer, cargar clases, cambiar propiedades del sistema, etc. Del mismo modo que las aplicaciones de programas *Java*, los *servlets* están restringidos por los permisos del sistema.
Son cargados la primera vez que son usados, y permanecen en memoria para satisfacer futuros requerimientos. Tiene un método *init*, donde el programador puede inicializar el estado del *servlet*, y un método *destroy* para administrar los recursos que son mantenidos por el *servlet.*

2.5.2.3 Java Server Pages

JSP (Páginas de Servidor *Java*) provee a los desarrolladores de *Web* de un entorno de desarrollo para crear contenidos dinámicos en el servidor usando plantillas HTML y XML (Lenguaje de Marcado Extensible), en código *Java*, encapsulando la lógica que genera el contenido de las páginas [17].

Cuando se ejecuta una página JSP es traducida a una clase de *Java*, la cual es compilada para obtener un *servlet*. Esta fase de traducción y compilación ocurre solamente cuando el archivo JSP es llamado la primera vez, o después de que ocurran cambios.
JSP y XML tienen una interesante relación, descrito en [18]. Así como pueden generarse páginas HTML dinámicas a partir de una fuente en JSP, pueden generarse dinámicamente en forma análoga documentos XML. Más adelante, en la sección de implementación de este artículo, podrá observarse un ejemplo concreto de generación de contenido XML a partir de páginas JSP.

2.5.2.4 eXtensible Markup Language

La familia XML es un conjunto de especificaciones que conforman el estándar que define las características de un mecanismo independiente de plataformas desarrollado para compartir datos. Se puede considerar a XML como un formato de transferencia de datos multi-plataforma. Es un subconjunto de SGML (Lenguaje de Marcado Generalizado Standard) y uno de sus objetivos es permitir que SGML genérico pueda ser servido, recibido y procesado en la *Web* de la misma manera que actualmente es posible con HTML.
XML ha sido diseñado de tal manera que sea fácil de implementar. No ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas.
XML hace uso de etiquetas (únicamente para delimitar datos) y atributos, y deja la interpretación de los datos a la aplicación que los utiliza. Por esta razón se van formando lenguajes a partir del XML, y desde este punto de vista XML es un metalenguaje.
El conjunto de reglas o convenciones que impone la especificación XML permite diseñar formatos de texto para los datos estructurados, haciendo que se almacenen de manera no ambigua, independiente de la plataforma y que en el momento de la recuperación se pueda verificar si la estructura es la correcta.
Para comprobar que los documentos estén bien formados se utiliza un DTD (Definición de Tipo de Documento). Se trata de una definición de los elementos que pueden incluirse en el documento XML, la relación entre ellos, sus atributos, posibles valores, etc. Es una definición de la gramática del documento, es decir, cuando se procesa cualquier información formateada mediante XML, el primer paso es comprobar si está bien formada, y luego, si incluye o referencia a un DTD, comprobar que sigue sus reglas gramaticales.

2.5.2.5 eXtensible Stylesheet Language

XSL (Lenguaje de Hojas de Estilo Extensible) [20] es una especificación desarrollada dentro del W3C (*World Wide Web Consortium*) para aplicar formato a los documentos XML de forma estandarizada.
Aunque se ha establecido un modo para que puedan usarse hojas de estilo CSS (Hojas de Estilo en Cascada) dentro de documentos XML, es lógico pensar que para aprovechar las características del nuevo lenguaje hace falta tener un estándar paralelo y similar asociado a él.
Según el W3C, XSL es "un lenguaje para transformar documentos XML", así como un vocabulario XML para especificar semántica de formateo de documentos.

Además del aspecto que ya incluía CSS referente a la presentación y estilo de los elementos del documento, añade una pequeña sintaxis de lenguaje de comandos para poder procesar los documentos XML de forma más cómoda. La XSL permite añadir lógica de procesamiento a la hoja de estilo.
La idea es asociar al documento XML con una hoja de estilo y a partir de esto visualizar el documento XML en cualquier plataforma: *PalmPC*, *PC*, *Internet Explorer*, *Netscape*, etc. y con el aspecto (colores, fuentes, etc) que se quiera utilizar.
En esencia, XSL son dos lenguajes: uno de transformación y otro de formateo. El lenguaje de transformación permite transformar un documento XML en otro con diferente formato, como HTML o texto plano, o bien en otro documento XML. El lenguaje de formateo no es más que un vocabulario XML para especificar objetos de formateo (FO).
Al igual que con HTML, se pueden especificar las hojas de estilo, CSS o XSL, dentro del propio documento XML o haciendo referencia a ellas en forma externa. Esto es muy útil para mover datos de una representación XML a otra representación, basada en correo electrónico, intercambio de datos electrónicos, intercambio de meta datos, y alguna aplicación que necesite convertir datos entre diferentes representaciones de XML a otro tipo de representación.
La gran ventaja de utilizar XML y XSL es que los datos y la presentación de estos quedan en dos archivos diferentes.
Existen tres opciones para transformar un documento XML a otro formato, como se describen en [21], tal como HTML, utilizando hojas de estilo XSL:

- En el cliente: los documentos XML y las hojas de estilo son enviados al cliente (*Web* Browser), el cual se encarga de transformar los documentos basándose en la especificación de las hojas de estilo, y luego de la transformación se presentan al usuario en el explorador.

- En el servidor: el servidor es el encargado de aplicar el XSL al documento XML para transformarlo en algún otro formato (generalmente HTML) y envía el documento transformado al cliente.

- Ni en el servidor, ni en el cliente: una tercera opción consiste en transformar el documento original XML en algún otro formato (usualmente HTML) antes de que el documento sea ubicado en el servidor.

2.5.3 Interfaz

Las interfaces Web tienen ciertas limitantes en la funcionalidad del cliente. Métodos comunes en las aplicaciones de escritorio como dibujar en la pantalla o arrastrar-y-soltar no están soportadas por las tecnologías Web estándar. Los desarrolladores Web comúnmente utilizan lenguajes interpretados del lado del cliente para añadir más funcionalidad, especialmente para crear una experiencia interactiva que no requiera recargar la página cada vez (cosa que suele molestar a los usuarios). Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con tecnologías del lado del

servidor, como por ejemplo PHP. AJAX, es una técnica de desarrollo Web que usa una combinación de varias tecnologías.

2.5.4 Consideraciones Técnicas

Una ventaja significativa en la construcción de aplicaciones Web que soporten las características de los browsers estándar es que deberían de funcionar igual independientemente de la versión del sistema operativo instalado en el cliente. En vez de crear clientes para Windows, Mac OS X, GNU/Linux, y otros sistemas operativos, la aplicación es escrita una vez y es mostrada casi en todos lados. Sin embargo, aplicaciones inconsistentes de HTML, CSS, DOM y otras especificaciones de browsers pueden causar problemas en el desarrollo y soporte de aplicaciones Web. Adicionalmente, la habilidad de los usuarios a costumizar muchas de características de display (como tamaño y color de fuentes, tipos de fuentes, inhabilitar Javascript) puede interferir con la consistencia de la aplicación Web.

Otra (poco común) aproximación es utilizar Macromedia Flash o Java applets para producir parte o toda la interfaz de usuario. Como casi todos los browsers incluyen soporte para estas tecnologías (usualmente por medio de plug-ins), aplicaciones basadas en Flash o Java pueden ser implementadas con aproximadamente la misma facilidad. Como hacen caso omiso de las configuraciones de los browsers estas tecnologías permiten más control sobre la interfaz, aunque incompatibilidad entre implementaciones de Flash o Java puedan traer nuevas complicaciones. Por las similitudes con una arquitectura clienta-servidor, con un cliente un poco "especializado", hay disputas sobre si llamara a estos sistemas "aplicaciones Web"; un termino alternativo es "aplicación rica de Internet".

2.5.5 Estructura

Aunque muchas variaciones son posibles, una aplicación Web está comúnmente estructurada como una aplicación de tres-capas. En su forma más común, el Web browser es la primer capa, un motor usando alguna tecnología Web dinámica (ejemplo: CGI, PHP, Java Servlets o ASP) es la capa de en medio, y una base de datos como última capa. El Web browser manda peticiones a la capa media, que la entrega valiéndose de consultas y actualizaciones a la base de datos generando una interfaz de usuario.

2.5.6 Uso en negocios

Una estrategia que esta emergiendo para las empresas proveedoras de software, es proveer acceso vía Web al software. Para aplicaciones previamente distribuidas como de escritorio, esto puede requerir el desarrollo de una totalmente nueva aplicación o simplemente adaptar la aplicación para usar una interfaz Web. Estos programas permiten al usuario pagar una cuota mensual o anual para usar la aplicación, sin necesidad de instalarla en la computadora del usuario. Las compañías que siguen esta estrategia son llamadas Proveedores de Aplicaciones de Servicio (ASP por sus siglas en ingles), este modelo de negocios esta atrayendo la atención de la industria del software.

2.6 Trasladando OOHDM a una implementación

En esta sección se describirán las etapas de desarrollo de una aplicación *Web* simple, siguiendo la metodología OOHDM. En cada capa de diseño, la implementación correspondiente se basa en diferentes tecnologías, elegidas con el propósito de minimizar la dificultad del desarrollo y aprovechar al máximo las virtudes de la metodología.

2.6.1 Capa Conceptual

En OOHDM, el desarrollo se inicia diseñando la capa conceptual, siendo el principal objetivo de esta etapa capturar los conceptos involucrados en el dominio de la aplicación y describirlos en detalle, haciendo uso de diagramas que permitan expresar con claridad el comportamiento, la estructura y las relaciones entre dichos conceptos. La Programación Orientada a Objetos facilita el traslado del diseño conceptual a la implementación, proveyendo al programador con herramientas que permiten reducir la distancia entre el problema del mundo real y la programación de la solución en la computadora.

El modelo de objetos del ejemplo a discutir consta de las entidades básicas de un dominio específico: un comercio de venta de productos B2C (Negocio a Consumidor). En este dominio, entidades como *producto*, *categoría de productos*, *carro de compras*, *usuario*, *venta*, se interrelacionan para responder a la navegación del usuario por la aplicación y a sus actividades transaccionales. Todas las entidades mencionadas se construyen a partir de información persistente, propiedad mantenida por la empresa en forma directa a través de un DBA (Administrador de Bases de Datos) o simplemente aprovechando una funcionalidad incorporada de la aplicación que permita manipular la base de datos. Esta última alternativa es un ejemplo, entre otros fundamentos, de la importancia del diseño conceptual en el desarrollo de aplicaciones: el mismo esquema de objetos y relaciones que en principio pudieron ser diseñadas únicamente para mostrar información navegacional, puede ser abastecida con una interfaz que lleve a cabo la interacción con la base de datos, concentrando así esta actividad en un único sector de la aplicación, sin afectar al resto del modelo.

En cada diseño conceptual existe comportamiento que escapa a la simple navegación de información. Se trata del comportamiento inherente de cada clase, y aunque la aplicación particular no requiera que se implemente, es importante destacar que si la aplicación crece el diseño conceptual debe estar preparado para ser extendido, tal como cualquier diseño orientado a objetos. Más adelante podrá observarse cómo un modelo robusto y eficiente puede facilitar el trabajo de las capas restantes de la aplicación.

Retomando el diseño conceptual del ejemplo, el análisis anterior de las entidades del dominio permite afirmar que dichas clases comparten (al menos) el comportamiento correspondiente a la interfaz con la capa de persistencia: todas las entidades fuertes son capaces de construirse a partir de identificadores, coincidentes con las claves primarias de las tablas correspondientes de la base de datos [23]. Las clases del diseño conceptual que representen a estas entidades podrán obtener sus atributos al iniciarse y actualizar los cambios cuando sea necesario (realizando eventualmente algún tipo de almacenamiento temporal para mejorar la eficiencia). La consistencia de la información queda asegurada, entonces, por el comportamiento de los objetos del modelo.

El lenguaje elegido para desarrollar la implementación de esta capa de OOHDM es *Java*, presentado en la sección de tecnologías de este artículo. Durante esta etapa se utilizará también JDBC como paquete de vital importancia para el manejo de base de datos.

En la Figura 4 pueden observarse las primeras capas de implementación descritas anteriormente.

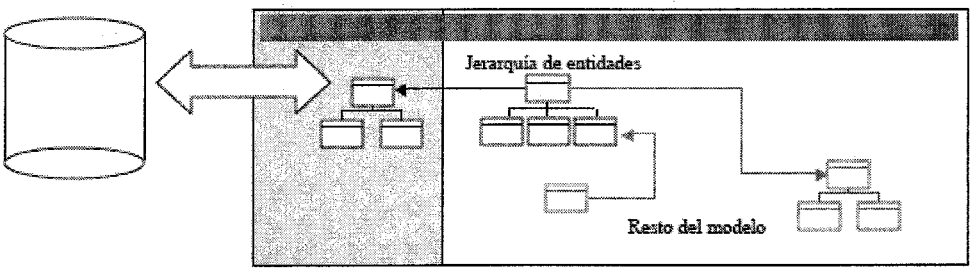


Figura 4: Paquete de interfaz con la base de datos, dentro del Diseño Conceptual

La clase abstracta que define el comportamiento básico de las entidades del modelo y concentra la lógica de interacción con la base de datos será denominada *EntidadAbstracta*. Para cumplir con los objetivos propuestos, esta clase debe ser capaz de crear una conexión con la base de datos, ejecutar consultas y retornar los resultados para ser procesados. Por una cuestión de eficiencia, la creación de conexiones a la base de datos podría ser delegada a un *singleton* [22], es decir, una clase capaz de controlar su instanciación para retornar siempre la misma instancia en reiteradas llamadas a su constructor. Una clase con estas características podría ser instanciada por *EntidadAbstracta* para luego solicitarle una conexión.
Recordemos que las subclases de *EntidadAbstracta* son entidades capaces de construirse a partir de una clave, realizando una consulta a la base de datos que involucre una o más tablas y que retorne una tupla unívocamente identificada por dicha clave. Los constructores de las subclases concretas de *EntidadAbstracta* pueden realizar estas consultas, utilizando en forma de *template method* [22] el comportamiento heredado. En la Figura 5 puede observarse con mayor claridad la secuencia de pasos involucrados en la instanciación de una entidad concreta; en este caso se instanciará la clase *Producto* para ejemplificar el proceso.
Sólo la información más importante y de menor volumen es cargada desde la base de datos en el momento de la instanciación. La información restante puede ser cargada bajo demanda, a partir de un eventual requerimiento de la aplicación. Para ilustrar esta idea con claridad puede considerarse cargar los siguientes atributos en la instanciación de un *Producto*: descripción, categoría, cantidad disponible y precio. En algún momento de la ejecución, la aplicación puede requerir los productos relacionados de un determinado producto (por ejemplo, el usuario podría solicitar una lista de productos relacionados con el producto televisor, tales como video grabadora, filmadora, mesa para televisor, etc.); dado el volumen de esta información y lo esporádico de su requerimiento, se sugiere entonces consultar a la base de datos para obtener esta información sólo cuando es requerida. Notar que la conexión a la base usada en cada consulta es siempre la solicitada

en el momento de la instanciación, evitando con esto creaciones y destrucciones reiteradas de conexiones a la base de datos.
Un modelo conceptual de estas características, obliga a considerar a todas las entidades del dominio como subclases de EntidadAbstracta (al menos todas aquellas entidades que se mantienen en la base de datos). Esta decisión de diseño puede refinarse aún más para evitar al máximo la incorporación de comportamiento de persistencia en las clases del dominio. Para alcanzar este objetivo puede utilizarse interfaces (mediante las cuales se pueden establecer contratos entre clases, permitiendo así ligar a las clases del dominio con las encargadas de interactuar con la base de datos) y/o un modelo paralelo de *wrappers* [22] que encapsulen a las clases del dominio (estos *wrappers* pueden encargarse directamente de la persistencia o interactuar con otras clases para factorizar código, dejando en cada *wrapper* el comportamiento específico requerido por la entidad encapsulada).
Es importante destacar que el diseño conceptual puede estar compuesto de otras clases que por su naturaleza no puedan considerarse subclases de *EntidadAbstracta*. Estos casos son simplemente colaboradores de entidades concretas, clases requeridas para realizar alguna actividad específica y de corta vida útil, o algunos casos de entidades débiles desde el punto de vista de diseño entidad-relación. Considérese como ejemplo la información relacionada con la navegación del usuario en una determinada sesión, irrelevante para otras sesiones e incluso para el resto de la aplicación. En este caso, se requiere una clase para contener la información mencionada y el comportamiento para manipularla, pero no requiere considerar la persistencia dentro de su funcionalidad.
El resto de las consideraciones a tener en cuenta para implementar el diseño conceptual depende del dominio específico. Hasta el momento se dispone de un modelo cuyas clases más importantes pueden instanciarse con facilidad para crear objetos de la capa navegacional y posteriormente decorarse para ser presentados en alguna interfaz. La generalidad del diseño conceptual es una característica determinante para llevar a cabo este tipo de construcciones: es necesario considerar a las entidades sin acotar su funcionalidad ni estructura por cuestiones de usuarios o contextos navegacionales.

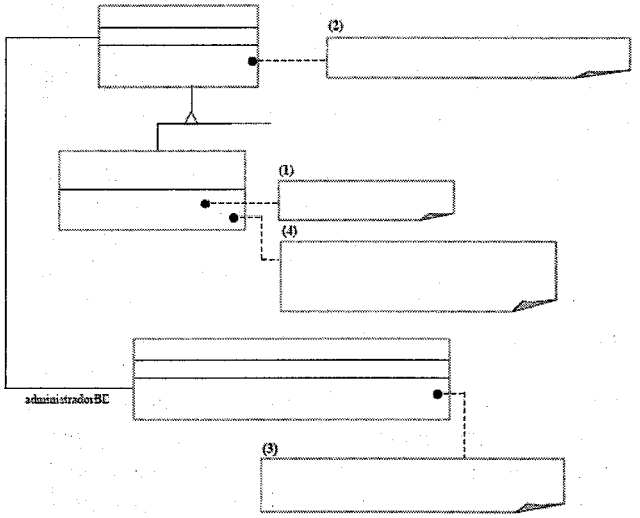


Figura 5: Instanciación de una subclase concreta de EntidadAbstracta

A continuación podrá observarse cómo un nodo de la capa navegacional es construido realizando los requerimientos convenientes a la entidad o entidades del diseño conceptual que observa, generando así una vista de dicha porción del diseño conceptual.

2.6.2 Capa Navegacional

La capa navegacional se compone de objetos construidos a partir de objetos conceptuales, y constituyen en general los elementos canónicos de las aplicaciones hipermedia tradicionales: *nodos*, *enlaces*, *anclas* y *estructuras de acceso*. Sin embargo, estas clases pueden extender el comportamiento característico para funcionar como *adaptadores* [22] de los objetos conceptuales y delegar así operaciones específicas del dominio.

Entonces, los objetos navegacionales pueden actuar como *observadores* [22], para construir vistas de objetos conceptuales, y como *adaptadores*, para extender la actividad navegacional de un nodo y poder aprovechar el comportamiento conceptual del objeto adaptado. Estas dos perspectivas pueden implementarse aprovechando las virtudes inherentes de diferentes tecnologías: JSP para observar y *Servlets* para adaptar.

Las páginas JSP serán responsables de construir los nodos de la capa navegacional. Esto se logra instanciando los objetos del diseño conceptual necesarios para mostrar la información del nodo y utilizando los datos solicitados a dichas instancias para generar árboles de elementos XML. Con este procedimiento se compone un nodo concentrando información relacionada en un documento XML generado dinámicamente con cada requerimiento, lo que permite además personalizar el contenido (datos sin presentación) a partir de infinitas configuraciones, tales como un perfil de usuario, la sobrecarga de requerimientos de la aplicación, la historia registrada de la navegación, políticas de protección de contenidos, seguridad, etc.

La Figura 6 ilustra la construcción del nodo a partir de un requerimiento HTTP proveniente de un cliente remoto.

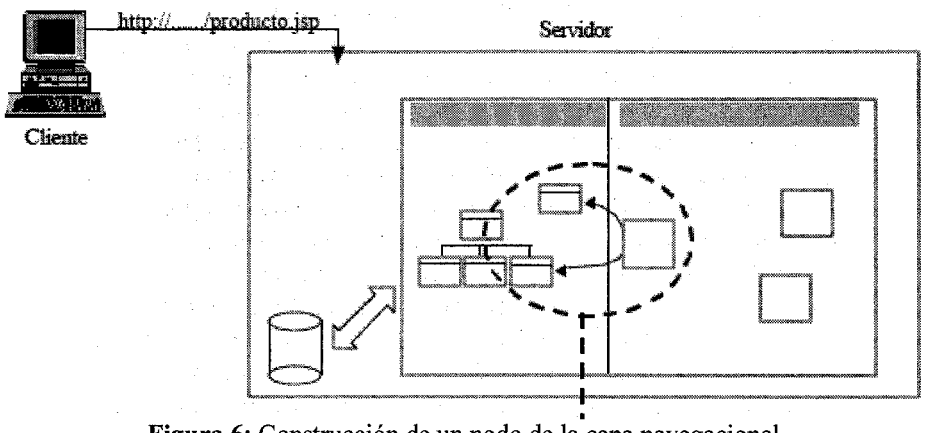


Figura 6: Construcción de un nodo de la capa navegacional

Como puede observarse, la cadena se inicia con un requerimiento del usuario que navega por la aplicación. Al acceder a un enlace, una página JSP es invocada para construir una página XML. En principio, el archivo XML generado puede ser útil para transferir información entre servidores cooperativos, o simplemente entre un cliente y un servidor

con un esquema tradicional. En cualquier caso, la portabilidad brindada por las características simples de XML hacen posible una transferencia transparente incluso entre plataformas completamente heterogéneas. Más adelante se podrá descubrir la verdadera razón por la cual se elige una salida con formato XML para los nodos de la capa navegacional.

Antes de abordar la implementación de la capa de presentación es necesario describir la segunda perspectiva de los nodos, vistos como *adaptadores* de objetos conceptuales. Un ejemplo donde se observa claramente este tipo de nodos es el *carro de compras*. Para mostrar el contenido del carro de compras del usuario es necesario instanciar el objeto conceptual *CarroDeCompras* (notar que el identificador del usuario que navega la aplicación y solicita acceder a su carro de compras es el único dato que se necesita para invocar al constructor de dicha entidad). De este modo, el nodo construido a partir de esta información no sólo concentra los datos de las compras sino que además ofrece un menú de operaciones para manipular el carro de compras, como borrar un elemento, cambiar la cantidad solicitada de un producto, vaciar el carro, finalizar la compra. Todas estas operaciones no son responsabilidad del nodo, sino que son realizadas por el objeto conceptual. Entonces, delegar responsabilidad es lo único que debe hacer el nodo navegacional en este caso: la actividad delegada continúa dentro de un *servlet* y finalmente es el *servlet* quien se encarga de redireccionar la navegación a una página JSP para eventualmente mostrar un resultado.
Los *servlets* son una herramienta muy útil para realizar actividades del lado del servidor y retornar información al cliente para informarle sobre el trabajo realizado o para solicitarle parámetros y retomar alguna operación. Continuando con el ejemplo del carro de compras, en caso que la operación elegida por el usuario sea vaciar el contenido (entre otras operaciones que puede seleccionar), una transacción debe ejecutarse a cargo del objeto *CarroDeCompras* correspondiente, a fin de modificar la base de datos satisfaciendo el deseo del usuario. Dado que este deseo es manifestado a través de un requerimiento (por ejemplo, realizando un *get* a través de una URL, o un *post* a través de un formulario HTML) la acción debe continuar en algún sector de la aplicación, sin necesidad de mostrar información durante el proceso, al menos en este caso puntual.
Entonces, puede construirse un paquete de *servlets* capaces de realizar operaciones específicas e incluso agruparse en jerarquías para aprovechar la herencia de comportamiento. A manera de ejemplo, considérese la jerarquía de *servlets* de la Figura 7, diseñados para servir acciones específicas referidas al *CarroDeCompras*.

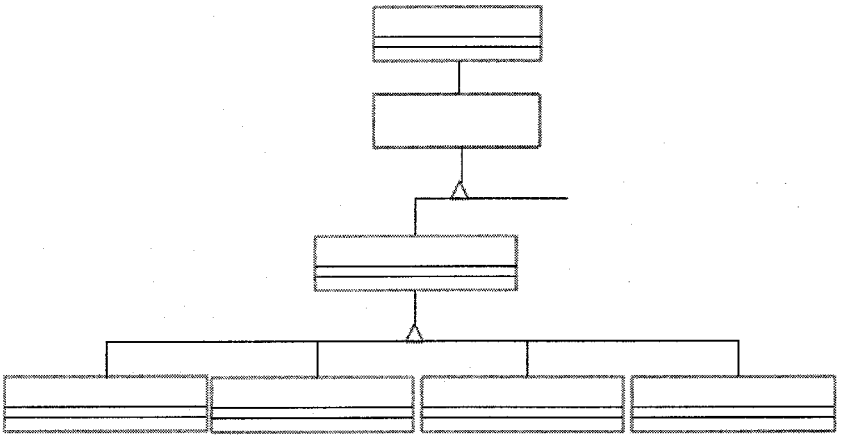


Figura 7: Jerarquía de *servlets* para administrar el carro de compras

El *servlet abstracto de la aplicación* es una clase que podría ser útil si se tiene comportamiento común a todos los *servlets* construidos para una aplicación específica. Luego, el *servlet* abstracto *CarroDeComprasServlet* define el comportamiento y la estructura de los *servlets* que encapsulan la lógica de una operación determinada sobre el carro de compras. No es necesario construir una subclase por operación concreta; varias operaciones pueden encapsularse en una única clase y parametrizarse adecuadamente si son semejantes.

En este escenario, cada subclase concreta de *servlet* tiene la responsabilidad de ejecutar una operación y retornar al cliente mostrando una página fija o calculada dinámicamente en función del resultado de la operación. Un ejemplo que puede considerarse para cualquier actividad es alternativa entre un resultado exitoso o uno erróneo, de la operación ejecutada. En el primer caso, la navegación podría continuar por una página JSP capaz de ilustrar la forma de proseguir la actividad, o de mostrar los resultados de la misma. En el segundo caso, la navegación se vería interrumpida por una página JSP que informara sobre el error producido y eventualmente solicitara la corrección de parámetros, u ofreciera reintentar la operación.

2.6.3 Capa de Interfaz Abstracta

Tanto un nodo actuando como observador, como un nodo actuando como adaptador, finalmente continúan por mostrar cierta información y para ello necesita definir la forma de presentación mediante la cual dicha información será visualizada en la interfaz. Para ello se incorporan las dos últimas tecnologías: XSL y un mecanismo de análisis sintáctico para obtener una página HTML en función de un par de documentos XML/XSL.
Las páginas XSL, ubicadas también del lado del servidor, definirán la apariencia de los nodos que se generaron en formato XML. Cada página XSL define la forma en que los elementos del XML asociados serán mostrados, haciendo uso de código HTML y eventualmente CSS [24], para dar el formato deseado a las páginas finales. Enlaces y estructuras de acceso también son presentados conforme con los estilos adoptados para este tipo de información navegacional.

El vínculo entre un XML y su apariencia puede establecerse en forma explícita, con una etiqueta especial situada en la primera línea del archivo XML, o bien podría ser calculado en forma dinámica para satisfacer demandas de personalización. En cualquier caso, retornar al cliente con un archivo XML que haga referencia a un archivo XSL ubicado en el servidor tiene al menos dos inconvenientes: genera un tráfico de ida y vuelta innecesario, ya que podría evitarse enviando directamente el código HTML al cliente, y puede requerir características especiales en el explorador del cliente que pueden privarlo de visualizar la página correctamente.
La transformación del lado del servidor es provista en la actualidad por paquetes como *Xalan-Java* [25] y *Saxon* [26]. Un esquema de la operación de transformación puede observarse en la Figura 8.

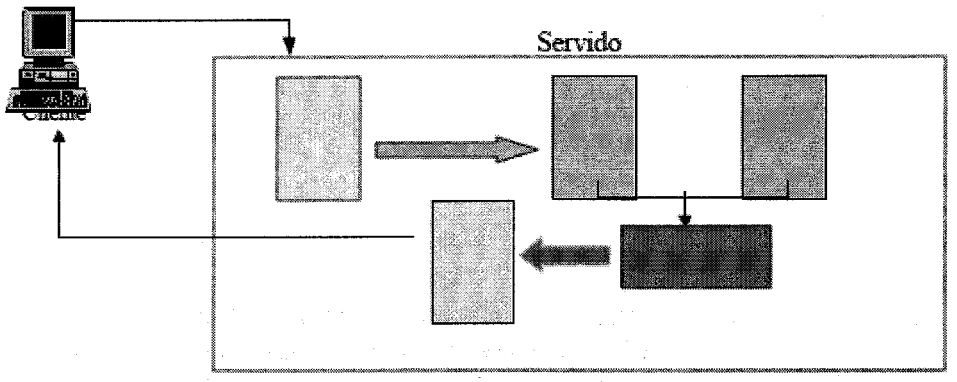


Figura 8: Generación de un documento HTML a partir de una fuente XML + XSL

El lenguaje XSLT (Transformaciones XSL40) [27] se utiliza para componer hojas de estilo XSL. Estos documentos contienen instrucciones que mediante el uso de *Xalan-Java*, por ejemplo, sirven para llevar a cabo las transformaciones y producir un documento de salida, una secuencia de caracteres o de *bytes*, un DOM (Modelo de Objetos de Documento), etc. En el caso particular mencionado anteriormente, se desea obtener un documento de salida con formato HTML a partir de un par de documentos XML / XSL.

2.7 Desarrollo de aplicaciones WEB con UML

Una de las características más relevantes de la notación UML es su capacidad para absorber nueva semántica sin romper su lógica interna.
La necesidad de implementar servicios Web a través de complejas arquitecturas con múltiples capas de componentes y una gran dispersión geográfica de nodos, ha supuesto todo un reto al abordar su modelado y especificación.

Jim Conallen ha desarrollado desde 1998 una extensión de la notación UML denominada WAE "Web Application Extensión" que permite rentabilizar toda la gramática interna de

UML para modelar aplicaciones con elementos específicos de la arquitectura de un entorno WEB.

No hay que confundir la implementación de un Website con el desarrollo de una aplicación Web. Como señala Conallen, un "Web site" es relativamente estático. En cambio, la aplicación Web es mucho más dinámica, dispone de una lógica de negocio que puede reaccionar y alterar su estado a partir de la interacción con un usuario.
Su contrapartida es la complejidad, ya que requiere implementar una arquitectura que se adapte a los cambios constantes, que facilite su ágil integración con otros sistemas y que resuelva picos variables de interacción con un buen rendimiento.

2.7.1 Modelado

Las aplicaciones Web son típicamente representadas por un conjunto de modelos: modelo de casos de usos, modelo de implementación, el modelo de despliegue, el modelo de seguridad, y algunos otros. Existe un modelo usado exclusivamente en sistemas Web: el mapa del sitio, el cual es una abstracción de las páginas Web y sus rutas de navegación a través del sistema.

Modelar el interior de el servidor Web, o los detalles de el browser no ayudará a los diseñadores y arquitectos de la aplicación Web. Modelar las páginas, sus enlaces y todo el contenido dinámico de crear las páginas del lado del servidor y el contenido dinámico en el lado del cliente es muy importante.
El paso siguiente es mapear estos artefactos y así modelar elementos. Por lo que las páginas pudieran mapearse a clases en el punto de vista lógico del modelo. Siguiendo esto, entonces los scripts pudieran también mapearse de forma natural dentro de la clase. Hay que considerar que un scrip se puede ejecutar en el cliente o en el servidor, y esto puede representar un problema. Aquí puede existir confusión entre cuales son los atributos y métodos, además de no saber si van del lado del cliente. Para esto UML nos permite extender su semántica, así nos podemos definir estereotipos, valores de etiquetas y restricciones que puedan ser aplicadas a los elementos del modelo.

Un estereotipo es una sutileza que nos permite definir un nuevo significado de la semántica para el elemento a modelar. Las restricciones son reglas que definen la buena forma del modelo. Pueden ser expresados como una forma libre de texto o con más formalidad con Object Constraint Language (OCL). Para una típica aplicación de negocio, solamente la lógica de negocio debería ser parte del modelo ADM. Los detalles tales como botones animados, botones de ayuda y otras mejoras, normalmente no pertenecen al modelo ADM. En pocas palabras la ADM necesita permanecer centrado sobre el problema del negocio y darle un espacio de solución.

2.7.2 Arquitectura de una aplicación Web

La arquitectura básica de una aplicación Web incluye browser, network, y un servidor Web. Algunas páginas incluyen scripts del lado del cliente que son interpretados por el browser con los cuales el usuario interactúa. A veces el usuario ingresa información en los campos de los formularios de la página y somete estos datos al procesamiento del servidor.

Actualmente los servidores Web son mucho más seguros, e incluyen características, como la administración de usuarios, el estado del servidor, proceso de la transacción, administración remota, etc. Hoy en día los servidores Web pueden ser divididos dentro de tres categorías: paginas con código (scripts, ejecutable del lado del servidor), paginas compiladas (carga y ejecuta un componente binario) y un híbrido entre los dos. Esta ultima categoría representa paginas con código, que una vez que son solicitadas son compiladas y usadas en lo sucesivo con esta misma compilación cuantas veces se requiera.

2.7.3 Modelando páginas Web

En las paginas Web se mapean uno a uno los componentes en UML. Un componente es una parte "física" y reemplazable del sistema. La vista de implementación del modelo describe los componentes del sistema y sus relaciones. En una aplicación Web, esta vista describe todas las paginas Web del sistema y sus relaciones con cada una de ellas.

Los componentes solamente representan el paquete físico de interfaces, por lo que no son adecuados para modelar la colaboración dentro de las páginas. Para empezar, podríamos decir que cada página Web es una clase UML en el modelo de diseño, y que las relaciones con otras páginas son los enlaces. Si consideramos que cualquier página Web puede representar un conjunto de funciones que existen solamente en el servidor y otra completamente diferente un conjunto que existe en el cliente se romperá nuestra abstracción.

Para solucionar este problema debemos considerar que el comportamiento de una página Web en el servidor es completamente diferente que una del lado del cliente. Mientras se ejecuta en el lado del servidor accesa a los recursos del lado del servidor. La página de salida tiene un diferente comportamiento y un conjunto de relaciones propias del lado del cliente.

Los estereotipos en UML nos permiten definir nuevas semánticas para modelar elementos. Las clases estereotipadas pueden ser dibujadas en un diagrama UML con su propio icono, o simplemente adornada con el nombre del estereotipo entre "<<>>". Cada página Web dinámica es construida por el servidor. Cada página del cliente es construida (en la mayoría) por un solo servidor de páginas, pero, es posible para un servidor de páginas construir múltiples páginas para el cliente. Un hyperlink en una aplicación Web representa una ruta de navegación por el sistema. Esta relación se expresa un estereotipo <<link>>. Esta asociación siempre se origina desde las paginas del cliente hacia el cliente o el servidor Web.

La ventaja principal de separar los aspectos de una pagina es que las paginas del cliente son modeladas considerando las relaciones y recursos del lado del cliente: java, DOM, applets, controles activex y plug-ins. Las paginas del lado del servidor son modeladas considerando las relaciones y los recursos del servidor, middle tier components, acceso a base de datos, sistema operativo, y mas. Las páginas del servidor típicamente asumen el papel de controladores, orquestando la actividad necesaria de los objetos del negocio para lograr la meta del negocio iniciada por la solicitud del browser. Una página del lado del

cliente es de la forma más simple un documento HTML que contiene tanta información de contenido y de presentación. En el modelo lógico estas relaciones pueden ser expresadas con una dependencia desde una página del cliente a un estereotipo <<style sheet>>. Las hojas de estilo son a menudo dejadas fuera del ADM.

2.7.4 Formularios

El principal mecanismos de entrada de datos en una pagina Web, son los formularios. Los formularios son definidos en el documento HTML con la etiqueta <form>. Los elementos de entrada de un formulario son todos atributos estereotipados de la clase <<form>>. Un estereotipo form puede tener relación con applets o controles activex que actúan como controles de entrada. Cada formulario también tiene una relación con el servidor de páginas que procesa todas las solicitudes del formulario. Esta relación es estereotipada <<submit>>.

2.7.5 Frames

Los frames permiten que múltiples páginas sean cargadas, estén activas y visibles del lado del cliente al mismo tiempo. Así para emplear frames o múltiples instancias de browsers en una aplicación es una decisión para el arquitecto. Para modelar el uso de un frame necesitamos definir dos clases estereotipadas mas: <<frameset>> y <<target>>, y una asociación estereotipada <<targeted link>>. Una clase frameset representa un contenedor de objetos y mapea directamente a una etiqueta HTML <frameset>. Una clase target es un nombre de un frame o una instancia del browser que es referenciada por otra pagina del cliente. Una asociación target link es un hyperlink a otra pagina, una ves que es colocada en un locación especifica (target). Es importante tener en mente que esta notación expresa una simple instancia de un cliente.

CAPÍTULO 3

Metodología UML.

3.1 ¿Qué es UML?

UML (Unified Modeling Language), es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.
Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa (principalmente Booch, OMT y OOSE). UML ha puesto fin a las llamadas "guerras de métodos" que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

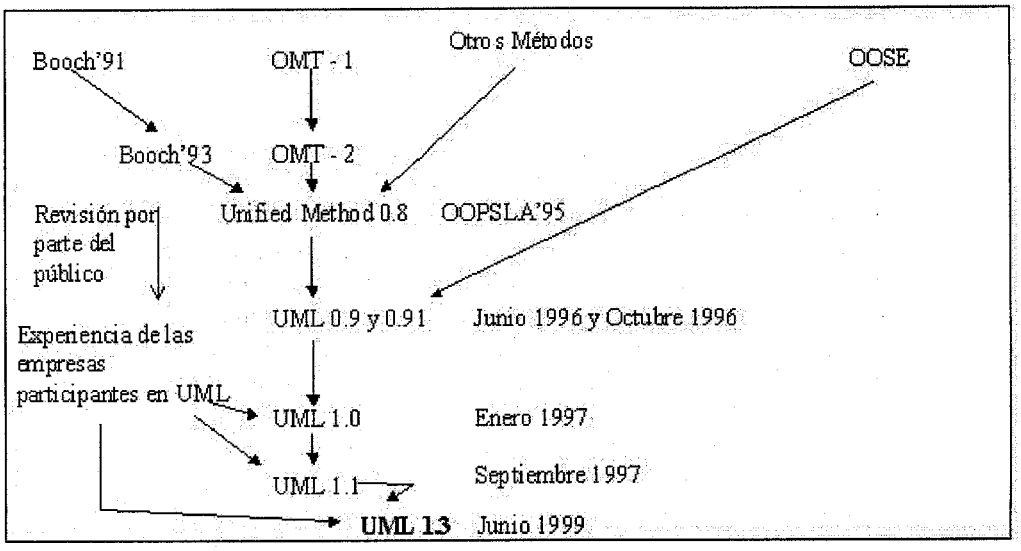


Fig. 1. Evolución de UML

Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. En la Figura 1 se puede ver cuál ha sido la evolución de UML hasta la creación de UML 1.3. Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación.

3.1.1 Visión general de UML

UML es un lenguaje para:

- Visualizar
- Especificar
- Construir
- Documentar

UML es un lenguaje de modelado es un lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema, por lo tanto es un lenguaje estándar para los planos del software.

3.1.2 UML es un lenguaje para visualizar

Un programador cuando esta modelando tiene que pensar en:

- **Primero**, la comunicación de esos modelos conceptuales a otros está sujeta a errores a menos que cualquier persona implicada hable del mismo lenguaje.
- **Segundo**, que el software no puede entender a menos que se construyan modelos que trasciendan el lenguaje de programación textual.
- **Tercero**, si el desarrollador que escribió el código no dejó documentación sobre los modelos, esa información se perderá y será parcialmente reproducible a partir de la implementación.
- UML es algo más que un simple montón de símbolos gráficos, en cada símbolo hay una semántica definida.

3.1.3 UML es un lenguaje para especificar

Especificar, significa construir modelos precisos, no complejos. Y UML cubre las especificaciones de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar un sistema de gran cantidad de software.

3.1.4 UML es un lenguaje para construir

En UML, sus modelos pueden conectarse en forma directa a gran variedad de lenguajes de programación, esta correspondencia permite ingeniería directa: la generación de código a partir de un modelo UML en un lenguaje de programación, también se puede reconstruir a partir de una implementación.

3.1.5 UML es un lenguaje para documentar

Una organización de software que trabaje bien produce:

- Requisitos
- Arquitectura

- Diseño
- Código fuente
- Planificación de proyectos
- Pruebas
- Prototipos
- Versiones

UML cubre la documentación de la arquitectura y proporciona requisitos y pruebas y las actividades de planificación de proyectos.

3.1.6 ¿Dónde puede utilizarse UML?

Sistemas de información de empresas
Bancos y servicios financieros
Telecomunicaciones
Transporte
Defensa / industrias aeroespacial
Comercio
Electrónica médica
Ámbito científico
Servicios distribuidos basados en la Web

3.1.7 Un modelo conceptual de UML

Para comprender UML, se necesita adquirir un modelo conceptual del lenguaje, y esto se requiere aprender tres elementos principales:

- Los bloques básicos de construcción de UML.
- Las reglas que dictan cómo se pueden combinar estos bloques básicos.
- Algunos mecanismos comunes que se aplican a través de UML.

3.1.8 Bloques de construcción de UML

UML tiene tres clases de bloques de construcción:

- Elementos
- Relaciones
- Diagramas

Los elementos son abstracciones que son cuidados de primera clase en un modelo, las relaciones ligan estos elementos entre sí; y los diagramas agrupan colecciones interesantes de elementos

3.1.9 Modelos

Un modelo representa a un sistema software desde una perspectiva específica. Al igual que la planta y el alzado de una figura en dibujo técnico nos muestran la misma figura vista desde distintos ángulos, cada modelo nos permite fijarnos en un aspecto distinto del sistema.

Los modelos de UML son los siguientes:
• Diagrama de Estructura Estática.
• Diagrama de Casos de Uso.

• Diagrama de Secuencia.
• Diagrama de Colaboración.
• Diagrama de Estados.

3.2. Elementos Comunes a Todos los Diagramas

3.2.1 Notas

Una nota sirve para añadir cualquier tipo de comentario a un diagrama o a un elemento de un diagrama. Es un modo de indicar información en un formato libre, cuando la notación del diagrama en cuestión no nos permite expresar dicha información de manera adecuada. Una nota se representa como un rectángulo con una esquina doblada con texto en su interior. Puede aparecer en un diagrama tanto sola, como unida a un elemento por medio de una línea discontinua. Puede contener restricciones, comentarios, el cuerpo de un procedimiento, etc.

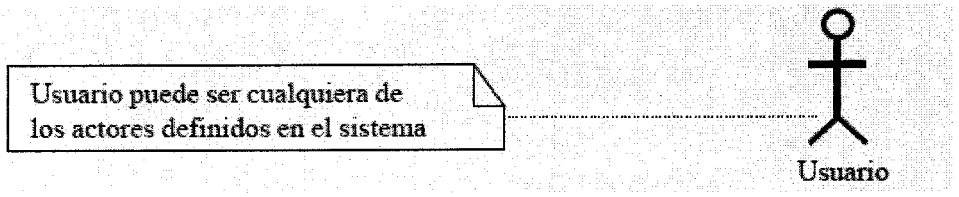


Fig.2 Nota

3.2.2 Dependencias

La relación de dependencia entre dos elementos de un diagrama significa que un cambio en el elemento destino puede implicar un cambio en el elemento origen (por tanto, si cambia el elemento destino habría que revisar el elemento origen). Una dependencia se representa por medio de una línea de trazo discontinuo entre los dos elementos con una flecha en su extremo. El elemento dependiente es el origen de la flecha y el elemento del que depende es el destino (junto a él está la flecha).

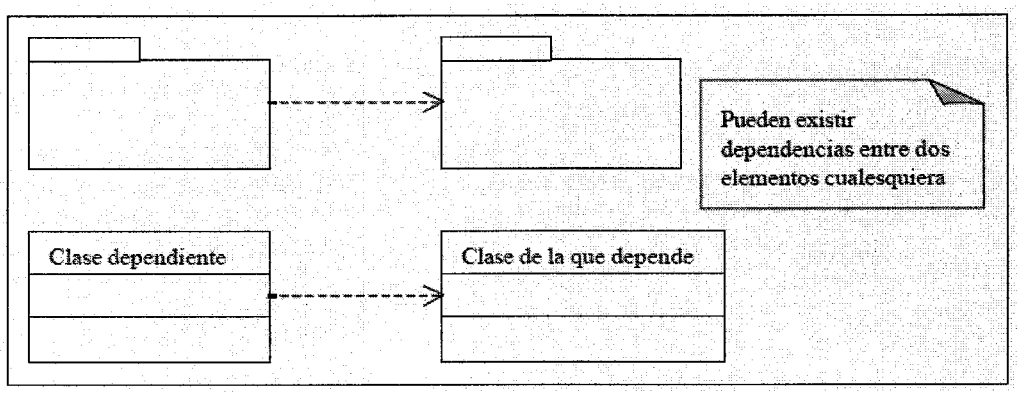


Fig. 3 Dependencia

3.3 Diagramas de Estructura Estática

Los Diagramas de Estructura Estática de UML se van a utilizar para representar tanto Modelos Conceptuales como Diagramas de Clases de Diseño. Ambos usos son distintos conceptualmente, mientras los primeros modelan elementos del dominio los segundos presentan los elementos de la solución software. Ambos tipos de diagramas comparten una parte de la notación para los elementos que los forman (clases y objetos) y las relaciones que existen entre los mismos (asociaciones). Sin embargo, hay otros elementos de notación que serán exclusivos de uno u otro tipo de diagrama.

3.3.1 Clases

Una clase se representa mediante una caja subdividida en tres partes: En la superior se muestra el nombre de la clase, en la media los atributos y en la inferior las operaciones. Una clase puede representarse de forma esquemática, con los atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase. En la Figura 4 se ve cómo una misma clase puede representarse a distinto nivel de detalle según interese, y según la fase en la que se esté.

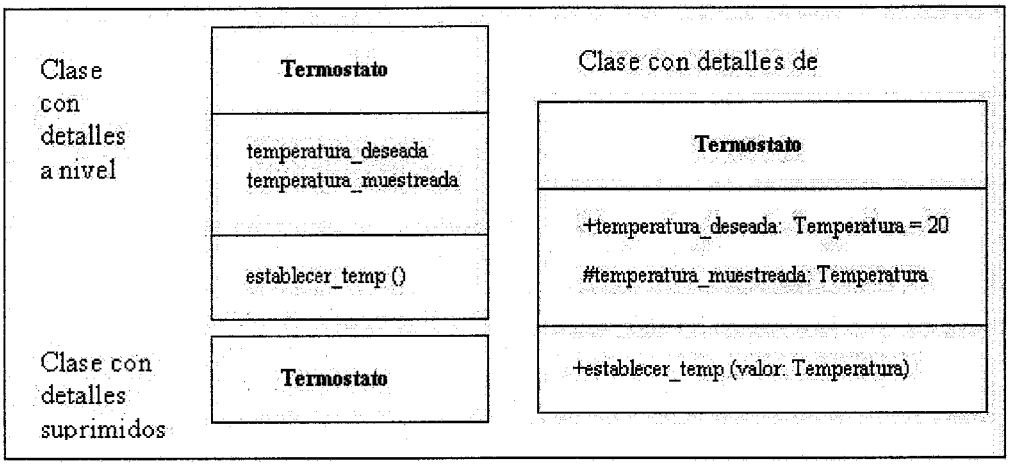


Fig. 4 Niveles de Detalle en las Clases

3.3.2 Objetos

Un objeto se representa de la misma forma que una clase. En el compartimiento superior aparecen el nombre del objeto junto con el nombre de la clase subrayados, según la siguiente sintaxis: nombre_del_objeto: nombre_de_la_clase Puede representarse un objeto sin un nombre específico, entonces sólo aparece el nombre de la clase.

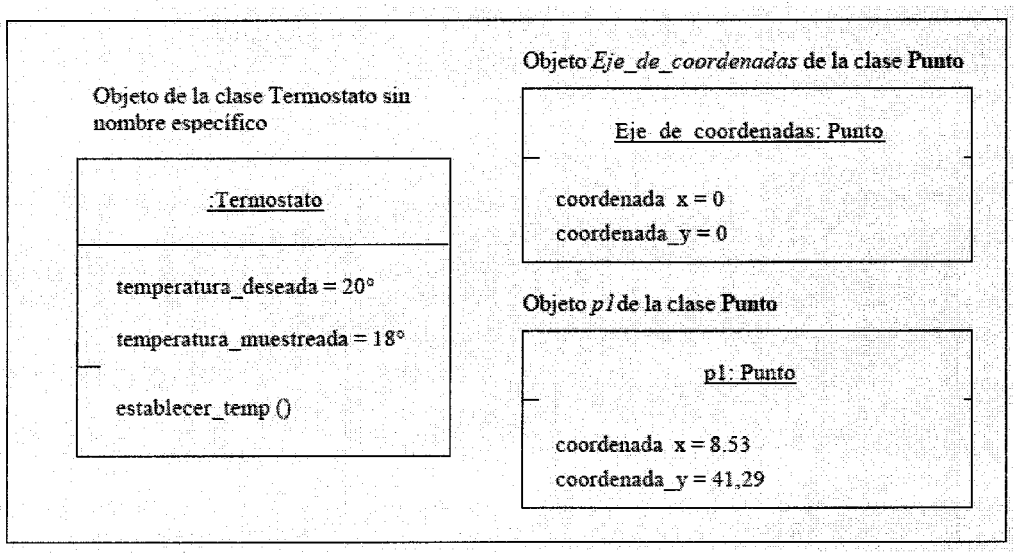


Fig. 5 Objetos

3.3.3 Asociaciones

Las asociaciones entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación. A continuación se verán los más importantes de entre dichos elementos gráficos.

3.3.3.1 Nombre de la Asociación y Dirección

El nombre de la asociación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir un pequeño triángulo negro sólido que indique la dirección en la cual leer el nombre de la asociación. En el ejemplo de la Figura 6 se puede leer la asociación como "Director manda sobre Empleado".

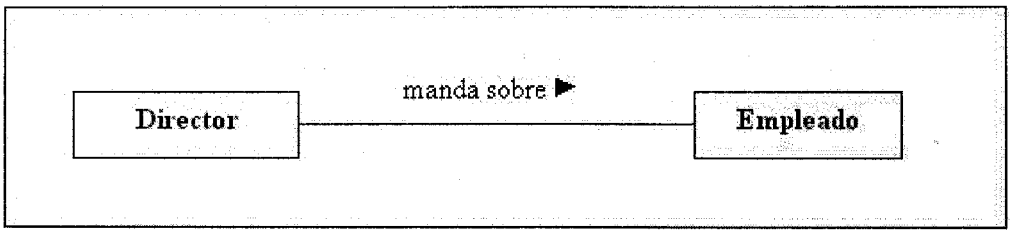


Fig. 6 Asociación

Los nombres de las asociaciones normalmente se incluyen en los modelos para aumentar la legibilidad. Sin embargo, en ocasiones pueden hacer demasiado abundante la información que se presenta, con el consiguiente riesgo de saturación. En ese caso se puede suprimir el nombre de las asociaciones consideradas como suficientemente

conocidas. En las asociaciones de tipo agregación y de herencia no se suele poner el nombre.

3.3.3.2 Multiplicidad

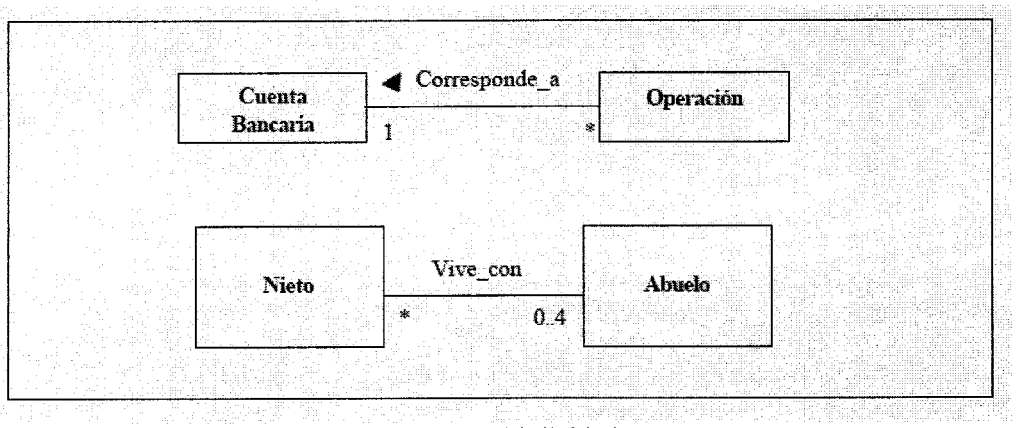


Fig. 7 Multiplicidad

La multiplicidad es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase. Puede expresarse de las siguientes formas:

- Con un número fijo: 1.
- Con un intervalo de valores: 2..5.
- Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, 2..* significa 2 o más.
- Con una combinación de elementos como los anteriores separados por comas: 1, 3..5, 7, 15..*.
- Con un asterisco: * . En este caso indica que puede tomar cualquier valor (cero o más).

3.3.3.3 Roles

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol.

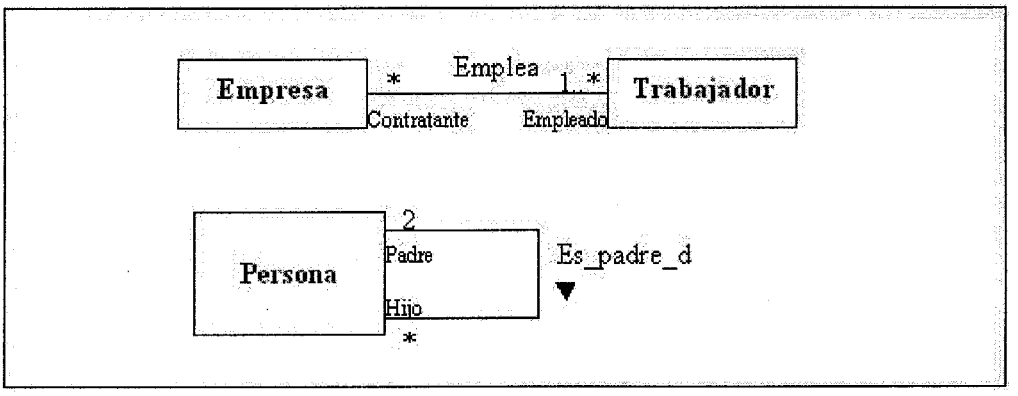


Fig. 8 Multiplicidad.

Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

3.3.3.4 Agregación

El símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el "todo".

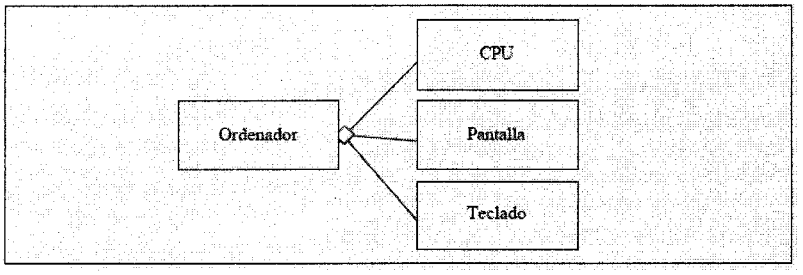


Fig. 9 Agregación.

3.3.3.5 Clases Asociación

Cuando una asociación tiene propiedades propias se representa como una clase unida a la línea de la asociación por medio de una línea a trazos. Tanto la línea como el rectángulo de clase representan el mismo elemento conceptual: la asociación. Por tanto ambos tienen el mismo nombre, el de la asociación. Cuando la clase asociación sólo tiene atributos el nombre suele ponerse sobre la línea (como ocurre en el ejemplo de la Figura 10). Por el contrario, cuando la clase asociación tiene alguna operación o asociación propia, entonces se pone el nombre en la clase asociación y se puede quitar de la línea.

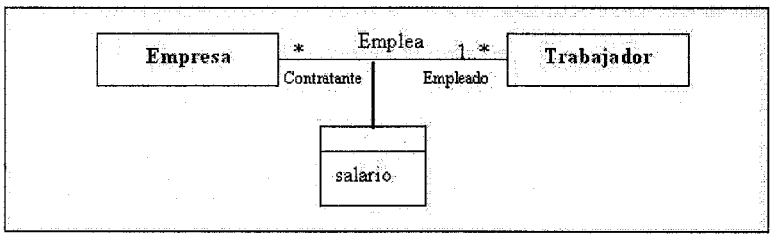


Fig. 10 Asociación

3.3.3.6 Asociaciones N-Arias

En el caso de una asociación en la que participan más de dos clases, las clases se unen con una línea a un diamante central. Si se muestra multiplicidad en un rol, representa el número potencial de tuplas de instancias en la asociación cuando el resto de los N-1 valores están fijos. En la Figura 11 se ha impuesto la restricción de que un jugador no puede jugar en dos equipos distintos a lo largo de una temporada, porque la multiplicidad de "Equipo" es 1 en la asociación ternaria.

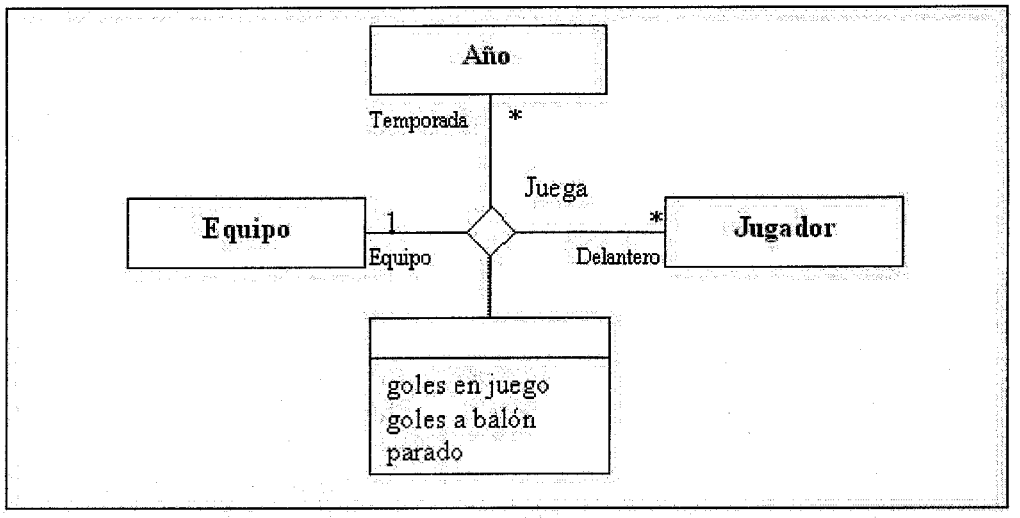


Fig. 11 Asociaciones N-arias

3.3.3.7 Navegabilidad

En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible "navegar" desde el objeto de la clase origen hasta el objeto de la clase destino. Se trata de un concepto de diseño, que indica que un objeto de la clase origen conoce al (los) objeto(s) de la clase destino, y por tanto puede llamar a alguna de sus operaciones.

3.3.4 Herencia

La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o clase "padre".

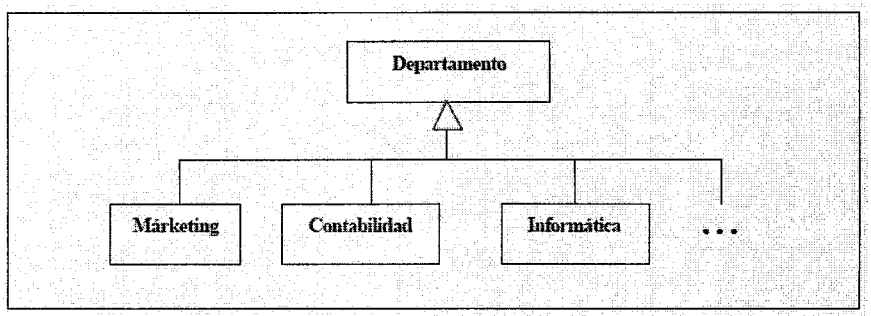


Fig. 12 Herencia

Si se tiene una relación de herencia con varias clases subordinadas, pero en un diagrama concreto no se quieren poner todas, esto se representa mediante puntos suspensivos. En el ejemplo de la Figura 12, sólo aparecen en el diagrama 3 tipos de departamentos, pero con los puntos suspensivos se indica que en el modelo completo (el formado por todos los diagramas) la clase "Departamento" tiene subclases adicionales, como podrían ser "Recursos Humanos" y "Producción".

3.3.5 Elementos Derivados

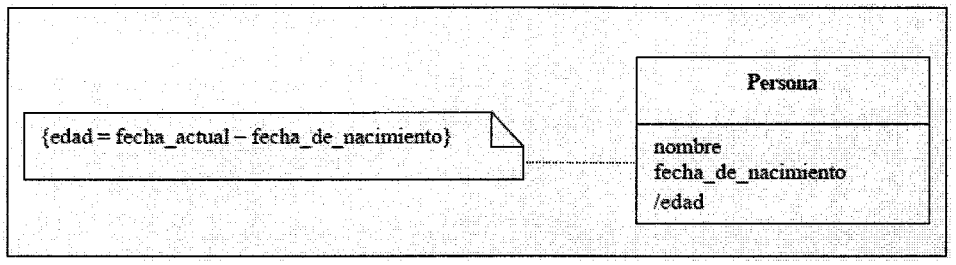


Fig. 13 Elemento Derivado.

Un elemento derivado es aquel cuyo valor se puede calcular a partir de otros elementos presentes en el modelo, pero que se incluye en el modelo por motivos de claridad o como decisión de diseño. Se representa con una barra "/" precediendo al nombre del elemento derivado.

3.4 Diagrama de Casos de Uso

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea. En la Figura 14 se muestra un ejemplo de Diagrama de Casos de Uso para un cajero automático.

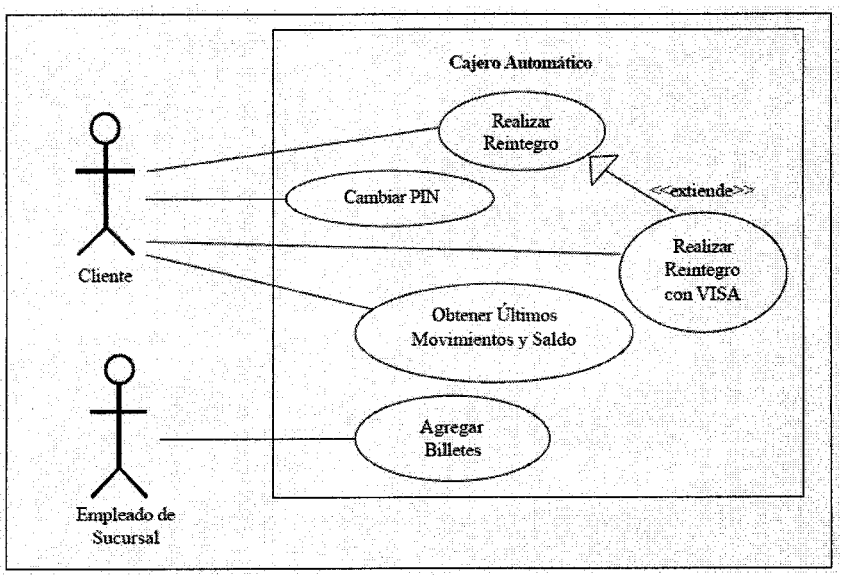


Fig. 14 Caso de uso para un cajeo automático.

3.4.1 Elementos

Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones entre casos de uso.

3.4.1.1 Actores

Un actor es algo con comportamiento, como una persona (identificada por un rol), un sistema informatizado u organización, y que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana dibujada con palotes. Esta representación sirve tanto para actores que son personas como para otro tipo de actores.

3.4.1.2 Casos de Uso

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

3.4.1.3 Relaciones entre Casos de Uso

Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo, hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos en algunos casos, es mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. Para el caso de que queramos utilizar estos casos de uso más pequeños, las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes tres tipos:

• Incluye (<>): Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso. En la Figura 15 se muestra cómo el caso de uso Realizar Reintegro puede incluir el comportamiento del caso de uso Autorización.

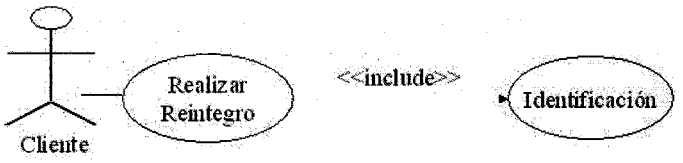


Fig. 15 - Ejemplo de Relación <>

• Extiende (<>): Cuando un caso de uso base tiene ciertos puntos (puntos de extensión) en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación incluye que se da siempre que se realiza la interacción descrita) En la Figura 16 se muestra como el caso de uso Comprar Producto permite explícitamente extensiones en el siguiente punto de extensión: info regalo. La interacción correspondiente a establecer los detalles sobre un producto que se envía como regalo están descritos en el caso de uso Detalles Regalo.

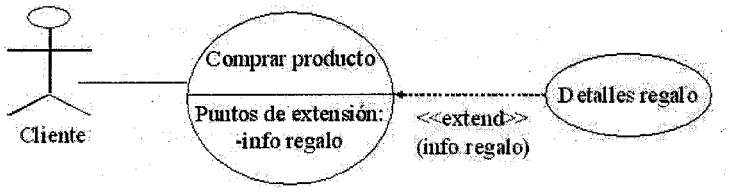


Fig. 16 - Ejemplo de Relación <>

Ambos tipos de relación se representan como una dependencia etiquetada con el estereotipo correspondiente (<> o <>), de tal forma que la flecha indique el sentido en el que debe leerse la etiqueta. Junto a la etiqueta <> se puede detallar el/los puntos de extensión del caso de uso base en los que se aplica la extensión.

• Generalización ( ): Cuando un caso de uso definido de forma abstracta se particulariza por medio de otro caso de uso más específico. Se representa por una línea continua entre los dos casos de uso, con el triángulo que simboliza generalización en UML (usado también para denotar la herencia entre clases) pegado al extremo del caso de uso más general. Al igual que en la herencia entre clases, el caso de uso hijo hereda las asociaciones y características del caso de uso padre. El caso de uso padre se trata de un caso de uso abstracto, que no está definido completamente. Este tipo de relación se utiliza mucho menos que las dos anteriores.

3.5 Diagramas de Interacción

En los diagramas de interacción se muestra un patrón de interacción entre objetos. Hay dos tipos de diagrama de interacción, ambos basados en la misma información, pero cada uno enfatizando un aspecto particular: Diagramas de Secuencia y Diagramas de Colaboración.

3.5.1 Diagrama de Secuencia

Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo. El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren.

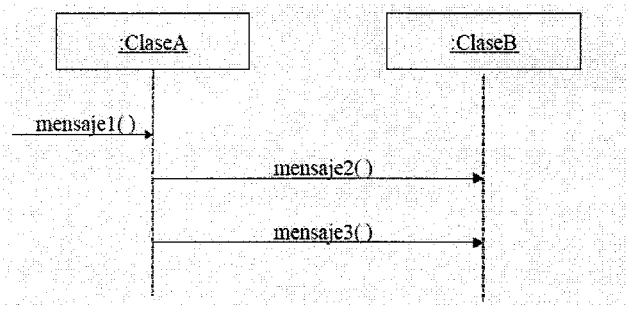


Fig. 17 Diagrama de Secuencia

3.5.2 Diagrama de Colaboración

Un Diagrama de Colaboración muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

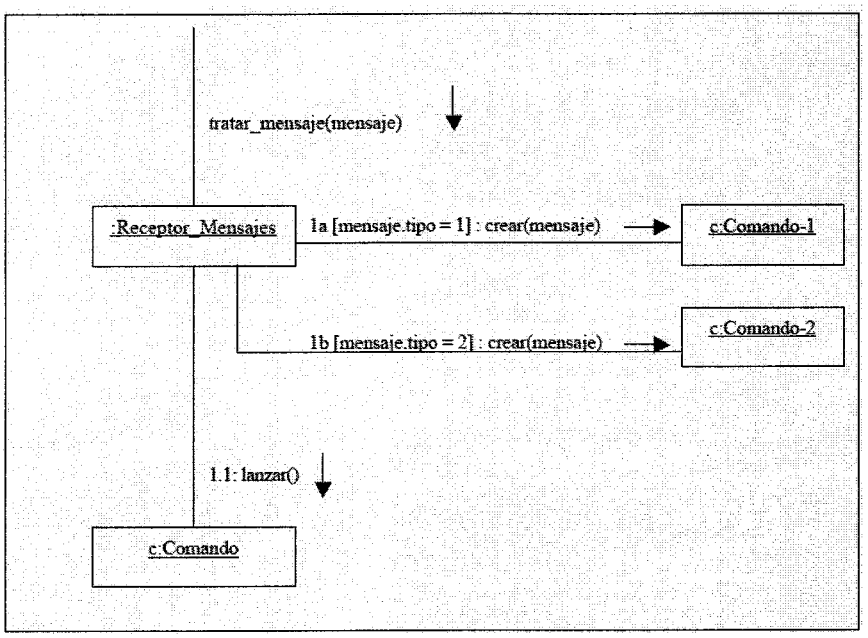


Fig. 18 Diagrama de Colaboración.

En cuanto a la representación, un Diagrama de Colaboración muestra a una serie de objetos con los enlaces entre los mismos, y con los mensajes que se intercambian dichos objetos. Los mensajes son flechas que van junto al enlace por el que "circulan", y con el nombre del mensaje y los parámetros (si los tiene) entre paréntesis. Cada mensaje lleva un número de secuencia que denota cuál es el mensaje que le precede, excepto el mensaje que inicia el diagrama, que no lleva número de secuencia. Se pueden indicar alternativas con condiciones entre corchetes (por ejemplo 3 [condición_de_test]: nombre_de_método()), tal y como aparece en el ejemplo de la Figura 18. También se puede mostrar el anidamiento de mensajes con números de secuencia como 2.1, que significa que el mensaje con número de secuencia 2 no acaba de ejecutarse hasta que no se han ejecutado todos los 2. x .

3.6 Diagramas de Estado

Un Diagrama de Estados muestra la secuencia de estados por los que pasa bien un caso de uso, bien un objeto a lo largo de su vida, o bien todo el sistema. En él se indican qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.
En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos.
Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimiento aparece el nombre del estado. El segundo compartimiento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas.
Una acción de entrada aparece en la forma entrada/acción_asociada donde acción_asociada es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta. Una acción de salida aparece en la forma salida/acción_asociada. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta.
Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma nombre_de_evento/acción_asociada.

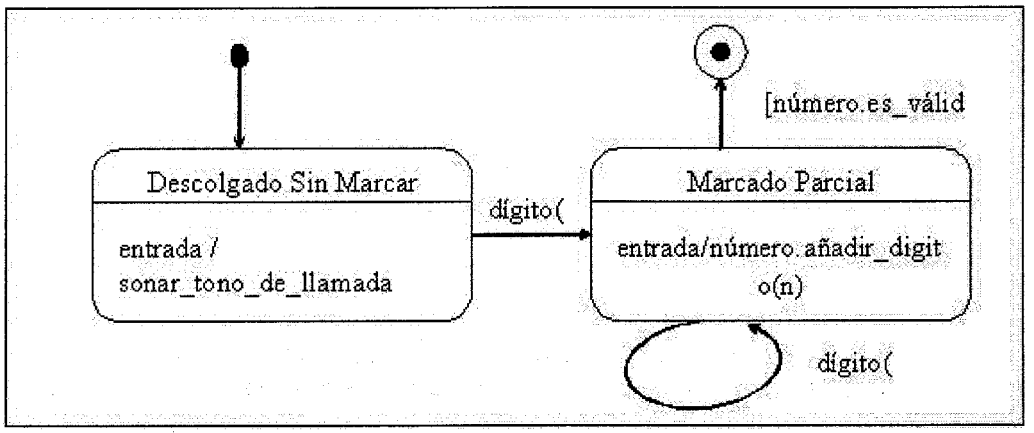


Fig. 19 Diagrama de Estados.

Un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (finalización del caso de uso o destrucción del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudo estados, pues un objeto no puede "estar" en esos estados, pero nos sirven para saber cuáles son las transiciones iniciales y final(es).

3.7 Modelado Dinámico

3.7.1 Diagramas De Actividades

Vamos a recordar los diferentes modelos que sirven para representar el aspecto dinámico de un sistema:
- Diagramas de secuencia
- Diagramas de colaboración
- Diagramas de estados
- Diagramas de casos de uso
- Diagramas de actividades

En este capítulo nos centraremos en los diagramas de actividades que sirven fundamentalmente para modelar el flujo de control entre actividades. La idea es generar

una especie de diagrama Pert, en el que se puede ver el flujo de actividades que tienen lugar a lo largo del tiempo, así como las tareas concurrentes que pueden realizarse a la vez. El diagrama de actividades sirve para representar el sistema desde otra perspectiva, y de este modo complementa a los anteriores diagramas vistos. Gráficamente un diagrama de actividades será un conjunto de arcos y nodos. Desde un punto de vista conceptual, el diagrama de actividades muestra cómo fluye el control de unas clases a otras con la finalidad de culminar con un flujo de control total que se corresponde con la consecución de un proceso más complejo. Por este motivo, en un diagrama de actividades aparecerán acciones y actividades correspondientes a distintas clases. Colaborando todas ellas para conseguir un mismo fin. Ejemplo: Hacer un pedido.

3.7.2 Contenido del diagrama de actividades

Básicamente un diagrama de actividades contiene:

- Estados de actividad
- Estados de acción
- Transiciones
- Objetos

3.7.2.1 Estados de actividad y estados de acción

La representación de ambos es un rectángulo con las puntas redondeadas, en cuyo interior se representa bien una actividad o bien una acción. La forma de expresar tanto una actividad como una acción, no queda impuesta por UML, se podría utilizar lenguaje natural, una especificación formal de expresiones, un metalenguaje, etc. La idea central es la siguiente: "Un estado que represente una acción es atómico, lo que significa que su ejecución se puede considerar instantánea y no puede ser interrumpida" En la Figura 20, podemos ver ejemplos de estados de acción.

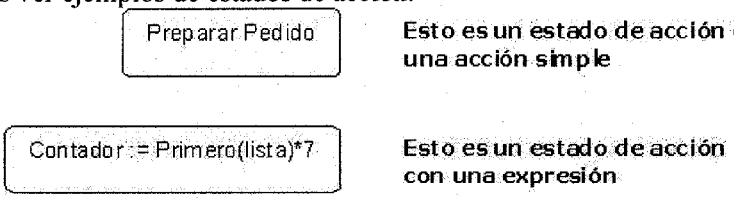


Fig. 20 Estados de Acción

En cambio un estado de actividad, sí puede descomponerse en más sub-actividades representadas a través de otros diagramas de actividades. Además estos estados sí pueden ser interrumpidos y tardan un cierto tiempo en completarse. En los estados de actividad podemos encontrar otros elementos adicionales como son: acciones de entrada (entry) y de salida (exit) del estado en cuestión, así como definición de sub-máquinas, como podemos ver en la Figura 21.

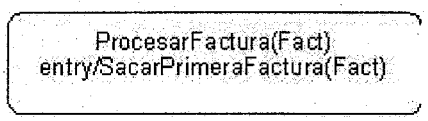


Fig. 21 Estado de Actividad

3.7.2.2 Transiciones

Las transiciones reflejan el paso de un estado a otro, bien sea de actividad o de acción. Esta transición se produce como resultado de la finalización del estado del que parte el arco dirigido que marca la transición. Como todo flujo de control debe empezar y terminar en algún momento, podemos indicar esto utilizando dos disparadores de inicio y fin tal y como queda reflejado en el ejemplo de la Figura 22.

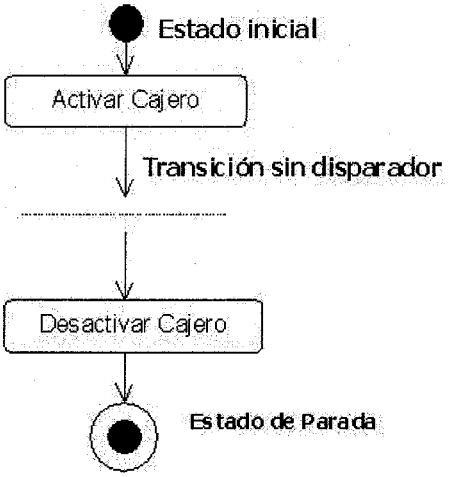


Fig. 22 Transiciones sin disparadores

3.7.2.3 Bifurcaciones

Un flujo de control no tiene porqué ser siempre secuencial, puede presentar caminos alternativos. Para poder representar dichos caminos alternativos o bifurcación se utilizará como símbolo el rombo. Dicha bifurcación tendrá una transición de entrada y dos o más de salida. En cada transición de salida se colocará una expresión booleana que será evaluada una vez al llegar a la bifurcación, las guardas de la bifurcación han de ser excluyentes y contemplar todos los casos ya que de otro modo la ejecución del flujo de control quedaría interrumpida. Para poder cubrir todas las posibilidades se puede utilizar la palabra ELSE, para indicar una transición obligada a un determinado estado cuando el resto de guardas han fallado. En la Figura 23 podemos ver un ejemplo de bifurcación.

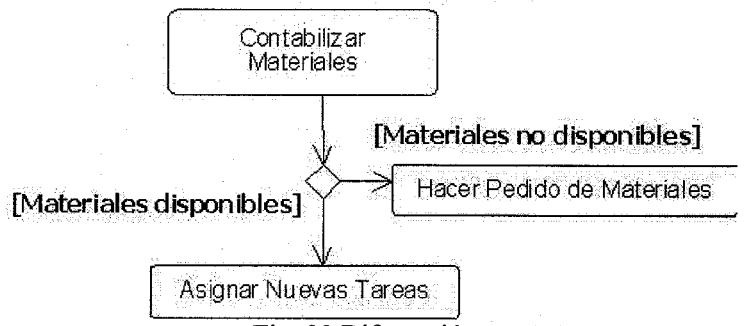


Fig. 23 Bifurcación

3.7.2.4 División y unión

No sólo existe el flujo secuencial y la bifurcación, también hay algunos casos en los que se requieren tareas concurrentes. UML representa gráficamente el proceso de división, que representa la concurrencia, y el momento de la unión de nuevo al flujo de control secuencial, por una línea horizontal ancha. En la Figura 24 podemos ver cómo se representa gráficamente.

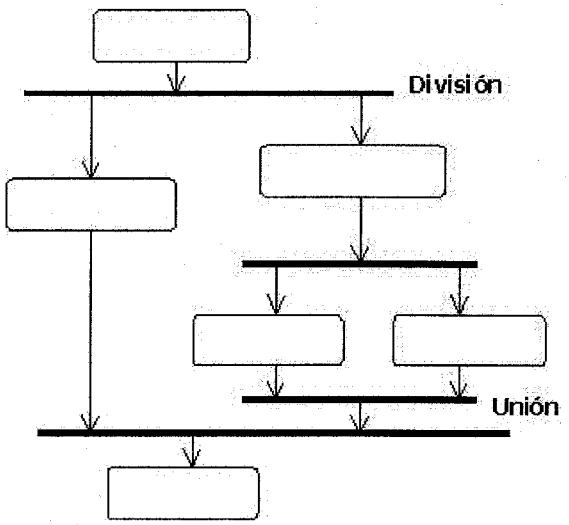


Fig. 24 División y unión

3.7.2.5 Calles

Cuando se modelan flujos de trabajo de organizaciones, es especialmente útil dividir los estados de actividades en grupos, cada grupo tiene un nombre concreto y se denominan calles. Cada calle representa a la parte de la organización responsable de las actividades que aparecen en esa calle. Gráficamente quedaría como se muestra en la Figura 25.

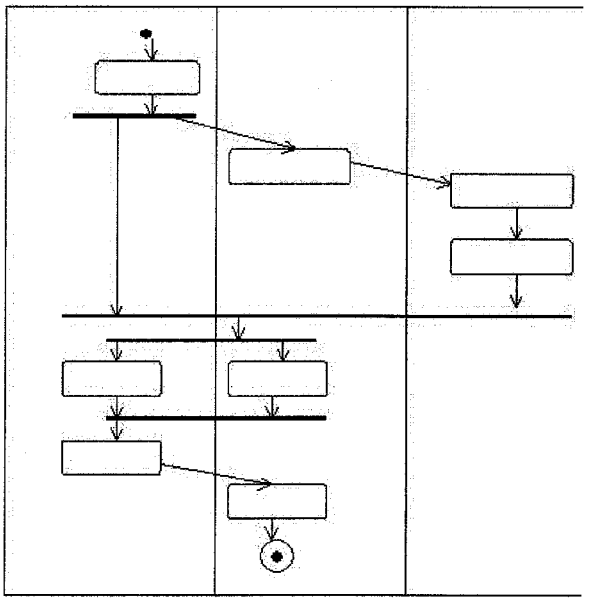


Fig. 25 Calles

3.8 Modelado Físico De Un Sistema OO

3.8.1 Componentes

Los componentes pertenecen al mundo físico, es decir, representan un bloque de construcción al modelar aspectos físicos de un sistema. Una característica básica de un componente es que:

"debe definir una abstracción precisa con una interfaz bien definida, y permitiendo reemplazar fácilmente los componentes más viejos con otros más nuevos y compatibles."

En UML todos los elementos físicos se modelan como componentes. UML proporciona una representación gráfica para estos como se puede ver en la Figura 26, en la que XXXX.dll, es el nombre del componente.

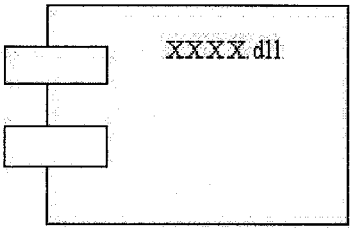


Fig. 26 Representación de un componente

Cada componente debe tener un nombre que lo distinga de los demás. Al igual que las clases los componentes pueden enriquecerse con compartimentos adicionales que muestran sus detalles, como se puede ver en la Figura 27.

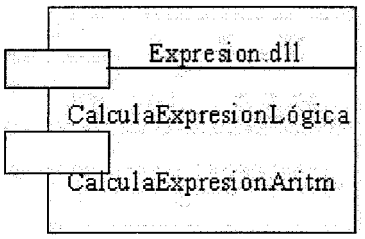


Fig. 27 Representación extendida de un componente

Vamos a ver con más detalle cuáles son los parecidos y las diferencias entre los componentes y las clases.

PARECIDOS
Ambos tienen nombre
Ambos pueden realizar un conjunto de interfaces.
Pueden participar en relaciones de dependencia, generalización y asociación.
Ambos pueden anidarse
Ambos pueden tener instancias
Ambos pueden participar en interacciones

DIFERENCIAS
Las Clases Los Componentes
Representan abstracciones lógicas Representan elementos físicos
Es decir los componentes pueden estar en nodos y las clases no
Pueden tener atributos y operaciones directamente accesibles. Sólo tienen operaciones y estas son alcanzables a través de la interfaz del componente.

3.8.1.1 Interfaces

Tanto los servicios propios de una clase como los de un componente, se especifican a través de una Interfaz. Por ejemplo, todas las facilidades más conocidas de los sistemas operativos, basados en componentes (COM+, CORBA, etc.), utilizan las interfaces como lazo de unión entre unos componentes y otros. La relación entre un componente y sus interfaces se puede representar de dos maneras diferentes, de forma icónica como se indica en la Figura 28, y de forma expandida como se muestra en la Figura 29.

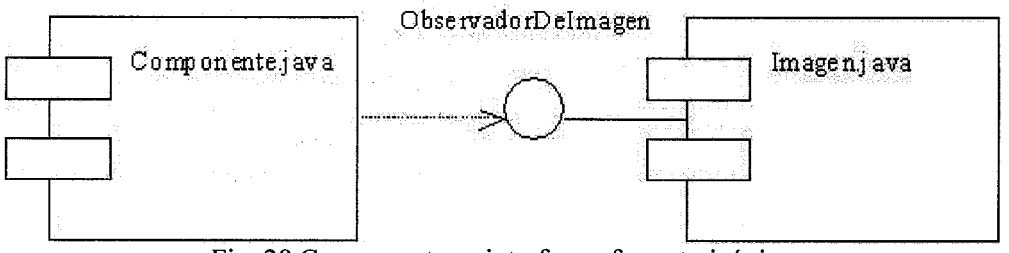


Fig. 28 Componentes e interfaces, formato icónico

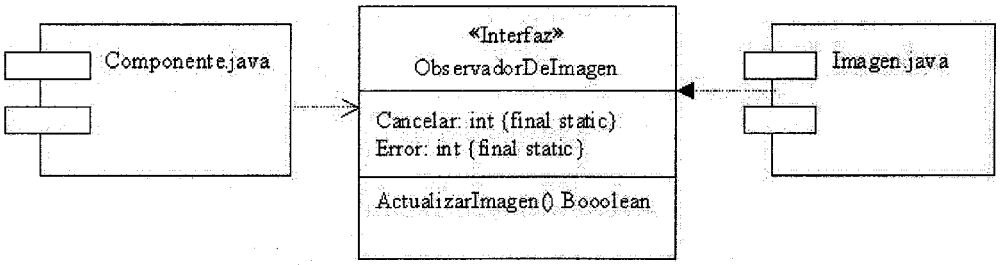


Fig. 29 Componentes e interfaces, formato extendido

3.8.1.2 Tipos de componentes

Existen básicamente tres tipos de componentes:

1. Componentes de despliegue: componentes necesarios y suficientes para formar un sistema ejecutable, como pueden ser las bibliotecas dinámicas (DLLs) y ejecutables (EXEs).
2. Componentes producto del trabajo: estos componentes son básicamente productos que quedan al final del proceso de desarrollo. Consisten en cosas como archivos de código fuente y de datos a partir de los cuales se crean los componentes de despliegue.
3. Componentes de ejecución: son componentes que se crean como consecuencia de un sistema en ejecución. Es el caso de un objeto COM+ que se instancia a partir de una DLL.

3.8.1.3 Organización de componentes

Los componentes se pueden agrupar en paquetes de la misma forma que se organizan las clases. Además se pueden especificar entre ellos relaciones de dependencia, generalización, asociación (incluyendo agregación), y realización.

3.8.1.4 Estereotipos de componentes

UML define cinco estereotipos estándar que se aplican a los componentes:
Executable Componente que se puede ejecutar en un nodo.
library Biblioteca de objetos estática o dinámica.
table Componentes que representa una tabla de una base de datos.
file Componente que representa un documento que contiene código fuente o datos.
document Componente que representa un documento.
UML no especifica iconos predefinidos para estos estereotipos.

3.8.2 Despliegue

3.8.2.1 Nodos

Al igual que los componentes los nodos pertenecen al mundo material. Vamos a definir un nodo como un elemento físico, que existe en tiempo de ejecución y representa un recurso computacional que generalmente tiene alguna memoria y, a menudo, capacidad de procesamiento. Los nodos sirven para modelar la topología del hardware sobre el que se ejecuta el sistema. Un nodo representa normalmente un procesador o un dispositivo sobre el que se pueden desplegar los componentes. Un nodo debe tener un nombre

asignado que lo distinga del resto de nodos. Además los nodos se representan gráficamente como se indica en la Figura 30.

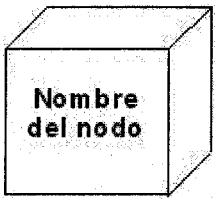


Fig. 30 Nodos

3.8.2.2 Nodos y componentes

En muchos aspectos los nodos y los componentes tienen características parecidas. Vamos a ver con más detalle cuales son los parecidos y las diferencias entre los componentes y los nodos.

PARECIDOS
Ambos tienen nombre
Pueden participar en relaciones de dependencia, generalización y asociación.
Ambos pueden anidarse
Ambos pueden tener instancias
Ambos pueden participar en interacciones

DIFERENCIAS
Los Nodos Los Componentes
Son los elementos donde se ejecutan los componentes. Son los elementos que participan en la ejecución de un sistema.
Representan el despliegue físico de los componentes. Representan el empaquetamiento físico de los elementos lógicos.
La relación entre un nodo y los componentes que despliega se pueden representar mediante una relación de dependencia como se indica en la Figura 31.
Los nodos se pueden agrupar en paquetes igual que los las clases y los componentes.
Los tipos de relación más común entre nodos es la asociación. Una asociación entre nodos viene a representar una conexión física entre nodos como se puede ver en la Figura 32.

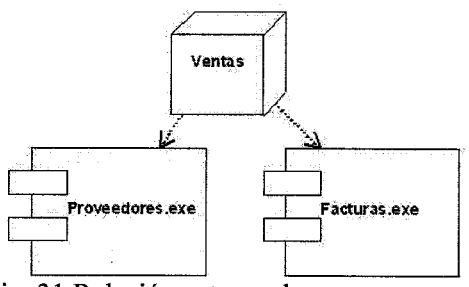


Fig. 31 Relación entre nodos y componentes

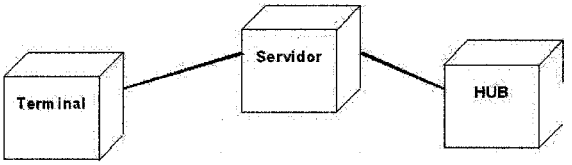


Fig. 32 Conexiones entre nodos

Existen dos tipos de diagramas que sirven para modelar los aspectos físicos de un sistema orientado a objetos:

- Diagramas de Componentes
- Diagramas de Despliegue

3.8.3 Diagramas de Componentes

Un diagrama de componentes muestra la organización y las dependencias entre un conjunto de componentes. Para todo sistema OO se han de construir una serie de diagramas que modelan tanto la parte estática (diagrama de clases), como dinámica (diagramas de secuencia, colaboración, estados y de actividades), pero llegado el momento todo esto se debe materializar en un sistema implementado que utilizará partes ya implementadas de otros sistemas, todo esto es lo que pretendemos modelar con los diagramas de componentes.

3.8.3.1 Algunos conceptos

Un diagrama de componentes muestra un conjunto de componentes y sus relaciones de manera gráfica a través del uso de nodos y arcos entre estos.

Normalmente los diagramas de componentes contienen:

- Componentes
- Interfaces
- Relaciones de dependencia, generalización, asociaciones y realización.
- Paquetes o subsistemas
- Instancias de algunas clases

Visto de otro modo un diagrama de componentes puede ser un tipo especial de diagrama de clases que se centra en los componentes físicos del sistema.

3.8.3.2 Usos más comunes

a) Modelado de Código Fuente
b) Los diagramas de componentes se pueden utilizar para modelar la gestión de la configuración de los archivos de código fuente, tomando como productos de trabajo precisamente estos ficheros. Esto resulta bastante útil por ejemplo cuando se han implementado unas partes con Java otras con C, etc. El resultado de esta implementación pueden ser multitud de ficheros ejecutables con características particulares, de manera que la mejor forma de controlarlos es estableciendo gestión de configuración.
c) Para poder llevar a cabo esta gestión con éxito será necesario definir los estereotipos de ficheros que se quieren tener bajo control así como las relaciones entre dichos tipos de ficheros.
d) Para modelar el código fuente de un sistema:
   - Hay que identificar el conjunto de archivos de código fuente de interés y modelarlos como componentes estereotipados como archivos.

• Si el sistema es muy grande es necesario utilizar los paquetes para agrupar los archivos de código fuente.
• Es necesario identificar la versión del componente.

1. Modelado de una versión ejecutable y bibliotecas.
La utilización de los componentes para modelar versiones ejecutables se centra en la definición de todos los elementos que componen lo que se conoce como versión ejecutable, es decir la documentación, los ficheros que se entregan etc.
Para modelar una versión ejecutable es preciso:
• Identificar el conjunto de componentes que se pretende modelar.
• Identificar el estereotipo de cada componente del conjunto seleccionado.
• Para cada componente de este conjunto hay que considerar las relaciones con los vecinos. Esto implica definir las interfaces importadas por ciertos componentes y las exportadas por otros.

2. Modelado de una base de datos física
Para modelar una base de datos física es necesario:
• Identificar las clases del modelo que representan el esquema lógico de la base de datos.
• Seleccionar una estrategia para hacer corresponder las clases con tablas. Así como la distribución física de la/s base/s de datos.
• Para poder visualizar, especificar, construir y documentar dicha correspondencia es necesario crear un diagrama de componentes que tenga componentes estereotipados como tablas.
• Donde sea posible es aconsejable utilizar herramientas que ayuden a transformar diseño lógico en físico.

3.8.4 Diagramas de Despliegue

3.8.4.1 Técnicas más comunes de modelado
Modelado de un sistema empotrado
El desarrollo de un sistema empotrado es más que el desarrollo de un sistema software. Hay que manejar el mundo físico. Los diagramas de despliegue son útiles para facilitar la comunicación entre los ingenieros de hardware y los de software.
Para modelar un sistema empotrado es necesario:
- Identificar los dispositivos y nodos propios del sistema.
- Proporcionar señales visuales, sobre todo para los dispositivos poco usuales.
- Modelar las relaciones entre esos procesadores y dispositivos en un diagrama de despliegue.
- Si es necesario hay que detallar cualquier dispositivo inteligente, modelando su estructura en un diagrama de despliegue más pormenorizado.

Modelado de un sistema cliente servidor

La división entre cliente y servidor en un sistema es complicada ya que implica tomar algunas decisiones sobre dónde colocar físicamente sus componentes software, qué cantidad de software debe residir en el cliente, etc.

Para modelar un sistema cliente/servidor hay que hace lo siguiente:

- Identificar los nodos que representan los procesadores cliente y servidor del sistema.
- Destacar los dispositivos relacionados con el comportamiento del sistema.
- Proporcionar señales visuales para esos procesadores y dispositivos a través de estereotipos.
- Modelar la tipología de esos nodos mediante un diagrama de despliegue.

3.8.5 Arquitectura del Sistema

3.8.5.1 Arquitectura de tres niveles

La llamada "Arquitectura en Tres Niveles", es la más común en sistemas de información que además de tener una interfaz de usuario contemplan la persistencia de los datos.

Una descripción de los tres niveles sería la siguiente:

Nivel 1: Presentación – ventanas, informes, etc.

Nivel 2: Lógica de la Aplicación – tareas y reglas que gobiernan el proceso.

Nivel 3: Almacenamiento – mecanismo de almacenamiento.

3.8.5.2 Arquitectura de tres niveles orientadas a objetos

a) Descomposición del nivel de lógica de la aplicación
b) En el diseño orientado a objetos, el nivel de lógica de la aplicación se descompone en sub-niveles que son los siguientes:
   Objetos del Dominio: son clases que representan objetos del dominio. Por ejemplo en un problema de ventas, una "Venta" sería un objeto del dominio.
   Servicios: se hace referencia a funciones de interacción con la base de datos, informes, comunicaciones, seguridad, etc.

3.8.5.3 Arquitectura MULTI-nivel

La arquitectura de tres niveles puede pasar a llamarse de Múltiples Niveles si tenemos en cuenta el hecho de que todos los niveles de la arquitectura de tres niveles se pueden descomponer cada uno de ellos cada vez más.

Por ejemplo el nivel de Servicios, se puede descomponer en servicios de alto y de bajo nivel, identificando como de alto nivel los servicios de generación de informes y como de bajo nivel los de manejo de ficheros de entrada y salida.

El motivo que lleva a descomponer la arquitectura del sistema en diferentes niveles es múltiple:

- Separación de la lógica de la aplicación en componentes separados que sean más fácilmente reutilizables.
- Distribución de niveles en diferentes nodos físicos de computación.
- Reparto de recursos humanos en diferentes niveles de la arquitectura.

3.8.5.4 Paquetes

La forma que tiene UML de agrupar elementos en subsistemas es a través del uso de Paquetes, pudiéndose anidar los paquetes formando jerarquías de paquetes. De hecho un sistema que no tenga necesidad de ser descompuesto en subsistemas se puede considerar como con un único paquete que lo abarca todo.

Gráficamente un paquete viene representado como se indica en la Figura 33.

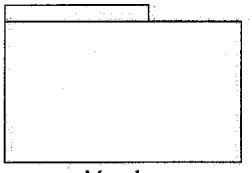


Fig. 33 Representación de un paquete en UML

En la Figura 34, vemos cómo se representa la arquitectura del sistema con la notación de paquetes.

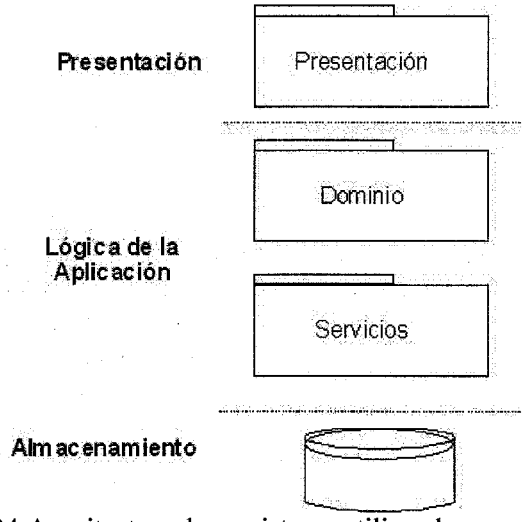


Fig. 34 Arquitectura de un sistema utilizando paquetes.

3.8.5.5 Identificación de Paquetes

Vamos a definir una serie de reglas que nos pueden ser de utilidad a la hora de agrupar los diferentes elementos en paquetes.

- Conviene agrupar elementos que proporcionen un mismo servicio.
- Los elementos que se agrupen en un mismo paquete han de presentar un alto grado de cohesión, es decir deben estar muy relacionados.
- Los elementos que estén en diferentes paquetes deben tener poca relación, es decir deben colaborar lo menos posible.

3.9 Proceso de Desarrollo

Cuando se va a construir un sistema software es necesario conocer un lenguaje de programación, pero con eso no basta. Si se quiere que el sistema sea robusto y mantenible, es necesario que el problema sea analizado y la solución sea cuidadosamente diseñada. Se debe seguir un proceso robusto, que incluya las actividades principales. Si se sigue un proceso de desarrollo que se ocupa de plantear cómo se realiza el análisis y el diseño, y cómo se relacionan los productos de ambos, entonces la construcción de sistemas software va a poder ser planificable y repetible, y la probabilidad de obtener un sistema de mejor calidad al final del proceso aumenta considerablemente, especialmente cuando se trata de un equipo de desarrollo formado por varias personas.
La notación que se usa para los distintos modelos, tal y como se ha dicho anteriormente, es la proporcionada por UML, que se ha convertido en el estándar de facto en cuanto a notación orientada a objetos. El uso de UML permite integrar con mayor facilidad en el equipo de desarrollo a nuevos miembros y compartir con otros equipos la documentación, pues es de esperar que cualquier desarrollador versado en orientación a objetos conozca y use UML (o se esté planteando su uso).
Se va a abarcar todo el ciclo de vida, empezando por los requisitos y acabando en el sistema funcionando, proporcionando así una visión completa y coherente de la producción de sistemas software. El enfoque que toma es el de un ciclo de vida iterativo incremental, el cual permite una gran flexibilidad a la hora de adaptarlo a un proyecto y a un equipo de desarrollo específicos. El ciclo de vida está dirigido por casos de uso, es decir, por la funcionalidad que ofrece el sistema a los futuros usuarios del mismo. Así no se pierde de vista la motivación principal que debería estar en cualquier proceso de construcción de software: el resolver una necesidad del usuario/cliente.

3.9.1 Visión General

El proceso a seguir para realizar desarrollo orientado a objetos es complejo, debido a la complejidad que nos vamos a encontrar al intentar desarrollar cualquier sistema software de tamaño medio-alto. El proceso está formado por una serie de actividades y subactividades, cuya realización se va repitiendo en el tiempo, aplicadas a distintos elementos.
En este apartado se va a presentar una visión general para poder tener una idea del proceso a alto nivel, y más adelante se verán los pasos que componen cada fase.
Las tres fases al nivel más alto son las siguientes:
• Planificación y Especificación de Requisitos: Planificación, definición de requisitos, construcción de prototipos, etc.
• Construcción: La construcción del sistema. Las fases dentro de esta etapa son las siguientes:

- Diseño de Alto Nivel: Se analiza el problema a resolver desde la perspectiva de los usuarios y de las entidades externas que van a solicitar servicios al sistema.
- Diseño de Bajo Nivel: El sistema se especifica en detalle, describiendo cómo va a funcionar internamente para satisfacer lo especificado en el Diseño de Alto Nivel.
- Implementación: Se lleva lo especificado en el Diseño de Bajo Nivel a un lenguaje de programación.

- Pruebas: Se llevan a cabo una serie de pruebas para corroborar que el software funciona correctamente y que satisface lo especificado en la etapa de Planificación y Especificación de Requisitos.
- Instalación: La puesta en marcha del sistema en el entorno previsto de uso.

De ellas, la fase de Construir es la que va a consumir la mayor parte del esfuerzo y del tiempo en un proyecto de desarrollo. Para llevarla a cabo se va adoptar un enfoque iterativo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del diseño de alto y bajo nivel hasta la implementación y pruebas, tal y como se muestra en la Figura 35. El sistema va creciendo incrementalmente en cada ciclo. Con esta aproximación se consigue disminuir el grado de complejidad que se trata en cada ciclo, y se tiene pronto en el proceso una parte del sistema funcionando que se puede contrastar con el usuario/cliente.

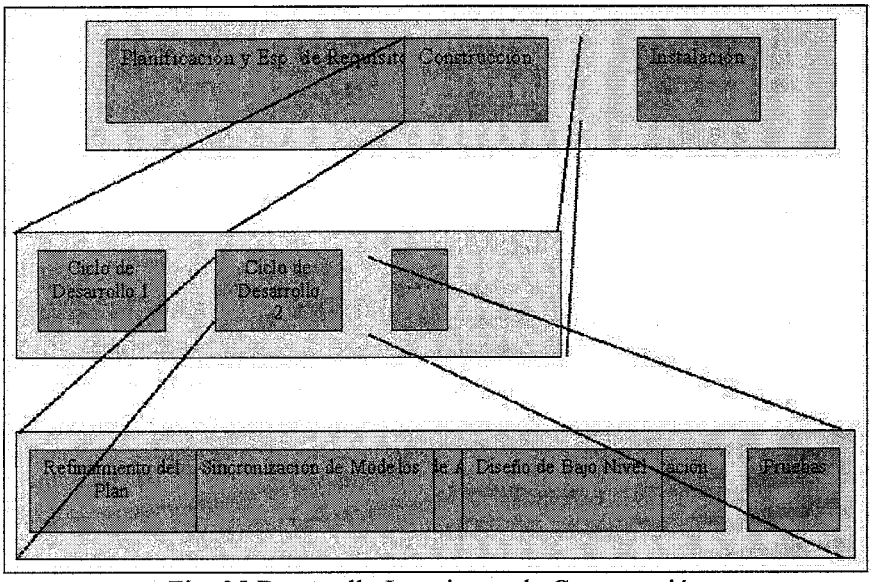


Fig. 35 Desarrollo Iterativo en la Construcción

Fase de Planificación y Especificación de Requisitos

Esta fase se corresponde con la Especificación de Requisitos tradicional ampliada con un Borrador de Modelo Conceptual y con una definición de Casos de Uso de alto nivel. En esta fase se decidiría si se aborda la construcción del sistema mediante desarrollo orientado a objetos o no, por lo que, en principio, es independiente del paradigma empleado posteriormente.

3.9.1.1 Actividades

Las actividades de esta fase son las siguientes:

- Definir el Plan-Borrador.
- Crear el Informe de Investigación Preliminar.

• Definir los Requisitos.
• Registrar Términos en el Glosario. (Continuado en posteriores fases)
• Implementar un Prototipo. (opcional)
• Definir Casos de Uso (de alto nivel y esenciales).
• Definir el Modelo Conceptual-Borrador. (puede retrasarse hasta una fase posterior)
• Definir la Arquitectura del Sistema-Borrador. (puede retrasarse hasta una fase posterior)
• Refinar el Plan.

El orden no es estricto, lo normal es que las distintas actividades se solapen en el tiempo. Esto sucede también en las actividades de las fases de diseño.
De estas actividades no se va a entrar en las que corresponden al campo de la planificación de proyectos software, como las correspondientes a creación de planes e informes preliminares.

3.9.1.2 Requisitos

El formato del documento de Especificación de Requisitos no está definido en UML, pero se ha incluido este punto para resaltar que la actividad de definición de requisitos es un paso clave en la creación de cualquier producto software. Para entender y describir los requisitos la técnica de casos de uso constituye una valiosa ayuda.

3.9.1.3 Casos de Uso

Un Caso de Uso es un documento narrativo que describe a los actores utilizando un sistema para satisfacer un objetivo. Es una historia o una forma particular de usar un sistema. Los casos de uso son requisitos, en particular requisitos funcionales.
Nótese que UML no define un formato para describir un caso de uso. Tan sólo define la manera de representar la relación entre actores y casos de uso en un diagrama. Sin embargo, un caso de uso individual no es un diagrama, es un documento de texto. En la siguiente sección se define el formato textual para la descripción de un caso de uso que se va a utilizar en este documento.
Un escenario es un camino concreto a través del caso de uso, una secuencia específica de acciones e interacciones entre los actores y el sistema. Más adelante se verá la representación de los escenarios correspondientes a un caso de uso por medio de Diagramas de Secuencia.
En un primer momento interesa abordar un caso de uso desde un nivel de abstracción alto, utilizando el formato de alto nivel. Cuando se quiere describir un caso de uso con más detalle se usa el formato expandido.

3.9.1.3.1 Casos de Uso de Alto Nivel

El siguiente Caso de Uso de Alto Nivel describe el proceso de sacar dinero cuando se está usando un cajero automático:
- Caso de Uso: Realizar Reintegro
- Actores: Cliente
- Tipo: primario
- Descripción: Un Cliente llega al cajero automático, introduce la tarjeta, se identifica y solicita realizar una operación de reintegro por una cantidad específica. El cajero le da el

dinero solicitado tras comprobar que la operación puede realizarse. El Cliente coge el dinero y la tarjeta y se va.
En un caso de uso descrito a alto nivel la descripción es muy general, normalmente se condensa en dos o tres frases. Es útil para comprender el ámbito y el grado de complejidad del sistema.

3.9.1.3.2 Casos de Uso Expandidos
Los casos de uso que se consideren los más importantes y que se considere que son los que más influencian al resto, se describen a un nivel más detallado: en el formato expandido.
La principal diferencia con un caso de uso de alto nivel está en que incluye un apartado de Curso Típico de Eventos, pero también incluye otros apartados como se ve en el siguiente ejemplo:
- Caso de Uso: Realizar Reintegro
- Actores: Cliente (iniciador)
- Propósito: Realizar una operación de reintegro de una cuenta del banco
- Visión General:

  Un Cliente llega al cajero automático, introduce la tarjeta, se identifica y solicita realizar una operación de reintegro por una cantidad específica. El cajero le da el dinero solicitado tras comprobar que la operación puede realizarse. El Cliente coge el dinero y la tarjeta y se va.
- Tipo: primario y esencial
- Referencias: Funciones: R1.3, R1.7
- Curso Típico de Eventos:

Acción del Actor Respuesta del Sistema
1. Este caso de uso empieza cuando un Cliente introduce una tarjeta en el cajero.
2. Pide la clave de identificación.
3. Introduce la clave.
4. Presenta las opciones de operaciones disponibles.
5. Selecciona la operación de Reintegro.
6. Pide la cantidad a retirar.
7. Introduce la cantidad requerida.
8. Procesa la petición y da el dinero solicitado. Devuelve la tarjeta y genera un recibo.
9. Recoge la tarjeta.
10. Recoge el recibo.
11. Recoge el dinero y se va.

Cursos Alternativos:
- Línea 4: La clave es incorrecta. Se indica el error y se cancela la operación.
- Línea 8: La cantidad solicitada supera el saldo. Se indica el error y se cancela la operación. El significado de cada apartado de este formato es como sigue:
  - Caso de Uso: Nombre del Caso de Uso
  - Actores: Lista de actores (agentes externos), indicando quién inicia el caso de uso. Los actores son normalmente roles que un ser humano desempeña, pero puede ser cualquier tipo de sistema.
  - Propósito: Intención del caso de uso.

- Visión General: Repetición del caso de uso de alto nivel, o un resumen similar.
- Tipo: 1. primario, secundario u opcional (descritos más adelante).
- Tipo: 2. esencial o real (descritos más adelante).
- Referencias: Casos de uso relacionados y funciones del sistema que aparecen en los requisitos.
- Curso Típico de Eventos: Descripción de la interacción entre los actores y el sistema mediante las acciones numeradas de cada uno. Describe la secuencia más común de eventos, cuando todo va bien y el proceso se completa satisfactoriamente. En caso de haber alternativas con grado similar de probabilidad se pueden añadir secciones adicionales a la sección principal, como se verá más adelante.
- Cursos Alternativos: Puntos en los que puede surgir una alternativa, junto con la descripción de la excepción.

3.9.1.3.3 Identificación de Casos de Uso

La identificación de casos de uso requiere un conocimiento medio acerca de los requisitos, y se basa en la revisión de los documentos de requisitos existentes, y en el uso de la técnica de brainstorming entre los miembros del equipo de desarrollo.
Como guía para la identificación inicial de casos de uso hay dos métodos:
a) Basado en Actores
1. Identificar los actores relacionados con el sistema y/o la organización.
2. Para cada actor, identificar los procesos que inicia o en los que participa.

b) Basado en Eventos
1. Identificar los eventos externos a los que el sistema va a tener que responder.
2. Relacionar los eventos con actores y casos de uso.

Ejemplos de casos de uso:
• Pedir un producto.
• Matricularse en un curso de la facultad.
• Comprobar la ortografía de un documento en un procesador de textos.
• Comprar un libro en una tienda de libros en Internet

3.9.1.3.4 Identificación de los Límites del Sistema

En la descripción de un caso de uso se hace referencia en todo momento al "sistema". Para que los casos de uso tengan un significado completo es necesario que el sistema esté definido con precisión.
Al definir los límites del sistema se establece una diferenciación entre lo que es interno y lo que es externo al sistema. El entorno exterior se representa mediante los actores.
Ejemplos de sistemas son:
- El hardware y software de un sistema informático.
- Un departamento de una organización.
- Una organización entera.

Si no se está haciendo reingeniería del proceso de negocio lo más normal es escoger como sistema el primero de los ejemplos: el hardware y el software del sistema que se quiere construir.

*3.9.*1.3.5 Tipos de Casos de Uso

a) Según Importancia.

Para establecer una primera aproximación a la priorización de casos de uso que identifiquemos los vamos a distinguir entre:

- Primarios: Representan los procesos principales, los más comunes, como Realizar Reintegro en el caso del cajero automático.
- Secundarios: Representan casos de uso menores, que van a necesitarse raramente, tales como Añadir Nueva Operación.
- Opcionales: Representan procesos que pueden no ser abordados en el presente proyecto.

c) Según el Grado de Compromiso con el Diseño

d) En las descripciones que se han visto anteriormente no se han hecho apenas compromisos con la solución, se han descrito los casos de uso a un nivel abstracto, independiente de la tecnología y de la implementación. Un caso de uso definido a nivel abstracto se denomina esencial. Los casos de uso definidos a alto nivel son siempre esenciales por naturaleza, debido a su brevedad y abstracción.

Por el contrario, un caso de uso real describe concretamente el proceso en términos del diseño real, de la solución específica que se va a llevar a cabo. Se ajusta a un tipo de interfaz específica, y se baja a detalles como pantallas y objetos en las mismas.

Como ejemplo de una parte de un Caso de Uso Real para el caso del reintegro en un cajero automático tenemos la siguiente descripción del Curso Típico de Eventos:
Acción del Actor Respuesta del Sistema
1. Este caso de uso empieza cuando un Cliente introduce una tarjeta en la ranura para tarjetas.
2. Pide el PIN (Personal Identification Number).
3. Introduce el PIN a través del teclado numérico.
4. Presenta las opciones de operaciones disponibles.
5. etc. 6. etc.

En principio, los casos de uso reales deberían ser creados en la fase de Diseño de Bajo Nivel y no antes. Sin embargo, en algunos proyectos se plantea la definición de interfaces en fases tempranas del ciclo de desarrollo, en base a que son parte del contrato. En este caso se pueden definir algunos o todos los casos de uso reales, a pesar de que suponen tomar decisiones de diseño muy pronto en el ciclo de vida.

No hay una diferencia estricta entre un Caso de Uso Esencial y uno Real, el grado de compromiso con el diseño es un continuo, y una descripción específica de un caso de uso estará situada en algún punto de la línea entre Casos de Uso Esenciales y Reales, normalmente más cercano a un extremo que al otro, pero es raro encontrar Casos de Uso Esenciales o Reales puros.

3.9.1.3.6 Consejos Relativos a Casos de Uso

a) Nombre

El nombre de un Caso de Uso debería ser un verbo, para enfatizar que se trata de un proceso, por ejemplo: Comprar Artículos o Realizar Pedido.

b) Alternativas equiprobables

Cuando se tiene una alternativa que ocurre de manera relativamente ocasional, se indica en el apartado Cursos Alternativos. Pero cuando se tienen distintas

opciones, todas ellas consideradas normales se puede completar el Curso Típico de Eventos con secciones adicionales.

Así, si en un determinado número de línea hay una bifurcación se pueden poner opciones que dirigen el caso de uso a una sección que se detalla al final del Curso Típico de Eventos, en la siguiente forma:

- Curso Típico de Eventos:
- Sección: Principal

Acción del Actor Respuesta del Sistema
1. Este caso de uso empieza cuando Actor llega al sistema.
2. Pide la operación a realizar.
3. Escoge la operación A.
4. Presenta las opciones de pago.
5. Selecciona el tipo de pago:
    a. Si se paga al contado ver sección Pago al Contado.
    b. Si se paga con tarjeta ver sección Pago con Tarjeta.
6. Genera recibo.
7. Recoge el recibo y se va

Cursos Alternativos:
• Líneas 3 y 5: Selecciona Cancelar. Se cancela la operación.
- Sección: Pago al Contado

Acción del Actor Respuesta del Sistema.
1. Mete los billetes correspondientes
2. Coge los billetes y sigue pidiendo dinero hasta que la cantidad está satisfecha
3. Devuelve el cambio.

Cursos Alternativos:
• Línea 3: No hay cambio suficiente. Se cancela la operación.
- Sección: Pago con Tarjeta

3.9.1.4 Construcción del Modelo de Casos de Uso

Para construir el Modelo de Casos de Uso en la fase de Planificación y Especificación de Requisitos se siguen los siguientes pasos:

1. Después de listar las funciones del sistema, se definen los límites del sistema y se identifican los actores y los casos de uso.
2. Se escriben todos los casos de uso en el formato de alto nivel. Se categorizan como primarios, secundarios u opcionales.
3. Se dibuja el Diagrama de Casos de Uso.
4. Se detallan relaciones entre casos de uso, en caso de ser necesarias, y se ilustran tales relaciones en el Diagrama de Casos de Uso.

5. Los casos de uso más críticos, importantes y que conllevan un mayor riesgo, se describen en el formato expandido esencial. Se deja la definición en formato expandido esencial del resto de casos de uso para cuando sean tratados en posteriores ciclos de desarrollo, para no tratar toda la complejidad del problema de una sola vez.
6. Se crean casos de uso reales sólo cuando:
   - Descripciones más detalladas ayudan significativamente a incrementar la comprensión del problema.
   - El cliente pide que los procesos se describan de esta forma.
7. Ordenar según prioridad los casos de uso.

3.9.1.5 Planificación de Casos de Uso según Ciclos de Desarrollo

La decisión de qué partes del sistema abordar en cada ciclo de desarrollo se va a tomar basándose en los casos de uso. Esto es, a cada ciclo de desarrollo se le va a asignar la implementación de uno o más casos de uso, o versiones simplificadas de casos de uso. Se asigna una versión simplificada cuando el caso de uso completo es demasiado complejo para ser tratado en un solo ciclo (ver Figura 36).

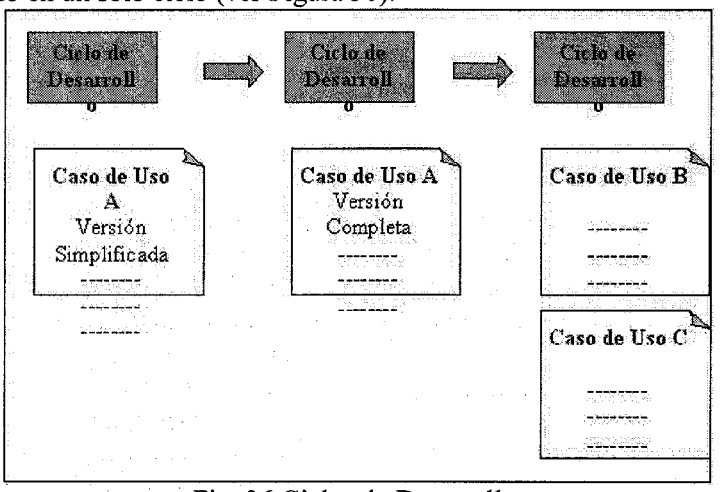


Fig. 36 Ciclos de Desarrollo.

Para tomar la decisión de qué casos de uso se van a tratar primero es necesario ordenarlos según prioridad. Las características de un caso de uso específico que van a hacer que un caso de uso tenga una prioridad alta son las siguientes:

a. Impacto significativo en el diseño de la arquitectura. Por ejemplo, si aporta muchas clases al modelo del dominio o requiere persistencia en los datos.
b. Se obtiene una mejor comprensión del diseño con un nivel de esfuerzo relativamente bajo.
c. Incluye funciones complejas, críticas en el tiempo o de nivel elevado de riesgo.
d. Implica bien un trabajo de investigación significante, o bien el uso de una tecnología nueva o arriesgada.
e. Representa un proceso de gran importancia en la línea de negocio.
f. Supone directamente un aumento de beneficios o una disminución de costes.

Para realizar la clasificación se puede asignar a cada caso de uso una valoración numérica de cada uno de estos puntos, para conseguir una puntuación total aplicando pesos a cada apartado.

3.9.1.5.1 Caso de Uso Inicialización

Prácticamente todos los sistemas van a tener un caso de uso Inicialización. Aunque puede ser que no tenga una prioridad alta en la clasificación realizada según el punto anterior, normalmente va a interesar que sea desarrollado desde el principio. Inicialmente se desarrolla una versión simplificada, que se va completando en cada ciclo de desarrollo para satisfacer las necesidades de inicialización de los casos de uso que se tratan en dicho ciclo. Así se tiene un sistema en cada ciclo de desarrollo que puede funcionar.

3.9.2 Fase de Construcción: Diseño de Alto Nivel

En la fase de Diseño de Alto Nivel de un ciclo de desarrollo se investiga sobre el problema, sobre los conceptos relacionados con el subconjunto de casos de uso que se esté tratando. Se intenta llegar a una buena comprensión del problema por parte del equipo de desarrollo, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación.

Cuando el ciclo de desarrollo no es el primero, antes de la fase de Diseño de Alto Nivel hay una serie de actividades de planificación. Estas actividades consisten en actualizar los modelos que se tengan según lo que se haya implementado, pues siempre se producen desviaciones entre lo que se ha analizado y diseñado y lo que finalmente se construye. Una vez se tienen los modelos acordes con lo implementado se empieza el nuevo ciclo de desarrollo con la fase de Diseño de Alto Nivel.

En esta fase se trabaja con los modelos de Diseño de Alto Nivel construidos en la fase anterior, ampliándolos con los conceptos correspondientes a los casos de uso que se traten en el ciclo de desarrollo actual.

3.9.2.1 Actividades

Las actividades de la fase de Diseño de Alto Nivel son las siguientes:

1. Definir Casos de Uso Esenciales en formato expandido. (si no están definidos)
2. Refinar los Diagramas de Casos de Uso.
3. Refinar el Modelo Conceptual.
4. Refinar el Glosario. (continuado en posteriores fases)
5. Definir los Diagramas de Secuencia del Sistema.
6. Definir Contratos de Operación.
7. Definir Diagramas de Estados. (opcional)

3.9.2.2 Modelo Conceptual

Una parte de la investigación sobre el dominio del problema consiste en identificar los conceptos que lo conforman. Para representar estos conceptos se va a usar un Diagrama de Estructura Estática de UML, al que se va a llamar Modelo Conceptual.

En el Modelo Conceptual se tiene una representación de conceptos del mundo real, no de componentes software.

El objetivo de la creación de un Modelo Conceptual es aumentar la comprensión del problema. Por tanto, a la hora de incluir conceptos en el modelo, es mejor crear un modelo con muchos conceptos que quedarse corto y olvidar algún concepto importante.

3.9.2.2.1 Identificación de Conceptos

Para identificar conceptos hay que basarse en el documento de Especificación de Requisitos y en el conocimiento general acerca del dominio del problema.
Otro consejo para identificar conceptos consiste en buscar sustantivos en los documentos de requisitos o, más concretamente, en la descripción de los casos de uso. No es un método infalible, pero puede servir de guía para empezar.
Para poner nombre a los conceptos se puede usar la analogía con el cartógrafo, resumida en los siguientes tres puntos:

- Usar los nombres existentes en el territorio: Hay que usar el vocabulario del dominio para nombrar conceptos y atributos.
- Excluir características irrelevantes: Al igual que el cartógrafo elimina características no relevantes según la finalidad del mapa (por ejemplo datos de población en un mapa de carreteras), un Modelo Conceptual puede excluir conceptos en el dominio que no son pertinentes en base a los requisitos.
- No añadir cosas que no están ahí: Si algo no pertenece al dominio del problema no se añade al modelo.

3.9.2.2.2 Creación del Modelo Conceptual

Para crear el Modelo Conceptual se siguen los siguientes pasos:

1. Hacer una lista de conceptos candidato usando una lista de Categorías de Conceptos y la búsqueda de sustantivos relacionados con los requisitos en consideración en este ciclo.
2. Representarlos en un diagrama.
3. Añadir las asociaciones necesarias para ilustrar las relaciones entre conceptos que es necesario conocer.
4. Añadir los atributos necesarios para contener toda la información que se necesite conocer de cada concepto.

3.9.2.2.3 Identificación de Asociaciones

Una asociación es una relación entre conceptos que indica una conexión con sentido y que es de interés en el conjunto de casos de uso que se está tratando.
Se incluyen en el modelo las asociaciones siguientes:

- Asociaciones para las que el conocimiento de la relación necesita mantenerse por un cierto período de tiempo (asociaciones "necesita-conocer").
- Asociaciones derivadas de la Lista de Asociaciones Típicas.

Una vez identificadas las asociaciones se representan en el Modelo Conceptual con la multiplicidad adecuada.

3.9.2.2.4 Identificación de Atributos

Es necesario incorporar al Modelo Conceptual los atributos necesarios para satisfacer las necesidades de información de los casos de uso que se estén desarrollando en ese momento.
Los atributos deben tomar valor en tipos simples (número, texto, etc.), pues los tipos complejos deberían ser modelados como conceptos y ser relacionados mediante asociaciones.
Incluso cuando un valor es de un tipo simple es más conveniente representarlo como concepto en las siguientes ocasiones:

- Se compone de distintas secciones. Por ejemplo: un número de teléfono, el nombre de una persona, etc.
- Tiene operaciones asociadas, tales como validación. Ejemplo: NIF.
- Tiene otros atributos. Por ejemplo un precio de oferta puede tener fecha de fin.
- Es una cantidad con una unidad. Ejemplo: El precio, que puede estar en pesetas o en euros.

Una vez definidos los atributos se tiene ya un Modelo Conceptual. Este modelo no es un modelo definitivo, pues a lo largo del desarrollo se va refinando según se le añaden conceptos que se habían pasado por alto.

3.9.2.3 Glosario

En el glosario debe aparecer una descripción textual de cualquier elemento de cualquier modelo, para eliminar toda posible ambigüedad. Se ordena alfabéticamente por término.

3.9.2.4 Diagramas de Secuencia del Sistema

Además de investigar sobre los conceptos del sistema y su estructura, también es preciso investigar en el Diseño de Alto Nivel sobre el comportamiento del sistema, visto éste como una caja negra. Una parte de la descripción del comportamiento del sistema se realiza mediante los Diagramas de Secuencia del Sistema.
En cada caso de uso se muestra una interacción de actores con el sistema. En esta interacción los actores generan eventos, solicitando al sistema operaciones. Por ejemplo, en el caso de una reserva de un billete de avión, el empleado de la agencia de viajes solicita al sistema de reservas que realice una reserva. El evento que supone esa solicitud inicia una operación en el sistema de reservas.
Los casos de uso representan una interacción genérica. Una instancia de un caso de uso se denomina escenario, y muestra una ejecución real del caso de uso, con las posibles bifurcaciones y alternativas resueltas de forma particular.
Un Diagrama de Secuencia de Sistema se representa usando la notación para diagramas de secuencia de UML. En él se muestra para un escenario particular de un caso de uso los eventos que los actores generan, su orden, y los eventos que se intercambian entre sistemas.
Para cada caso de uso que se esté tratando se realiza un diagrama para el curso típico de eventos, y además se realiza un diagrama para los cursos alternativos de mayor interés.
En la Figura 37 se muestra el Diagrama de Secuencia del Sistema para el caso de uso Realizar Reintegro de un cajero automático.

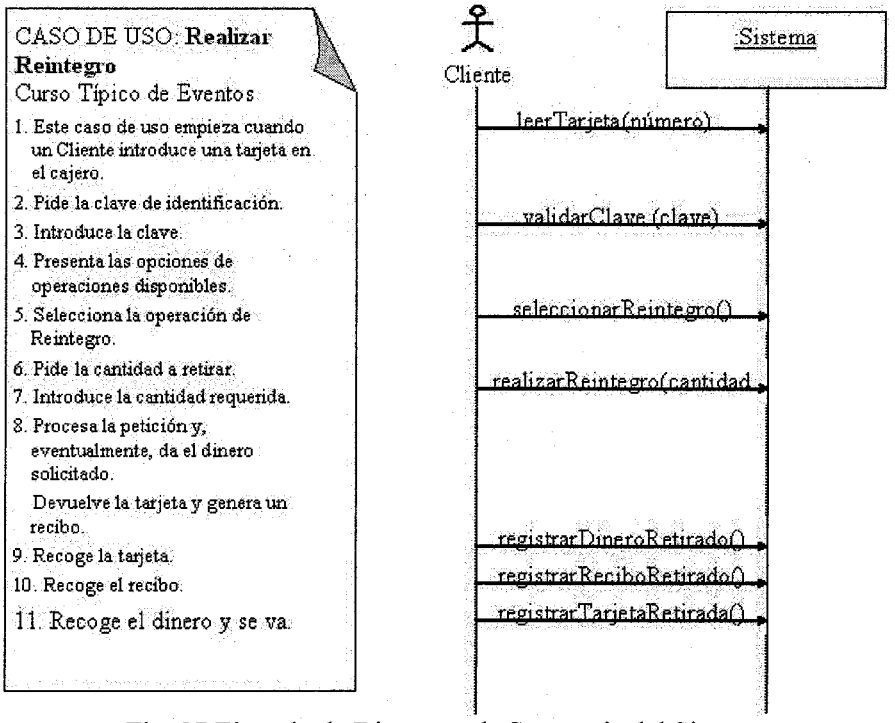


Fig. 37 Ejemplo de Diagrama de Secuencia del Sistema

3.9.2.4.1 Construcción de un Diagrama de Secuencia del Sistema

Para construir un Diagrama de Secuencia del Sistema para el curso típico de eventos de un caso de uso, se siguen los siguientes pasos:

1. Representar el sistema como un objeto con una línea debajo.
2. Identificar los actores que directamente operan con el sistema, y dibujar una línea para cada uno de ellos.
3. Partiendo del texto del curso típico de eventos del caso de uso, identificar los eventos (externos) del sistema que cada actor genera y representarlos en el diagrama.
4. Opcionalmente, incluir el texto del caso de uso en el margen del diagrama.

Los eventos del sistema deberían expresarse en base a la noción de operación que representan, en vez de en base a la interfaz particular. Por ejemplo, se prefiere "finalizarOperación" a "presionadaTeclaEnter", porque captura la finalidad de la operación sin realizar compromisos en cuanto a la interfaz usada.

3.9.2.5 Contratos de Operaciones

Una vez se tienen las Operaciones del Sistema identificadas en los Diagramas de Secuencia, se describe mediante contratos el comportamiento esperado del sistema en cada operación.

Un Contrato es un documento que describe qué es lo que se espera de una operación. Tiene una redacción en estilo declarativo, enfatizando en el qué más que en el cómo. Lo

más común es expresar los contratos en forma de pre- y post-condiciones en torno a cambios de estado.
Se puede escribir un contrato para un método individual de una clase software, o para una operación del sistema completa. En este punto se verá únicamente éste último caso.
Un Contrato de Operación del Sistema describe cambios en el estado del sistema cuando una operación del sistema es invocada.
A continuación se ve un ejemplo de Contrato:
Contrato
Nombre: leerTarjeta (número_tarjeta: número)
Responsabilidades: Comprobar que la tarjeta numero_tarjeta es correcta y presentar las opciones disponibles.
Referencias Cruzadas: Funciones del Sistema: R1.2, R1.6, R1.7
Casos de Uso: Reintegro
Notas:
Excepciones: Si la tarjeta es ilegible o no pertenece a los tipos de tarjetas aceptadas, indicar que ha habido un error.
Salida:
Pre-condiciones: No hay una operación activa.
Post-condiciones:

- Una nueva Operación se ha creado. (creación de instancia).
- La Operación se ha asociado con Cajero. (asociación formada).

La descripción de cada apartado de un contrato es como sigue:
Nombre: Nombre de la operación y parámetros.
Responsabilidades: Una descripción informal de las responsabilidades que la operación debe desempeñar.
Referencias Cruzadas: Números de referencia en los requisitos de funciones del sistema, casos de uso, etc.
Notas: Comentarios de diseño, algoritmos, etc.
Excepciones: Casos excepcionales. Situaciones que debemos tener en cuenta que pueden pasar. Se indica también qué se hace cuando ocurre la excepción.
Salida: Salidas que no corresponden a la interfaz de usuario, como mensajes o registros que se envían fuera del sistema. (En la mayor parte de las operaciones del sistema este apartado queda vacío)
Pre-condiciones: Suposiciones acerca del estado del sistema antes de ejecutar la operación.
Post-condiciones: El estado del sistema después de completar la operación.

3.9.2.5.1 Construcción de un Contrato

Los pasos a seguir para construir un contrato son los siguientes:

1. Identificar las operaciones del sistema a partir de los Diagramas de Secuencia del Sistema.
2. Para cada operación del sistema construir un contrato.
3. Empezar escribiendo el apartado de Responsabilidades, describiendo informalmente el propósito de la operación. Este es el apartado más importante del contrato.

4. A continuación rellenar el apartado de Post-condiciones, describiendo declarativamente los cambios de estado que sufren los objetos en el Modelo Conceptual. Puede ser que este apartado quede vacío si no cambia el valor de ningún dato de los maneja el sistema (por ejemplo en una operación del sistema que tan solo se encarga de sacar por pantalla algo al usuario).
5. Para describir las post-condiciones, usar las siguientes categorías:
   - Creación y borrado de instancias.
   - Modificación de atributos.
   - Asociaciones formadas y retiradas.
6. Completar el resto de apartados en su caso.

3.9.2.5.2 Post-condiciones

Las post-condiciones se basan en el Modelo Conceptual, en los cambios que sufren los elementos del mismo una vez se ha realizado la operación.
Es mejor usar el tiempo pasado o el pretérito perfecto al redactar una post-condición, para enfatizar que se trata de declaraciones sobre un cambio en el estado que ya ha pasado. Por ejemplo es mejor decir "se ha creado un nuevo Cliente" que decir "crear un Cliente".
Cuando se ha creado un objeto, lo normal es que se haya asociado a algún otro objeto ya existente, porque si no queda aislado del resto del sistema. Por tanto, al escribir las post-condiciones hay que acordarse de añadir asociaciones a los objetos creados. Olvidar incluir estas asociaciones es el fallo más común cometido al escribir las post-condiciones de un contrato.

3.9.2.6 Diagramas de Estados

Para modelar el comportamiento del sistema pueden usarse los Diagramas de Estados que define UML.

Se puede aplicar un Diagrama de Estados al comportamiento de los siguientes elementos:
- Una clase software.
- Un concepto.
- Un caso de uso.

En la fase de Diseño de Alto Nivel sólo se haría para los dos últimos tipos de elemento, pues una clase software pertenece al Diagrama de Clases de Diseño.
Puesto que el sistema entero puede ser representado por un concepto, también se puede modelar el comportamiento del sistema completo mediante un Diagrama de Estados.
La utilidad de un Diagrama de Estados en esta fase reside en mostrar la secuencia permitida de eventos externos que pueden ser reconocidos y tratados por el sistema. Por ejemplo, no se puede insertar una tarjeta en un cajero automático si se está en el transcurso de una operación.
Para los casos de uso complejos se puede construir un Diagrama de Estados. El Diagrama de Estados del sistema sería una combinación de los diagramas de todos los casos de uso.
El uso de Diagramas de Estados es opcional. Tan solo los usaremos cuando consideremos que nos ayudan a expresar mejor el comportamiento del elemento descrito.

Fase de Construcción: Diseño de Bajo Nivel

En la fase de Diseño de Bajo Nivel se crea una solución a nivel lógico para satisfacer los requisitos, basándose en el conocimiento reunido en la fase de Diseño de Alto Nivel. Los modelos más importantes en esta fase son el Diagrama de Clases de Diseño y los Diagramas de Interacción, que se realizan en paralelo y que definen los elementos que forman parte del sistema orientado a objetos que se va a construir (clases y objetos) y cómo colaboran entre sí para realizar las funciones que se piden al sistema, según éstas se definieron en los contratos de operaciones del sistema.

3.9.3 Actividades

Las actividades que se realizan en la etapa de Diseño de Bajo Nivel son las siguientes:

1. Definir los Casos de Uso Reales.
2. Definir Informes e Interfaz de Usuario.
3. Refinar la Arquitectura del Sistema.
4. Definir los Diagramas de Interacción.
5. Definir el Diagrama de Clases de Diseño. (en paralelo con los Diagramas de Interacción)
6. Definir el Esquema de Base de Datos.

El paso de Refinar la Arquitectura del Sistema no tiene por qué realizarse en la posición 3, puede realizarse antes o después.

3.9.3.1 Casos de Uso Reales

Un Caso de Uso Real describe el diseño real del caso de uso según una tecnología concreta de entrada y de salida y su implementación. Si el caso de uso implica una interfaz de usuario, el caso de uso real incluirá bocetos de las ventanas y detalles de la interacción a bajo nivel con los widgets (botón, lista seleccionable, campo editable, etc.) de la ventana.

Como alternativa a la creación de los Casos de Uso Reales, el desarrollador puede crear bocetos de la interfaz en papel, y dejar los detalles para la fase de implementación.

3.9.3.2 Diagramas de Interacción

Los Diagramas de Interacción muestran el intercambio de mensajes entre instancias del modelo de clases para cumplir las post-condiciones establecidas en un contrato

Hay dos clases de Diagramas de Interacción:

1. Diagramas de Colaboración.
2. Diagramas de Secuencia.

De entre ambos tipos, los Diagramas de Colaboración tienen una mayor expresividad y mayor economía espacial (una interacción compleja puede ser muy larga en un Diagrama de Secuencia), sin embargo en ellos la secuencia de interacción entre objetos es más difícil de seguir que en un Diagrama de Secuencia. Ambos tipos de diagramas expresan la misma información, por lo que se puede usar cualquiera de los dos para expresar la interacción entre los objetos del sistema.

La creación de los Diagramas de Interacción de un sistema es una de las actividades más importantes en el desarrollo orientado a objetos, pues al construirlos se toman unas decisiones clave acerca del funcionamiento del futuro sistema. La creación de estos

diagramas, por tanto, debería ocupar un porcentaje significativo en el esfuerzo dedicado al proyecto entero.

3.9.3.2.1 Creación de Diagramas de Interacción

Para crear los Diagramas de Colaboración o de Secuencia se pueden seguir los siguientes consejos:

• Crear un diagrama separado para cada operación del sistema en desarrollo en el ciclo de desarrollo actual.

- Para cada evento del sistema, hacer un diagrama con él como mensaje inicial.

• Usando los apartados de responsabilidades y de post-condiciones del contrato de operación, y la descripción del caso de uso como punto de partida, diseñar un sistema de objetos que interaccionan para llevar a cabo las tareas requeridas.

• Si el diagrama se complica, dividirlo en dos diagramas más pequeños. Para ello se termina la secuencia de mensajes en un mensaje determinado, y en el segundo diagrama se comienza con el mensaje que terminó el primero. Debe indicarse en el primer diagrama que el resto de la interacción se detalla en el segundo.

El comportamiento dinámico del sistema que se describe en un Diagrama de Interacción debe ser acorde con la estructura estática del sistema que se refleja en el Diagrama de Clases de Diseño. Es aconsejable realizar un Diagrama de Clases de Diseño borrador antes de comenzar con los Diagramas de Interacción. La capacidad de realizar una buena asignación de responsabilidades a los distintos objetos es una habilidad clave, y se va adquiriendo según aumenta la experiencia en el desarrollo orientado a objetos.

Responsabilidad es como un contrato u obligación de una clase o tipo. Las responsabilidades están ligadas a las obligaciones de un objeto en cuanto a su comportamiento. Básicamente, estas responsabilidades pueden ser de tipo Conocer o de tipo Hacer:

• Conocer:
- Conocer datos privados encapsulados.
- Conocer los objetos relacionados.
- Conocer las cosas que puede calcular o derivar.

• Hacer:
- Hacer algo él mismo.
- Iniciar una acción en otros objetos.
- Controlar y coordinar actividades en otros objetos.

Por ejemplo, puedo decir que "un Recibo es responsable de calcular el total" (tipo hacer), o que "una Transacción es responsable de saber su fecha" (tipo conocer). Las responsabilidades de tipo "conocer" se pueden inferir normalmente del Modelo Conceptual.

Una responsabilidad no es lo mismo que un método, pero los métodos se implementan para satisfacer responsabilidades.

3.9.3.3 Diagrama de Clases de Diseño

Un Diagrama de Clases de Diseño muestra la especificación para las clases software de una aplicación. Incluye la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias.

A diferencia del Modelo Conceptual, un Diagrama de Clases de Diseño muestra definiciones de entidades software más que conceptos del mundo real. En la Figura 38 se muestra un ejemplo de Diagrama de Clases de Diseño sencillo.

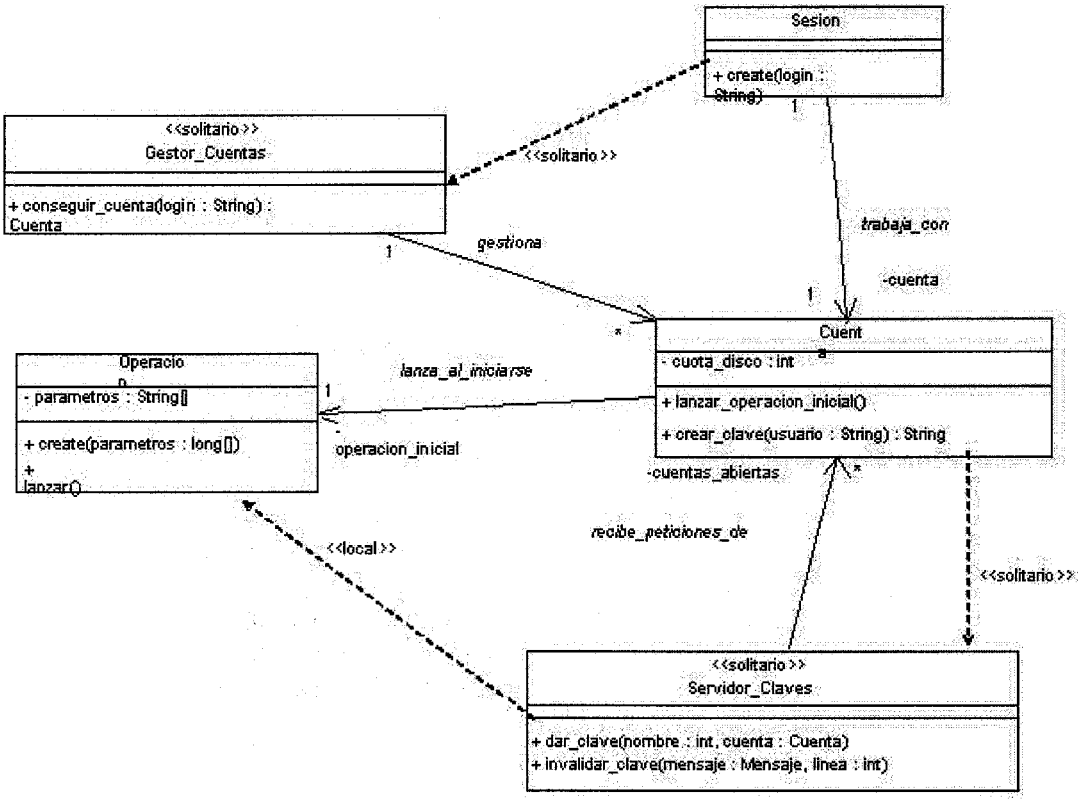


Fig. 38 Ejemplo de Diagrama de Clases de Diseño

3.9.3.3.1 Relaciones de Dependencia para Representar Visibilidad entre Clases

Cuando una clase conoce a otra por un medio que no es a través de un atributo (una asociación con la navegabilidad adecuada), entonces es preciso indicar esta situación por medio de una dependencia.

Un objeto debe conocer a otro para poder llamar a uno de sus métodos, se dice entonces que el primer objeto tiene "visibilidad" sobre el segundo. La visibilidad más directa es

por medio de atributo, cuando hay una asociación entre ambas clases y se puede navegar de la primera a la segunda (un atributo de la primera es un puntero a un objeto de la segunda). Hay otros tres tipos de visibilidad que hay que representar en el diagrama de clases mediante relaciones de dependencia:

- Parámetro: Cuando a un método de una clase se le pasa como parámetro un objeto de otra clase, se dice que la primera tiene visibilidad de parámetro sobre la segunda. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <> (<> en inglés).

- Local: Cuando en un método de una clase se define una variable local que es un objeto de otra clase, se dice que la primera tiene visibilidad local sobre la segunda. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <>.
- Global: Cuando hay una variable global en el sistema, instancia de una clase A, y un método de una clase B llama a un método de A, se dice que la clase B tiene visibilidad global sobre la clase A. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <>.

No es necesario representar la relación de dependencia entre clases que ya están relacionadas por medio de una asociación, que se trata de una "dependencia" más fuerte. Las relaciones de dependencia se incluyen tan solo para conocer qué elementos hay que revisar cuando se realiza un cambio en el diseño de un elemento del sistema.

Solitario: Caso Particular de Visibilidad global

El uso de variables globales no se aconseja por los efectos laterales que se pueden presentar, pero hay un caso en el que sí hay cierta globalidad: las clases que sólo van a tener una instancia. Suele tratarse de clases que se han creado en el Diseño de Bajo Nivel, que no aparecían en el Modelo Conceptual.

Varias clases de nuestro sistema pueden querer llamar a los métodos de la única instancia de una clase de ese tipo, entonces sí se considera que es beneficioso que se pueda acceder a esa instancia como un valor accesible de forma global. Una solución elegante para este caso se implementa mediante un método de clase para obtener esa única instancia (los métodos de clase los permiten algunos lenguajes orientados a objetos, por ejemplo Java), en vez de definirla directamente como una variable global. Para indicar que una clase sólo va a tener una instancia, se etiqueta la clase con el estereotipo <> (<> en inglés), y las relaciones de dependencia entre las clases que la usan y se etiquetan también <> en vez de <>.

A continuación se muestra un ejemplo del código en Java de una clase solitario:

```
public class Solitario {
// se define la instancia como atributo de clase (static)
Solitario static instancia := null;
// método de clase que devuelve la instancia
public static Solitario dar_instancia() {
if (instancia = = null) {
// si no está creada la instancia la crea
instancia := new Solitario();
}
return instancia;
```

```
}
... // otros métodos de la clase
}
```

Cuando otra clase quiere llamar a un método de la instancia incluye el siguiente código:

```
variable Solitario;
variable = Solitario.dar_instancia();
variable.método(); // llamada al método que necesitamos
```

3.9.3.3.2 Construcción de un Diagrama de Clases de Diseño

Normalmente se tiene una idea de un Diagrama de Clases, con una asignación de responsabilidades inicial. En caso de que no se tenga dicho Diagrama de Clases Borrador, puede seguirse la siguiente estrategia:

1. Identificar todas las clases participantes en la solución software. Esto se lleva a cabo analizando los Diagramas de Interacción.
2. Representarlas en un diagrama de clases.
3. Duplicar los atributos que aparezcan en los conceptos asociados del Modelo Conceptual.
4. Añadir los métodos, según aparecen en los Diagramas de Interacción.
5. Añadir información de tipo a los atributos y métodos.
6. Añadir las asociaciones necesarias para soportar la visibilidad de atributos requerida.
7. Añadir flechas de navegabilidad a las asociaciones para indicar la dirección de visibilidad de los atributos.
8. Añadir relaciones de dependencia para indicar visibilidad no correspondiente a atributos.

Algunos de estos pasos se van realizando según se vayan completando los Diagramas de Interacción correspondientes. No existe precedencia entre la realización del Diagrama de Clases de Diseño y los Diagramas de Interacción. Ambos tipos de diagramas se realizan en paralelo, y unas veces se trabaja primero más en el de clases y otras veces se trabaja primero más en los de interacción.

No todas las clases que aparecían en el Modelo Conceptual tienen por qué aparecer en el Diagrama de Clases de Diseño. De hecho, tan solo se incluirán aquellas clases que tengan interés en cuanto a que se les ha asignado algún tipo de responsabilidad en el diseño del sistema. No hay, por tanto, una transición directa entre el Modelo Conceptual y el Diagrama de Clases de Diseño, debido a que ambos se basan en enfoques completamente distintos: el primero en comprensión de un dominio, y el segundo en una solución software.

En el Diagrama de Clases de Diseño se añaden los detalles referentes al lenguaje de programación que se vaya a usar. Por ejemplo, los tipos de los atributos y parámetros se expresarán en el lenguaje de implementación escogido.

3.9.3.3.3 Navegabilidad

La navegabilidad es una propiedad de un rol (un extremo de una asociación) que indica que es posible "navegar" unidireccionalmente a través de la asociación, desde objetos de

la clase origen a objetos de la clase destino. Como se vio en la parte II, se representa en UML mediante una flecha.
La navegabilidad implica visibilidad, normalmente visibilidad por medio de un atributo en la clase origen. En la implementación se traducirá en la clase origen como un atributo que sea una referencia a la clase destino.
Las asociaciones que aparezcan en el Diagrama de Clases deben cumplir una función, deben ser necesarias, si no es así deben eliminarse.
Las situaciones más comunes en las que parece que se necesita definir una asociación con navegabilidad de A a B son:

- A envía un mensaje a B.
- A crea una instancia B.
- A necesita mantener una conexión con B.

3.9.3.3.4 Visibilidad de Atributos y Métodos

Los atributos y los métodos deben tener una visibilidad asignada, que puede ser:
+ Visibilidad pública.
# Visibilidad protegida.
- Visibilidad privada.
También puede ser necesario incluir valores por defecto, y todos los detalles ya cercanos a la implementación que sean necesarios para completar el Diagrama de Clases.

3.9.3.4 Otros Aspectos en el Diseño del Sistema

En los puntos anteriores se ha hablado de decisiones de diseño a un nivel de granularidad fina, con las clases y objetos como unidades de decisión. En el diseño de un sistema es necesario también tomar decisiones a un nivel más alto sobre la descomposición de un sistema en subsistemas y sobre la arquitectura del sistema. Esta parte del Diseño de Bajo Nivel es lo que se denomina Diseño del Sistema. Estas decisiones no se toman de forma distinta en un desarrollo orientado a objetos a como se llevan a cabo en un desarrollo tradicional. Por tanto, no se va a entrar en este documento en cómo se realiza esta actividad.
Sí hay que tener en cuenta que las posibles divisiones en subsistemas tienen que hacerse en base a las clases definidas en el Diagrama de Clases del Diseño.

Fases de Implementación y Pruebas.
Una vez se tiene completo el Diagrama de Clases de Diseño, se pasa a la implementación en el lenguaje de programación elegido.
El programa obtenido se depura y prueba, y ya se tiene una parte del sistema funcionando que se puede probar con los futuros usuarios, e incluso poner en producción si se ha planificado una instalación gradual.
Una vez se tiene una versión estable se pasa al siguiente ciclo de desarrollo para incrementar el sistema con los casos de uso asignados a tal ciclo.

CAPÍTULO 4

Desarrollo de la Aplicación.

4.1 Definición del problema

En el Laboratorio de Oceanografía Física del Instituto de Ciencias del Mar y Limnología de la UNAM, se cuenta con un registro sistemático de imágenes de satélite (NOAA-12 y NOAA-14) de la Temperatura Superficial del Mar (TSM); este registro esta almacenado en una Base de Datos (BITSMEX), el contar con la información organizada permite agilizar la búsqueda para conocer el estado actual del acervo de imágenes. Además de poner a disposición del publico esta valiosa información mediante el uso de la red Internet.

Con este acervo acumulativo de información se podrían hacer análisis locales y regionales para identificar y estudiar procesos climáticos. Tales estudios sentarían bases sólidas para actualizar, sólo por mencionar una aplicación relevante, modelos de clima regional para nuestro país con los que se podrían hacer pronósticos cada vez mas acertados de la distribución y el volumen de lluvia en las diversas regiones agrícolas de México.

El objetivo es desarrollar un sistema que permita procesar la información contenida en imágenes satelitales. Los requerimientos permitirán definir rutinas para procesar la información de las imágenes satelitales de la temperatura superficial del mar de los mares mexicanos y aguas internacionales adyacentes.

4.2 Requerimientos

La aplicación Web, además de contar con una interfaz bien estructurada, deberá ser capaz de procesar la información contenida en las imágenes satelitales; para este fin el usuario podrá hacer uso de herramientas tales como:

- Obtención del histograma de la imagen satelital.
- Obtención de la tabla de temperaturas proveniente de un transecto especificado por el usuario.
- Obtención de la temperatura presente en un punto de la imagen, así como la georeferenciación de dicho punto.
- Realización de recortes.
- Aplicación de zoom.
- Interpolación o decimación de imágenes.
- Realce de una imagen.
- Binarización de una imagen.
- Segmentación de una imagen.

Además a esta aplicación, deberán ser adheridas a estas, herramientas que permitan procesamientos convencionales de imágenes, así como también todo aquello que se considere de utilidad para que el objetivo se lleve acabo de manera eficaz.

4.3 Casos de uso.

A continuación se hace una descripción de los casos de uso que se desprenden de los requerimientos, planteando los posibles escenarios que podrían presentarse (necesidades del usuario) al hacer uso de la aplicación Web y considerando todas las herramientas que serán útiles para el buen funcionamiento del sistema.

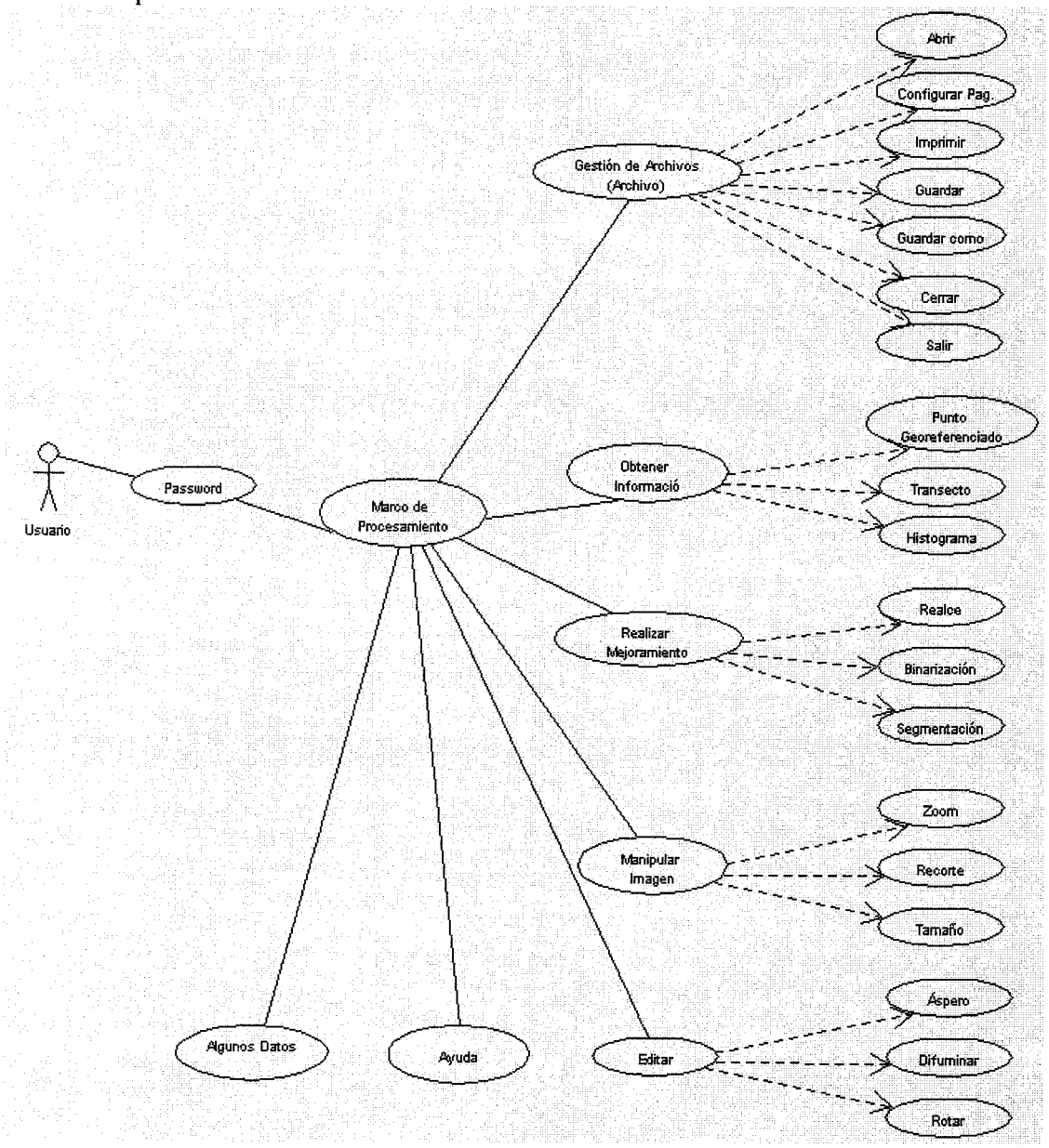
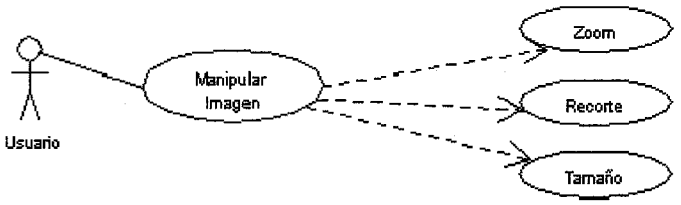


Diagrama general de los Casos de Uso.

4.3.1 Caso de uso Manipular Imagen.



Breve Descripción

El caso de uso *Manipular Imagen* permite a un usuario del Sistema aplicar un Zoom a una imagen, realizar un Recorte o modificar el Tamaño de la imagen.

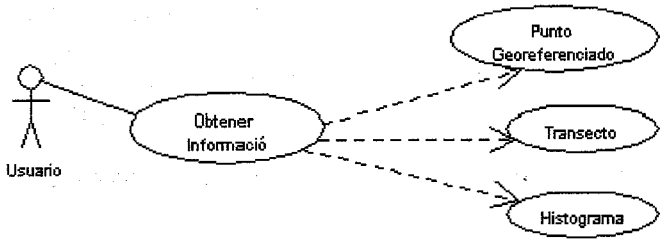
Descripción paso por paso.

1. El usuario especifica la imagen que va a ser manipulada.

2. El usuario puede realizar alguna de las siguientes acciones:
   - 2.1 Aplicar un Zoom a la imagen. El Zoom puede ser Zoom In o bien Zoom Out para lo cual:
     a) El usuario especifica el tipo de Zoom deseado y el porcentaje en el que este se realizará.
     b) El sistema responde con la visualización de la imagen con el porcentaje de Zoom especificado.
   - 2.2 Para realizar un recorte:
     a) El sistema responde transformando el cursor dentro del área de la imagen.
     b) El usuario especifica mediante un trazo rectangular la región que desea recortar.
     c) El sistema responde separando la región seleccionada de la imagen original.
     d) La región recortada podrá ser guardada como una imagen nueva.
   - 2.3 Para modificar el tamaño de una imagen:
     a) El usuario especifica el factor en que será modificado el tamaño de la imagen original (1/4, 1/2, 0, 2, 4).
     b) Si el porcentaje especificado es 2 o 4 el Sistema identificará que corresponde a una interpolación y si es 1/2 0 1/4 menor corresponde a una decimación.
     c) El Sistema entonces responderá con la modificación del tamaño de la imagen según el factor especificado.

3. Los cambios realizados por el usuario pueden ser guardados e impresos.

4.3.2 Caso de uso Obtener Información.



Breve Descripción

El caso de uso *Obtener Información* permite a un usuario del Sistema visualizar el histograma de una imagen, conocer la temperatura en un punto de la imagen o bien la temperatura el un conjunto lineal de puntos sobre la imagen, georeferenciando los puntos involucrados.

Descripción paso por paso.

1. El usuario especifica la imagen de la cual se extraerá la información.

2. El usuario puede solicitar información de los siguientes tipos:
   2.1 Visualizar el Histograma de la imagen:
      a) El usuario selecciona la visualización del histograma correspondiente a la imagen especificada.
      b) El sistema responde con la gráfica del histograma correspondiente a la imagen.
      c) La gráfica del Histograma mostrado puede ser guardada o impresa si el usuario así lo desea.
   2.2 Obtener la temperatura de un punto en especifico:
      a) El sistema responde transformando el cursor dentro del área de la imagen.
      b) El sistema responde de manera simultánea con una etiqueta sobre el cursor la cual muestra la temperatura correspondiente a dicho punto, y su georeferanciación.
   2.3 Obtener la temperatura de los puntos que se sitúan a lo largo de una línea definida por dos puntos (P_1 y P_2):
      c) El sistema responde transformando el cursor dentro del área de la imagen.
      d) Con el nuevo cursor el usuario puede trazar una línea recta entre dos puntos de interés dentro del área de la imagen.
      e) El sistema responde mostrando una tabla la cual especifica la temperatura en cada punto de la línea trazada.
      f) La tabla mostrada puede ser impresa si el usuario así lo desea.

4.3.3 Caso de uso Realizar Mejoramiento.

Breve Descripción

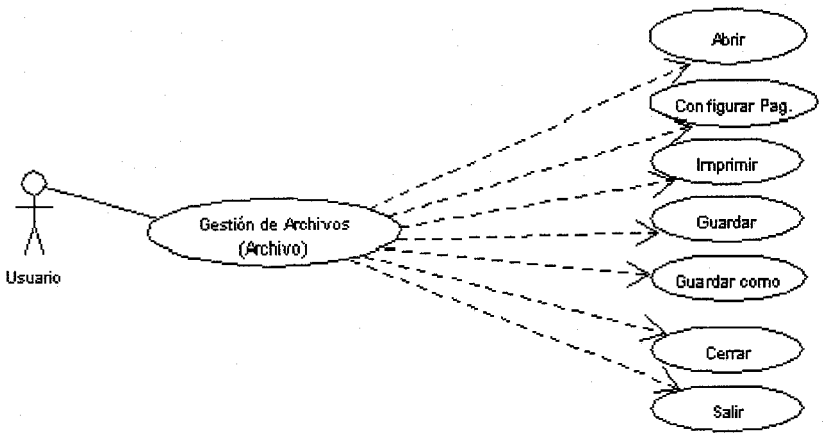
El caso de uso *Realizar Mejoramiento* permite a un usuario del Sistema operar imágenes para obtener una mejor visualización u obtener transformaciones de esta y partes de interés mediante el Realce, la binarización y la Segmentación.

Descripción paso por paso.

1. El usuario especifica la imagen a la cual se le realizaran las operaciones.

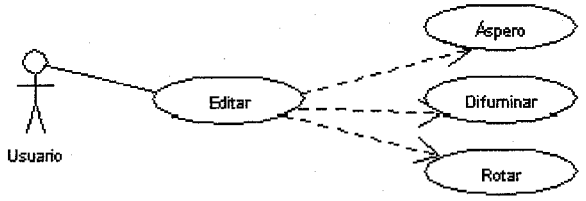
2. El usuario puede hacer uso de las siguientes herramientas:
   - 2.1 Al aplicar un realce:
     - 2.1.1 El usuario indica que desea realizar un realce.
     - 2.1.2 El sistema responde mostrando una nueva imagen como resultado de la aplicación del realce.
     - 2.1.3 El usuario puede guardar o imprimir la nueva imagen realzada.
   - 2.2 Efectuar una binarización:
     - 2.2.1 El usuario indica la aplicación de una binarización a la imagen.
     - 2.2.2 El sistema responde con la nueva imagen correspondiente a la binarización
     - 2.2.3 El usuario puede guardar o imprimir la nueva imagen binarizada.
   - 2.3 Efectuar una segmentación:
     - 2.3.1 El usuario indica la aplicación de una segmentación a la imagen.
     - 2.3.2 El sistema responde con la nueva imagen correspondiente a la segmentación.
     - 2.3.3 El usuario puede guardar o imprimir la nueva imagen segmentada.

4.3.4 Caso de uso Gestión de Archivos.



Breve Descripción
El caso de uso *Archivos* permite a un usuario del Sistema hacer gestión de archivos.
Descripción paso por paso. 1. La gestión de archivos incluye:: 1.1 Abrir un archivo de imagen (*.gif, *.jpeg, *jpg). 1.2 Configuración de la hoja de impresión. 1.3 Impresión del contenido de los archivos de imagen. 1.4 Guardar archivos. 1.5 Guardar archivos o imágenes procesadas con distinto nombre. 1.5 Cerrar archivos. 1.6 Salir del sistema

4.3.4 Caso de uso Editar.



Breve Descripción

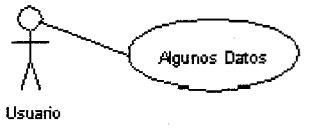
El caso de uso *Editar* permite a un usuario del Sistema operar imágenes para obtener transformaciones de esta y partes de interés mediante la difuminación, la transformación áspera y la rotación de esta.

Descripción paso por paso.

1. El usuario especifica la imagen a la cual se le realizaran las transformaciones.

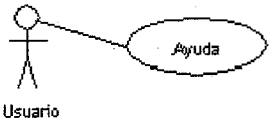
2. El usuario puede hacer uso de las siguientes herramientas:
   - 2.1 Al aplicar una apariencia áspera:
     - 2.1.1 El usuario indica que desea aplicar una transformación áspera.
     - 2.1.2 El sistema responde mostrando una nueva imagen como resultado de la transformación áspera.
     - 2.1.3 El usuario puede guardar o imprimir la nueva imagen ásperada.
   - 2.2 Efectuar una difuminación:
     - 2.2.1 El usuario indica la aplicación de una difuminación a la imagen.
     - 2.2.2 El sistema responde con la nueva imagen correspondiente a la difuminación.
     - 2.2.3 El usuario puede guardar o imprimir la nueva imagen difuminada.
   - 2.3 Efectuar una rotación:
     - 2.3.1 El usuario indica la aplicación de una rotación proporcionando el ángulo.
     - 2.3.2 El sistema responde con la nueva imagen rotada según el ángulo especificado.
     - 2.3.3 El usuario puede guardar o imprimir la nueva imagen rotada.

4.3.5 Caso de uso Algunos Datos.



Breve Descripción
El caso de uso *Algunos Datos,* permite a un usuario del Sistema conocer información sobre el desarrollo del sistema y algunos datos bibliográficos.
Descripción paso por paso. 1. El usuario especifica que desea observar la ventana de Algunos Datos. 2. El usuario puede hacer lectura de la información presentada por el sistema y los datos Bibliográficos contenidos.

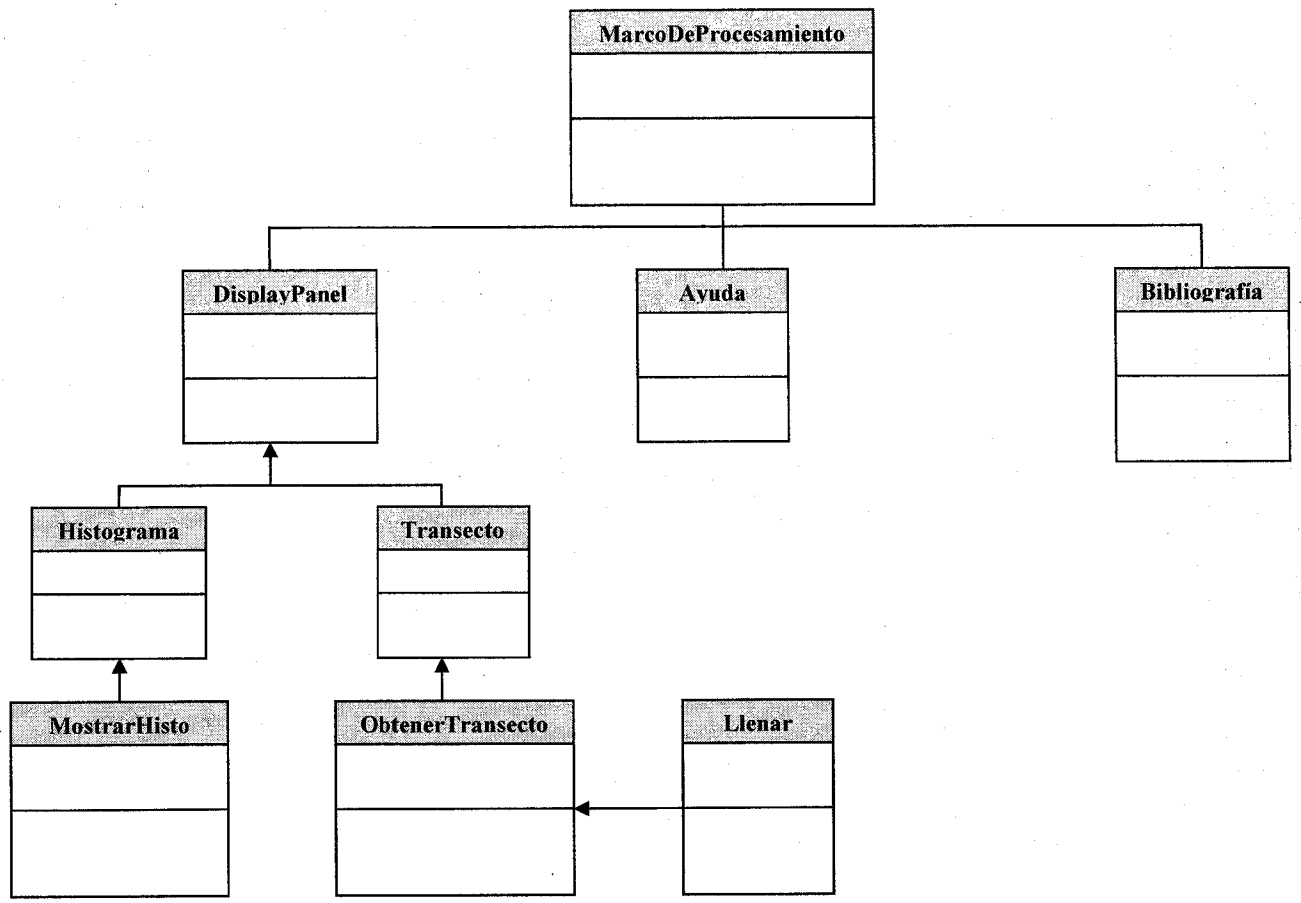
4.3.6 Caso de uso Ayuda.



Breve Descripción
El caso de uso *Ayuda,* permite a un usuario del Sistema obtener ayuda mediante la presentación del algunos temas de ayuda.
Descripción paso por paso. 1. El usuario especifica la que desea observar la ventana de ayuda. 2. El usuario puede observar los siguientes temas de ayuda: 2.1 Abrir: este tema contiene documentos que describen el funcionamiento de los componentes del menú Abrir en la barra de Menú. 2.2 Editar: este tema contiene documentos que describen el funcionamiento de los componentes del menú Editar en la barra de Menú. 2.3 Herramientas: este tema contiene documentos que describen el funcionamiento de los componentes del menú Herramientas en la barra de Menú.

4.4 Diagrama de Clases.

A continuación se presenta el diagrama de clases diseñado para llevar acabo el desarrollo de la programación de la aplicación



4.4.1 Atributos y métodos de las Clases

MarcoDeProcesamiento
JMenu menuArchivo, menuEditar, rotar, menuBarras, menuAyuda **JMenuBar** menuBar **JMenuItem** elementoAbrir, configPag, imprimir, elementoGuardar, elementoCerrar, salir, blurItem, sharpenItem, brillo, edgeDetectItem, negativo, noventaItem, cochentaItem, dsetentaItem, ObInformacion, ObManipulacion, ObMejoramiento, textos, algunos **JButton** anterior, posterior **JScrollPane** sp, sp2, sp3 **Border** brd, brd2, brd3 **BufferedImage** mBufferedImage **File** fFile **Container** contentPane **DisplayPanel** displayPanel **JToolBar** bar, bar2, bar3 **JLabel** regla11, 21 **JButton** boton1, boton2, boton3, boton21, boton22, boton23, boton31, boton34, boton32, boton33 **Icon** punto, transecto, histograma, recortar, modiftamaño, zoom, realce1, realce2, binarización, segmentación **String** tamaños, mensaje **int** bandera, inicio, res
MarcoDeProcesamiento() centrar() actionPerformed() removeScrollPane() ObInf() ObManip() ObMejor()

DisplayPanel
BufferedImage mBufferedImage, BufferOriginal, BufferAnterior, BufferPosterior, BufferDeRecorte, BufferDePunto, BufferDeTransecto, BufferDeTemperatura, BufferDeZoom, BufferAnterior5, BufferAnterior4, BufferAnterior3, BufferAnterior2, BufferAnterior1, BufferPosterior5, BufferPosterior4, BufferPosterior3, BufferPosterior2, BufferPosterior1. **int** posX1, posY1, posX2, posY2, puntoX, puntoY, longitud, fuera, recorte, puntogeoref, transecto, ratonsuelto, puntoselec, pintainicio, recorterealizado, returnVal, factor, valorDePunto, coordenada, diferencia, aproximado, escalar, prefImp, lati, longi, eszoom, haciendoZoom, regionDeZoom. **double** pospixelX1, pospixelY1, pospixelX2, pospixelY2, m, b, pixelX, pixelY,ancho, alto, temperature, multdiv. **float** a, b. **String** nombre, nombreNuevo; **JFileChooser** eleccion; **Image** image; **Graphics** g; **PageFormat** pf; **PrinterJob** job;

DisplayPanel()
loadImage();
createBufferedImage();
addMouseMotionListener();
addMouseListener();
guardarImagen()
guardarComoImagen()
paintComponent()
binarización()
difuminar()
áspero()
segmentación()
masrealzar()
menosrealzar()
rotar()
convolve()
filter()
recortar()
puntogeo()
transecto()
mtamaño()
zoomin()
zoomout()
verHistograma()
configurarPagina()
imprimir()
anterior()
posterior()

Histograma
Image imagenNueva; **BufferedImage** bufferFuente, BufferedImage bufer; **MostrarHisto** mostrar; **JScrollPane** jsp **Border** brd **JButton** botonImprimir, botonGuardar
Histograma() centrar()

MostrarHisto
BufferedImage buffer, BufferDeHistograma, BufferDeEscala. **int** contador, i, j, k, l, m, n, posición, aproximado, diferencia, coordenada, returnVal **int[]** paleta = new int[258]; tinferiorX = {650, 650, 660}; tinferiorY = {495, 505, 500}; tsuperiorX = {95, 100, 105}; tsuperiorY = {100, 90, 100}; **double** escala, función; **String** nombreNuevo; **MediaTracker** m **Graphics** g **PrinterJob** pj **JFileChooser** eleccion
MostrarHisto() paintComponent() conteo() imprimir() guardarComoImagen()

Transecto
BufferedImage BufferDeTemperatura **PagingModel** pm; **JTable** table; **Image** temperatura **MediaTracker** t **JScrollPane** jsp **Border** brd **JButton** botonImprimir
Transecto() centrar()

ObtenerTransecto
int pageSize, pageOffset, numRows, size I, j, k, aproximado, diferencia; **int[]** valores **Record[]** data; **BufferedImage** Transecto;
ObtenerTransecto() getRowCount() getColumnName() getColumnCount()

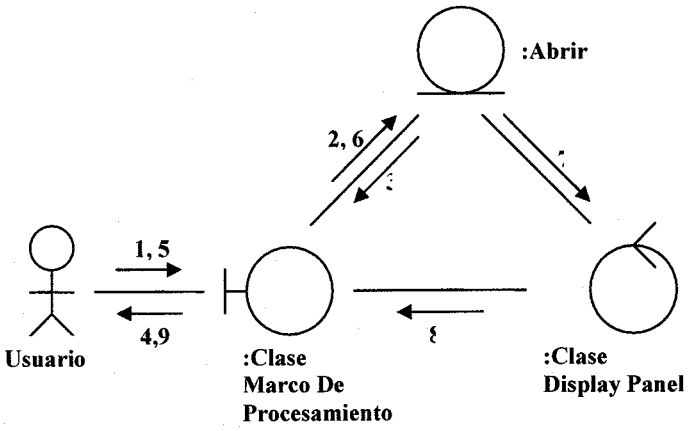
Llenar
String[] headers, data **int** counter; **int[]** valor
Llenar( ) getColumnName() getColumnCount() getValueAt()

Ayuda
JEditorPane htmlPane; **JTree** tree; **URL** helpURL; **String** lineStyle **JPanel** panel **DefaultMutableTreeNode** top **JScrollPane** htmlView **Dimension** minimumSize **DefaultMutableTreeNode** category, book, top
centrar() createNodes() displayURL() initHelp() toString() BookInfo() valueChanged()

Bibliografía
JPanel panel **Border** brd **JTextArea** texto **JScrollPane** Info **String** información
Bibliografía() centrar()

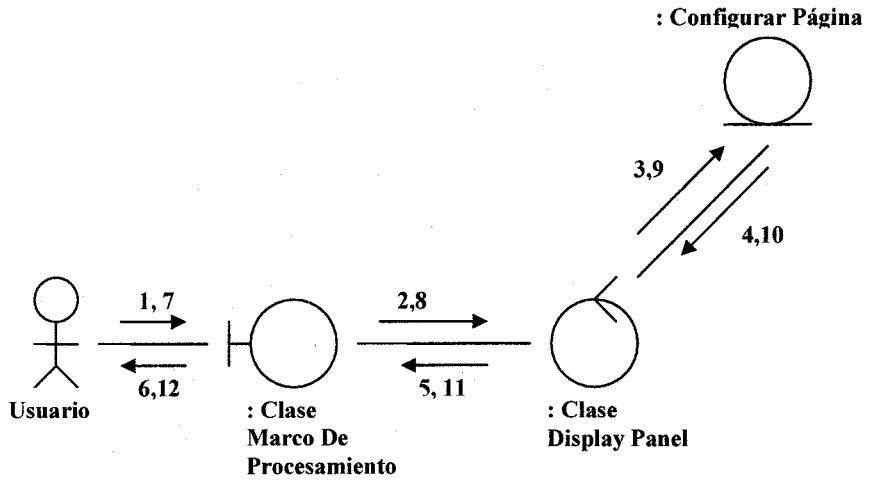
4.5 Diagramas de Colaboración obtenidos para cada uno de los Casos de Uso.

4.5.1 Diagrama de colaboración para el caso de uso particular <u>Abrir</u> a través de la Gestión de Archivos.



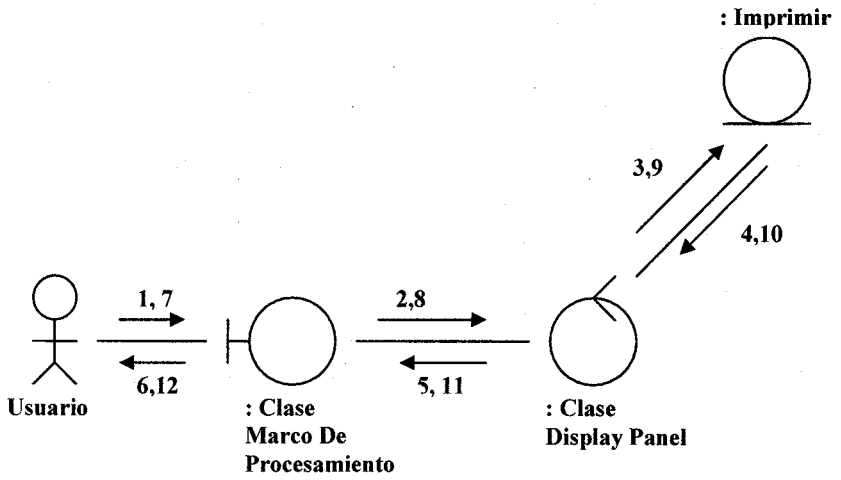
1: Solicitar la apertura de un archivo.
2: Solicitud para mostrar ventana de dialogo para apertura de archivo.
3: Ventana mostrada.
4: El usuario visualiza la ventana y selecciona un archivo.
5: Archivo seleccionado.
6: Datos del archivo seleccionado se reciben.
7: Información de archivo seleccionado descargada.
8: Información (imagen) enviada al marco para su despliegue.
9: El usuario observa la imagen seleccionada.

4.5.2 Diagrama de colaboración para el caso de uso particular <u>Configurar Página</u> a través de la Gestión de Archivos.



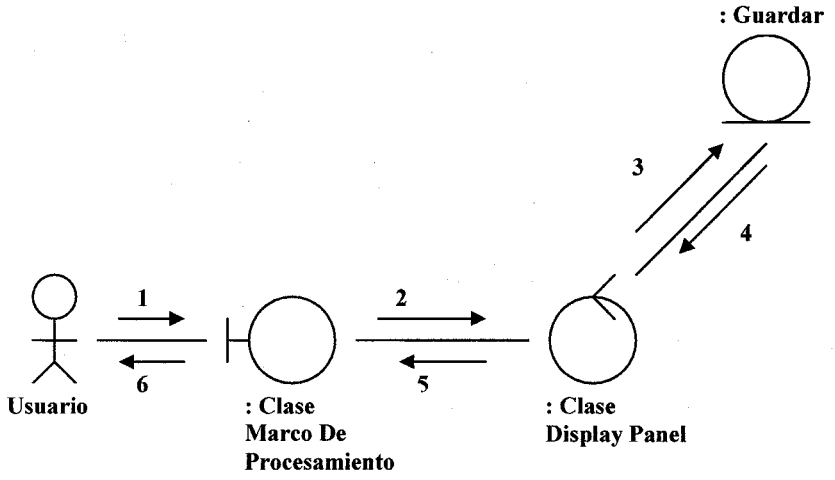
1: Solicitar la configuración de la hoja de impresión.
2: Transferir la solicitud para la configuración.
3: Solicitud para mostrar ventana de dialogo para la configuración.
3: Ventana mostrada.
4: Apertura realizada.
5: Ventana visualizada.
6: El usuario observa la ventana.
7: El usuario realiza la configuración.
8: Datos de la configuración se reciben.
9: Información de la nueva configuración.
10: Conclusión satisfactoria.
11: Información de conclusión satisfactoria.
12: El usuario puede continuar trabajando.

4.5.3 Diagrama de colaboración para el caso de uso particular <u>Imprimir</u> a través de la Gestión de Archivos.



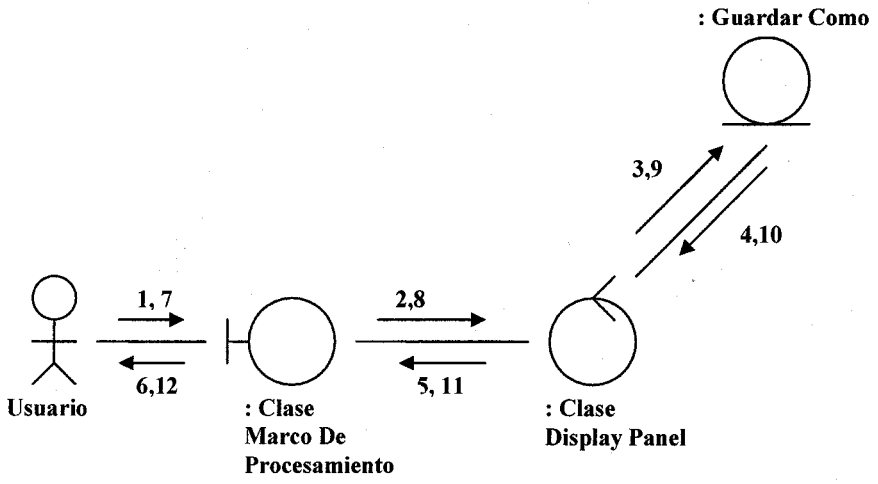
1: Solicitar la impresión de la imagen actual.
2: Transferir la solicitud para la impresión.
3: Solicitud para mostrar ventana de dialogo para la impresión.
4: Apertura de la ventana realizada.
5: Ventana visualizada.
6: El usuario observa la ventana.
7: El usuario realiza la configuración y ordena la impresión.
8: Datos de la configuración para la impresión se reciben.
9: Información para realizar la impresión.
10: Conclusión satisfactoria de el envío de datos a la impresora.
11: Información de conclusión de impresión satisfactoria.
12: El usuario puede continuar trabajando.

4.5.4 Diagrama de colaboración para el caso de uso particular <u>Guardar</u> a través de la Gestión de Archivos.



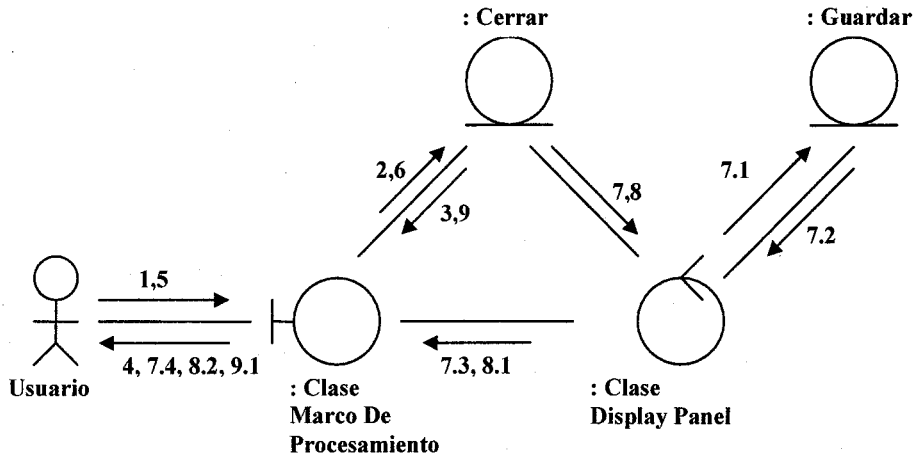
1: Solicitar guardar la imagen actual.
2: Transferir la solicitud para guardar.
3: Solicitud para guardar la imagen actual.
4: Operación de guardar realizada.
5: Información de conclusión satisfactoria.
6: El usuario puede continuar trabajando.

4.5.5 Diagrama de colaboración para el caso de uso particular <u>Guardar Como</u> a través de la Gestión de Archivos.



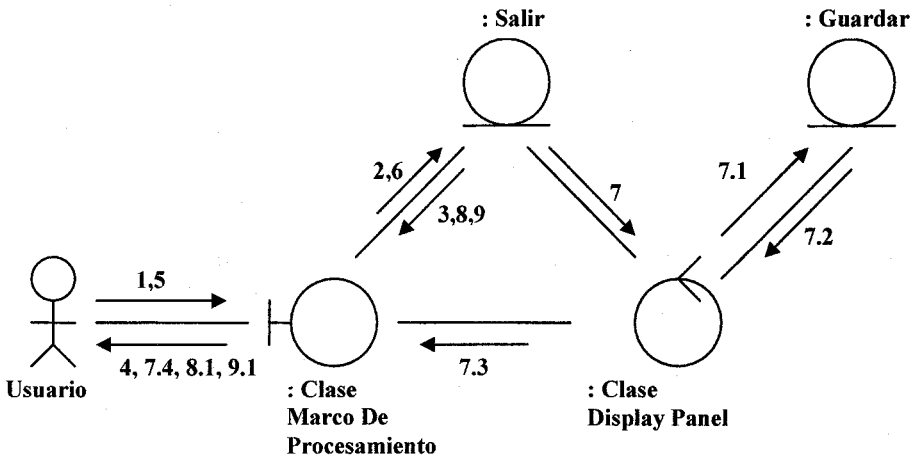
1: Solicitar guardar la imagen actual con nuevo nombre.
2: Transferir la solicitud para guardar.
3: Solicitud para mostrar ventana de dialogo para guardar.
4: Apertura de la ventana realizada.
5: Ventana visualizada.
6: El usuario observa la ventana.
7: El usuario aporta en nuevo nombre y ejecuta la acción.
8: Nuevo nombre recibido.
9: Información para guardar con el nuevo nombre.
10: Conclusión satisfactoria de la operación.
11: Información de conclusión satisfactoria.
12: El usuario puede continuar trabajando.

4.5.6 Diagrama de colaboración para el caso de uso particular Cerrar a través de la Gestión de Archivos.



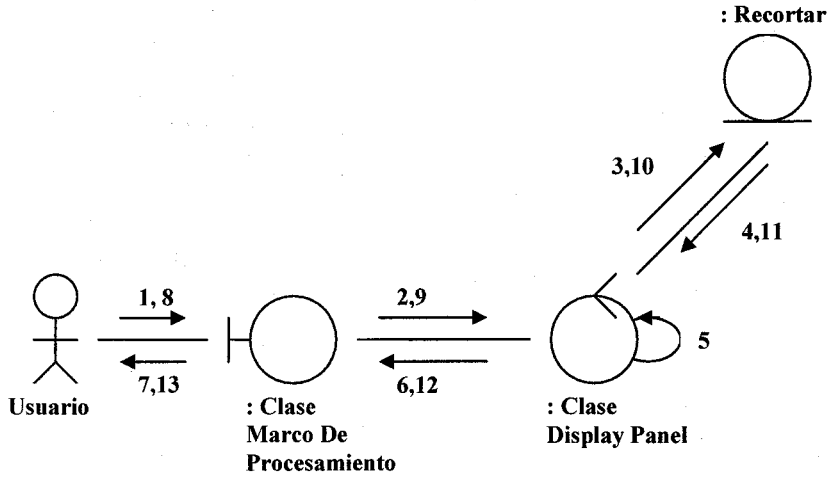
1: Solicitar el cierre de la imagen actual.
2: Solicitud de cierre.
3: Mostrar mensaje de guardar antes de cerrar.
4: El usuario visualiza el mensaje.
5: Opción seleccionada.
6: La opción seleccionada se transfiere.
7: La opción es guardar.
7.1: Se informa que se debe guardar la imagen.
7.2: La imagen ha sido guardada satisfactoriamente.
7.3: Informar que ahora ya se puede limpiar el marco.
7.4: El usuario observa el marco limpio y elementos del menú bloqueados.
8: La opción es no guardar.
8.1: Informar que ahora ya se puede limpiar el marco.
8.2: El usuario observa el marco limpio y elementos del menú bloqueados.
9: La opción es cancelar.
9.1: El usuario observa el marco sin cambios.

4.5.7 Diagrama de colaboración para el caso de uso particular <u>Salir</u> a través de la Gestión de Archivos.



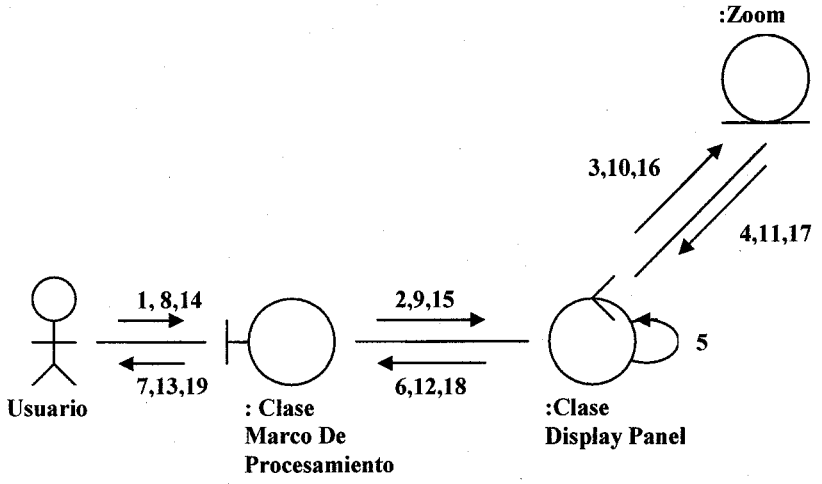
1: Solicitar la salida del sistema.
2: Solicitud de salida.
3: Mostrar mensaje de guardar antes de salir.
4: El usuario visualiza el mensaje.
5: Opción seleccionada.
6: La opción seleccionada se transfiere.
7: La opción es guardar.
7.1: Se informa que se debe guardar la imagen.
7.2: La imagen ha sido guardada satisfactoriamente.
7.3: Informar que ahora ya se puede salir del sistema.
7.4: El usuario ha salido del sistema.
8: La opción es no guardar.
8.1: El usuario ha salido del sistema.
9: La opción es cancelar.
9.1: El usuario observa el marco sin cambios.

4.5.8 Diagrama de colaboración para el caso de uso particular <u>Recortar</u> a través de Manipular una Imagen.



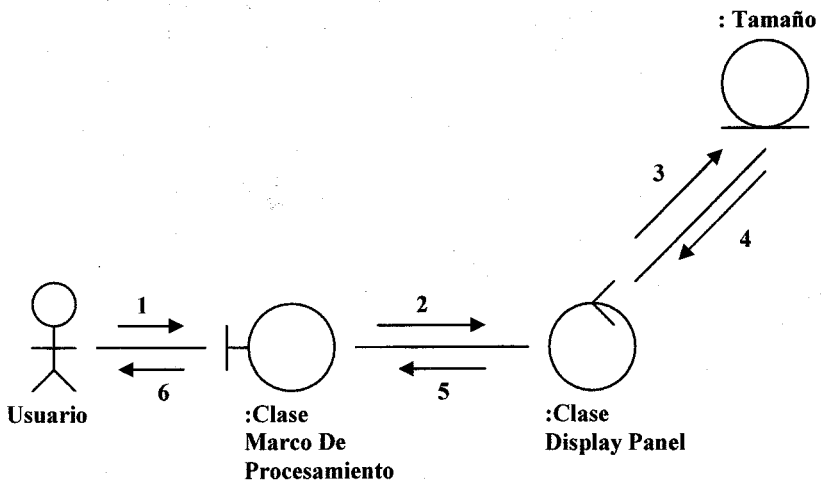
1: Solicitar realizar un recorte a la imagen actual.
2: Transferir la solicitud para recortar.
3: Solicitud para recortar establecida.
4: Disponibilidad de la imagen para trazar recorte.
5: Transformación de apuntador.
6: Disposición de trazo de región en la imagen.
7: El usuario observa en nuevo apuntador dentro de la región de la imagen.
8: El usuario realiza el trazo de la región a recortar.
9: Informar que se ha seleccionado una región.
10: Información del área a recortar enviada.
11: Conclusión satisfactoria de la operación de recorte.
12: Recorte como nueva imagen a mostrar.
13: El usuario observa la región seleccionada como la nueva imagen en el marco.

4.5.9 Diagrama de colaboración para el caso de uso particular <u>Zoom</u> a través de Manipular una Imagen.



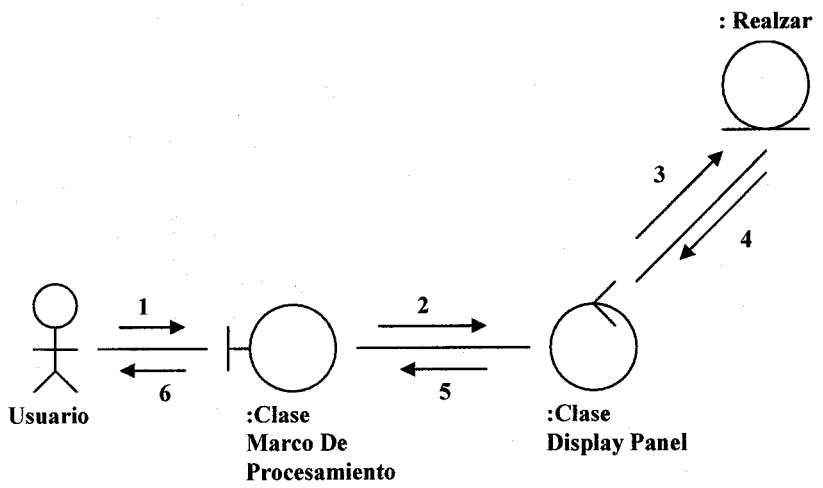
1: Solicitar realizar un zoom a la imagen actual.
2: Transferir la solicitud para realizar el zoom.
3: Solicitud para realizar zoom establecida.
4: Disponibilidad de la imagen para trazar el área donde se desea realizar el zoom.
5: Transformación de apuntador.
6: Disposición de trazo de la región en la imagen.
7: El usuario observa en nuevo apuntador dentro de la región de la imagen.
8: El usuario realiza el trazo de la región en la que desea realizar el zoom.
9: Informar que se ha seleccionado una región.
10: Información del área seleccionada enviada.
11: Conclusión satisfactoria de la operación.
12: Región seleccionada como nueva imagen a mostrar.
13: El usuario observa la región seleccionada como la nueva imagen en el marco.
14: El usuario entonces indica la magnitud del zoom que desea observar.
15: Solicitud de magnitud de zoom transferida.
16: Solicitud de magnitud de zoom informada.
17: Solicitud ejecutada.
18. Nueva imagen con el zoom realizado se transfiere al marco.
19. El usuario observa la imagen con el zoom solicitado.

4.5.10 Diagrama de colaboración para el caso de uso particular Tamaño, a través de Manipular una Imagen.



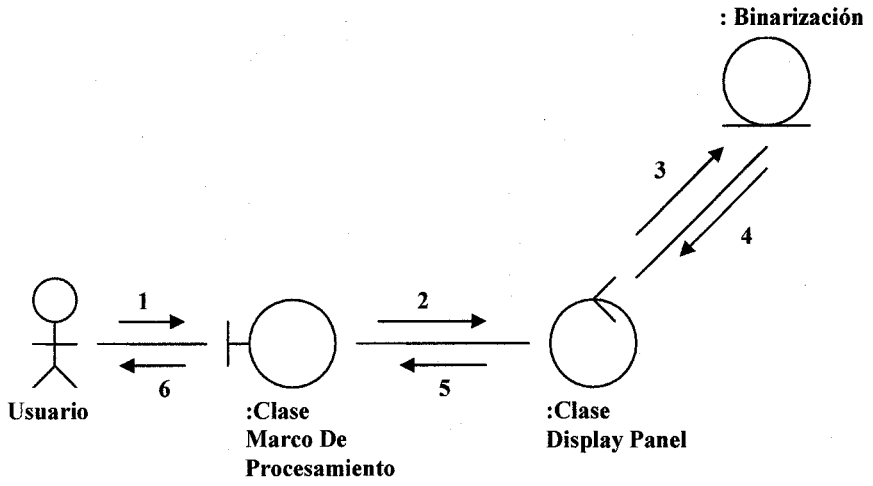
1: Solicitar guardar cambiar el tamaño de la imagen a un factor seleccionado.
2: Transferir la solicitud para modificar el tamaño de la imagen.
3: Solicitud para modificar el tamaño de la imagen a un factor ya proporcionado.
4: Imagen escalada al factor proporcionado.
5: Imagen escalada es enviada al marco para ser visualizada.
6: La imagen escalada al factor seleccionado es visualizada por el usuario.

4.5.11 Diagrama de colaboración para el caso de uso particular <u>Realzar,</u> a través de Mejorar una Imagen.



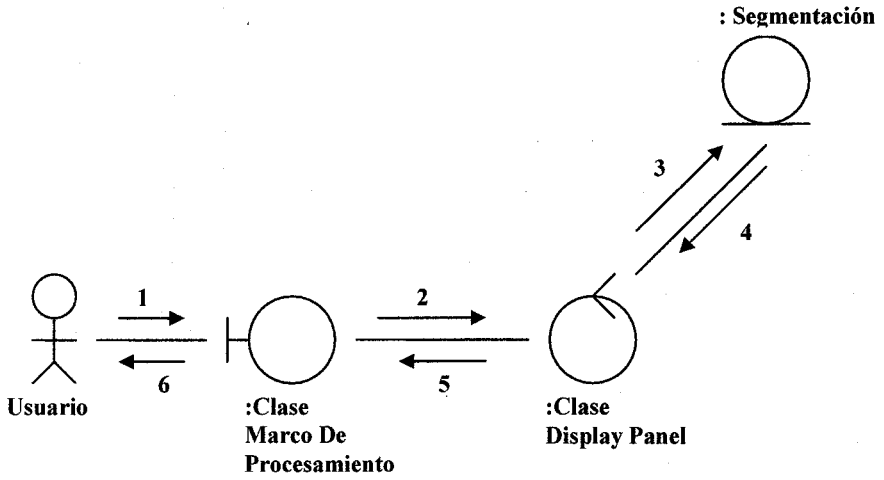
1: Solicitar aplicar un realce a la imagen.
2: Transferir la solicitud para el realce de la imagen.
3: Solicitud para realzar la imagen.
4: Imagen realzada.
5: Imagen realzada es enviada al marco para ser visualizada.
6: La imagen realzada es visualizada por el usuario.

4.5.12 Diagrama de colaboración para el caso de uso particular <u>Binarización,</u> a través de Mejorar una Imagen.



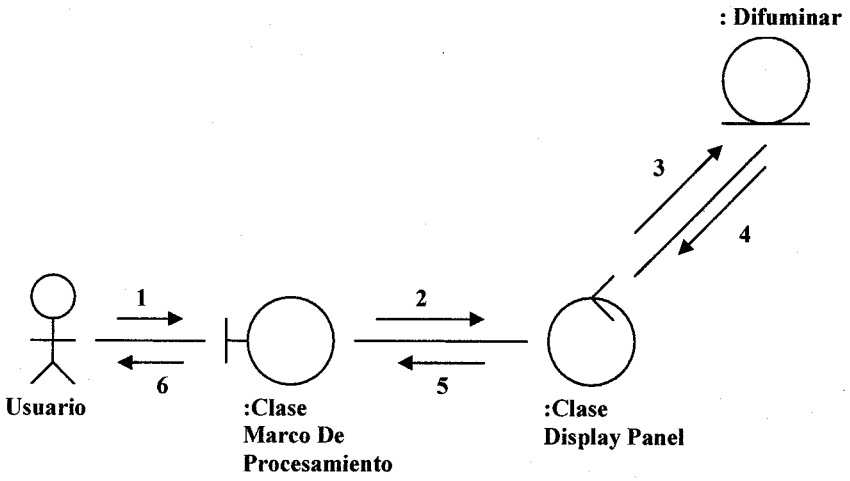
1: Solicitar aplicar una binarización a la imagen.
2: Transferir la solicitud para la binarización de la imagen.
3: Solicitud para binarizar la imagen.
4: Imagen binarizada.
5: La imagen binarizada es enviada al marco para ser visualizada.
6: La imagen binarizada es visualizada por el usuario.

4.5.13 Diagrama de colaboración para el caso de uso particular Segmentación a través de Mejorar una Imagen.



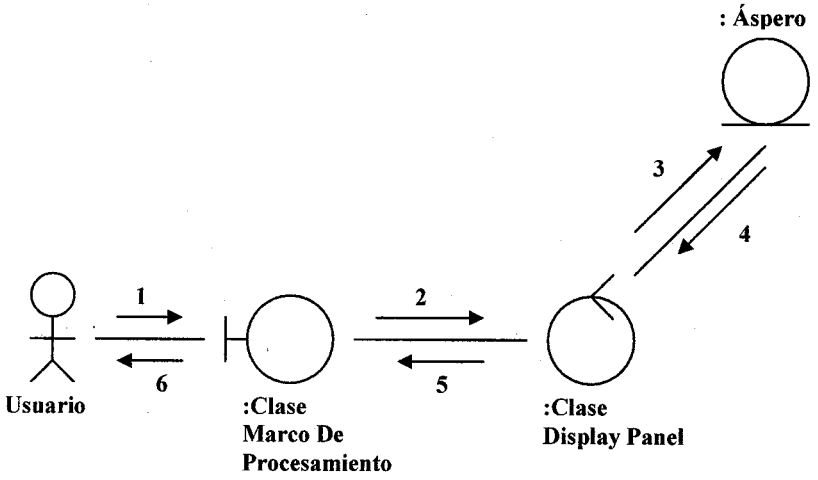
1: Solicitar aplicar una segmentación a la imagen.
2: Transferir la solicitud para la segmentación de la imagen.
3: Solicitud para segmentar la imagen.
4: Imagen segmentada.
5: La imagen segmentada es enviada al marco para ser visualizada.
6: La imagen segmentada es visualizada por el usuario.

4.5.14 Diagrama de colaboración para el caso de uso particular <u>Difuminar</u> a través de Editar.



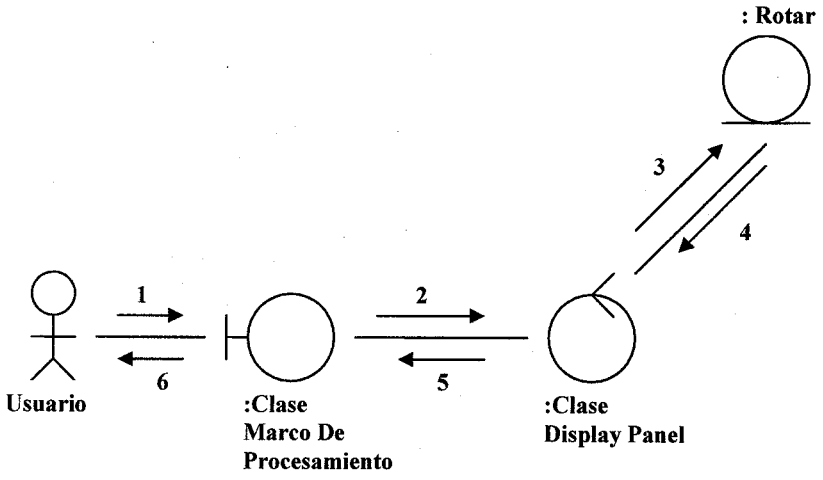
1: Solicitar la difuminación de la imagen actual.
2: Transferir la solicitud para la difuminación de la imagen.
3: Solicitud para difuminar la imagen.
4: Imagen difuminada.
5: La imagen difuminada es enviada al marco para ser visualizada.
6: La imagen difuminada es visualizada por el usuario.

4.5.15 Diagrama de colaboración para el caso de uso particular Áspero a través de Editar.



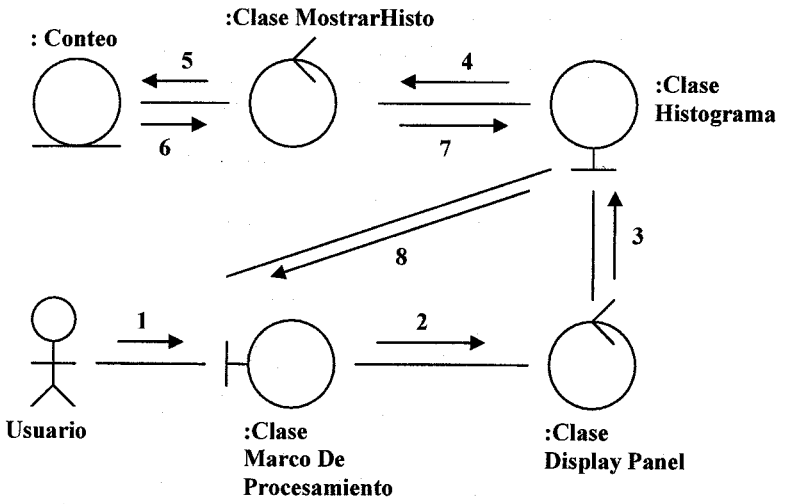
1: Solicitar tornar áspera la imagen actual.
2: Transferir la solicitud para tornar áspera la imagen.
3: Solicitud para tornar áspera la imagen.
4: Imagen áspera.
5: La imagen áspera es enviada al marco para ser visualizada.
6: La imagen áspera es visualizada por el usuario.

4.5.16 Diagrama de colaboración para el caso de uso particular <u>Rotar</u> a través de Editar.



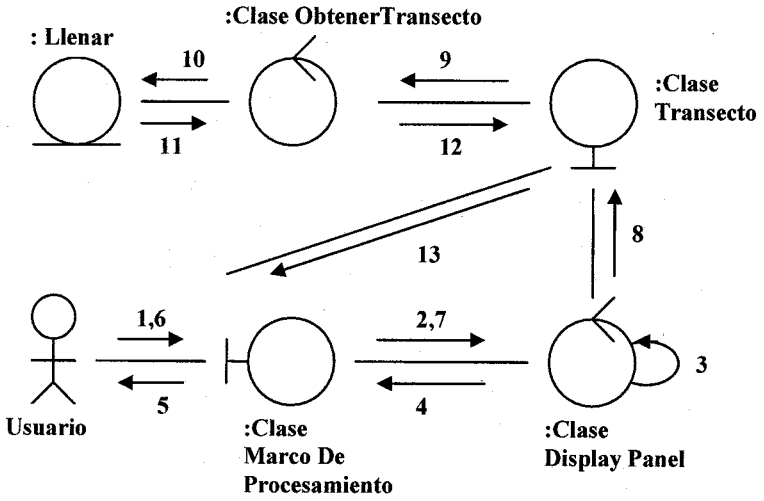
1: Solicitar rotar la imagen actual a un ángulo especificado.
2: Transferir la solicitud para rotar la imagen.
3: Solicitud para rotar la imagen.
4: Imagen rotada al ángulo seleccionado.
5: La imagen ya rotada es enviada al marco para ser visualizada.
6: La imagen rotada es visualizada por el usuario.

4.5.17 Diagrama de colaboración para el caso de uso particular Histograma a través de Información de Imágenes.



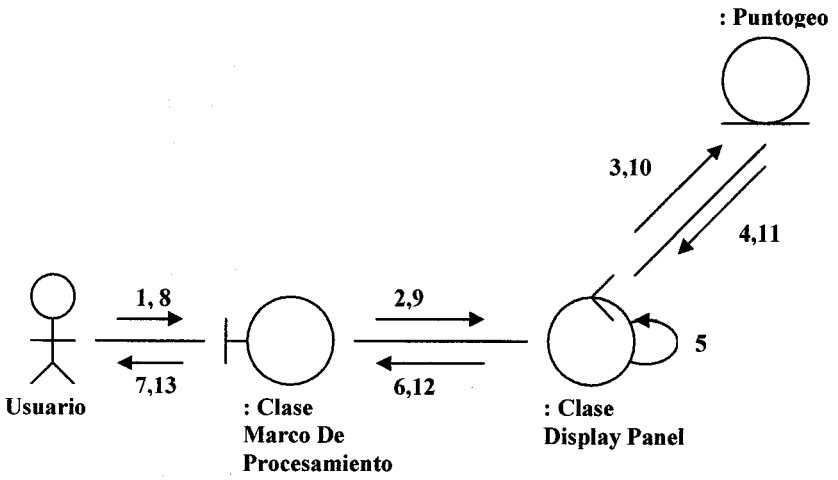
1: Solicitar la presentación del histograma de la imagen actual.
2: Transferir la solicitud para la obtención del histograma.
3: Envío de los datos de la imagen actual para la obtención de su histograma.
4: Solicitud para la obtención del histograma de la imagen.
5: Solicitud de presentación del histograma.
6: Histograma obtenido, es enviado para su presentación.
7: Histograma obtenido es desplegado en el marco destinado para esto.
8: El usuario visualiza el histograma de la imagen actual.

4.5.18 Diagrama de colaboración para el caso de uso particular <u>Transecto,</u> a través de Información de Imágenes.



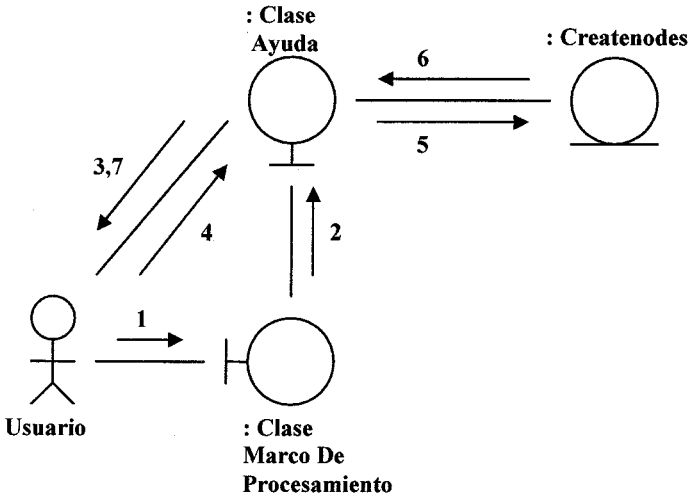
1: Solicitar el trazado de un transecto.
2: Transferir la solicitud para la obtención de un transecto.
3: Transformación del apuntador.
4: Disponibilidad de obtención de información.
5: El usuario puede trazar el transecto deseado.
6: El usuario traza el transecto.
7: Solicitar la información del transecto trazado.
8: Envío de la información del transecto trazado.
9: Solicitud de procesamiento de la información del transecto.
10: Procesamiento de la información del transecto.
11: Información obtenida es enviada para su presentación en tabla.
12: Tabla desplegada en el marco predispuesto para esto.
13: El usuario visualiza la tabla correspondiente al transecto trazado.

4.5.19 Diagrama de colaboración para el caso de uso particular Punto Georeferenciado, a través de Información de Imágenes.



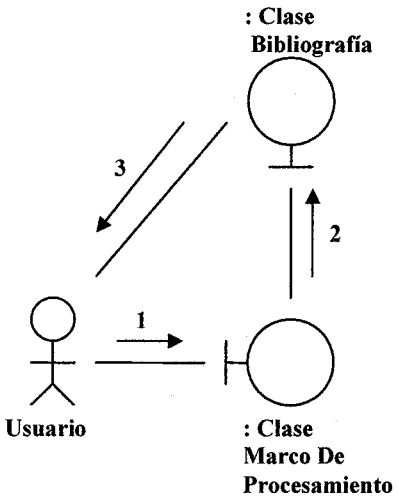
1: Solicitar la información de temperatura de un punto en la imagen actual.
2: Transferir la solicitud para obtener la información de un punto en la imagen actual.
3: Solicitud establecida para obtener la información.
4: Disponibilidad de la imagen para elegir un punto.
5: Transformación de apuntador.
6: Disposición de selección de un punto.
7: El usuario observa en nuevo apuntador dentro de la región de la imagen.
8: El usuario selecciona un punto de interés.
9: Informar que se ha seleccionado un punto.
10: Información enviada del punto georeferenciado.
11: Conclusión satisfactoria de la operación.
12: Información del punto es enviada al marco para presentarla.
13: El usuario observa la información de la temperatura y la georeferencia del punto seleccionado.

4.5.20 Diagrama de colaboración para el caso de uso Ayuda.



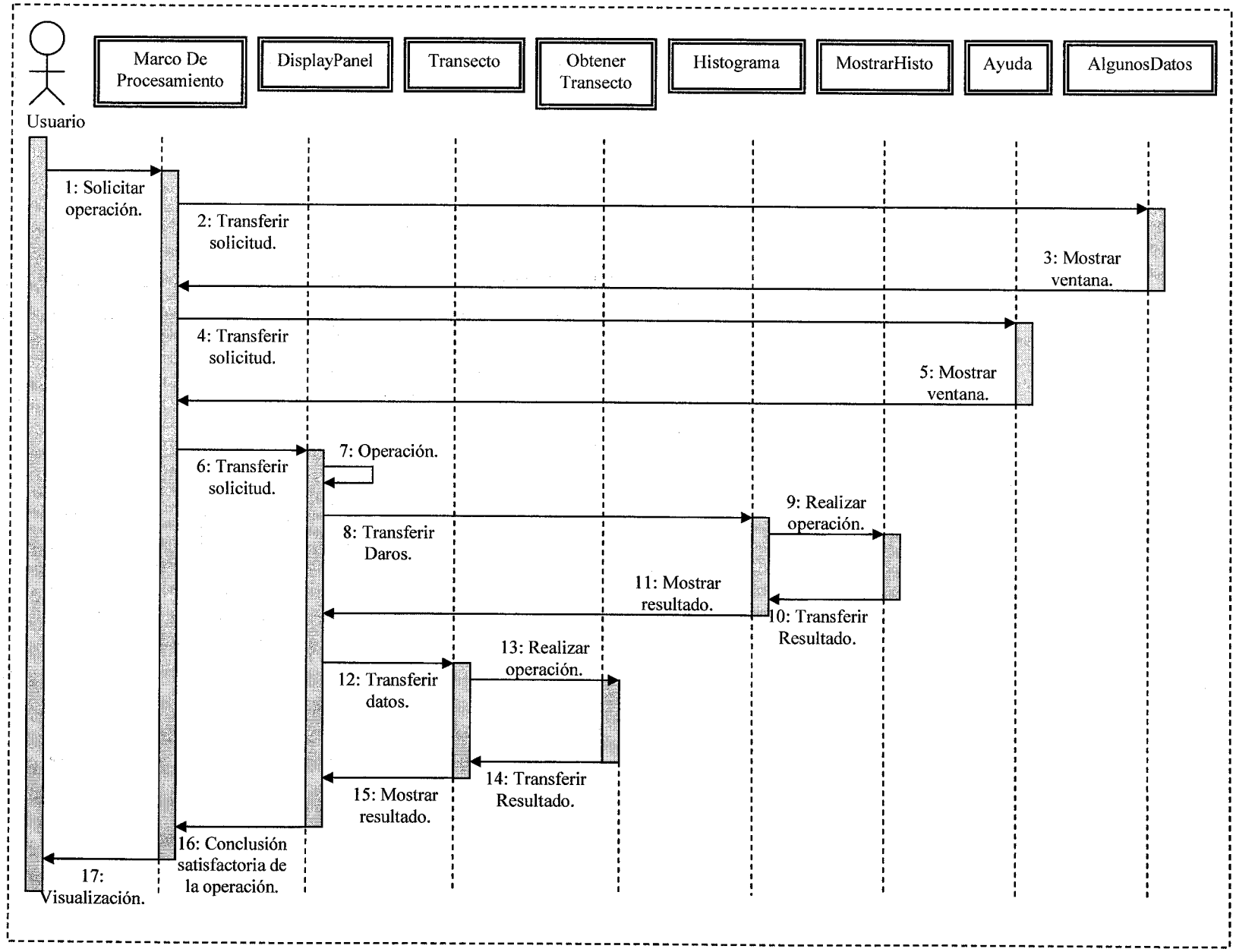
1: Solicitar ayuda.
2: Solicitud de ayuda establecida.
3: El usuario visualiza la ventana con los temas de ayuda.
4: Solicitar la presentación de un tema de ayuda.
5: Solicitud del tema de ayuda establecida.
6: Se realiza la apertura del tema de ayuda para su presentación en la ventana.
7: El usuario visualiza el tema de ayuda solicitado.

4.5.21 Diagrama de colaboración para el caso de uso <u>Algunos Datos.</u>



1: Solicitar algunos datos.
2: Solicitud de presentar algunos datos establecida.
3: El usuario visualiza la ventana con algunos datos.

4.6 Diagrama de Secuencia para un escenario general en la aplicación Web.


Usuario
Marco De Procesamiento
DisplayPanel
Transecto
Obtener Transecto
Histograma
MostrarHisto
Ayuda
AlgunosDatos
1: Solicitar operación.
2: Transferir solicitud.
3: Mostrar ventana.
4: Transferir solicitud.
5: Mostrar ventana.
6: Transferir solicitud.
7: Operación.
8: Transferir Daros.
9: Realizar operación.
10: Transferir Resultado.
11: Mostrar resultado.
12: Transferir datos.
13: Realizar operación.
14: Transferir Resultado.
15: Mostrar resultado.
16: Conclusión satisfactoria de la operación.
17: Visualización.

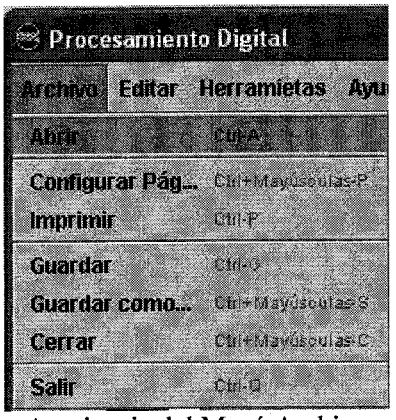
CAPÍTULO 5

Resultados.

5.1 Del diseño.

A partir del diseño planteado en el capitulo anterior se obtuvo una aplicación Web de la cual se describen sus partes a continuación.

5.1.1 Menú Archivo



Apariencia del Menú Archivo

<<Abrir>>

Este elemento del menú Archivo permite abrir archivos de imagen en formatos ya sea ".gif", ".jpg", o "jpeg" exclusivamente, dado que este sistema esta diseñado para procesar digitalmente imágenes con dichos formatos.
La ventana que permite seleccionar el archivo de imagen para abrir, esta provista de una región ubicada a la derecha de la ventana que permite una visualización en miniatura de la imagen seleccionada.

<<Configurar Pág...>>

Este elemento del menú Archivo permite ajustar los márgenes de la hoja donde será impresa la imagen así como también la orientación (horizontal o vertical); de no especificar el tamaño de los márgenes que se desea, estos aparecerán de forma predeterminada con un tamaño de 2.5cm cada uno y una orientación vertical también predeterminada.

<<Imprimir>>

Este elemento del menú Archivo permite visualizar el cuadro de diálogo para realizar la impresión de la imagen contenida actualmente en el panel, dicho cuadro de diálogo cuenta con todas las opciones de configuración de impresión estándares

<<Guardar>>

Este elemento del menú Archivo permite guardar los cambios realizados a la imagen origen es decirse guarda la imagen contenida actualmente en el panel; se sobre escribe la imagen origen, es decir se actualiza la imagen origen con las modificaciones realizadas (el nombre y la extensión permanecen iguales) por lo que se debe tener cuidado si no se desea perder la imagen origen.

<<Guardar como...>>

Este elemento del menú Archivo permite observar el cuadro de diálogo para guardar los cambios realizados a la imagen origen es decir se guarda la imagen contenida actualmente en el panel pero como una nueva imagen; no se sobre escribe la imagen origen, es decir debemos especificar un nombre y una extensión (otro directorio si se desea) para el nuevo archivo que se generará. Cabe mencionar que además del nombre, se debe especificar la extensión que puede ser ".gif", ".jpg" o bien ".jpeg" de lo contrario la imagen no se guardará de forma correcta y los datos pueden perderse.

<<Cerrar>>

Este elemento del menú Archivo cierra la imagen contenida en el panel, antes de realizar el cierre informa al usuario si desea guardar los cambios realizados independientemente de si se halla realizado algún cambio o no; en caso de que la opción seleccionada sea "No", el programa simplemente cierra la imagen y limpia por completo el panel, ocultando las barras de herramientas que hubieran sido utilizadas además de deshabilitar algunas opciones del menú; en caso de que la opción seleccionada sea "Si", el programa guarda la imagen actual sobre escribiendo la imagen origen tal como sí se hubiera seleccionado el elemento "Guardar" del menú Archivo e inmediatamente limpia el panel y deshabilita algunos elementos del menú; en caso de que la opción seleccionada sea "Cancelar", el programa cierra el diálogo y permite continuar trabajando en el panel sin ningún cambio.

<<Salir>>

Este elemento del menú Archivo cierra el sistema, antes de realizar el cierre informa al usuario si desea guardar los cambios realizados independientemente de si se halla realizado algún cambio o no; en caso de que la opción seleccionada sea "No", el programa simplemente cierra el sistema; en caso de que la opción seleccionada sea "Si", el programa guarda la imagen actual sobre escribiendo la imagen origen tal como sí se hubiera seleccionado el elemento "Guardar" del menú Archivo e inmediatamente cierra

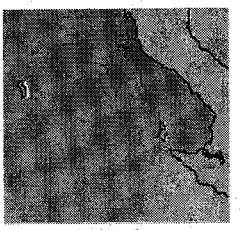
el sistema; en caso de que la opción seleccionada sea "Cancelar", el programa cierra el diálogo y permite continuar trabajando en el panel sin ningún cambio.

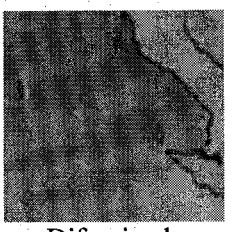
5.1.2 Menú Editar

Apariencia del Menú Editar

<<Difuminar>>

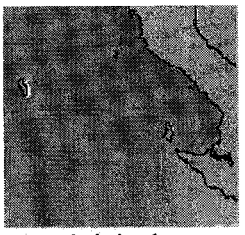
Este elemento del menú Editar aplica una transformación a la imagen contenida actualmente en el panel; esta transformación como su nombre lo dice, difumina la apariencia de la imagen (la hace un poco más opaca); las imágenes presentadas a continuación son un ejemplo de antes y después de aplicar esta transformación.

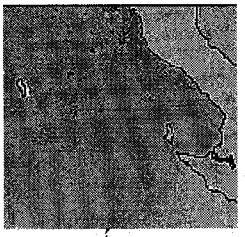

Original


Difuminada

<<Áspero>>

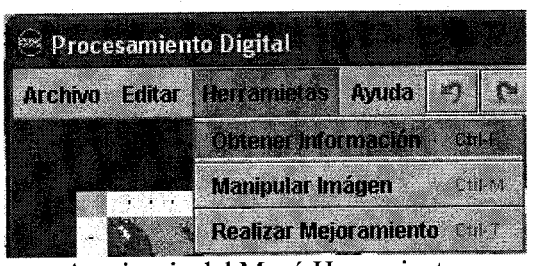
Este elemento del menú Editar aplica una transformación a la imagen contenida actualmente en el panel; esta transformación como su nombre lo dice, hace áspera la apariencia de la imagen (la sobre enfoca); las imágenes presentadas a continuación son un ejemplo de antes y después de aplicar esta transformación.


Original


Áspero

<<Rotar>>
Este elemento del menú Editar permite rotar la imagen contenida actualmente en el panel; la rotación puede ser de 90°, 180° o 270°.

5.1.3 Menú Herramientas



Apariencia del Menú Herramientas

<<Obtener Información>>

Este elemento del menú Herramientas, despliega una barra de herramientas que permiten obtener información de la imagen contenida en el panel; la barra de herramientas desplegada contiene los siguientes elementos:

Punto Georeferenciado

Al seleccionar esta herramienta, el apuntador del Mouse cambia de figura dentro del área de la imagen; podemos entonces hacer clic en el punto en que deseemos conocer la temperatura e inmediatamente aparecerá junto al punto seleccionado la información de la temperatura y su ubicación en cuanto a Longitud y Latitud se refiere.

Transecto

Al seleccionar esta herramienta, el apuntador del Mouse cambia de figura dentro del área de la imagen; podemos entonces presionar el botón izquierdo del Mouse el un punto de nuestra preferencia y sin soltarlo trazar una línea recta hacia cualquier otro punto, luego de soltarlo aparecerá una nueva ventana con la lista de puntos y su temperatura correspondiente tal como muestra la imagen (las temperaturas se presentan en el orden en que hicimos crecer la recta).

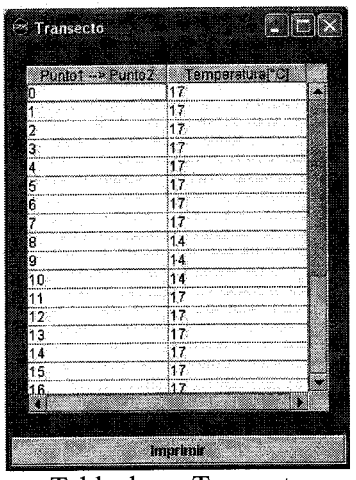
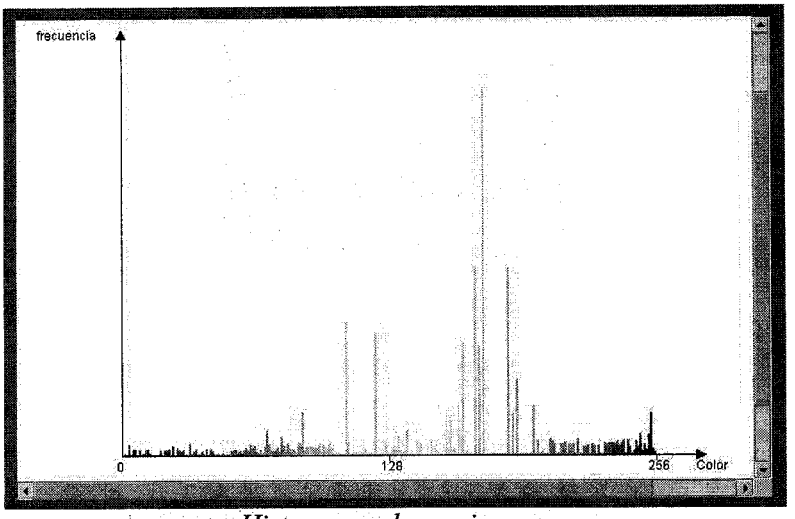


Tabla de un Transecto

Histograma

Al seleccionar esta herramienta, el sistema comienza a procesar la información de la imagen contenida en el panel para obtener su respectivo histograma; el histograma resultante se mostrará en una nueva ventana la cual aparecerá una vez que se halla generado por completo el histograma, cabe resaltar que esta operación puede llegar a tardar algunos minutos dependiendo del tamaño de la imagen, el histograma podrá ser impreso si así se desea presionando el botón inferior de la ventana del histograma.



Histograma de una imagen

<<Manipular Imagen>>

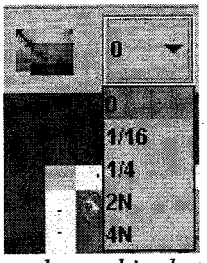
Este elemento del menú Herramientas, despliega una barra de herramientas cuyos elementos permiten manipular la imagen contenida en el panel; la barra de herramientas desplegada contiene los siguientes elementos:

Recortar

Al seleccionar esta herramienta, el apuntador del Mouse cambia de figura dentro del área de la imagen; podemos entonces presionar el botón izquierdo del Mouse el un punto de nuestra preferencia y sin soltarlo trazar un rectángulo que encierre la región de la imagen que deseamos recortar, luego de soltarlo la región seleccionada aparecerá como la imagen actual en el panel y esta podrá ser procesada y guardada según lo deseemos; cabe mencionar que podemos deshacer esta selección y volver a la imagen anterior si seleccionamos el botón deshacer que aparece en la barra de menú principal, esta acción solo será valida si la imagen recortada no ha sido ya procesada.

Modificar Tamaño

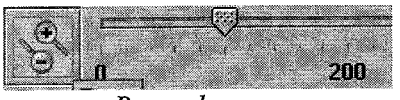
Al seleccionar esta herramienta, se habilita una lista de opciones de factores de tamaño en que se puede escalar la imagen, estos factores son: "1/4", "1/2", "2", "4" o bien "0" para reestablecer la imagen a su tamaño original. Al seleccionar un factor, la imagen contenida en el panel modifica automáticamente su tamaño según el factor. Cabe mencionar que si la imagen es demasiado grande y no es escalable, aparecerá un informe de que la imagen no se puede escalar y no será modificada.



Opciones de cambio de tamaño.

Realizar Zoom

Al seleccionar esta herramienta, se habilitará la barra para realizar el zoom deseado sobre la imagen del panel, si la imagen es demasiado grande, aparecerá un informe de que se debe seleccionar una región para poder realizar el zoom (la región seleccionada no podrá ser mayor a 200x200), entonces el apuntador del Mouse cambia de figura dentro del área de la imagen, podemos entonces presionar el botón izquierdo del Mouse el un punto de nuestra preferencia y sin soltarlo trazar un rectángulo que encierre la región de la imagen a la que deseamos realzarle el zoom, luego de soltarlo la región seleccionada aparecerá como la imagen actual en el panel y a esta si se le podrá realizar el zoom de hasta 8 veces el tamaño de la región seleccionada; a este zoom no se le podrá realizar ninguna operación (si esto sucede aparecerá la imagen original con la operación realizada), para salir del zoom basta con presionar el botón deshacer.



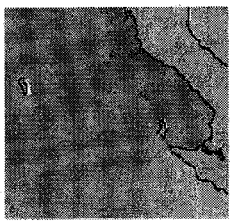
Barra de zoom

<<Realizar Mejoramiento>>

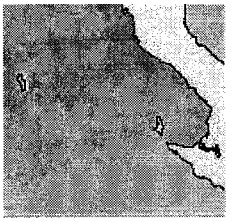
Este elemento del menú Herramientas, despliega una barra de herramientas cuyos elementos permiten aplicar ciertos tipos de transformación a la imagen contenida en el panel; la barra de herramientas desplegada contiene los siguientes elementos:

Realzar

Este elemento de la barra de herramientas aplica una transformación a la imagen contenida actualmente en el panel; esta transformación consiste en realzar el color en cada uno de los píxeles de la imagen tal como si aplicáramos brillo a la imagen; las imágenes presentadas a continuación son un ejemplo de antes y después de aplicar esta transformación.



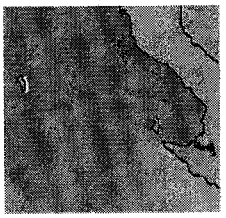
Original



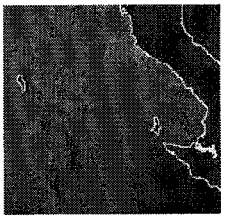
Realzada

Binarización

Este elemento de la barra de herramientas aplica una transformación a la imagen contenida actualmente en el panel; esta transformación consiste en transformar cada color de píxel a un equivalente binario en el espectro de colores rojo, verde y azul (tal como si obtuviéramos el negativo de la imagen); las imágenes presentadas a continuación son un ejemplo de antes y después de aplicar esta transformación.



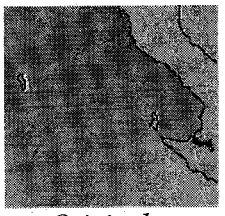
Original



Binarizada

Segmentación

Este elemento de la barra de herramientas aplica una transformación a la imagen contenida actualmente en el panel; esta transformación consiste en detectar regiones de color predominante (gradientes), dichas regiones son detectadas y representadas médiate una línea que hace frontera con can cada cambio drástico de color en la imagen; las imágenes presentadas a continuación son un ejemplo de antes y después de aplicar esta transformación.



Original

Segmentada

ANEXO

CÓDIGO FUENTE (JAVA)

//Main.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.awt.print.*;
import javax.imageio.*;
import java.io.*;
import java.beans.*;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JOptionPane;
import javax.swing.JDialog;
import javax.swing.border.Border;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;

import java.awt.Cursor.*;

import java.net.*;
import java.awt.image.BufferedImage;
import java.awt.MenuItem;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.CheckboxMenuItem;

import javax.swing.JApplet;
import javax.swing.JOptionPane;
import java.io.FilePermission;

import javax.swing.event.UndoableEditEvent;
import javax.swing.event.UndoableEditListener;
import javax.swing.undo.CannotRedoException;
import javax.swing.undo.UndoManager;


public class Main  extends JApplet
{
    public void init()
    //public static void main(String[] args)
        {
        FilePermission perm = new
FilePermission("/Documents and
Settings/Visitante/Desktop/Sistema/Icmyl.gif",
"read, write, execute");

        FilePermission perm1 = new
FilePermission(System.getProperty("user.home"
),"write, execute");


        JFrame frame = new
MarcoDeProcesamiento();
        frame.show();
        }
}
class MarcoDeProcesamiento extends JFrame
        implements ActionListener
{
    private JMenuItem elementoAbrir;
    private JMenuItem configPag;
    private JMenuItem imprimir;
    private JMenuItem elementoGuardar;
    private JMenuItem elementoGuardarComo;
    private JMenuItem elementoCerrar;
    private JMenuItem salir;
        private JMenuItem blurItem;
        private JMenuItem sharpenItem;
        private JMenuItem brillo;
        private JMenuItem edgeDetectItem;
        private JMenuItem negativo;
        private JMenuItem noventaItem;
        private JMenuItem cochentaItem;
        private JMenuItem dsetentaItem;
    private JMenuItem ObInformacion;
    private JMenuItem ObManipulacion;
    private JMenuItem ObMejoramiento;
    private JMenuItem textos;
    private JMenuItem algunos;
    private JButton anterior;
    private JButton posterior;
    public int bandera = 0;
    public JScrollPane sp2;
    public BufferedImage mBufferedImage;
```

```
    public File fFile = new File ("default.java");
    public Container contentPane =
getContentPane();
    private DisplayPanel displayPanel;
        public MarcoDeProcesamiento()
        {
                setTitle("Procesamiento
Digital");

        setIconImage(Toolkit.getDefaultToolkit
().getImage("Icmyl.gif"));
         setSize(800, 600);
                centrar();
         addWindowListener(new
WindowAdapter()
                {
                        public void
windowClosing(WindowEvent e)
                        {

        System.exit(0);
                        }
                }
                );
         JScrollPane sp = new
JScrollPane(JScrollPane.

VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.
HORIZONTAL_SCROLLBAR_ALWAYS);
Border brd =
BorderFactory.createMatteBorder(40, 35, 40, 35,
Color.getColor("azul", -16490900));
         sp.setBorder(brd);
         contentPane.add(sp,
BorderLayout.CENTER);



//**********************************
**********************************
         //Componentes del MENÚ ARCHIVO
//**********************************
**********************************
                JMenu menuArchivo = new
JMenu("Archivo");

         elementoAbrir = new
JMenuItem("Abrir");

elementoAbrir.setAccelerator(KeyStroke.getKey
Stroke(
              KeyEvent.VK_A,
Event.CTRL_MASK));

elementoAbrir.addActionListener(this);

         menuArchivo.add(elementoAbrir);

         menuArchivo.addSeparator();

         configPag = new
JMenuItem("Configurar Pág...");

configPag.setAccelerator(KeyStroke.getKeyStro
ke(
              KeyEvent.VK_P,
Event.CTRL_MASK | Event.SHIFT_MASK));
         configPag.addActionListener(this);
         configPag.setEnabled(false);
              menuArchivo.add(configPag);

         imprimir = new
JMenuItem("Imprimir");

imprimir.setAccelerator(KeyStroke.getKeyStrok
e(
              KeyEvent.VK_P,
Event.CTRL_MASK));
         imprimir.addActionListener(this);
         imprimir.setEnabled(false);
              menuArchivo.add(imprimir);

         menuArchivo.addSeparator();

         elementoGuardar = new
JMenuItem("Guardar");

elementoGuardar.setAccelerator(KeyStroke.getK
eyStroke(
              KeyEvent.VK_G,
Event.CTRL_MASK));

elementoGuardar.addActionListener(this);
         elementoGuardar.setEnabled(false);

        menuArchivo.add(elementoGuardar);

         elementoGuardarComo = new
JMenuItem("Guardar como...");

elementoGuardarComo.setAccelerator(KeyStrok
e.getKeyStroke(
              KeyEvent.VK_S,
Event.CTRL_MASK | Event.SHIFT_MASK));

elementoGuardarComo.addActionListener(this);

elementoGuardarComo.setEnabled(false);

        menuArchivo.add(elementoGuardarCo
mo);
```

```
        elementoCerrar = new
JMenuItem("Cerrar");

elementoCerrar.setAccelerator(KeyStroke.getKe
yStroke(
                KeyEvent.VK_C,
Event.CTRL_MASK | Event.SHIFT_MASK));

elementoCerrar.addActionListener(this);
        elementoCerrar.setEnabled(false);

       menuArchivo.add(elementoCerrar);

        menuArchivo.addSeparator();

        salir = new JMenuItem("Salir");

salir.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_Q,
Event.CTRL_MASK));
        salir.addActionListener(this);
                menuArchivo.add(salir);

//************************************
**********************************
        //Componentes del MENÚ EDITAR
//************************************
**********************************
                JMenu menuEditar = new
JMenu("Editar");

        blurItem = new
JMenuItem("Difuminar");

       blurItem.addActionListener(this);
                blurItem.setEnabled(false);
        menuEditar.add(blurItem);

        menuEditar.addSeparator();

                sharpenItem = new
JMenuItem("Áspero");

       sharpenItem.addActionListener(this);
        sharpenItem.setEnabled(false);
                menuEditar.add(sharpenItem);

        menuEditar.addSeparator();

                JMenu rotar = new
JMenu("Rotar");
        noventaItem = new JMenuItem("90°");
        rotar.add(noventaItem);
        cochentaItem = new
JMenuItem("180°");
        rotar.add(cochentaItem);
        dsetentaItem = new
JMenuItem("270°");
        rotar.add(dsetentaItem);

       noventaItem.addActionListener(this);

       cochentaItem.addActionListener(this);
        dsetentaItem.addActionListener(this);
        noventaItem.setEnabled(false);
        cochentaItem.setEnabled(false);
        dsetentaItem.setEnabled(false);
                menuEditar.add(rotar);

//************************************
**********************************
        //Componentes del MENÚ BARRAS
DE HERRAMIENTAS
//************************************
**********************************
        JMenu menuBarras = new
JMenu("Herramietas");

        ObInformacion = new
JMenuItem("Obtener Información");

ObInformacion.setAccelerator(KeyStroke.getKe
yStroke(
                KeyEvent.VK_I,
Event.CTRL_MASK));

ObInformacion.addActionListener(this);
        ObInformacion.setEnabled(false);

       menuBarras.add(ObInformacion);

        menuBarras.addSeparator();

        ObManipulacion = new
JMenuItem("Manipular Imágen");

ObManipulacion.setAccelerator(KeyStroke.getK
eyStroke(
                KeyEvent.VK_M,
Event.CTRL_MASK));

ObManipulacion.addActionListener(this);
        ObManipulacion.setEnabled(false);

       menuBarras.add(ObManipulacion);

        menuBarras.addSeparator();

        ObMejoramiento = new
JMenuItem("Realizar Mejoramiento");
```

```
ObMejoramiento.setAccelerator(KeyStroke.getK
eyStroke(
                KeyEvent.VK_T,
Event.CTRL_MASK));

ObMejoramiento.addActionListener(this);
          ObMejoramiento.setEnabled(false);

          menuBarras.add(ObMejoramiento);

//*************************************
**********************************
//*************************************
**********************************
          //Componentes del MENÚ AYUDA
//*************************************
**********************************

          JMenu menuAyuda = new
JMenu("Ayuda");

          textos = new JMenuItem("Textos de
Ayuda");

textos.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_T,
Event.CTRL_MASK));
          textos.addActionListener(this);
                    menuAyuda.add(textos);

          algunos = new JMenuItem("Algunos
Datos");

algunos.setAccelerator(KeyStroke.getKeyStroke
(
                KeyEvent.VK_D,
Event.CTRL_MASK));
          algunos.addActionListener(this);
                    menuAyuda.add(algunos);

//*************************************
**********************************
          //Boton para regresar acciones
//*************************************
**********************************
          Icon regresar = new
ImageIcon("anterior.PNG");
          anterior = new JButton(regresar);
          anterior.setMaximumSize(new
Dimension(30, 25));
          anterior.setToolTipText("Anterior");
          //contentPane.add(anterior,
BorderLayout.SOUTH);
          anterior.setEnabled(false);
          anterior.addActionListener(new
ActionListener()
          {
            public void
actionPerformed(ActionEvent ev)
            {
            displayPanel.anterior();
            }
          });
          Icon siguiente = new
ImageIcon("posterior.PNG");
          posterior = new JButton(siguiente);
          posterior.setMaximumSize(new
Dimension(30, 25));
          posterior.setToolTipText("Posterior");
          //contentPane.add(posterior,
BorderLayout.SOUTH);
          posterior.setEnabled(false);
          posterior.addActionListener(new
ActionListener()
          {
            public void
actionPerformed(ActionEvent ev)
            {
            displayPanel.posterior();
            }
          });
//*************************************
**********************************
          //Alta de componentes del Menu
//*************************************
**********************************
          JMenuBar menuBar = new
JMenuBar();
                    menuBar.add(menuArchivo);
          menuBar.add(menuEditar);
                    menuBar.add(menuBarras);
          menuBar.add(menuAyuda);
          menuBar.add(anterior);
          menuBar.add(posterior);
          setJMenuBar(menuBar);
        }

    protected void centrar()
    {
        Dimension screen =
Toolkit.getDefaultToolkit().getScreenSize();
        Dimension us = getSize();
        int x = (screen.width - us.width) / 2;
        int y = (screen.height - us.height) / 2;
        setLocation(x, y);
    }

          public void
actionPerformed(ActionEvent evt)
          {
```

```
                Object source =
evt.getSource();
                if(source == elementoAbrir)
                {
            JFileChooser eleccion = new
JFileChooser();

        eleccion.setCurrentDirectory(new
File("."));
                Miniatura previa = new
Miniatura(eleccion);
                eleccion.setAccessory(previa);

        eleccion.setFileFilter(new

        javax.swing.filechooser.FileFilter()
                                {

        public boolean accept(File f)
                                {

        String name =
f.getName().toLowerCase();

        return name.endsWith(".gif")

        || name.endsWith(".jpg")

        || name.endsWith(".jpeg")

        || f.isDirectory();
                                }

        public String getDescription()
                                {

        return "Archivos de Imágen (*.gif,
*.jpg, *.jpeg)";
                                }
                            });
            int returnVal =
eleccion.showOpenDialog(this);
            if(returnVal ==
JFileChooser.APPROVE_OPTION)
            {
              if (eleccion.getSelectedFile() ==
null)
                return;
              File file =
eleccion.getSelectedFile();
              String path = file.getPath();
              displayPanel = new
DisplayPanel(path);
              removeScrollPane(contentPane);
              JScrollPane sp2 = new
JScrollPane(displayPanel, JScrollPane.
VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.

HORIZONTAL_SCROLLBAR_ALWAYS);
              JLabel regla11 = new JLabel(new
ImageIcon("reglahorizontal.gif", "regla
vertical"));
              JLabel regla21 = new JLabel(new
ImageIcon("reglavertical.gif", "regla
Horizontal"));

sp2.setColumnHeaderView(regla11);
              sp2.setRowHeaderView(regla21);
              Border brd =
BorderFactory.createMatteBorder(40, 35, 40, 35,
Color.getColor("azul", -16490900));
              sp2.setBorder(brd);
              contentPane.add(sp2,
BorderLayout.CENTER);
              contentPane.validate();
              bandera = 1;

              //Habilitacion de componentes del
menú
              configPag.setEnabled(true);
              imprimir.setEnabled(true);

elementoGuardar.setEnabled(true);

elementoGuardarComo.setEnabled(true);
              elementoCerrar.setEnabled(true);
              blurItem.setEnabled(true);
              sharpenItem.setEnabled(true);
              noventaItem.setEnabled(true);
              cochentaItem.setEnabled(true);
              dsetentaItem.setEnabled(true);
              ObInformacion.setEnabled(true);

ObManipulacion.setEnabled(true);

ObMejoramiento.setEnabled(true);
              anterior.setEnabled(true);
              posterior.setEnabled(true);
              //contentPane.repaint();

            }
                }
          else if (source == configPag)
          {
            displayPanel.configurarPagina();
          }
          else if (source == imprimir)
          {
            displayPanel.imprimir();
          }
```

```
                else if (source ==
elementoGuardar)
        {
          displayPanel.guardarImagen();
                }

        else if (source ==
elementoGuardarComo)
        {
        displayPanel.guardarComoImagen();
                }

        else if (source == elementoCerrar)
        {
          int res;
          String mensaje = "¿Guardar los
Cambios?";
          res =
JOptionPane.showInternalConfirmDialog(conten
tPane,

mensaje, null,


JOptionPane.YES_NO_CANCEL_OPTION,

JOptionPane.QUESTION_MESSAGE);
          if (res ==
JOptionPane.NO_OPTION)
          {
            contentPane.removeAll();
            contentPane.repaint();
            JScrollPane sp3 = new
JScrollPane(JScrollPane.

VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.

HORIZONTAL_SCROLLBAR_ALWAYS);
              //JLabel regla13 = new
JLabel(new ImageIcon("reglahorizontal.gif",
"regla vertical"));
              //JLabel regla23 = new
JLabel(new ImageIcon("reglavertical.gif", "regla
Horizontal"));

//sp3.setColumnHeaderView(regla13);

//sp3.setRowHeaderView(regla23);
              Border brd3 =
BorderFactory.createMatteBorder(40, 35, 40, 35,
Color.getColor("azul", -16490900));
              sp3.setBorder(brd3);
              contentPane.add(sp3,
BorderLayout.CENTER);

              //deshabilitacion de componentes
del menú
              configPag.setEnabled(false);
              imprimir.setEnabled(false);

elementoGuardar.setEnabled(false);

elementoGuardarComo.setEnabled(false);
              elementoCerrar.setEnabled(false);
              blurItem.setEnabled(false);
              sharpenItem.setEnabled(false);
              noventaItem.setEnabled(false);
              cochentaItem.setEnabled(false);
              dsetentaItem.setEnabled(false);
              ObInformacion.setEnabled(false);

ObManipulacion.setEnabled(false);

ObMejoramiento.setEnabled(false);
              anterior.setEnabled(false);
              posterior.setEnabled(false);
              setVisible(true);
            }
            else if(res ==
JOptionPane.YES_OPTION)
            {
              bandera = 0;
              displayPanel.guardarImagen();
              contentPane.removeAll();
              contentPane.repaint();
              JScrollPane sp3 = new
JScrollPane(JScrollPane.

VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.

HORIZONTAL_SCROLLBAR_ALWAYS);
              JLabel regla13 = new JLabel(new
ImageIcon("reglahorizontal.gif", "regla
vertical"));
              JLabel regla23 = new JLabel(new
ImageIcon("reglavertical.gif", "regla
Horizontal"));

sp3.setColumnHeaderView(regla13);
              sp3.setRowHeaderView(regla23);
              Border brd3 =
BorderFactory.createMatteBorder(40, 35, 40, 35,
Color.getColor("azul", -16490900));
              sp3.setBorder(brd3);
              contentPane.add(sp3,
BorderLayout.CENTER);

              //deshabilitacion de componentes
del menú
              configPag.setEnabled(false);
```

```
                imprimir.setEnabled(false);

elementoGuardar.setEnabled(false);

elementoGuardarComo.setEnabled(false);
                elementoCerrar.setEnabled(false);
                blurItem.setEnabled(false);
                sharpenItem.setEnabled(false);
                noventaItem.setEnabled(false);
                cochentaItem.setEnabled(false);
                dsetentaItem.setEnabled(false);
                ObInformacion.setEnabled(false);

ObManipulacion.setEnabled(false);

ObMejoramiento.setEnabled(false);
                anterior.setEnabled(false);
                posterior.setEnabled(false);
                setVisible(true);
              }
              else{}

            }

          else if (source == salir)//
System.exit(0);
          {
            if (bandera == 1)
            {
              int res;
              String mensaje = "¿Guardar los
Cambios antes de Salir?";
              res =
JOptionPane.showInternalConfirmDialog(conten
tPane,

mensaje, null,


JOptionPane.YES_NO_CANCEL_OPTION,

JOptionPane.QUESTION_MESSAGE);
              if (res ==
JOptionPane.NO_OPTION)
              {
                System.exit(0);
              }
              else if(res ==
JOptionPane.YES_OPTION)
              {
                displayPanel.guardarImagen();
                System.exit(0);
              }
              else{}
            }
            else
            {
              System.exit(0);
            }
          }
                else if (source == blurItem)
displayPanel.difuminar();
                else if (source == sharpenItem)
displayPanel.áspero();
          else if (source == noventaItem)
displayPanel.rotar(90);
          else if (source == cochentaItem)
displayPanel.rotar(180);
          else if (source == dsetentaItem)
displayPanel.rotar(270);
          else if (source == ObInformacion) new
ObInf();
          else if (source == ObManipulacion)
new ObManip();
          else if (source == ObMejoramiento)
new ObMejor();
          else if (source == textos)
          {
            Ayuda help = new Ayuda();
          }
          else if (source == algunos)
          {
            Bibliografía bliblio = new
Bibliografía();
          }
        }

    private void removeScrollPane(Container c)
    {
    Component[] components =
c.getComponents();
    for(int i = 0; i < components.length; i++)
      if(components[i] instanceof JScrollPane)
      {
        c.remove(components[i]);
        break;
      }
    }

    public class ObInf
    {
      private JButton boton1, boton2, boton3;
      public ObInf() {
      JToolBar bar = new
JToolBar("Herramientas de Información", 1);
      bar.setFloatable(false);
      bar.setRollover(true);
      bar.addSeparator();
      Icon punto = new
ImageIcon("punto.PNG");
      boton1 = new JButton(punto);
      boton1.setToolTipText("Punto");
```

```
                }
            }
        });

        bar2.add(boton22);
        bar2.addSeparator();
        Icon zoom = new
ImageIcon("zoom.PNG");
        boton23 = new JButton(zoom);
        boton23.setToolTipText("Zoom");
        boton23.addActionListener(new
ActionListener()
        {
          public void
actionPerformed(ActionEvent e)
          {
          slider.setEnabled(true);
          slider.setValue(100);
          }
        });
        slider.setMinorTickSpacing(20);
        slider.setMajorTickSpacing(100);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);

slider.setLabelTable(slider.createStandardLabels
(200));
        slider.setEnabled(false);
        slider.addChangeListener(new
ChangeListener()
        {
           public void
stateChanged(ChangeEvent e) {
                JSlider valor =
(JSlider)e.getSource();
                if(slider.getValue() > 100)
                {

displayPanel.zoomin(slider.getValue());
                }
                else if(slider.getValue() < 100 ||
slider.getValue() == 100)
                {

displayPanel.zoomout(slider.getValue());
                }
            }
        });
        bar2.add(selectamaño);
        bar2.add(boton23);
        bar2.add(slider);
        contentPane.add(bar2,
BorderLayout.NORTH);
        setVisible(true);
        }
    }
```

```
    public class ObMejor
    {
        private JButton boton31, boton34,
boton32, boton33;
        public ObMejor() {
        JToolBar bar3 = new
JToolBar("Herramientas de Mejoramiento", 1);
        bar3.setFloatable(false);
        bar3.setRollover(true);
        bar3.addSeparator();
        Icon realce1 = new
ImageIcon("realce1.PNG");
        boton31 = new JButton(realce1);
        boton31.setToolTipText("Realce");
        boton31.addActionListener(new
ActionListener()
        {
          public void
actionPerformed(ActionEvent e)
          {
            displayPanel.masrealzar();
          }
        });
        bar3.add(boton31);
        bar3.addSeparator();
        Icon binarización = new
ImageIcon("binarizacion.PNG");
        boton32 = new JButton(binarización);

boton32.setToolTipText("Binarización");
        boton32.addActionListener(new
ActionListener()
        {
          public void
actionPerformed(ActionEvent e)
          {
            displayPanel.binarización();
          }
        });
        bar3.add(boton32);
        bar3.addSeparator();
        Icon segmentación = new
ImageIcon("segmentacion.PNG");
        boton33 = new JButton(segmentación);

boton33.setToolTipText("Segmentación");
        boton33.addActionListener(new
ActionListener()
        {
          public void
actionPerformed(ActionEvent e)
          {
            displayPanel.segmentación();
          }
        });
```

```
        bar3.add(boton33);
        bar3.addSeparator();
        Icon realce2 = new
ImageIcon("realce2.PNG");
        boton34 = new JButton(realce2);
        boton34.setToolTipText("Realce");
        boton34.addActionListener(new
ActionListener()
        {
          public void
actionPerformed(ActionEvent e)
          {
            displayPanel.menosrealzar();
          }
        });
        bar3.add(boton34);
        contentPane.add(bar3,
BorderLayout.EAST);
        setVisible(true);
        }
    }


class DisplayPanel extends JPanel implements
Printable, MouseMotionListener, MouseListener
{
  public int factor;
  public Image image;
  public String nombre;
  public String nombreNuevo;
  public JFileChooser eleccion;
  public int returnVal;
  private BufferedImage mBufferedImage;
  private BufferedImage BufferOriginal;
  private BufferedImage BufferAnterior;
  private BufferedImage BufferPosterior;
  private BufferedImage BufferDeRecorte;
  private BufferedImage BufferDePunto;
  private BufferedImage BufferDeTransecto;
  private BufferedImage BufferDeTemperatura;
  private BufferedImage BufferDeZoom;
  private BufferedImage BufferAnterior5;
  private BufferedImage BufferAnterior4;
  private BufferedImage BufferAnterior3;
  private BufferedImage BufferAnterior2;
  private BufferedImage BufferAnterior1;
  private BufferedImage BufferPosterior5;
  private BufferedImage BufferPosterior4;
  private BufferedImage BufferPosterior3;
  private BufferedImage BufferPosterior2;
  private BufferedImage BufferPosterior1;


  public int posX1, posY1, posX2, posY2,
puntoX, puntoY;

  public double pospixelX1, pospixelY1,
pospixelX2, pospixelY2;
  public double m, b;
  public double pixelX, pixelY;
  public int longitud;
  public int fuera = 0;
  public int recorte = 0;
  public int puntogeoref = 0;
  public int transecto = 0;
  public int ratonsuelto = 0;
  public int puntoselec = 0;
  public int pintainicio = 0;
  public int recorterealizado = 0;
  public Graphics g;
  public int valorDePunto;
  public int coordenada = 0;
  public int diferencia, aproximado;
  public double ancho, alto;
  public double temperatura;
  public int escalar = 0;
  public PageFormat pf;
  public PrinterJob job;
  public int prefImp = 0;
  public int lati, longi;
  public double multdiv;
  public int eszoom=0;
  public int haciendoZoom;
  public int regionDeZoom = 0;


  DisplayPanel(String path)
  {
    loadImage(path);
    createBufferedImage();
    addMouseMotionListener(this);
    addMouseListener(this);
    setBackground(Color.WHITE);

  }

  public void loadImage(String fileName)
  {
    image =
Toolkit.getDefaultToolkit().getImage(fileName);
    nombre= fileName;
    MediaTracker mt = new
MediaTracker(this);
    mt.addImage(image, 0);
    try { mt.waitForID(0); }
    catch (InterruptedException ie)
    {
      return;
    }
    if (mt.isErrorID(0)) return;
    //revalidate();
```

```
////////////////////////////////////////////////////////////////
////////
      Image temperatura =
Toolkit.getDefaultToolkit().getImage("reftempe.
jpg");
      MediaTracker t = new MediaTracker(this);
      t.addImage(temperatura, 0);
      try { t.waitForID(0); }
      catch (InterruptedException ie) { return; }
      if (t.isErrorID(0)) return;

      BufferDeTemperatura = new
BufferedImage(74,973,

BufferedImage.TYPE_INT_BGR);
      Graphics2D g9 =
BufferDeTemperatura.createGraphics();
      g9.drawImage(temperatura, 0, 0, this);

////////////////////////////////////////////////////////////////
////////
   }

   public void createBufferedImage()
   {
      if(image.getWidth(this) > 1410 ||
image.getHeight(this) > 892 &&
            image.getWidth(this) > 4 &&
image.getHeight(this) > 4)
      {
         mBufferedImage = new
BufferedImage(image.getWidth(this),
                              image.getHeight(this),

BufferedImage.TYPE_INT_BGR);
         try
         {
            JOptionPane.showMessageDialog(
                  null, "          La Imágen es
demasiado " +
                  "grande." + "\nRedusca el tamaño
de esta o " +
                  "bien seleccione otra.",
                  "Error de Carga",
JOptionPane.INFORMATION_MESSAGE);
         }
         catch (RuntimeException e) {return;}

         BufferOriginal = new BufferedImage(
                              image.getWidth(this),
                              image.getHeight(this),

BufferedImage.TYPE_INT_BGR);
         escalar = 2;
         Graphics2D g2 =
mBufferedImage.createGraphics();
         g2.drawImage(image, 0, 0, this);
         Graphics2D g3 =
BufferOriginal.createGraphics();
         g3.drawImage(image, 0, 0, this);
         mtamaño(7);

      }
      else if(image.getWidth(this) > 705 ||
image.getHeight(this) > 446 &&
            image.getWidth(this) > 2 &&
image.getHeight(this) > 2)
      {
         mBufferedImage = new
BufferedImage(image.getWidth(this),
                              image.getHeight(this),

BufferedImage.TYPE_INT_BGR);
         BufferOriginal = new BufferedImage(
                              image.getWidth(this),
                              image.getHeight(this),

BufferedImage.TYPE_INT_BGR);
         escalar = 1;
         Graphics2D g2 =
mBufferedImage.createGraphics();
         g2.drawImage(image, 0, 0, this);
         Graphics2D g3 =
BufferOriginal.createGraphics();
         g3.drawImage(image, 0, 0, this);
         mtamaño(6);

      }
      else
      {
         mBufferedImage = new
BufferedImage(image.getWidth(this),
                              image.getHeight(this),

BufferedImage.TYPE_INT_BGR);
         BufferOriginal = new BufferedImage(
                              image.getWidth(this),
                              image.getHeight(this),

BufferedImage.TYPE_INT_BGR);
         Graphics2D g2 =
mBufferedImage.createGraphics();
         g2.drawImage(image, 0, 0, this);
         Graphics2D g3 =
BufferOriginal.createGraphics();
         g3.drawImage(image, 0, 0, this);
         repaint();
      }
      BufferAnterior = BufferOriginal;
      BufferPosterior = BufferOriginal;
      BufferAnterior5 = BufferAnterior;
      BufferAnterior4 = BufferAnterior;
```

```
    BufferAnterior3 = BufferAnterior;
    BufferAnterior2 = BufferAnterior;
    BufferAnterior1 = BufferAnterior;
    BufferPosterior5 = BufferPosterior;
    BufferPosterior4 = BufferPosterior;
    BufferPosterior3 = BufferPosterior;
    BufferPosterior2 = BufferPosterior;
    BufferPosterior1 = BufferPosterior;
  }

  public void guardarImagen()
  {

            Graphics2D g2 =
mBufferedImage.createGraphics();
            g2.dispose();
            try
            {
              ImageIO.write(mBufferedImage,
"jpg", new File(nombre));
            }
            catch(IOException ioe)
            {

System.out.println(ioe.getMessage());
            }

  }

  public void guardarComoImagen()
  {
              JFileChooser eleccion = new
JFileChooser();

        eleccion.setCurrentDirectory(new
File("."));
            FilePreviewer previewer = new
FilePreviewer(eleccion);
            eleccion.setAccessory(previewer);

        eleccion.setFileFilter(new

        javax.swing.filechooser.FileFilter()
                                {

        public boolean accept(File f)
                                        {

        String name =
f.getName().toLowerCase();

        return name.endsWith(".gif")

        || name.endsWith(".jpg")

        || name.endsWith(".jpeg")

        || f.isDirectory();

                                }

        public String getDescription()
                                {

        return "Archivos de Imágen (*.gif,
*.jpg, *.jpeg)";
                                }
                        });
            returnVal =
eleccion.showSaveDialog(null);
            if(returnVal ==
JFileChooser.APPROVE_OPTION)
            {
              if (eleccion.getSelectedFile() ==
null)
                return;
              File file =
eleccion.getSelectedFile();
              nombreNuevo = file.getPath();
              Graphics2D g2 =
mBufferedImage.createGraphics();
              g2.dispose();
              try
              {

ImageIO.write(mBufferedImage, "jpg",
                      new File(nombreNuevo));
              }
              catch(IOException ioe)
              {

System.out.println(ioe.getMessage());
              }
            }
  }

//////////////////////////

  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    /*int width = getWidth();
    int height = getHeight();
    int imageWidth = image.getWidth(this);
    int imageHeight = image.getHeight(this);
    int x = (width - imageWidth)/2;
    int y = (height - imageHeight)/2;*/
    if (pintainicio == 0)
    {
     g.drawImage(mBufferedImage, 0, 0, this);
```

```
g.drawLine(0,mBufferedImage.getHeight(),mBu
fferedImage.getWidth()
                ,mBufferedImage.getHeight());

g.drawLine(mBufferedImage.getWidth(),0,mBuf
feredImage.getWidth()
                ,mBufferedImage.getHeight());

      }
      else if(pintainicio == 2)
      {
        g.drawImage(mBufferedImage, 0, 0, this);
        g.setColor(Color.BLACK);
        g.drawString((int)temperatura+"°C",
puntoX+8, puntoY);
        g.drawOval(puntoX-2, puntoY-2, 4, 4);
        if(puntoX <= (612*multdiv) && puntoY
<= (384*multdiv))
        {
          longi = 20;
          lati = 30;
          g.drawString("Longitud: "+ longi,
puntoX+8, puntoY+12);
          g.drawString("Latitud:  "+ lati,
puntoX+8, puntoY+24);
        }
        else if(puntoX > (612*multdiv) &&
puntoY < (384*multdiv))
        {
          longi = 200;
          lati = 30;
          g.drawString("Longitud: "+ longi,
puntoX+8, puntoY+12);
          g.drawString("Latitud:  "+ lati,
puntoX+8, puntoY+24);
        }
        else if(puntoX < (612*multdiv) &&
puntoY > (384*multdiv))
        {
          longi = 400;
          lati = 30;
          g.drawString("Longitud: "+ longi,
puntoX+8, puntoY+12);
          g.drawString("Latitud:  "+ lati,
puntoX+8, puntoY+24);
        }
        else//(puntoX > (612*multdiv) &&
puntoY > (384*multdiv))
        {
          longi = 500;
          lati = 30;
          g.drawString("Longitud: "+ longi,
puntoX+8, puntoY+12);
          g.drawString("Latitud:  "+ lati,
puntoX+8, puntoY+24);
        }

      }
      else if(pintainicio == 3)
      {
       g.drawImage(BufferDeZoom, 0, 0, this);
      }

      else{}
      repaint();
//////////////////////////////////////////////
      if((recorte == 1 || haciendoZoom == 1)&&
fuera == 0)
      {
       if (pintainicio == 0 || pintainicio == 3)
       {
          if(posX2 > posX1 && posY2 > posY1)
          {
              g.setColor(Color.GREEN);
              g.drawLine(posX2, posY2, posX2,
0);
              g.drawLine(posX2, posY2, 0,
posY2);
              g.setColor(Color.BLUE);
              g.drawRect(posX1, posY1, posX2-
posX1, posY2-posY1);
          }
          else if(posX2 > posX1 && posY2 <
posY1)
          {
              g.setColor(Color.GREEN);
              g.drawLine(posX2, posY2, posX2,
0);
              g.drawLine(posX2, posY2, 0,
posY2);
              g.setColor(Color.BLUE);
              g.drawRect(posX1, posY2, posX2-
posX1, posY1-posY2);
          }
          else if(posX2 < posX1 && posY2 >
posY1)
          {
              g.setColor(Color.GREEN);
              g.drawLine(posX2, posY2, posX2,
0);
              g.drawLine(posX2, posY2, 0,
posY2);
              g.setColor(Color.BLUE);
              g.drawRect(posX2, posY1, posX1-
posX2, posY2-posY1);
          }
          else if(posX2 < posX1 && posY2 <
posY1)
          {
              g.setColor(Color.GREEN);
```

```
            g.drawLine(posX2, posY2, posX2,
0);
            g.drawLine(posX2, posY2, 0,
posY2);
            g.setColor(Color.BLUE);
            g.drawRect(posX2, posY2, posX1-
posX2, posY1-posY2);
         }
         else{}
      }
      else{}
      if(ratonsuelto == 1)
      {
         if(posX2 > posX1 && posY2 > posY1)
         {
            BufferDeRecorte = new
BufferedImage(
                  posX2-posX1,
                  posY2-posY1,

BufferedImage.TYPE_INT_BGR);

            Graphics2D g4 =
BufferDeRecorte.createGraphics();
            g4.drawImage(mBufferedImage,
0, 0, posX2-posX1,
                  posY2-posY1, posX1,
posY1, posX2, posY2, this);
         }
         else if(posX2 > posX1 && posY2 <
posY1)
         {
            BufferDeRecorte = new
BufferedImage(
                  posX2-posX1,
                  posY1-posY2,

BufferedImage.TYPE_INT_BGR);

            Graphics2D g4 =
BufferDeRecorte.createGraphics();
            g4.drawImage(mBufferedImage,
0, 0, posX2-posX1,
                  posY1-posY2, posX1,
posY2, posX2, posY1, this);
         }
         else if(posX2 < posX1 && posY2 >
posY1)
         {
            BufferDeRecorte = new
BufferedImage(
                  posX1-posX2,
                  posY2-posY1,

BufferedImage.TYPE_INT_BGR);

            Graphics2D g4 =
BufferDeRecorte.createGraphics();
            g4.drawImage(mBufferedImage,
0, 0, posX1-posX2,
                  posY2-posY1, posX2,
posY1, posX1, posY2, this);
         }
         else if(posX2 < posX1 && posY2 <
posY1)
         {
            BufferDeRecorte = new
BufferedImage(
                  posX1-posX2,
                  posY1-posY2,

BufferedImage.TYPE_INT_BGR);

            Graphics2D g4 =
BufferDeRecorte.createGraphics();
            g4.drawImage(mBufferedImage,
0, 0, posX1-posX2,
                  posY1-posY2, posX2,
posY2, posX1, posY1, this);
         }
         else//
         {
         }

         g.drawImage(BufferDeRecorte, 0, 0,
this);
         pintainicio = 1;
         recorterealizado = 1;
         if(recorterealizado == 1)
         {
            if(haciendoZoom == 1)
            {
            BufferAnterior5 = BufferAnterior4;
            BufferAnterior4 = BufferAnterior3;
            BufferAnterior3 = BufferAnterior2;
            BufferAnterior2 = BufferAnterior1;
            BufferAnterior1 = BufferAnterior;
            BufferAnterior = mBufferedImage;
            mBufferedImage =
BufferDeRecorte;
            BufferDeZoom = BufferDeRecorte;
            regionDeZoom = 1;
            haciendoZoom = 0;
            pintainicio = 3;
            }
            else
            {
            BufferAnterior5 = BufferAnterior4;
            BufferAnterior4 = BufferAnterior3;
            BufferAnterior3 = BufferAnterior2;
            BufferAnterior2 = BufferAnterior1;
            BufferAnterior1 = BufferAnterior;
```

```
            BufferAnterior = mBufferedImage;
            mBufferedImage =
BufferDeRecorte;
            BufferOriginal = mBufferedImage;

            pintainicio = 0;
            }
            recorte = 0;
            ratonsuelto = 0;
            recorterealizado = 0;
            posX1 = 0;
            posX2 = 0;
            posY1 = 0;
            posY2 = 0;
          }
          else{}
          repaint();
       }
       else
       {
       }
     }
     else
     {
     }
////////////////////////////////////////////////
      if(puntoselec == 1)
      {
          BufferDePunto = new BufferedImage(
              1,
              1,

BufferedImage.TYPE_INT_BGR);

          Graphics2D g5 =
BufferDePunto.createGraphics();
          g5.drawImage(mBufferedImage, 0, 0,
1,
              1, puntoX, puntoY, puntoX+1,
puntoY+1, this);
          g.drawImage(BufferDePunto, 0, 0,
this);
          valorDePunto =
BufferDePunto.getRGB(0,0);
         System.out.println("el valor es:"+
valorDePunto);
          aproximado = 17000000;
          for(int i = 0; i <
BufferDeTemperatura.getHeight(); i++ )
          {
            for(int j = 0; j <
BufferDeTemperatura.getWidth(); j++)
            {
              diferencia =
Math.abs(BufferDeTemperatura.getRGB(j,i) -
valorDePunto);
              if(diferencia < aproximado)
              {
                coordenada = i;
                aproximado = diferencia;
              }
              else{}
            }
          }

          temperatura = ((((double)coordenada *
25)/973)*(-1))+35;
          //System.out.println("la temperatura
es:"+ (int)temperatura);
          pintainicio = 1;
          recorterealizado = 1;
          if(recorterealizado == 1)
          {
            // mBufferedImage =
BufferDePunto;
            // BufferOriginal = mBufferedImage;
             pintainicio = 2;
             puntoselec = 0;
          }
          else{}
          repaint();
       }
       else{}

////////////////////////////////////////////////////////////////
//////
     if(transecto == 1 && fuera == 0)
     {
      if (pintainicio == 0)
      {
         g.setColor(Color.CYAN);
         g.drawLine((int)pospixelX1,
(int)pospixelY1,
              (int)pospixelX2, (int)pospixelY2);
         g.setColor(Color.GREEN);
         g.drawLine((int)pospixelX2,
(int)pospixelY2, (int)pospixelX2, 0);
         g.drawLine((int)pospixelX2,
(int)pospixelY2, 0, (int)pospixelY2);
       }
       else{}
       if(ratonsuelto == 1)
       {
         ////////////////cuadrante II
         if(pospixelX2 > pospixelX1 &&
pospixelY2 > pospixelY1)
         {
           if((pospixelX2-pospixelX1 >
pospixelY2-pospixelY1) ||
               (pospixelX2-pospixelX1 ==
pospixelY2-pospixelY1))
           {
```

```
                longitud = (int)pospixelX2-
(int)pospixelX1;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelY2-
pospixelY1)/(pospixelX2-pospixelX1));
                b = pospixelY1-(pospixelX1 *
m);

                for(int i = 0; i<longitud; i++)
                {
                  pixelX = pospixelX1 +
(double)i;
                  pixelY = (pixelX * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                    1, (int)pixelX, (int)pixelY,
                      (int)pixelX+1,
(int)pixelY+1, this);
                }
              }
              else
              {
                longitud = (int)pospixelY2-
(int)pospixelY1;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelX2-
pospixelX1)/(pospixelY2-pospixelY1));
                b = pospixelX1-(pospixelY1 *
m);

                for(int i = 0; i<longitud; i++)
                {
                  pixelY = pospixelY1 +
(double)i;
                  pixelX = (pixelY * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                    1, (int)pixelX, (int)pixelY,
                      (int)pixelX+1,
(int)pixelY+1, this);
                }
              }
            }
            ////////cuadrante III
            else if(pospixelX2 < pospixelX1 &&
pospixelY2 > pospixelY1)
            {
              if((pospixelX1-pospixelX2 >
pospixelY2-pospixelY1) ||
                  (pospixelX1-pospixelX2 ==
pospixelY2-pospixelY1))
              {
                longitud = (int)pospixelX1-
(int)pospixelX2;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelY1-
pospixelY2)/(pospixelX1-pospixelX2));
                b = pospixelY2-(pospixelX2 *
m);

                for(int i = 0; i<longitud; i++)
                {
                  pixelX = pospixelX1 -
(double)i;
                  pixelY = (pixelX * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                    1, (int)pixelX, (int)pixelY,
                      (int)pixelX+1,
(int)pixelY+1, this);
                }
              }
              else
              {
                longitud = (int)pospixelY2-
(int)pospixelY1;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelX2-
pospixelX1)/(pospixelY2-pospixelY1));
```

```
                b = pospixelX1-(pospixelY1 *
m);

                for(int i = 0; i<longitud; i++)
                {
                  pixelY = pospixelY1 +
(double)i;
                  pixelX = (pixelY * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                    1, (int)pixelX, (int)pixelY,
                      (int)pixelX+1,
(int)pixelY+1, this);
                }
              }
            }
            //////////cuadrante IV
            else if(pospixelX2 < pospixelX1 &&
pospixelY2 < pospixelY1)
            {
              if((pospixelX1-pospixelX2 >
pospixelY1-pospixelY2) ||
                 (pospixelX1-pospixelX2 ==
pospixelY1-pospixelY2))
              {
                longitud = (int)pospixelX1-
(int)pospixelX2;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelY2-
pospixelY1)/(pospixelX2-pospixelX1));
                b = pospixelY1-(pospixelX1 *
m);

                for(int i = 0; i<longitud; i++)
                {
                  pixelX = pospixelX1 -
(double)i;
                  pixelY = (pixelX * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                    1, (int)pixelX, (int)pixelY,
                      (int)pixelX+1,
(int)pixelY+1, this);
                }
              }
              else
              {
                longitud = (int)pospixelY1-
(int)pospixelY2;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelX2-
pospixelX1)/(pospixelY2-pospixelY1));
                b = pospixelX1-(pospixelY1 *
m);

                for(int i = 0; i<longitud; i++)
                {
                  pixelY = pospixelY1 -
(double)i;
                  pixelX = (pixelY * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                    1, (int)pixelX, (int)pixelY,
                      (int)pixelX+1,
(int)pixelY+1, this);
                }
              }
            }
            //////////cuadrante I
            else if(pospixelX2 > pospixelX1 &&
pospixelY2 < pospixelY1)
            {
              if((pospixelX2-pospixelX1 >
pospixelY1-pospixelY2) ||
                 (pospixelX2-pospixelX1 ==
pospixelY1-pospixelY2))
              {
                longitud = (int)pospixelX2-
(int)pospixelX1;

                BufferDeTransecto = new
BufferedImage(
                    longitud,
                    1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelY2-
pospixelY1)/(pospixelX2-pospixelX1));
                b = pospixelY1-(pospixelX1 *
m);

                for(int i = 0; i<longitud; i++)
                {
```

```
                    pixelX = pospixelX1 +
(double)i;
                    pixelY = (pixelX * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                        1, (int)pixelX, (int)pixelY,
                            (int)pixelX+1,
(int)pixelY+1, this);
                }
            }
            else
            {
                longitud = (int)pospixelY2-
(int)pospixelY1;

                BufferDeTransecto = new
BufferedImage(
                        longitud,
                        1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();
                m = ((pospixelX2-
pospixelX1)/(pospixelY2-pospixelY1));
                b = pospixelX1-(pospixelY1 *
m);

                for(int i = 0; i<longitud; i++)
                {
                    pixelY = pospixelY1 -
(double)i;
                    pixelX = (pixelY * m) + b;

g6.drawImage(mBufferedImage, i, 0, i+1,
                        1, (int)pixelX, (int)pixelY,
                            (int)pixelX+1,
(int)pixelY+1, this);
                }
            }
        }
        /////////eje derecho
        else if(pospixelX2 > pospixelX1 &&
pospixelY2 == pospixelY1)
        {

                longitud = (int)pospixelX2-
(int)pospixelX1;

                BufferDeTransecto = new
BufferedImage(
                        longitud,
                        1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();

                for(int i = 0; i<longitud; i++)
                {
                    pixelX = pospixelX1 +
(double)i;
                    pixelY = pospixelY1;

g6.drawImage(mBufferedImage, i, 0, i+1,
                        1, (int)pixelX, (int)pixelY,
                            (int)pixelX+1,
(int)pixelY+1, this);
                }
        }
        /////////eje izquierdo
        else if(pospixelX2 < pospixelX1 &&
pospixelY2 == pospixelY1)
        {

                longitud = (int)pospixelX1-
(int)pospixelX2;

                BufferDeTransecto = new
BufferedImage(
                        longitud,
                        1,

BufferedImage.TYPE_INT_BGR);
                Graphics2D g6 =
BufferDeTransecto.createGraphics();

                for(int i = 0; i<longitud; i++)
                {
                    pixelX = pospixelX1 -
(double)i;
                    pixelY = pospixelY1;

g6.drawImage(mBufferedImage, i, 0, i+1,
                        1, (int)pixelX, (int)pixelY,
                            (int)pixelX+1,
(int)pixelY+1, this);
                }
        }
        /////////eje seperior
        else if(pospixelX2 == pospixelX1 &&
pospixelY2 < pospixelY1)
        {

                longitud = (int)pospixelY1-
(int)pospixelY2;

                BufferDeTransecto = new
BufferedImage(
                        longitud,
                        1,
```

```
BufferedImage.TYPE_INT_BGR);
            Graphics2D g6 =
BufferDeTransecto.createGraphics();

            for(int i = 0; i<longitud; i++)
            {
              pixelY = pospixelY1 -
(double)i;
              pixelX = pospixelX1;

g6.drawImage(mBufferedImage, i, 0, i+1,
                1, (int)pixelX, (int)pixelY,
                  (int)pixelX+1,
(int)pixelY+1, this);
            }
        }
        /////////eje inferior
        else if(pospixelX2 == pospixelX1 &&
pospixelY2 > pospixelY1)
        {

            longitud = (int)pospixelY2-
(int)pospixelY1;

            BufferDeTransecto = new
BufferedImage(
                longitud,
                1,

BufferedImage.TYPE_INT_BGR);
            Graphics2D g6 =
BufferDeTransecto.createGraphics();

            for(int i = 0; i<longitud; i++)
            {
              pixelY = pospixelY1 +
(double)i;
              pixelX = pospixelX1;

g6.drawImage(mBufferedImage, i, 0, i+1,
                1, (int)pixelX, (int)pixelY,
                  (int)pixelX+1,
(int)pixelY+1, this);
            }
        }

        else//(pospixelX2 == pospixelX1 &&
pospixelY2 == pospixelY1)
        {
        }

        g.drawImage(BufferDeTransecto, 0, 0,
this);
        Transecto linea = new
Transecto(BufferDeTransecto);
        pintainicio = 1;
        recorterealizado = 1;
        if(recorterealizado == 1)
        {
          //mBufferedImage =
BufferDeTransecto;
          //BufferOriginal = mBufferedImage;
          pintainicio = 0;
          transecto = 0;
          ratonsuelto = 0;
          posX1 = 0;
          posX2 = 0;
          posY1 = 0;
          posY2 = 0;
        }
        else{}
        repaint();
      }
      else{}
    }
    else
    {
    }
//////////////////////////
  }

  public Dimension getPreferredSize()
  {
    return new
Dimension(mBufferedImage.getWidth(this)*8,
        mBufferedImage.getHeight(this)*8);
  }


  void binarización()
  {
    regionDeZoom = 0;
    haciendoZoom = 0;
    pintainicio = 0;
    byte negative[] = new byte[256];
      for(int i = 0; i < 256; i++)
          negative[i] = (byte) (255 - i);
          ByteLookupTable table = new
ByteLookupTable(0, negative);
          LookupOp op = new
LookupOp(table, null);
          filter(op, 1);
  }

  public void difuminar()
  {
    pintainicio = 0;
    regionDeZoom = 0;
    haciendoZoom = 0;
      float weight = 1.0f/9.0f;
      float[] elements = new float[9];
```

```
        for (int i = 0; i<9; i++)
                elements[i] = weight;
        convolve(elements);
    }

    public void áspero()
    {
      pintainicio = 0;
      regionDeZoom = 0;
      haciendoZoom = 0;
        float[] elements =
                {
                /**/0.0f, -1.0f, 0.0f,/**/
         /**/-1.0f, 5.f, -1.0f,/**/
         /**/0.0f, -1.0f, 0.0f /**/
                };
        convolve(elements);
    }

    void segmentación()
    {
      pintainicio = 0;
      regionDeZoom = 0;
      haciendoZoom = 0;
        float[] elements =
                {
                /**/0.0f, -1.0f, 0.0f,/**/
         /**/-1.0f, 4.f, -1.0f,/**/
         /**/0.0f, -1.0f, 0.0f /**/
                };
        convolve(elements);
    }

    public void masrealzar()
    {
      pintainicio = 0;
      regionDeZoom = 0;
      haciendoZoom = 0;
        float a = 1.5f;
      float b = 0;//-20.0f;
        RescaleOp op = new RescaleOp(a, b,
null);
        filter(op, 1);
    }

    public void menosrealzar()
    {
      pintainicio = 0;
      regionDeZoom = 0;
      haciendoZoom = 0;
        float a = 0.5f;
      float b = 64f;
        RescaleOp op = new RescaleOp(a, b,
null);
        filter(op, 1);
    }

    void rotar(int vrotación)
    {
      pintainicio = 0;
      if(vrotación == 90)
      {
        AffineTransform transform =

AffineTransform.getRotateInstance(Math.toRadi
ans(90),
              BufferOriginal.getHeight()/2,
BufferOriginal.getHeight()/2);
        AffineTransformOp op = new
AffineTransformOp(transform,

AffineTransformOp.TYPE_NEAREST_NEIGH
BOR);
        filter(op, 9);
      }
      else if(vrotación == 180)
      {

        AffineTransform transform =

AffineTransform.getRotateInstance(Math.toRadi
ans(180),
              BufferOriginal.getWidth()/2,
BufferOriginal.getHeight()/2);
        AffineTransformOp op = new
AffineTransformOp(transform,

AffineTransformOp.TYPE_NEAREST_NEIGH
BOR);
        filter(op, 10);
      }
      else
      {
        AffineTransform transform =

AffineTransform.getRotateInstance(Math.toRadi
ans(270),
              BufferOriginal.getWidth()/2,
BufferOriginal.getWidth()/2);
        AffineTransformOp op = new
AffineTransformOp(transform,

AffineTransformOp.TYPE_NEAREST_NEIGH
BOR);
        filter(op, 11);
      }
    }

    private void convolve(float[] elements)
    {
          Kernel kernel = new Kernel(3, 3,
elements);
```

```
        ConvolveOp op = new
ConvolveOp(kernel);
        filter(op, 1);
    }

    private void filter(BufferedImageOp op, int
factor)
        {

         if(factor != 5 && factor != 6 &&
factor != 1 &&
                factor != 7 && factor != 8 &&
factor != 9
                && factor != 10 && factor != 11)
          {
            if(eszoom == 0)
            {
              BufferAnterior5 =
BufferAnterior4;
              BufferAnterior4 =
BufferAnterior3;
              BufferAnterior3 =
BufferAnterior2;
              BufferAnterior2 =
BufferAnterior1;
              BufferAnterior1 =
BufferAnterior;
              BufferAnterior =
mBufferedImage;
              BufferedImage filteredImage =
new BufferedImage(

BufferOriginal.getWidth()*factor,

BufferOriginal.getHeight()*factor,
                BufferOriginal.getType());
              try
              {

JOptionPane.showInternalMessageDialog(
                getContentPane(), "        La
Imágen es demasiado " +
                "grande por lo que no se puede
escalar al factor " +
                "seleccionado.",
                "Imagen No Escalable",

JOptionPane.INFORMATION_MESSAGE);
              }
              catch (RuntimeException e)
{return;}

              op.filter(BufferOriginal,
filteredImage);
              mBufferedImage = filteredImage;
            }
            else
            {
              BufferedImage filteredImage =
new BufferedImage(

BufferOriginal.getWidth()*factor,

BufferOriginal.getHeight()*factor,
                BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              mBufferedImage = filteredImage;
             // BufferPosterior =
mBufferedImage;
              eszoom = 0;
            }
          }

          else if(factor == 1)
          {
            if(eszoom == 0)
            {
              BufferAnterior5 =
BufferAnterior4;
              BufferAnterior4 =
BufferAnterior3;
              BufferAnterior3 =
BufferAnterior2;
              BufferAnterior2 =
BufferAnterior1;
              BufferAnterior1 =
BufferAnterior;
              BufferAnterior =
mBufferedImage;
              BufferedImage filteredImage =
new BufferedImage(
                BufferOriginal.getWidth(),
                BufferOriginal.getHeight(),
                BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              BufferOriginal = filteredImage;
              mBufferedImage = filteredImage;
            }
            else
            {
              BufferedImage filteredImage =
new BufferedImage(
                BufferOriginal.getWidth(),
                BufferOriginal.getHeight(),
                BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              mBufferedImage = filteredImage;
             // BufferOriginal = filteredImage;
```

```
                // BufferPosterior =
mBufferedImage;
                eszoom = 0;
              }
            }

            else if(factor == 5)
            {
              if(eszoom == 0)
              {
                BufferAnterior5 =
BufferAnterior4;
                BufferAnterior4 =
BufferAnterior3;
                BufferAnterior3 =
BufferAnterior2;
                BufferAnterior2 =
BufferAnterior1;
                BufferAnterior1 =
BufferAnterior;
                BufferAnterior =
mBufferedImage;
                BufferedImage filteredImage =
new BufferedImage(
                  BufferOriginal.getWidth()/4,
                  BufferOriginal.getHeight()/4,
                  BufferOriginal.getType());
                op.filter(BufferOriginal,
filteredImage);
                mBufferedImage = filteredImage;
              }
              else
              {
                BufferedImage filteredImage =
new BufferedImage(
                  BufferOriginal.getWidth()/4,
                  BufferOriginal.getHeight()/4,
                  BufferOriginal.getType());
                op.filter(BufferOriginal,
filteredImage);
                mBufferedImage = filteredImage;
               // BufferPosterior =
mBufferedImage;
                eszoom = 0;
              }
            }

            else if(factor == 6)
            {
              if(eszoom == 0)
              {
                BufferAnterior5 =
BufferAnterior4;
                BufferAnterior4 =
BufferAnterior3;
                BufferAnterior3 =
BufferAnterior2;
                BufferAnterior2 =
BufferAnterior1;
                BufferAnterior1 =
BufferAnterior;
                BufferAnterior =
mBufferedImage;
                BufferedImage filteredImage =
new BufferedImage(
                  BufferOriginal.getWidth()/2,

BufferOriginal.getHeight()/2,
                  BufferOriginal.getType());
                op.filter(BufferOriginal,
filteredImage);
                mBufferedImage = filteredImage;
              }
              else
              {

                BufferedImage filteredImage =
new BufferedImage(
                  BufferOriginal.getWidth()/2,

BufferOriginal.getHeight()/2,
                  BufferOriginal.getType());
                op.filter(BufferOriginal,
filteredImage);
                mBufferedImage = filteredImage;
                // BufferPosterior =
mBufferedImage;
                eszoom = 0;
              }
            }

            else if(factor == 7)
            {
            BufferedImage filteredImage = new
BufferedImage(

(int)(mBufferedImage.getWidth()*8),

          (int)(mBufferedImage.getHeight()*8),
                mBufferedImage.getType());
                  op.filter(mBufferedImage,
filteredImage);
                BufferDeZoom =
filteredImage;
            pintainicio = 3;
            }

            else if(factor == 8)
            {
            BufferedImage filteredImage = new
BufferedImage(
```

```
                mBufferedImage.getWidth(),

          mBufferedImage.getHeight(),
                mBufferedImage.getType());
                   op.filter(mBufferedImage,
filteredImage);
                   BufferDeZoom =
filteredImage;
          pintainicio = 3;
          }

          else if(factor == 9)
          {
            if(eszoom == 0)
            {
              BufferAnterior5 =
BufferAnterior4;
              BufferAnterior4 =
BufferAnterior3;
              BufferAnterior3 =
BufferAnterior2;
              BufferAnterior2 =
BufferAnterior1;
              BufferAnterior1 =
BufferAnterior;
              BufferAnterior =
mBufferedImage;
              BufferedImage filteredImage =
new BufferedImage(
                   BufferOriginal.getHeight(),
                   BufferOriginal.getWidth(),
                   BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              BufferOriginal = filteredImage;
              mBufferedImage = filteredImage;

            }
            else
            {
              BufferedImage filteredImage =
new BufferedImage(
                   BufferOriginal.getHeight(),
                   BufferOriginal.getWidth(),
                   BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              mBufferedImage = filteredImage;
             // BufferPosterior =
mBufferedImage;
              eszoom = 0;
            }
          }

          else if(factor == 10)
          {
            if(eszoom == 0)
            {
              BufferAnterior5 =
BufferAnterior4;
              BufferAnterior4 =
BufferAnterior3;
              BufferAnterior3 =
BufferAnterior2;
              BufferAnterior2 =
BufferAnterior1;
              BufferAnterior1 =
BufferAnterior;
              BufferAnterior =
mBufferedImage;
              BufferedImage filteredImage =
new BufferedImage(
                   BufferOriginal.getWidth(),
                   BufferOriginal.getHeight(),
                   BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              BufferOriginal = filteredImage;
              mBufferedImage  =
filteredImage;
            }
            else
            {
              BufferedImage filteredImage =
new BufferedImage(
                   BufferOriginal.getWidth(),
                   BufferOriginal.getHeight(),
                   BufferOriginal.getType());
              op.filter(BufferOriginal,
filteredImage);
              mBufferedImage = filteredImage;
             // BufferPosterior =
mBufferedImage;
              eszoom = 0;
            }
          }
          else if(factor == 11)
          {
            if(eszoom == 0)
            {
              BufferAnterior5 =
BufferAnterior4;
              BufferAnterior4 =
BufferAnterior3;
              BufferAnterior3 =
BufferAnterior2;
              BufferAnterior2 =
BufferAnterior1;
              BufferAnterior1 =
BufferAnterior;
              BufferAnterior =
mBufferedImage;
```

```
                BufferedImage filteredImage =
new BufferedImage(
                    BufferOriginal.getHeight(),
                    BufferOriginal.getWidth(),
                    BufferOriginal.getType());
                op.filter(BufferOriginal,
filteredImage);
                BufferOriginal = filteredImage;
                mBufferedImage = filteredImage;
              }
              else
              {
                BufferedImage filteredImage =
new BufferedImage(
                    BufferOriginal.getHeight(),
                    BufferOriginal.getWidth(),
                    BufferOriginal.getType());
                op.filter(BufferOriginal,
filteredImage);
                mBufferedImage = filteredImage;
              // BufferPosterior =
mBufferedImage;
                eszoom = 0;
              }
            }
          }
////////////////////////////////////
  public void recortar()
  {
    recorte = 1;
    puntogeoref = 0;
    transecto = 0;
    fuera = 0;
    pintainicio = 0;
    regionDeZoom = 0;
    haciendoZoom = 0;
    mBufferedImage = BufferOriginal;
    repaint();
  }

  public void puntogeo()
  {
    puntogeoref = 1;
    recorte = 0;
    transecto = 0;
    fuera = 0;
    regionDeZoom = 0;
    haciendoZoom = 0;
    mBufferedImage = BufferOriginal;
    pintainicio = 0;
    repaint();
  }

  public void transecto()
  {
    transecto = 1;
    puntogeoref = 0;
    recorte = 0;
    fuera = 0;
    regionDeZoom = 0;
    haciendoZoom = 0;
    mBufferedImage = BufferOriginal;
    pintainicio = 0;
    repaint();
  }


  public void mtamaño(int tamaño)
  {
    pintainicio = 0;
    regionDeZoom = 0;
    haciendoZoom = 0;
    repaint();
    if(tamaño == 0)
    {
        factor = 1;
        multdiv = 1;
        AffineTransform transform =
        AffineTransform.getScaleInstance(1,
1);
        AffineTransformOp op = new
AffineTransformOp(transform,

      AffineTransformOp.TYPE_BILINEAR
);
              filter(op, factor);
    }
    else if(tamaño == 1)
    {
        factor = 5;
        multdiv = 0.25;
        AffineTransform transform =

AffineTransform.getScaleInstance(0.25, 0.25);
              AffineTransformOp op = new
AffineTransformOp(transform,

      AffineTransformOp.TYPE_BILINEAR
);
              filter(op, factor);
    }
    else if(tamaño == 2)
    {
        factor = 6;
        multdiv = 0.5;
        AffineTransform transform =
        AffineTransform.getScaleInstance(0.5,
0.5);
              AffineTransformOp op = new
AffineTransformOp(transform,
```

```
            AffineTransformOp.TYPE_BILINEAR
);
                        filter(op, factor);
        }
        else if(tamaño == 3)
        {
            factor = 2;
            multdiv = 2;
            AffineTransform transform =
            AffineTransform.getScaleInstance(2,
2);
                        AffineTransformOp op = new
AffineTransformOp(transform,

            AffineTransformOp.TYPE_BILINEAR
);
                        filter(op, factor);
        }
        else if(tamaño == 6 && escalar == 1)
        {
            factor = 6;
            multdiv = 0.5;
            AffineTransform transform =
            AffineTransform.getScaleInstance(0.5,
0.5);
                        AffineTransformOp op = new
AffineTransformOp(transform,

            AffineTransformOp.TYPE_BILINEAR
);
                        filter(op, factor);
        }
        else if(tamaño == 7 && escalar == 2)
        {
            factor = 5;
            multdiv = 0.25;
            AffineTransform transform =

AffineTransform.getScaleInstance(0.25, 0.25);
                        AffineTransformOp op = new
AffineTransformOp(transform,

            AffineTransformOp.TYPE_BILINEAR
);
                        filter(op, factor);
        }
        else
        {
            multdiv = 4;
            factor = 4;
            AffineTransform transform =
            AffineTransform.getScaleInstance(4,
4);
                        AffineTransformOp op = new
AffineTransformOp(transform,
            AffineTransformOp.TYPE_BILINEAR
);
            try
            {

JOptionPane.showInternalMessageDialog(
                getContentPane(), "          La
Imágen es demasiado " +
                "grande por lo que no se puede
escalar al factor " +
                "seleccionado.",
                "Imagen No Escalable",

JOptionPane.INFORMATION_MESSAGE);
            }
            catch (RuntimeException e) {return;}

                        filter(op, factor);
        }
    }
//////////
    public void zoomin(double valormas)
    {
        if(mBufferedImage.getHeight() > 300 ||
mBufferedImage.getWidth() > 300
            && regionDeZoom == 0)
        {

JOptionPane.showInternalMessageDialog(getCo
ntentPane(), "Debe seleccionar " +
                "un área para poder realizar el Zoom
(el área no podrá ser mayor a 200x200)",
                "Zoom",
JOptionPane.INFORMATION_MESSAGE);
            System.out.println("Es más grande");
            haciendoZoom = 1;
        }
        else{}
            eszoom = 1;
            double val1 = (valormas/100);
            factor = 7;
            AffineTransform transform =

AffineTransform.getScaleInstance(val1, val1);
                        AffineTransformOp op = new
AffineTransformOp(transform,

            AffineTransformOp.TYPE_NEAREST
_NEIGHBOR);
                        filter(op, factor);
    }
    public void zoomout(double valormenos)
    {
        if(mBufferedImage.getHeight() > 300 ||
mBufferedImage.getWidth() > 300
```

```
            && regionDeZoom == 0)
        {

JOptionPane.showInternalMessageDialog(getCo
ntentPane(), "Debe seleccionar " +
                "un área para poder realizar el Zoom
(el área no podrá ser mayor a 200x200)",
                "Zoom",
JOptionPane.INFORMATION_MESSAGE);
            haciendoZoom = 1;
        }
        else{}
            eszoom=1;
            double val2 = (valormenos/100);
            if(valormenos > 10)
            {
               factor = 8;
               AffineTransform transform =

AffineTransform.getScaleInstance(val2, val2);
               AffineTransformOp op = new
AffineTransformOp(transform,

            AffineTransformOp.TYPE_NEAREST
_NEIGHBOR);
               filter(op, factor);
            }
            else{}
     }
     public void verHistograma()
     {

JOptionPane.showInternalMessageDialog(getCo
ntentPane(), "Espere un moneto " +
          "mientras se genera el Histograma.\n(esta
operacion puede tardar algunos minutos)",
          "Histograma",
JOptionPane.INFORMATION_MESSAGE);

          Histograma histo = new
Histograma(mBufferedImage);
          regionDeZoom = 0;
          pintainicio = 0;
          repaint();

     }
//////////////////////
     public void mouseClicked(MouseEvent
mousee) {
       if(puntogeoref == 1)
       {
        if(mousee.getX() <
mBufferedImage.getWidth() &&
              mousee.getY() <
mBufferedImage.getHeight())
        {
          puntoX = mousee.getX();
          puntoY = mousee.getY();
          puntoselec = 1;
        }
        else
        {
          puntogeoref = 0;
          fuera = 1;
        }
       }
       else{}


     }
     public void mouseDragged(MouseEvent
mousee) {
       if(recorte == 1 && fuera == 0)
       {
          if(mousee.getX() >
mBufferedImage.getWidth() &&
              mousee.getY() <
mBufferedImage.getHeight())
          {
            posX2 = mBufferedImage.getWidth();
            posY2 = mousee.getY();
            repaint();
          }

          else if(mousee.getX() <
mBufferedImage.getWidth() &&
              mousee.getY() >
mBufferedImage.getHeight())
          {
            posX2 = mousee.getX();
            posY2 = mBufferedImage.getHeight();
            repaint();
          }
          else if(mousee.getX() >
mBufferedImage.getWidth() &&
              mousee.getY() >
mBufferedImage.getHeight())
          {
            posX2 = mBufferedImage.getWidth();
            posY2 = mBufferedImage.getHeight();
            repaint();
          }
          else if(mousee.getX() <
mBufferedImage.getWidth() &&
              mousee.getY() <
mBufferedImage.getHeight())
          {
            posX2 = mousee.getX();
            posY2 = mousee.getY();
            repaint();
          }
          else{}
```

```
    }

    if(transecto == 1 && fuera == 0)
    {
      if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
      {
        pospixelX2 =
mBufferedImage.getWidth();
        pospixelY2 = mousee.getY();
        repaint();
      }

      else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
      {
        pospixelX2 = mousee.getX();
        pospixelY2 =
mBufferedImage.getHeight();
        repaint();
      }
      else if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
      {
        pospixelX2 =
mBufferedImage.getWidth();
        pospixelY2 =
mBufferedImage.getHeight();
        repaint();
      }
      else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
      {
        pospixelX2 = mousee.getX();
        pospixelY2 = mousee.getY();
        repaint();
      }
      else{}
    }

    if(haciendoZoom == 1 && fuera == 0)
    {
      if(Math.abs(mousee.getX()-posX1) < 200
&& Math.abs(mousee.getY()-posY1) < 200)
      {
        if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
        {
          posX2 = mBufferedImage.getWidth();
          posY2 = mousee.getY();
          repaint();
        }

        else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
        {
          posX2 = mousee.getX();
          posY2 = mBufferedImage.getHeight();
          repaint();
        }
        else if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
        {
          posX2 = mBufferedImage.getWidth();
          posY2 = mBufferedImage.getHeight();
          repaint();
        }
        else
        {
          posX2 = mousee.getX();
          posY2 = mousee.getY();
          repaint();
        }
      }
      else if(Math.abs(mousee.getX()-posX1) >
200 && Math.abs(mousee.getY()-posY1) < 200)
      {
        if(posX2 > posX1)
        {
          posX2 = posX1+200;
        }
        else
        {
          posX2 = posX1-200;
        }
        posY2 = mousee.getY();
        if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
        {
          posX2 = mBufferedImage.getWidth();
          posY2 = mousee.getY();
          repaint();
        }
```

```
        else if(mousee.getX() <
mBufferedImage.getWidth() &&
                mousee.getY() >
mBufferedImage.getHeight())
        {
            posX2 = mousee.getX();
            posY2 =
mBufferedImage.getHeight();
             repaint();
        }
        else if(mousee.getX() >
mBufferedImage.getWidth() &&
                mousee.getY() >
mBufferedImage.getHeight())
        {
            posX2 =
mBufferedImage.getWidth();
            posY2 =
mBufferedImage.getHeight();
            repaint();
        }
        else{}
      repaint();
    }

    else if(Math.abs(mousee.getX()-posX1) <
200 && Math.abs(mousee.getY()-posY1) > 200)
    {
      if(posY2 > posY1)
      {
        posY2 = posY1+200;
      }
      else
      {
        posY2 = posY1-200;
      }
      posX2 = mousee.getX();
      if(mousee.getX() >
mBufferedImage.getWidth() &&
         mousee.getY() <
mBufferedImage.getHeight())
      {
        posX2 = mBufferedImage.getWidth();
        posY2 = mousee.getY();
        repaint();
      }

     else if(mousee.getX() <
mBufferedImage.getWidth() &&
          mousee.getY() >
mBufferedImage.getHeight())
      {
          posX2 = mousee.getX();
          posY2 =
mBufferedImage.getHeight();
          repaint();
      }
      else if(mousee.getX() >
mBufferedImage.getWidth() &&
          mousee.getY() >
mBufferedImage.getHeight())
      {
          posX2 = mBufferedImage.getWidth();
          posY2 =
mBufferedImage.getHeight();
          repaint();
      }

      else{}

      repaint();
    }
    else if(Math.abs(mousee.getX()-posX1) >
200 && Math.abs(mousee.getY()-posY1) > 200)
    {
      if(posY2 > posY1)
      {
        posY2 = posY1+200;
      }
      else
      {
        posY2 = posY1-200;
      }
      if(posX2 > posX1)
      {
        posX2 = posX1+200;
      }
      else
      {
        posX2 = posX1-200;
      }
      if(mousee.getX() >
mBufferedImage.getWidth() &&
         mousee.getY() <
mBufferedImage.getHeight())
      {
          posX2 =
mBufferedImage.getWidth();
          posY2 = mousee.getY();
          repaint();
      }

     else if(mousee.getX() <
mBufferedImage.getWidth() &&
          mousee.getY() >
mBufferedImage.getHeight())
      {
          posX2 = mousee.getX();
          posY2 =
mBufferedImage.getHeight();
          repaint();
      }
```

```
        else if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
        {
            posX2 =
mBufferedImage.getWidth();
            posY2 =
mBufferedImage.getHeight();
            repaint();
        }
        else{}
        repaint();
      }
      else{}
    }
  }
  public void mouseEntered(MouseEvent
mousee) {
    if(recorte == 1)
    {
      setCursor(new
Cursor(Cursor.CROSSHAIR_CURSOR));
    }

    if(puntogeoref == 1)
    {
      //Cursor
custom=Toolkit.getDefaultToolkit().createCusto
mCursor(new ImageIcon("pen.gif").getImage(),
new Point(6, 15), "MyCursor");
      //setCursor(custom);
      setCursor(new
Cursor(Cursor.HAND_CURSOR));
    }
    if(transecto == 1)
    {
      setCursor(new
Cursor(Cursor.HAND_CURSOR));
    }
    if(haciendoZoom == 1)
    {
      setCursor(new
Cursor(Cursor.MOVE_CURSOR));
    }
  }
  public void mouseExited(MouseEvent mousee)
{
    if(recorte == 1 || puntogeoref == 1 || transecto
== 1 || haciendoZoom == 1)
    {
      setCursor(new
Cursor(Cursor.DEFAULT_CURSOR));
    }
  }
  public void mouseMoved(MouseEvent mousee)
{
    if(recorte == 0 && puntogeoref == 0 &&
transecto == 0 && haciendoZoom == 0)
    {
      setCursor(new
Cursor(Cursor.DEFAULT_CURSOR));
    }
  }
  public void mousePressed(MouseEvent
mousee) {
    if(recorte == 1)
    {
      if(mousee.getX() <
mBufferedImage.getWidth() &&
          mousee.getY() <
mBufferedImage.getHeight())
      {
        //System.out.println("se presiono en:"+
mousee.getX() + "," + mousee.getY());
        posX1 = mousee.getX();
        posY1 = mousee.getY();
        fuera = 0;
        repaint();
      }
      else
      {
        fuera = 1;
        recorte = 0;
      }
    }

    if(transecto == 1)
    {
      if(mousee.getX() <
mBufferedImage.getWidth() &&
          mousee.getY() <
mBufferedImage.getHeight())
      {
        //System.out.println("se presiono en:"+
mousee.getX() + "," + mousee.getY());
        pospixelX1 = mousee.getX();
        pospixelY1 = mousee.getY();
        fuera = 0;
        repaint();
      }
      else
      {
        fuera = 1;
        transecto = 0;
      }
    }
    if(haciendoZoom == 1)
    {
      if(mousee.getX() <
mBufferedImage.getWidth() &&
```

```
            mousee.getY() <
mBufferedImage.getHeight())
      {
       //System.out.println("se presiono en:"+
mousee.getX() + "," + mousee.getY());
       posX1 = mousee.getX();
       posY1 = mousee.getY();
       fuera = 0;
       repaint();
      }
      else
      {
        fuera = 1;
      }
   }
  }
  public void mouseReleased(MouseEvent
mousee) {
    if(recorte == 1 && fuera == 0)
    {
      if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        posX2 = mBufferedImage.getWidth();
        posY2 = mousee.getY();
        ratonsuelto = 1;
        repaint();
      }

      else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        posX2 = mousee.getX();
        posY2 = mBufferedImage.getHeight();
        ratonsuelto = 1;
        repaint();
      }
      else if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        posX2 = mBufferedImage.getWidth();
        posY2 = mBufferedImage.getHeight();
        ratonsuelto = 1;
        repaint();
      }
      else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        posX2 = mousee.getX();
        posY2 = mousee.getY();
        ratonsuelto = 1;
        repaint();
      }
      else{}
    }
    if(transecto == 1 && fuera == 0)
    {
      if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() <
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        pospixelX2 =
mBufferedImage.getWidth();
        pospixelY2 = mousee.getY();
        ratonsuelto = 1;
        repaint();
      }

      else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        pospixelX2 = mousee.getX();
        pospixelY2 =
mBufferedImage.getHeight();
        ratonsuelto = 1;
        repaint();
      }
      else if(mousee.getX() >
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        pospixelX2 =
mBufferedImage.getWidth();
        pospixelY2 =
mBufferedImage.getHeight();
        ratonsuelto = 1;
```

```
        repaint();
      }
      else if(mousee.getX() <
mBufferedImage.getWidth() &&
           mousee.getY() <
mBufferedImage.getHeight())
      {
        //System.out.println("se solto en:"+
mousee.getX() + "," + mousee.getY());
        pospixelX2 = mousee.getX();
        pospixelY2 = mousee.getY();
        ratonsuelto = 1;
        repaint();
      }
      else{}
    }
    if(haciendoZoom == 1 && fuera == 0)
    {
      if(Math.abs(mousee.getX()-posX1) < 200
&& Math.abs(mousee.getY()-posY1) < 200)
      {
        if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() <
mBufferedImage.getHeight())
        {
          posX2 = mBufferedImage.getWidth();
          posY2 = mousee.getY();
          repaint();
        }

        else if(mousee.getX() <
mBufferedImage.getWidth() &&
            mousee.getY() >
mBufferedImage.getHeight())
        {
           posX2 = mousee.getX();
           posY2 = mBufferedImage.getHeight();
          repaint();
        }
        else if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
        {
           posX2 = mBufferedImage.getWidth();
           posY2 = mBufferedImage.getHeight();
           repaint();
        }
        else
        {
           posX2 = mousee.getX();
           posY2 = mousee.getY();
           repaint();
        }
      }
      else if(Math.abs(mousee.getX()-posX1) >
200 && Math.abs(mousee.getY()-posY1) < 200)
      {
        if(posX2 > posX1)
        {
          posX2 = posX1+200;
        }
        else
        {
          posX2 = posX1-200;
        }
        posY2 = mousee.getY();
        if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() <
mBufferedImage.getHeight())
        {
          posX2 = mBufferedImage.getWidth();
          posY2 = mousee.getY();
          repaint();
        }

        else if(mousee.getX() <
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
        {
           posX2 = mousee.getX();
           posY2 =
mBufferedImage.getHeight();
           repaint();
        }
        else if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
        {
           posX2 =
mBufferedImage.getWidth();
           posY2 =
mBufferedImage.getHeight();
           repaint();
        }
        else{}
       repaint();
      }

      else if(Math.abs(mousee.getX()-posX1) <
200 && Math.abs(mousee.getY()-posY1) > 200)
      {
        if(posY2 > posY1)
        {
          posY2 = posY1+200;
        }
        else
        {
```

```
            posY2 = posY1-200;
          }
          posX2 = mousee.getX();
          if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() <
mBufferedImage.getHeight())
          {
            posX2 = mBufferedImage.getWidth();
            posY2 = mousee.getY();
            repaint();
          }

          else if(mousee.getX() <
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
          {
             posX2 = mousee.getX();
             posY2 =
mBufferedImage.getHeight();
             repaint();
          }
          else if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
          {
             posX2 = mBufferedImage.getWidth();
             posY2 =
mBufferedImage.getHeight();
             repaint();
          }

          else{}

          repaint();
        }
        else if(Math.abs(mousee.getX()-posX1) >
200 && Math.abs(mousee.getY()-posY1) > 200)
        {
          if(posY2 > posY1)
          {
            posY2 = posY1+200;
          }
          else
          {
            posY2 = posY1-200;
          }
          if(posX2 > posX1)
          {
            posX2 = posX1+200;
          }
          else
          {
            posX2 = posX1-200;
          }
          if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() <
mBufferedImage.getHeight())
          {
             posX2 =
mBufferedImage.getWidth();
             posY2 = mousee.getY();
             repaint();
          }

          else if(mousee.getX() <
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
          {
             posX2 = mousee.getX();
             posY2 =
mBufferedImage.getHeight();
             repaint();
          }
          else if(mousee.getX() >
mBufferedImage.getWidth() &&
             mousee.getY() >
mBufferedImage.getHeight())
          {
             posX2 =
mBufferedImage.getWidth();
             posY2 =
mBufferedImage.getHeight();
             repaint();
          }
          else{}
          repaint();
        }
        else{}
        ratonsuelto = 1;
        repaint();
      }
    }

    ////////////////////////////////////////////////////
     public void configurarPagina()
     {
       // Get a PrinterJob
       job = PrinterJob.getPrinterJob();
       // Ask user for page format (e.g.,
portrait/landscape)
       pf = job.pageDialog(job.defaultPage());
       prefImp=1;
     }

     public void imprimir()
     {
       if(prefImp == 1)
```

```
    {
      job.setPrintable(this, pf);
      if(job.printDialog())
      {
        System.out.println("hasta aqui llego
1");
        try
        {
          job.print();
          System.out.println("hasta aqui llego
2");
        }
        catch(PrinterException pe)
        {
          System.out.println(pe);
        }
      }
    }
    else
    {
      PrinterJob pj =
PrinterJob.getPrinterJob();
      pj.setPrintable(this);
      if(pj.printDialog())
      {
        System.out.println("hasta aqui llego
1");
        try
        {
          pj.print();
          System.out.println("hasta aqui llego
2");
        }
        catch(PrinterException pe)
        {
          System.out.println(pe);
        }
      }
    }
  }

  public int print(Graphics g, PageFormat pf, int
pageIndex) throws
      java.awt.print.PrinterException
  {
    if(pageIndex !=0) return NO_SUCH_PAGE;
    Graphics2D g2 = (Graphics2D)g;
    g2.translate(pf.getImageableX(),
pf.getImageableY());
    paint(g2);
    System.out.println("hasta aqui llego 3");
    return PAGE_EXISTS;
  }
////////////////////////////////////////////
  public void anterior()
  {
    BufferAnterior5 = BufferPosterior4;
    BufferPosterior4 = BufferPosterior3;
    BufferPosterior3 = BufferPosterior2;
    BufferPosterior2 = BufferPosterior1;
    BufferPosterior1 = BufferPosterior;
    BufferPosterior  = mBufferedImage;
    mBufferedImage = BufferAnterior;
    BufferOriginal = BufferAnterior;
    BufferAnterior = BufferAnterior1;
    BufferAnterior1 = BufferAnterior2;
    BufferAnterior2 = BufferAnterior3;
    BufferAnterior3 = BufferAnterior4;
    BufferAnterior4 = BufferAnterior5;


    pintainicio = 0;
    repaint();
  }
////////////////////////////////////////////////////////////
  public void posterior()
  {
    BufferAnterior5 = BufferAnterior4;
    BufferAnterior4 = BufferAnterior3;
    BufferAnterior3 = BufferAnterior2;
    BufferAnterior2 = BufferAnterior1;
    BufferAnterior1 = BufferAnterior;
    BufferAnterior = mBufferedImage;
    mBufferedImage = BufferPosterior;
    BufferOriginal = BufferPosterior;
    BufferPosterior = BufferPosterior1;
    BufferPosterior1 = BufferPosterior2;
    BufferPosterior2 = BufferPosterior3;
    BufferPosterior3 = BufferPosterior4;
    BufferPosterior4 = BufferPosterior5;
    pintainicio = 0;
    repaint();
  }

}

}


class Miniatura extends JComponent implements
PropertyChangeListener
{
    ImageIcon thumbnail = null;
    File f = null;
    public Miniatura(JFileChooser fc)
    {
      setPreferredSize(new Dimension(200,
100));
      fc.addPropertyChangeListener(this);
    }

    public void loadImage()
```

```
    {
      if(f != null)
      {
        ImageIcon tmpIcon = new
          ImageIcon(f.getPath());
        thumbnail = new ImageIcon(

tmpIcon.getImage().getScaledInstance
            (90, -1,
Image.SCALE_DEFAULT));
      }
    }

    public void
propertyChange(PropertyChangeEvent e)
    {
      String prop = e.getPropertyName();
      if(prop == JFileChooser.

SELECTED_FILE_CHANGED_PROPERTY)
      {
        f = (File) e.getNewValue();
        loadImage();
        repaint();
      }
    }
    public void paint(Graphics g)
    {
      if(thumbnail == null)
      {
        loadImage();
      }
      if(thumbnail != null)
      {
        int x = getWidth()/2 -
          thumbnail.getIconWidth()/2;
        int y = getHeight()/2 -
          thumbnail.getIconHeight()/2;
        if(y < 0)
        {
          y = 0;
        }
        if(x < 5)
        {
          x = 5;
        }
        thumbnail.paintIcon(this, g, x, y);
      }
    }
}
```

//Histograma.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import javax.swing.border.Border;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;

public class Histograma extends JFrame
{
   Image imagenNueva;
   BufferedImage bufferFuente;
   BufferedImage bufer;
   private MostrarHisto mostrar;



  public void main(String[] args)
  {
     Histograma obj = new
Histograma(bufferFuente);
     obj.repaint();

  }


   public Histograma(BufferedImage imagen)
   {
      bufferFuente = imagen;
      mostrar = new MostrarHisto(bufferFuente);
      JScrollPane jsp = new JScrollPane(mostrar,
JScrollPane.

VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.

HORIZONTAL_SCROLLBAR_ALWAYS);
      Border brd =
BorderFactory.createMatteBorder(40, 35, 40, 35,
Color.getColor("azul", -16490900));
      jsp.setBorder(brd);
      getContentPane().add(jsp);
      JButton botonImprimir = new
JButton("Imprimir");
      getContentPane().add(botonImprimir,
BorderLayout.SOUTH);
      botonImprimir.addActionListener(new
ActionListener()
         {
            public void
 actionPerformed(ActionEvent ev)
            {
             mostrar.imprimir();
            }
          });
      JButton botonGuardar = new
JButton("Guardar Histograma");
      getContentPane().add(botonGuardar,
BorderLayout.NORTH);
      botonGuardar.addActionListener(new
ActionListener()
         {
           public void
actionPerformed(ActionEvent ev)
            {
             mostrar.guardarComoImagen();
            }
          });
      setTitle("Histograma");

setIconImage(Toolkit.getDefaultToolkit().getIma
ge("Icmyl.gif"));
      setSize(800, 600);
      centrar();
      addWindowListener(new WindowAdapter()
      {
        public void
windowClosing(WindowEvent e)
        {
                  setVisible(false);
        }
         });
      setVisible(true);
   }


   protected void centrar()
   {
      Dimension screen =
Toolkit.getDefaultToolkit().getScreenSize();
      Dimension us = getSize();
      int x = (screen.width - us.width) / 2;
      int y = (screen.height - us.height) / 2;
      setLocation(x, y);
   }


}


////////////////////////////////////////////////////////////////
////////

class MostrarHisto extends JPanel implements
Printable
```

```
{
   public BufferedImage buffer;
   public BufferedImage BufferDeHistograma;
   public int contador, i, j, k, l, m, n, posición=0,
         aproximado, diferencia, coordenada;
   public int[] paleta = new int[258];
   public int[] tinferiorX = {650, 650, 660};
   public int[] tinferiorY = {495, 505, 500};
   public int[] tsuperiorX = {95, 100, 105};
   public int[] tsuperiorY = {100, 90, 100};
   public double escala;
   public double función;
//   public Barra mostrar;
   public BufferedImage BufferDeEscala;
   public  int returnVal;
   public  String nombreNuevo;


   MostrarHisto(BufferedImage laImagen)
   {
  // new Barra();
    buffer = laImagen;
    BufferDeHistograma = new
BufferedImage(711, 650,

BufferedImage.TYPE_INT_BGR);

              for(k=0; k <
BufferDeHistograma.getHeight(); k++)
              {
                 for(l=0; l <
BufferDeHistograma.getWidth(); l++)
                 {

BufferDeHistograma.setRGB(l,k,-1);
                 }
              }
    Image escala =
Toolkit.getDefaultToolkit().getImage("Paleta256
.jpg");
    MediaTracker m = new MediaTracker(this);
    m.addImage(escala, 0);
    try { m.waitForID(0); }
    catch (InterruptedException ie) { return; }
    if (m.isErrorID(0)) return;

    BufferDeEscala = new BufferedImage(1,258,

BufferedImage.TYPE_INT_BGR);
    Graphics2D g8 =
BufferDeEscala.createGraphics();
    g8.drawImage(escala, 0, 0, this);
    BufferDeEscala.setRGB(0, 256, -5328189);
    BufferDeEscala.setRGB(0, 257, -16777216);

    conteo();
   }


   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      g.drawImage(BufferDeHistograma, 0, 0,
this);
      g.drawLine(100, 500, 650, 500);
      g.fillPolygon(tinferiorX, tinferiorY,3);
      g.drawLine(100, 500, 100, 100);
      g.fillPolygon(tsuperiorX, tsuperiorY,3);
      g.drawString("frecuencia", 20, 100);
      g.drawString("Color", 650, 515);
      g.drawLine(356, 500, 356, 505);
      g.drawString("128",348,515);
      g.drawLine(612, 500, 612, 505);
      g.drawString("256",604,515);
      g.drawString("0",95,515);
      repaint();


   }

   public Dimension getPreferredSize()
   {
      return new Dimension(800, 600);
   }

   public void conteo()
   {
      setCursor(new
Cursor(Cursor.WAIT_CURSOR));

        for(k = 0; k < buffer.getHeight(); k++)
        {
          for(l = 0; l < buffer.getWidth(); l++)
          {
            aproximado = 17000000;
            for(i=0; i < 258; i++)
            {
               diferencia =
Math.abs(BufferDeEscala.getRGB(0,i) -
buffer.getRGB(l,k));
               if(diferencia < aproximado)
               {
                  coordenada = i;
                  aproximado = diferencia;
               }
               else{}
            }
            paleta[coordenada]=
paleta[coordenada]+1;
          }
        }
```

```
      for(m=255; m > 0; m--)
      {
        posición = posición+2;
        función =
((paleta[m]*3984)/(buffer.getWidth()*buffer.get
Height()))+1;
        for(n = 0; n < (int)función; n++)
        {
          if(n < 499)
          {
          BufferDeHistograma.setRGB(posición
+ 100,499-n, BufferDeEscala.getRGB(0,m) );
          BufferDeHistograma.setRGB(posición
+ 1 + 100,499-n, BufferDeEscala.getRGB(0,m)
);
          }
        else{}
        }
      }
      setCursor(new
Cursor(Cursor.DEFAULT_CURSOR));
   }
////////////////////////////////////////////////////////////////
////////
  public void imprimir()
  {
      PrinterJob pj = PrinterJob.getPrinterJob();
      pj.setPrintable(this);
      if(pj.printDialog())
      {
        System.out.println("hasta aqui llego 1");
        try
        {
          pj.print();
          System.out.println("hasta aqui llego
2");
        }
        catch(PrinterException pe)
        {
          System.out.println(pe);
        }
      }
  }

  public int print(Graphics g, PageFormat pf, int
pageIndex) throws
        java.awt.print.PrinterException
  {
      if(pageIndex !=0) return NO_SUCH_PAGE;
      Graphics2D g2 = (Graphics2D)g;
      g2.translate(pf.getImageableX(),
pf.getImageableY());
      paint(g2);
      System.out.println("hasta aqui llego 3");
      return PAGE_EXISTS;
  }
////////////////////////////////////////////////////////////////
////////
  public void guardarComoImagen()
   {
            JFileChooser eleccion = new
JFileChooser();

        eleccion.setCurrentDirectory(new
File("."));
            FilePreviewer previewer = new
FilePreviewer(eleccion);
            eleccion.setAccessory(previewer);

        eleccion.setFileFilter(new

        javax.swing.filechooser.FileFilter()
                                {

        public boolean accept(File f)
                                        {

        String name =
f.getName().toLowerCase();

        return name.endsWith(".gif")

        || name.endsWith(".jpg")

        || name.endsWith(".jpeg")

        || f.isDirectory();

                                        }

        public String getDescription()
                                        {

        return "Archivos de Imágen (*.gif,
*.jpg, *.jpeg)";
                                        }
                                });
            returnVal =
eleccion.showSaveDialog(null);
            if(returnVal ==
JFileChooser.APPROVE_OPTION)
            {
              if (eleccion.getSelectedFile() ==
null)
                return;
            File file =
eleccion.getSelectedFile();
            nombreNuevo = file.getPath();
            Graphics2D g2 =
BufferDeHistograma.createGraphics();
            g2.dispose();
```

```
                try
                {

ImageIO.write(BufferDeHistograma, "jpg",
                        new File(nombreNuevo));
                }
                catch(IOException ioe)
                {

System.out.println(ioe.getMessage());
                }
            }
    }


}
```

//Transecto.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableModel;
import javax.swing.JTable;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;



public class Transecto extends JFrame
implements Printable{

   private PagingModel pm;
   public JTable table;
   public Transecto(BufferedImage
BufferDeTransecto)
   {
////////////////////////////////////////////////////////////////
////////
      Image temperatura =
Toolkit.getDefaultToolkit().getImage("reftempe.
jpg");
      MediaTracker t = new MediaTracker(this);
      t.addImage(temperatura, 0);
      try { t.waitForID(0); }
      catch (InterruptedException ie) { return; }
      if (t.isErrorID(0)) return;

      BufferedImage BufferDeTemperatura =
new BufferedImage(74,973,

BufferedImage.TYPE_INT_BGR);
      Graphics2D g9 =
BufferDeTemperatura.createGraphics();
      g9.drawImage(temperatura, 0, 0, this);
////////////////////////////////////////////////////////////////
////////
      pm = new
PagingModel(BufferDeTransecto.getWidth(),
            BufferDeTransecto.getTileWidth(),
BufferDeTransecto,
            BufferDeTemperatura);
      table = new JTable(pm);
      //MostrarTabla mostrar = new
MostrarTabla(BufferDeTransecto);
      JScrollPane jsp = new JScrollPane(table,
JScrollPane.

VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.

HORIZONTAL_SCROLLBAR_ALWAYS);
      Border brd =
BorderFactory.createMatteBorder(20, 15, 20, 15,
Color.getColor("azul", -16490900));
      jsp.setBorder(brd);
      getContentPane().add(jsp);
      JButton botonImprimir = new
JButton("Imprimir");
      getContentPane().add(botonImprimir,
BorderLayout.SOUTH);
      botonImprimir.addActionListener(new
ActionListener()
         {
           public void
actionPerformed(ActionEvent ev)
           {
           imprimir();
           }
         });
      setTitle("Transecto");

setIconImage(Toolkit.getDefaultToolkit().getIma
ge("Icmyl.gif"));
      setSize(300, 400);
      centrar();
      addWindowListener(new WindowAdapter()
      {
        public void
windowClosing(WindowEvent e)
        {
                  setVisible(false);
        }
         });
      setVisible(true);

   }
////////////////////////////////////////////////////////////////
////////
  public void imprimir()
  {
      PrinterJob pj = PrinterJob.getPrinterJob();
      pj.setPrintable(Transecto.this);
      if(pj.printDialog())
      {
        System.out.println("hasta aqui llego 1");
        try
        {
          pj.print();
          System.out.println("hasta aqui llego
2");
```

```
        }
        catch(PrinterException pe)
        {
          System.out.println(pe);
        }
      }
  }

  public int print(Graphics g, PageFormat pf, int
pageIndex) throws
        java.awt.print.PrinterException
  {
     if(pageIndex !=0) return NO_SUCH_PAGE;
     Graphics2D g2 = (Graphics2D)g;
     g2.translate(pf.getImageableX(),
pf.getImageableY());
     //paint(g2);
     System.out.println("hasta aqui llego 3");
     return PAGE_EXISTS;
  }
////////////////////////////////////////////////////////////////
///////


    protected void centrar()
    {
       Dimension screen =
Toolkit.getDefaultToolkit().getScreenSize();
       Dimension us = getSize();
       int x = (screen.width - us.width) / 2;
       int y = (screen.height - us.height) / 2;
       setLocation(x, y);
    }

}


class ObtenerTransecto extends
AbstractTableModel{

  protected int pageSize;

  protected int pageOffset;

  protected  Record[] data;

  protected BufferedImage Transecto;


  public ObtenerTransecto(int numRows, int size,
BufferedImage BufferDeTransecto,
       BufferedImage BufferDeTemperatura) {
    data = new Record[numRows];
    pageSize = size;

    int  aproximado, diferencia;

    BufferedImage Transecto =
BufferDeTransecto;

    int[] valores = new int[500];

    for(int i = 0; i < Transecto.getWidth(); i++)
    {
          aproximado = 17000000;
          for(int j = 0; j <
BufferDeTemperatura.getHeight(); j++ )
          {
            for(int k = 0; k <
BufferDeTemperatura.getWidth(); k++)
            {
              diferencia =
Math.abs(BufferDeTemperatura.getRGB(k,j) -
Transecto.getRGB(i,0));
              if(diferencia < aproximado)
              {
                valores[i] = j;
                aproximado = diferencia;
              }
              else{}
            }
          }
          valores[i] = (((valores[i] * 25)/973)*(-
1))+35;
    }



    // Fill our table with random data (from the
Record() constructor).
    for (int i = 0; i < data.length; i++) {
      data[i] = new Record(valores);
    }
  }

  // Return values appropriate for the visible table
part.
  public int getRowCount() {
    return Math.min(pageSize, data.length);
  }

  public String getColumnName(int col) {
    return Record.getColumnName(col);
  }


  public int getColumnCount() {
    return Record.getColumnCount();
  }

  // Work only on the visible part of the table.
  public Object getValueAt(int row, int col) {
```

```
    int realRow = row + (pageOffset * pageSize);
    return data[realRow].getValueAt(col);
   }
}


class Llenar{
  static String[] headers = { "Punto1 --> Punto2",
"Temperatura[°C]"};

  static int counter;

  String[] data;

  int[] valor = new int[500];

  public Llenar( int valores[]) {

    valor = valores;

    data = new String[] { "" + (counter++),
       "" + valor[counter-1]};
  }

  public String getValueAt(int i) {
    return data[i];
  }

  public static String getColumnName(int i) {
    return headers[i];
  }

  public static int getColumnCount() {
    return headers.length;
  }

}
```

//Ayuda.java

```
import java.awt.*;
import java.awt.event.*;

import java.awt.Dimension;
import java.awt.GridLayout;
import java.io.IOException;
import java.net.URL;

import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTree;
import javax.swing.UIManager;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import
javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.border.Border;
import javax.swing.*;



public class Ayuda extends JFrame implements
TreeSelectionListener
{
   private JEditorPane htmlPane;
   private JTree tree;
   private URL helpURL;
   private static boolean DEBUG = false;

   //Optionally play with line styles.  Possible
values are
   //"Angled" (the default), "Horizontal", and
"None".
   private static boolean playWithLineStyle =
false;
   private static String lineStyle = "Horizontal";
   //Optionally set the look and feel.
   private static boolean useSystemLookAndFeel
= false;


   public Ayuda() {
      JPanel panel = new JPanel();
      setTitle("Ayuda");

setIconImage(Toolkit.getDefaultToolkit().getIma
ge("Icmyl.gif"));
      setSize(700, 545);
      addWindowListener(new WindowAdapter()
      {
         public void
windowClosing(WindowEvent e)
         {
            setVisible(false);
         }
      });

      //Create the nodes.
      DefaultMutableTreeNode top =
         new DefaultMutableTreeNode("Menú");
      createNodes(top);

      //Create a tree that allows one selection at a
time.
      tree = new JTree(top);
      tree.getSelectionModel().setSelectionMode

(TreeSelectionModel.SINGLE_TREE_SELECT
ION);

      //Listen for when the selection changes.
      tree.addTreeSelectionListener(this);

      if (playWithLineStyle) {
         System.out.println("line style = " +
lineStyle);
         tree.putClientProperty("JTree.lineStyle",
lineStyle);
      }

      //Create the scroll pane and add the tree to
it.
      JScrollPane treeView = new
JScrollPane(tree);

      //Create the HTML viewing pane.
      htmlPane = new JEditorPane();
      htmlPane.setEditable(false);
      initHelp();
      JScrollPane htmlView = new
JScrollPane(htmlPane);

      //Add the scroll panes to a split pane.
      JSplitPane splitPane = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
      splitPane.setTopComponent(treeView);
      splitPane.setBottomComponent(htmlView);

      Dimension minimumSize = new
Dimension(200, 100);
      htmlView.setMinimumSize(minimumSize);
      treeView.setMinimumSize(minimumSize);
      splitPane.setDividerLocation(200); //XXX:
ignored in some releases
```

```
                                //of Swing. bug
4101306
        //workaround for bug 4101306:
        //treeView.setPreferredSize(new
Dimension(100, 100));

        splitPane.setPreferredSize(new
Dimension(700, 500));

        //Add the split pane to this panel.
       //  add(splitPane);

        //JScrollPane scp = new
JScrollPane(splitPane);
        Border brd =
BorderFactory.createMatteBorder(20, 15, 20, 15,
Color.getColor("azul", -16490900));
        splitPane.setBorder(brd);
        panel.add(splitPane);
        centrar();
        getContentPane().add(panel,
BorderLayout.CENTER);
        setVisible(true);
    }

    /** Required by TreeSelectionListener
interface. */
    public void valueChanged(TreeSelectionEvent
e) {
        DefaultMutableTreeNode node =
(DefaultMutableTreeNode)

tree.getLastSelectedPathComponent();

        if (node == null) return;

        Object nodeInfo = node.getUserObject();
        if (node.isLeaf()) {
            BookInfo book = (BookInfo)nodeInfo;
            displayURL(book.bookURL);
            if (DEBUG) {
                System.out.print(book.bookURL + ":
\n    ");
            }
        } else {
            displayURL(helpURL);
        }
        if (DEBUG) {
            System.out.println(nodeInfo.toString());
        }
    }

    private class BookInfo {
        public String bookName;
        public URL bookURL;

        public BookInfo(String book, String
filename) {
            bookName = book;
            bookURL =
Ayuda.class.getResource(filename);
            if (bookURL == null) {
                System.err.println("Couldn't find file: "
                        + filename);
            }
        }

        public String toString() {
            return bookName;
        }
    }

    private void initHelp() {
        String s = "Bienvenido.htm";
        helpURL = Ayuda.class.getResource(s);
        if (helpURL == null) {
            System.err.println("Couldn't open help
file: " + s);
        } else if (DEBUG) {
            System.out.println("Help URL is " +
helpURL);
        }

        displayURL(helpURL);
    }

    private void displayURL(URL url) {
        try {
            if (url != null) {
                htmlPane.setPage(url);
            } else { //null url
                htmlPane.setText("File Not Found");
                if (DEBUG) {
                    System.out.println("Attempted to
display a null URL.");
                }
            }
        } catch (IOException e) {
            System.err.println("Attempted to read a
bad URL: " + url);
        }
    }

    private void
createNodes(DefaultMutableTreeNode top) {
        DefaultMutableTreeNode category = null;
        DefaultMutableTreeNode book = null;

        category = new
DefaultMutableTreeNode("Archivo");
        top.add(category);
```

```
        book = new DefaultMutableTreeNode(new
BookInfo
            ("Abrir",
             "Abrir.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Configurar Pág...",
             "ConfigPág.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Imprimir",
             "Imprimir.htm"));
        category.add(book);


        book = new DefaultMutableTreeNode(new
BookInfo
            ("Guardar",
           "Guardar.htm"));
        category.add(book);


        book = new DefaultMutableTreeNode(new
BookInfo
            ("Guardar como...",
"GuardarComo.htm"));
        category.add(book);


        book = new DefaultMutableTreeNode(new
BookInfo
            ("Cerrar",
             "Cerrar.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Salir",
             "Salir.htm"));
        category.add(book);



        category = new
DefaultMutableTreeNode("Editar");
        top.add(category);


        book = new DefaultMutableTreeNode(new
BookInfo
            ("Difuminar",
             "Difuminar.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Áspero",
             "Áspero.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Rotar",
             "Rotar.htm"));
        category.add(book);


        category = new
DefaultMutableTreeNode("Herramientas");
        top.add(category);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Obtener Información",
             "ObInfo.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Manipular Imágen",
             "ManImag.htm"));
        category.add(book);

        book = new DefaultMutableTreeNode(new
BookInfo
            ("Realizar Mejoramiento",
             "RealMej.htm"));
        category.add(book);



    }

    protected void centrar()
    {
        Dimension screen =
Toolkit.getDefaultToolkit().getScreenSize();
        Dimension us = getSize();
        int x = (screen.width - us.width) / 2;
        int y = (screen.height - us.height) / 2;
        setLocation(x, y);
    }


}
```

//Bibliografía.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.border.Border;
import javax.swing.*;


public class Bibliografía extends JFrame{

   /** Creates a new instance of Bibliografía */
   public Bibliografía()
   {
      JPanel panel = new JPanel();
      setTitle("Algunos Datos");

setIconImage(Toolkit.getDefaultToolkit().getIma
ge("Icmyl.gif"));
      setSize(450, 350);
      addWindowListener(new WindowAdapter()
      {
         public void
windowClosing(WindowEvent e)
         {
            setVisible(false);
         }
      });

      Border brd =
BorderFactory.createMatteBorder(20, 15, 20, 15,
Color.getColor("azul", -16490900));
      JTextArea texto = new
JTextArea(información, 16, 35);
      JScrollPane Info = new JScrollPane(texto);
      texto.setEditable(false);

texto.setBackground(Color.LIGHT_GRAY);
      texto.setSelectedTextColor(Color.BLACK);
      texto.setFont(Font.decode("ITALIC"));
      Info.setBorder(brd);
      panel.add(Info);
      centrar();
      getContentPane().add(panel,
BorderLayout.CENTER);
      setVisible(true);
   }

   static String información =

   "Título: Desarrollo De Una Aplicación Web
Para El Procesamiento\n" +
   "        Digital De Imagenes Satélitales\n\n" +
   "Tesista: Rodrigo Cázares Castro\n\n" +
   "Director de Tesis: M.I.Ranulfo Rodríguez
Sobreyra\n\n" +
   "Icmyl Junio 2006.\n\n" +
   "Para el desarrollo de este sistema se hizo uso
de la siguiente refere-\n" +
   "ncia bibliográfica.\n\n" +

   "*Geary, David M."+
      "\nGraphic Java : Mastering the AWT"+
      "\nCalifornia : SunSoft, c1997,600 p." +

   "\n\n*Geary, David M."+
      "\nGraphic Java 2 : mastering the JFC, 3rd
ed."+
      "\nPalo Alto, California : Sun
Microsystems, m"+
      "\n1999" +

   "\n\n*Zukowski, John"+
      "\nDefinitive guide to Swing for Java 2, 2nd
ed."+
      "\nNew York : Apress, c2000, 890 p." +

   "\n\n*Efford, Nick"+
      "\nDigital image processing : a practical
introduction"+
      "\nusing Java."+
      "\nHarlow, England : Pearson Education :"+
      "\nAddison-Wesley,2000, 340 p" +

   "\n\n*Campesato, Oswald"+
      "\nJava graphics programming library :
concepts to source"+
      "\nHingham, Massachusetts : Charles River
Media,"+
      "\nc2002, 524 p." +

   "\n\n*Subrahmanyam Allamaraju[et al.]"+
      "\nProgramacion Java Server con J2EE
edicion 1.3"+
      "\nMadrid : Anaya Multimedia, c2002"+
      "\n1245 p." +

   "\n\n*Pratdepadua Bufill, Joan Josep"+
      "\nProgramacion en 3D con Java 3D"+
      "\nMadrid : Ra-Ma; Mexico : Alfaomega,
c2003" +

   "\n\n*Horstmann, Cay S., Core Java 2"+
      "\n7th ed.,Santa Clara, California : Sun
Microsystems, " +

    "\n\ny la dirección
http://www.java2s.com/Code/Java/CatalogJava.h
tm";
```

```
    protected void centrar()
    {
        Dimension screen =
Toolkit.getDefaultToolkit().getScreenSize();
        Dimension us = getSize();
        int x = (screen.width - us.width) / 2;
        int y = (screen.height - us.height) / 2;
        setLocation(x, y);
    }

}
```

BIBLIOGRAFÍA

- [1] W. De Muynck. Bridging the Gap between XML and Hypermedia: a Layered Transformational Approach, Tesis. Approach, Vrije Universiteit Brussel, Belgium, 2000.

- [2] D. Schwave and G. Rossi. An Object Oriented Approach to Web-Based Application Design. En: Theory and Practice of Object Systems (TAPOS), October 1998.

- [3] D. Schwave *et al.* Engineering Web Applications for Reuse. IEEE Multimedia, Vol. 8 Núm. 1, pp. 20-31.

- [4] G. Rossi; D. Schwave and Fernando Lyardet. Web application models are more than conceptual models. En: Proceedings of the First Internation Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999.

- [5] D. Cowan and C. Lucena. Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. IEEE Transactions on Software Engineering. Vol. 21, No. 3, March 1995.

- [6] N. Koch. Comparing Development Methods for Web Applications. Ludwig-Maximilians-University Munich, Institute of Computer Science Oettingenstr. 67, 80538 München, Germany. 2000.

- [7] F. Garzotto; L. Mainetti and P. Paolini. Hypermedia design analysis. Communications of the ACM, 8(38), 74-86. 1995.

- [8] T. Isakowitz; E. Stohr and P. Balasubramanian. A methodology for the design of structured hypermedia applications. Communications of the ACM, 8(38), 34-44. 1995.

- [9] D. Lange. An object-oriented design approach for developing hypermedia information systems. Journal of Organizational Computing and Electronic Commerce, 6(3),269-293. 1996.

- [10] H. Lee; C. Lee and C. Yoo. A scenario-based object-oriented methodology for developing hypermedia information systems. En: Proceedings of 31st Annual Conference on Systems Science, Eds. Sprague R. 1998.

- [11] O. De Troyer and C. Leune. WSDM: A user-centered design method for Web sites. En: Proceedings of the 7th International World Wide Web Conference. 1997.

- [12] J. Conallen. Building Web application with UML. Addison Wesley. 1999.

- [13] Java Sun, The Java Tutorial: A practical guide for programmers,
- http://Java.sun.com/docs/books/tutorial, Marzo de 2001.

- [14] Java Sun, JDBC API tutorial and reference - Second Edition, http://Java.sun.com/products/jdbc/, Marzo de 2001.

- [15] Java Sun, JavaTM Servlet Technology: The Power Behind the Server,
- http://Java.sun.com/products/servlet, Marzo de 2001.

- [16] Eva.Arderiu, Javier.Conde. Objectivity/DB and JAVA,
- http://wwwinfo.cern.ch/asd/cernlib/rd45/objy_Java_info.htm#Servlets, Marzo de 2001.

- [17] K. Avedal *et al.* Professional JSP, Wrox Press 2000.

- [18] Java Sun, Java Server PagesTM: Dinamically Generated Web Content,
- http://www.Javasoft.com/products/jsp/, Marzo de 2001.

- [19] World Wide Web Consortium (W3C), Extensible Markup Language (XML). The base specifications
- are XML 1.0, W3C Recommendation Feb '98, and Namespaces, Jan '99. http://www.w3.org/XML,

- [20] Joyanes Luis, ***Programación orientada a objetos***, 2ª edición, McGraw-Hill, España 1998.

- [21] Ceballos Javier, P***rogramación orientada a objetos con C++***, 2ª edición, Alfaomega, México 1998.

- [Web_1]http://hostutn.frt.utn.edu.ar/sistemas/paradigmas/poo.htm#_Programación_orientada_a

- [Web_2]http://ccc.inaoep.mx/~labvision/doo/proy/T32.pdf#search='metodologias%20oo'

- [Web_3]http://es.wikipedia.org/wiki/Aplicacion_web

- [Web_4]http://www.avidos.net/blogold/aplicaciones-web/

- [Web_5]http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node17.html
- [Web_6]http://www.uoc.edu/masters/esp/img/873.pdf
- [Web_7]http://www.vico.org/TRAD_obert/TRAD_WAE_abierto.pdf
- [Web_8]http://usuario.cicese.mx/~jburci/Docs/art6.htm
- [Web_9]http://www.java2s.com/Code/Java/CatalogJava.htm