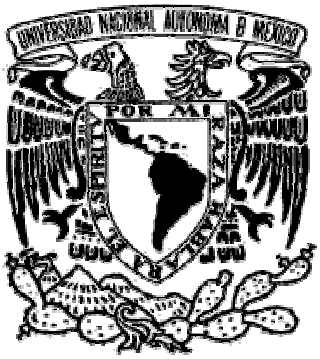


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UN SISTEMA PARA EL PROCESAMIENTO DE IMÁGENES SATELITALES, BASADO EN UML.

T E S I S

QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A :

KARENNINA GÓMEZ RAMÍREZ

DIRECTOR DE TESIS:
M. en C. RANULFO RODRIGUEZ SOBREYRA

MÉXICO, D.F. NOVIEMBRE DEL 2005

AGRADECIMIENTOS

A mi Madre,

A ti Emigdia Ramírez Camacho, por apoyarme siempre, porque te has entregado y das lo mejor de ti para que tus cuatro hijos se convirtieran en lo que son hasta el día de hoy. Gracias a ti me he convertido en la mujer que soy, tú y yo logramos juntas esta meta, gracias por el amor y confianza que me has dado siempre me has brindado.

A mi hermano Javier Giovanni,

Este trabajo representa la culminación de un trabajo constante, y que gracias a ese gran apoyo que me has dado lo pude lograr. Gracias por la confianza, el amor y porque siempre me impulsaste a ser mejor.

A mi hermano Netzer,

Gracias por dejarme siempre tomar mis decisiones, porque siempre me has dejado saber que cuento contigo, por la confianza que siempre me has tenido, por que siempre has sido un excelente hermano.

A mi hermana Cosijoesa,

A ti hermanita, porque juntas hemos logrado llegar hasta aquí, porque cada palabra de animo me ha ayudado a seguir adelante y porque siempre me has apoyado. Gracias por el amor y confianza que siempre me has demostrado.

A Raúl Moreno, por tu compañía en la realización de este trabajo y por esos buenos consejos.

AGRADECIMIENTOS

A la UNAM y en particular a la Facultad de Ingeniería por la oportunidad de pertenecer a ella y cursar mis estudios profesionales; agradezco a todos los profesores por su entusiasmo y trabajar para que la educación de México sea cada día mejor.

Al MC Ranulfo Rodríguez Sobreyra, por el apoyo, la paciencia y la ayuda que me brindo para realizar este trabajo.

Al Instituto de Ciencias del Mar y Limnología, en especial al Laboratorio de Oceanografía Física, sobre todo al Dr. Artemio Gallegos García, por permitirme pertenecer al equipo del laboratorio de Oceanografía Física para poder realizar este trabajo.

TABLA DE CONTENIDO

INTRUDUCCION

CAPITULO 1

PROGRAMACIÓN ORIENTADA A OBJETOS

1.1 Introducción

1.2 ¿Qué es programar?

1.3 Programación Orientada a objetos

1.4 Mecanismos Básicos de la Programación Orientada a objetos

1.4.1 Objetos
1.4.2 Comunicación: Mensajes
1.4.3 Métodos
1.4.4 Clases

1.5 Estructura de las Clases

1.6 Características de una programación Orientada a Objetos

1.6.1 Encapsulado
1.6.2 Herencia
1.6.3 Polimorfismo
1.6.4 Abstracción

1.7 Influencia e importancia de la Programación Orientada a Objetos

CAPITULO 2

METODOLOGÍAS ORIENTADAS A OBJETOS PARA EL ANÁLISIS, DISEÑO Y MODELACION.

2.1 Introducción

2.2 Modelación, Análisis y Diseño de Software Orientado a Objetos

2.2.1 Definiciones
2.2.2 Componentes

2.3 Metodologías Orientadas a Objetos

2.3.1 Metodologías de Booch

2.3.1.1 Definición
2.3.1.2 Características
2.3.1.3 Descripción General

2.3.2 Metodología Rumbaugh - OMT

2.3.2.1 Definición
2.3.2.2 Características
2.3.2.3 Descripción General

2.3.3 Metodología Jacobson - OOSE

2.3.3.1 Definición
2.3.3.2 Características
2.3.3.3 Descripción General

2.3.4 Metodología Jackson –JSD

2.3.4.1 Definición
2.3.4.2 Características
2.3.4.3 Descripción General

2.3.5 Metodología Yourdon & Coad

2.3.5.1 Definición
2.3.5.2 Características
2.3.5.3 Descripción General

2.3.6 Metodología Lenguaje de Modelación Unificado (UML)
2.3.6.1 Definición
2.3.6.2 Características
2.3.6.3 Descripción General

2.4 Herramientas para el desarrollo de Objetos Orientados.
2.4.1 Definición
2.4.2 Características
2.4.3 Ejemplos de herramientas CASE

CAPITULO 3

APLICACIÓN DE UNA METODOLOGÍA UML PARA EL DESARROLLO DEL SISTEMA

3.1 Implementación del Sistema Modelado3

3.1.1 Introducción
3.1.2 Objetivos del Sistema
3.1.3 Antecedentes
3.1.4 Requerimientos
3.1.6 Tabla de requerimientos

3.2 Metodología UML empleando la herramienta Rational Rose

3.2.1 Vistas de los objetos de usuario
3.2.2 Definir ventanas
3.2.3 Diseño
3.2.3.1 Encontrando Casos de Uso
3.2.3.2 Encontrando Diagrama de Secuencia
3.2.3.3 Encontrando Diagramas de Colaboración
3.2.3.4 Encontrando Clases

3.2.3.5 Descripción de pasos para Generar la estructura del sistema
3.2.3.6 Reportes obtenidos a partir de la generación de código

3.3 Programa Final

Resultados

Conclusiones

Anexos

Bibliografía

INTRODUCCION

Las metodologías de ingeniería de software indican como construir técnicamente el software. Estas abarcan un amplio espectro de y tareas incluyen: planificación y estimación de proyectos, análisis, de los requisitos del sistema y del software, diseño de estructuras de datos, lectura de programas y procedimientos algorítmicos, codificación, prueba y mantenimiento. Así mismo introducen una notación especial orientada a un lenguaje y aun conjunto de criterios para la calidad del software. (Presuman, 1993).

Una metodología reciente es UML (Unified Modeling Language), que es un lenguaje que ha crecido de manera increíble en los últimos años. Durante la década de los noventa, diferentes metodologías salieron al mercado, y para los usuarios manejar diferentes notaciones se volvió una guerra. Finalmente, tratando de estandarizar las metodologías, tres de los más importantes métodos se unieron para dar vida a UML, en Octubre de 1995. Debido a la explotación de este lenguaje en el área empresarial, ha sido necesario la publicación de gran número de libros, mostrando la información de una manera sencilla, tal es el caso del libro de Terry Quatrini (ref). En donde el usuario ve de manera simple el uso de este lenguaje.

En el laboratorio de Oceanografía Física (ICML-UNAM) se requiere desarrollar una interfaz grafica para el despliegue y procesamiento de imágenes satelitales, ya que estas proporcionan información por ejemplo de la temperatura superficial del mar. Ya que se cuenta con una base de datos de imágenes satelitales para hacer análisis locales y regionales de procesos oceánicos.

Objetivo General:

- Desarrollar una aplicación para el procesamiento de imágenes satelitales.

Objetivos Particulares:

- EL estudio de las metodologías para el desarrollo de software.
- Emplear la metodología UML para implementar la aplicación de procesamiento de imágenes.
- Evaluación de la metodología usada.

Este trabajo se divide en cuatro capítulos que abarcan los siguientes temas:

Capitulo 1: Programación Orientada a Objetos.

En este se describen los conceptos principales de la programación orientada a objetos.

Capitulo 2: Metodologías Orientadas a Objetos para el análisis, diseño y modelación.

Se presenta un estudio de las metodologías más destacadas como: James Rumbaugh, Ivar Jacobson, Grady Booch, Jackson, Yourdon &Coad y UML.

Capitulo 3: Aplicación de la metodología UML para el desarrollo del sistema.

Se muestra la aplicación paso a paso de la metodología UML incluyendo los diagramas de casos de uso, diagramas de secuencia y diagramas de clases.

Capitulo 4: Resultados

Se presenta un cuadro comparativo de las metodologías estudiadas y la estructura del programa desarrollado.

CAPITULO 1: PROGRAMACIÓN ORIENTADA A OBJETOS

1.1 Introducción

La Programación Orientada a Objetos (POO) esta compuesta de objetos. Se entiende como objeto, un ente, el cual tiene atributos particularmente únicos, por lo tanto cuenta con ciertas características tales como: tamaño, color, barras de desplazamiento, titulo y ciertas funciones que nos van a permitir dirigir a éstas datos (Cevallos, 1998). Los objetos pueden establecer una comunicación entre ellos y esto se logra a través del envío de mensajes. Un ejemplo claro esta en "Windows" en la cual encontramos varias ventanas, nosotros podemos abrirlas, y realizar cambios.

Pero no todo lo que se encuentra a nuestro alrededor son objetos. Tenemos de ellos el concepto pero no son palpables como el tiempo, el color, las emociones y todo lo incorpóreo. En muchas universidades se han preocupado por el avance en la programación, y en las herramientas de trabajo que han ayudado a un aprendizaje autodidáctico de la misma programación. Se entiende que la programación es una su forma de pensar, que es esta nueva forma de pensar que es lo hoy llamamos "Programación Orientada a Objetos" (Cevallos, 1998).

Los lenguajes básicos que han contribuido a la evolución de los lenguajes orientados es el primer lugar LISP de la década de los 50, en la década de los 60 PASCAL, C, y Ada. Aunque estos lenguajes no incluyen mecanismo para la programación orientada a objetos, sus características sirvieron para la base de la construcción de la POO. En la década de los 70, aparece Smalltalk como un lenguaje orientado a objetos (Cevallos, 1998).

1.2 ¿Qué es programar?

Todo programa parte de lo más esencial; el lenguaje. Un lenguaje es un conjunto de sentencias que pueden ser formadas a partir de cadenas, las cuales cumplen con reglas gramaticales. Así que programas consiste en definir las sentencias de un conjunto de cadenas que compondrán nuestro programa. El orden o la combinación dentro de una sentencia quedaran definidos por las reglas sintácticas para ejecutar una serie de acciones.

1.3 Programación Orientada a Objetos.

La programación orientada a objetos es una forma de programar que utiliza objetos (Joyanes, 1998). Todos estos procesos producen un flujo de mensajes, lo cual va a dar origen a un sin fin de cambios en el estado del objeto. La estructura básica está formada por dos objetos y un mensaje.

La programación estructurada se comenzó a utilizar a principios de los años 60, no había en ese entonces ninguna otra alternativa para programar. Este tipo de programación esta compuesta de subrutinas, se le llama estructurada porque esta compuesta por varias partes, similar a lo que es un rompe cabezas. (Cevallos, 1998). No es tan importante como se realizan las funciones, si no como estas deben hacer su función correctamente. Los tipos de datos se procesan en muchas funciones dentro de un programa estructurado. La programación orientada a objetos es una técnica de estructuración (Voss, 1998). En esta los principales elementos son los objetos para la construcción de un programa, la esencia fundamental es entender como los objetos se comunican entre si. La diferencias entre una programación tradicional y programación orientada a objetos es que la primera esta compuesta de datos y procedimientos, la segunda consiste en objetos (Joyanes, 1998).

La POO es una forma especial de programar, tiene un mayor acercamiento a como expresaríamos las cosas en la vida real, esto lo diferencia de otros tipos de programación. En la POO se tiene que aprender a pensar las cosas de una manera distinta, para escribir los programas en términos de objetos, propiedades, métodos o clases. Ahora la programación estructurada, hasta cierto nivel ya no es muy útil, ya que el desarrollo de software aumenta con mayor rapidez y muchas veces necesitamos re-utilizar el código, algo que no podemos hacer con la programación estructurada.

1.4 Mecanismos Básicos de la Programación Orientada a objetos

1.4.1 Objetos

Los objetos como las personas no trabajan individualmente, realizan actividades individuales, pero que finalmente forman parte de un conjunto, solo así los gobiernos, escuelas y empresas pueden funcionar, lo mismo pasa con los objetos cada uno tiene sus actividades, dentro de una estructura. Los objetos tienen atributos que los ayuda a distinguirse de otros. Además que estos objetos pueden realizar operaciones sobre otros objetos, estas operaciones puede ser (Cevallos, 1998) :

i. Modificación
ii. Selección
iii. Iteración
iv. Destructor
v. Constructor.

La comunicación entre los objetos es mediante el envió de mensajes. La creación de un objeto se realiza por medio de clases. Una clase es la que nos permite describir los atributos de un objeto.

1.4.2 Comunicación: Mensajes

Los mensajes son el medio el cual un objeto le indica a otro que se va a iniciar una actividad, y están formados por:

i. Mensaje: destino, operación, argumentos (Rumbaugh et al, 1999).
ii. Destino: Manda una señal a uno o más objetos. Se define quien es el que manda y quien recibe en este medio.
iii. Operación: Proceso de una llamada; una señal; una operación local sobre el que envía a un objeto y eventos explícitos.
iv. Argumentos: La comunicación de el que envía y recibe la información por medio de valores, es una expresión incluye especificaciones y condiciones e iteraciones para el mensaje en ejecución.

Aún cuando cada persona realice actividades por separado, es necesario tener comunicación con nuestros compañeros de trabajo, solo así se logra una coordinación. Saber quien realiza que actividades y quien da solución en cada caso de problemas.

Una vez más ejemplificamos como se expresa y plasma la realidad en la POO, la comunicación entre objetos también es de suma importancia y puede ser de dos formas; dinámica y estática (Joyanes, 1998).

La comunicación dinámica se refiere a que durante un tiempo los objetos pueden tener comunicación, se crea esta comunicación cuando un objeto le manda un mensaje a otro. A diferencia de la comunicación estática, en donde los objetos pueden estar en contacto en un periodo de tiempo muy largo y ambos saben que existen.

1.4.3 Métodos

Los métodos son las operaciones que pueden realizarse sobre el objeto, estos normalmente estarán incorporados en forma de código, el objeto es capaz de ejecutar y también pone a disposición de sus descendientes a través de la herencia (Cevallos, 1998). Cada uno de estos componentes desempeña un papel totalmente independiente. La descripción más clara de lo que es un objeto es lo que llamamos operaciones. Las relaciones permiten que el objeto se inserte en la organización, están representadas esencialmente por flechas, a otros objetos. Las propiedades distinguen a un objeto de los restantes, que forman parte de una misma organización, tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes. (Joyanes, 1998).

1.4.4 Clases

Una clase describe los métodos y atributos, definiendo con ellos las características comunes de todos los objetos (Cevallos, 1998). Las clases son agrupaciones dentro de las cuales tenemos objetos con las mismas características. Con frecuencia suele confundirse la clase con objeto cuando no se tiene un concepto claro de ambos términos. La clase es un tipo de dato, mientras que el objeto es la instancia del tipo de dato (Joyanes, 1998).

1.5 Estructura de una clase

Dentro de las clases se tienen los atributos y las operaciones que se pueden dividir en tres (Cevallos, 1998):

i. Publico
ii. Protegido
iii. Privado

Es importante saber que público hace referencia a la clase que puede ser vista por todas las demás, es decir pueden acceder a ellas desde cualquier parte del programa. Las clases privadas se refieren a que son accesibles sólo por funciones integrantes de la clase. Y las clases protegidas son accesibles por las funciones de ella misma y por las clases que deriven de ella.

Subclases

Dentro de una clase puede existir lo que denominamos subclase estas heredan características de la clase principal (Cevallos, 1998). Una clase bien diseñada es aquella fácil de entender y de usar. Por ejemplo si estamos tratando de obtener las características de una pintura, encontramos diferentes colores, no podemos definir una clase con un solo color para describir todos los colores de la pintura. Un atributo representa un valor. El atributo tiene relación con las clases que a su vez es compartido por todos los objetos que se encuentren en la clase (Cevallos, 1998). La utilidad de las subclases en la programación se debe a que muchas veces hay un exceso o repetición en el almacenamiento de un mismo atributo. Para evitar que esto suceda se define una jerarquía. Y heredar todos los atributos de la clase principal a la subclase.

1.6 Características de la programación Orientada a Objetos

Se dice que un lenguaje es orientado a objetos, si cumple con ciertas características específicas (Joyanes, 1998):

I. Debe de soportar objetos que son abstracciones de datos.
II. Los objetos tienen una asociación, es decir una clase.
III. Las clases pueden heredar atributos de supertipos que son las superclases.

En la actualidad se tiene varios lenguajes que utilizan el modelo de Objetos, por mencionar algunos:

C++: El lenguaje C evoluciono a partir de dos lenguajes previos, BCPL y B.BCPL fue desarrollado en 1967 por Martín Richards, como un lenguaje para escribir sofware y compiladores de sistemas operativos.(Deitel & Deitel, 1995)

Object Pascal: El lenguaje Pascal se creó en la década de los 70 con el objetivo de disponer de un lenguaje de programación de alto nivel y propósito general (se utiliza para gran diversidad de aplicaciones) orientado hacia los nuevos conceptos de programación, desarrollado por el profesor suizo Niklaus Wirth.

Smalltalk: Fue diseñado en el centro de Investigaciones Xerox de Palo Alto durante 1970. SmallTalk fue diseñada para ser un lenguaje simple de un Sistema de Programación Interactivo en el cual los programas puedan ser caracterizados por un alto grado de Modularidad y extensión Dinámica. El Procesador típico de SmallTalk permite al programador "hojear" a traves de de extensa libreria de definición de clases para crear, editar, compilar y verificar la definición de nuevas clases y la construcción de programas.

Todo software tiene sus propias características y reglas de funcionamiento. En el caso de la programación orientada a objetos estas son: Encapsulado, Herencia, Polimorfismo y Abstracción. Cada una de ellas trata de cumplir con los requerimientos del usuario, manipular la información de una manera sencilla y reducir el volumen del código.

1.6.1 Encapsulado

El encapsulado es una envoltura o protector, que evita que otro código que no pertenezca a esa sección pueda acceder al código. Es decir, es un mecanismo que nos permite unir el código y los datos que se manipulan, manteniéndolos alejados para que no haya interferencia entre ellos. El acceso al código y a los datos se realiza de una forma controlada, por medio de una interfaz definida (Cevallos, 1998). Por ejemplo, en el caso del radio podemos controlar el volumen, nosotros como usuarios sólo tenemos una forma de actuar sobre el encapsulado, es a través de la perilla y del ecualizador para obtener una mejor fidelidad sin afectar a otras operaciones como el grabar un cassette o cambiar de estación. La perilla de volumen es una interfaz bien definida, lo que ocurra dentro de ella no afecta a lo que este a su alrededor. El poder del encapsulado se debe a que todo mundo puede, sabe y conoce como acceder a él y se puede utilizar independientemente a los detalles de la implementación.

1.6.2. Herencia

Se debe tener claro que el diseño de un objeto orientado, es diferente a una estructura tradicional. Requiere por lo tanto, una forma diferente de pensar. La razón principal de la descomposición en pequeñas partes, es que, de esta forma se definen las restricciones que se tiene en un programa o en un sistema.

Ya que las descomposiciones están basadas en objetos se les llama "descomposición de un objeto orientado" (Cevallos, 1998). Si hacemos una analogía entre la programación orientada a objetos y los idiomas, observamos que al aprender un idioma se complica un poco, desde la estructura hasta la pronunciación, una vez terminado se inicia con un segundo idioma (una subclase) ahora es mas fácil porque mantenemos el mismo método de aprendizaje. Así "La herencia es un proceso mediante el cual un objeto va a adquirir propiedades de otro" (Cevallos, 1998). Es importante la existencia de la jerarquía de clases para hacer uso de la herencia. De esta manera un objeto tendría que dar explícitamente algún atributo único dentro de la clase. "Una subclase hereda todos los atributos de cada uno de sus antecesores en la jerarquía de clases".

1.6.3. Polimorfismo

Polimorfismo es un concepto que se aplica dentro de la programación. Se define la palabra polimorfismo del griego como: poly significa muchos y morphos significa forma. En la programación orientada a objetos se utiliza el término objetos polimórficos, estos objetos pueden ser, por ejemplo una variable, que puede contener varios valores. Para poder explicar mejor este concepto primero definiremos instancia. Una instancia u objeto de una clase es una representación concreta y específica de una clase, que se encuentra en la memoria (Cevallos, 1998). Cuando las instancias u objetos empiezan a tener comunicación entre sí, a veces, cuando se manda un mensaje a una instancia, la que envía no sabe a que clase pertenece la instancia receptora del mensaje. Aquí es donde se emplea el término polimorfismo. O bien, se utiliza este término, cuando una operación se usa o se implementa en varias clases.

El polimorfismo es una característica fundamental dentro de la programación orientada a objetos. Es la que va a permitir que una clase pueda utilizar una interfaz de distintas formas, esta interfaz se ocupa para un conjunto de actividades que tienen algo en común (Cevallos, 1998). Es de gran ayuda debido a que reduce la complejidad para el programador. En otras palabras, no es otra cosa, que la posibilidad de construir varios métodos con el mismo nombre, pero con una relación en cada clase a la que pertenece de cada uno de los métodos y con comportamientos diferentes. Esto proporciona la habilidad de enviar un mismo mensaje a objetos de clases diferentes.

Para ejemplificar, en el caso de los Bancos se realiza la operación pago, todos acuden a pagar cuentas del gas, teléfono, luz, etc. En este caso los bancos serían las clases, todos los objetos las personas que acuden al banco y realizan la operación pago de diferente forma, según sea el caso, una señora, un joven, etc. Todos los objetos recurren al método por lo tanto ha este proceso le llamamos polimorfismo.

En general, hacer un buen uso de estas características es muy útil, en relación al tiempo, porque con una buena estructura y un buen modelo se hace la migración del código.

1.6.4 Abstracción

Si se estudia un fenómeno y se ignoran todas las partes que son tangibles se puede comprender más de ese fenómeno gracias a la abstracción. Por ejemplo, cuando se conecta el radio no se piensa en un conjunto de miles de partes individuales, sino en un objeto bien definido con un comportamiento propio. Este concepto de abstracción permite utilizar el radio para escuchar noticias sin preocuparnos por las complejidades de las partes que integran dicho aparato. Se ignora básicamente como se alimenta de la corriente eléctrica, como capta la señal en frecuencia modulada. Visto por fuera el radio es un objeto sencillo, profundizando un poco el radio esta compuesto de varios subsistemas: el encendido, los transistores, las pilas, conexiones, etc. Lo importante es operar la complejidad del radio (o de cualquier otro sistema), esto se logra a través de las abstracciones jerárquicas. En los programas de las computadoras también se aplica la abstracción en objetos. Los objetos responden a mensajes que le dicen que hagan determinada acción; es decir, la esencia de la programación orientada a objetos (Cevallos, 1998).

1.7 Influencia e Importancia de la Programación Orientada a Objetos.

Cuando se programaba en un inicio los códigos eran inmensos y no habían herramientas necesarias que generaran un código, posteriormente con el tiempo y la tecnología se podía lograr a través del uso de herramientas simples. Sin embargo, no se descarto la programación manual. Además de extensos, los códigos eran complicados. En la programación convencional, los datos asumen estructura y los procesos hacen de los datos lo que el programador desee.

En el mundo de los datos orientados a objetos, las estructuras de datos se relacionan con los objetos (Martín, 199). De acuerdo a como se percibe la realidad se determina que todo esta construido por objetos y cada uno tiene características que lo hacen único, comparten también muchos atributos. Por lo cual, cada objeto a nuestro alrededor tiene una función, y es más fácil de entender la programación orientada a objetos.

Analizando nuestro alrededor casi todo esta construido con objetos ya existentes, de aquí nace otra característica de la POO, dentro de la programación se re-utilizar el código, lo que implica un ahorro en tiempo y dinero. Al igual que los edificios inteligentes (nuevamente recurriendo a nuestro alrededor), estos son de gran complejidad, ya que contienen un sistema de alarma, elevadores, drenaje, luz, abastecimiento de agua etc., los mismo pasa con los programas; se hacen de lo mas simple, que sólo desplieguen unos números, o cosas (mucho) más complejas como el encriptado de un código que a su vez puede ser también complejo (Joyanes, 1998).

Aunque los programadores expertos no recomiendan hacer programas tan complejos porque de esta forma no se pierde confiabilidad. Por otro lado se tiene, una gran ventaja con el uso de las clases, se diseñan no importando en que plataforma se utilicen.

El beneficio más importante y por el cuál es una opción más viable al elegir la POO, es la forma diferente de pensar, es plasmar nuestra realidad en un programa y entender mejor el mundo.

CAPITULO 2: METODOLOGÍAS EN LA PROGRAMACIÓN ORIENTADAS A OBJETOS PARA LA MODELACIÓN, ANÁLISIS Y DISEÑO.

2.1 Introducción

Un modelo es la simplificación de la realidad, proporciona un anteproyecto del sistema. Es una abstracción de la realidad dependiendo del contexto. Se modela para entender mejor el problema que se desarrolla. Es decir, visualizamos como es el sistema, especificamos la conducta de la estructura del sistema. Dando una guía básica para la construcción del sistema (Joyanes, 1998).

La dinámica de trabajo para el desarrollo de un sistema, indica que la información es reunida y canalizada por un grupo que permite tener una visión completa. En esta condición de trabajo, junto con la habilidad de cada integrante, a veces se tiene la clave para la finalización de un proyecto.

La tecnología de vanguardia que colabora y sirve de materia prima para el analista. El hecho de tener una metodología es con el propósito de evitar el factor incertidumbre, que es percibido al inicio de cada proyecto. Depende mucho de la creatividad del analista y del diseñador, el procesar la información con rapidez sin perder el hilo del proyecto en el menor tiempo posible. Esto se logra a través del tiempo y la experiencia (Joyanes, 1998).

La mente humana es compleja dentro de ella se almacena información, se procesa y envía respuestas, lleva dentro códigos y símbolos. Los analistas deben plasmar en sus diseños una realidad que parte de la pregunta ¿Qué es lo que debemos realizar?. Por consiguiente no es fácil dar respuestas al empezar un proyecto, se debe pensar, observar con el empleo (ayudándonos) de metodologías, conceptos y expresiones.

Las metodologías que se describirán en éste capítulo son los siguientes: Booch, Rumbaugh – OMT, Jacobson – OOSE, Jackson – JSD, Yourdon & Coad, y UML.

2.2 Modelación, Análisis y Diseño de Software Orientado a Objetos.

2.2.1 Definiciones

Metodología

En diferentes metodologías, se ocupan modelos que ayudan a representar algo. Dentro de la Ingeniería, la Arquitectura, el Diseño grafico, la Biología, etc., se utilizan los modelos para poder capturar e implementar los nuevos descubrimientos o estructuras, para un fácil entendimiento. Se plasma un bosquejo de lo que realmente es un sistema, en donde se pueden hacer cálculos del tiempo en que se llevará en realizarlo o el costo económico. Se determinan cuantas personas participaran el proyecto, además de cual será el lenguaje de programación adecuado. El siguiente paso en el proyecto será el análisis, donde poco a poco, se profundiza en el panorama a detalle del proyecto por medio de preguntas (Joyanes, 1998).

Análisis

El concepto de análisis tiene su origen en las discusiones de los programadores con los operadores y usuarios, en donde surgían grandes contrapuntos y cada uno tendrá una idea de cómo desarrollar un proyecto. Debido a la necesidad de crear un enfoque homogéneo, surge el concepto de análisis, que podía ser plasmado con diagramas. Además que estos pueden y en algunos casos deben ser documentados. El hombre es el único animal que "entiende que sabe" y, en consecuencia el proceso de "análisis" es algo inherente al él, donde un proceso mental es llevado a cabo, llevando al máximo todos sus conocimientos y capacidades. En este proceso se muestran sus fronteras, en la búsqueda de interminables preguntas y soluciones. Con gran perseverancia se da a la búsqueda y seguimiento de un problema para la solución más optima. Dentro del análisis es frecuente preguntarse por la localización de errores que por fallas de la modelación, y por la validez en el uso de los conceptos. (Joyanes, 1998)

Composición del Análisis de Sistemas

Durante el análisis del sistema, se conocen las operaciones del sistema y de la organización del mismo; además de identificar los requerimientos del usuario.

Es necesario realizar varias entrevistas a nuestro usuario para que se tengan especificadas, cuales son sus necesidades. El sistema se debe de desarrollar de tal manera, que el usuario final comprenda perfectamente como se maneja el sistema (Joyanes, 1998). En base a:

i. Identificar las responsabilidades de los usuarios
ii. Desarrollar un panorama inicial
iii. Establecer cuales son las metas a las que debemos de llegar
iv. Especificar una estructura tanto del medio como el comportamiento del modelo.

En la identificación de la responsabilidad, como por ejemplo, no todos los usuarios pueden tener los mismos permisos en el acceso de los recursos en la máquina, no pueden modificar o acceder a información confidencial de la empresa. En el desarrollo del panorama, se establecen metas. Debe quedar completamente entendido, cual es el objetivo final tanto del diseñador y como del programador. Saber que herramienta vamos a ocupar, cuales son sus características y como se explotarán al máximo, es parte de cómo vamos a estructurar el sistema (Joyanes, 1998).

Diseño

Para establecer los requerimientos de los usuarios, se debe de tomar en cuenta cinco puntos importantes. Cada uno de ellos dará no solamente un resultado satisfactorio, si no que se verá reflejado el profesionalismo del equipo de trabajo (Joyanes, 1998).

a) Documentación: El desarrollo y explicación del sistema para quienes vayan usar la herramienta.
b) Completo: Cubre todos los requerimientos del sistema.
c) No ambigüedad: No permitir que los recursos o conceptos tomen más de un concepto para el entendimiento del usuario.
d) Consistencia: No hay conflictos e incompatibilidades.
e) Precisión: Requerimientos sean claros y específicos. Entender los requerimientos del usuario. Buena comunicación entre los usuarios, entre los analistas y otros que estén relacionados en el desarrollo del proyecto. Prevenir errores graves, para minimizar el tiempo de la construcción del sistema.

2.3 Metodologías Orientadas a Objetos

En el análisis del problema se emplean conceptos, y términos lógicos. Con la finalidad de encontrar una idea clara y simple para la solución. Una vez analizado el problema es momento de dar el siguiente paso, que es, crear un diseño que pueda ser implementado en un ambiente donde pueda ser ejecutado adecuadamente. Algunas metodologías no hacen diferencia entre lo que es el análisis y diseño, muchas veces van a la par.

Para llevar acabo una buena estructura de la arquitectura de nuestro proyecto, es necesario seguir ciertas reglas, que harán que nuestra implementación sea la más eficiente. Tenemos diferentes prototipos a seguir, en los siguientes puntos se hace mención de las metodologías más importantes.

2.3.1 Metodología Booch Grady

La metodología Booch, parte de que cada etapa no es un proceso aislado, si no que ha de interactuar con sus siguientes y precedentes en una especie de bucle, del que se tiene el conocimiento completo del proceso del modelo en estudio.

2.3.1.1 Características

Proporciona una compresión sólida en la construcción de sistemas orientados a objetos (Booch, 1994). Además, de que es sólido en los conceptos fundamentales de los modelos de objetos. Facilita el dominio de la notación y el proceso de análisis y diseño de objetos orientados (Booch, 1994).

Ventajas (Booch, 1994):

- Es una metodología de propósito general.
- Herramientas y notaciones comprensibles.
- Proporciona una gran cantidad de información, sobre la semántica de los objetos.

Desventajas (Booch, 1994):

- Notaciones poco precisas.
- Poca ayuda para el desarrollador.
- Herramientas orientadas al texto más que a los gráficos.

2.3.1.2 Descripción General

En un principio se tienen una serie de objetos y clases que forman el sistema, a continuación se construye el modelo de interfaz y se examinan las relaciones entre las clases lo que, a su vez, genera la adición de nuevos interfaces que generarán nuevas relaciones iterándose hasta llegar al estado final.

El método Booch proporciona un conjunto de herramientas gráficas y notaciones que ayudan a representar visualmente los modelos definidos en las fases de análisis y diseño (Booch, 1994).

Algunas de estas herramientas son (Booch, 1994): Los diagramas y actividades, que se describen en las siguientes tablas.

Diagramas de clase	Se trata de una variación de los diagramas de entidad relación en los que se añaden nuevos tipos de relaciones como la herencia. Además permite agrupar las clases y relaciones en categorías para diagramas demasiado complejos.
Diagramas de objeto	En este tipo de gráfico de muestran los objetos y sus relaciones, mostrando la forma en la que los objetos se pasan mensajes entre ellos. Así mismo, en esto diagramas es posible representar la visibilidad de los objetos siendo ésta la que determina que objetos se pueden comunicar con otros.
Diagramas temporales	Muestran la secuencia temporal de creación y destrucción de objetos. Suelen ir acompañados de pseudocódigo en el que se explica el flujo de mensajes de control entre los objetos del sistema.
Diagramas de transición de estados.	Permiten definir como las instancias de las clases pasan de un estado a otro a causa de ciertos eventos y que acciones se desencadenan de esos cambios de estado.

Tabla 2.1 Descripción de diagramas de la metodología Booch

Actividades de Análisis y Diseño

Análisis de requerimientos	En esta etapa se define qué quiere el usuario del sistema. Es una etapa de alto nivel que identifica las funciones principales del sistema, el alcance de la modelación del mundo, documenta los procesos principales y las políticas que el sistema va a soportar. No se definen pasos formales, ya que éstos dependen de qué tan nuevo es el proyecto, la disponibilidad de expertos y usuarios y la disponibilidad de documentos adicionales (Booch, 1994).
Análisis de Dominio	Es el proceso de definir de una manera concisa, precisa y POO la parte del modelo del mundo del sistema. Las siguientes actividades son parte de esta etapa (Booch, 1994): Definir Clases Definir relaciones de contenencia Encontrar atributos Definir herencia Definir operaciones Validar e iterar sobre el modelo

Tabla 2.2 Tabla de las actividades de Análisis y diseño de la metodología Booch

Diseño

Es el proceso de determinar una implementación efectiva y eficiente que realice las funciones y tenga la información del análisis de dominio (Booch, 1994). En la siguiente tabla se muestran actividades que se plantean en esta etapa.

Diseño 1	Determinar la arquitectura inicial: decisiones acerca de recursos de implementación, categorías y prototipos a desarrollar.
Diseño 2	Determinar el diseño lógico: detallar al diagrama de clases.
Diseño 3	Implementación física: interfaz a dispositivos o características propias de la implementación.
Diseño 4	Determinar la arquitectura inicial: decisiones acerca de recursos de implementación, categorías y prototipos a desarrollar.

Tabla 2.3 Tabla del diseño de la metodología Booch

La metodología de Booch usa los diferentes tipos de diagramas para describir las decisiones de análisis y diseño, tácticas y estratégicas, que deben ser hechas en la creación de un sistema orientado a objetos. Engloba una selección de diversos dominios de problemas: adquisición de datos, entorno de aplicaciones y gestión de información.

2.3.2 Metodología de Rumbaugh-OMT

El Grupo de Objetos Manejables traducción del inglés Object Management Group, Inc. (OMT), es una organización fundada en 1989, cuyos propósitos son el reuso y portabilidad de los objetos. Considera al modelo de objetos como el más importante.

2.1.1.1 Definición

OMT describe a los objetos orientados como un camino para organizar software en una colección de objetos discretos, que incorporan tanto la estructura de los datos como su comportamiento (Rumbaugh, 1991).

2.3.2.2. Características

Las características de OMT se encuentran esencialmente en la identidad, la clasificación (el grupo de objetos en las clases), el polimorfismo y la herencia.

La esencia del desarrollo de los Objetos Orientados, con la técnica de modelación (OMT), es la identificación y la organización de la aplicación de los conceptos, en vez de la implementación de los conceptos.

De acuerdo a Rumbaugh los beneficios de los objetos orientados, es el acercamiento al gran énfasis de las propiedades esenciales de un objeto, el cual fuerza a los desarrolladores a pensar cuidadosamente y profundamente que es un objeto, y lo que hace. El costo del beneficio no se encuentra en la reducción del tiempo de desarrollo, pero si en el reuso de las clases, al reducir los errores y el mantenimiento.

Para Rumbaugh los aspectos más importantes son (Rumbaugh, 1991):

1) El esfuerzo en el análisis del desarrollo.
2) Enfatizar la estructura de los datos antes que la función. Esto provee estabilidad.
3) EL desarrollo del proceso.
4) El acercamiento interactivo, en vez que un acercamiento secuencial.

2.3.2.3 Descripción General

El método OMT describe el análisis, diseño e implementación, divide el análisis y diseño en tres partes (Rumbaugh, 1991):

1.- Análisis: Construye un modelo del mundo real, a partir de un problema.

2.- Diseño del sistema: Diseño de toda la arquitectura del sistema.

3.- Diseño de objetos: Refinamiento de la estructura de los objetos, a lado de la eficiencia de la implementación, y de los detalles que son añadidos a los objetos.

El método OMT es muy extenso en cuanto a notaciones, para el desarrollo de muchos sistemas únicamente dos o tres partes de la notación son utilizadas, pero algunos sistemas necesitan modelos avanzados de notación.

OMT menciona que la descripción del método no es secuencial. Pero la iteración es requerida y clarifica un camino correcto. En el acercamiento OMT, el sistema parte desde tres puntos de vista diferentes (Rumbaugh, 1991):

I. Un panorama de los objetos.- El objeto del modelo describe un objeto en donde algo sucede (la información fluye).
II. Panorama dinámico.- El modelo dinámico describe cuando sucede (el control del flujo).
III. Panorama funcional.- El modelo funcional describe que sucede (el proceso computacional).

Y Desde el punto de vista técnico (Rumbaugh, 1991), se considera:

a. La modelación de objetos que describen la estructura estática de cada objeto en un sistema y su relación. El objeto del modelo consiste de un diagrama de objetos, un cierto tipo de modelos.

b. Los modelos dinámicos, que describe el control del aspecto de un sistema. Esta parte dinámica para cada clase esta descrita en el diagrama.

c. Estados. Para identificar los eventos, estados y transiciones primero se dibujan los diagramas de flujo.

d. Los modelos funcionales, que describen las capacidades del sistema, estos se logran con el diagrama de flujo.

Procesos del análisis y diseño

El análisis y diseño se hacen por medio de la descripción de las actividades. Cada una da un panorama general de cómo, dentro de un proyecto se debe de desarrollar. La unidad de cada actividad es fundamental para la construcción. En la siguiente tabla se describen tales actividades:

Actividades

Actividad 1	Construcción del modelo de Análisis
Actividad 1.1	Describir el problema: La construcción del problema, comienza escribiendo el problema y cuáles son todos los errores o problemas que surgirán mas adelante.
Actividad 1.2.2	Identificar clases que pertenecen al Diccionario de clases, en el cual las clases son descritas.
Actividad 1.2.3	Después de identificar objetos y clases es necesaria la asociación entre clases. La asociación puede ser encontrada, al buscar los verbos en las frases en el problema escrito. Para mantener la asociación la siguiente guía es recomendada: Eliminar asociaciones entre clases que hayan sido eliminadas. No modelar acciones. Descomponer asociaciones terciarias, si es posible en asociaciones binarias. No modelar relaciones que se puedan derivar de ellas alguna otra.
Actividad 1.2.4	Dentro de esta actividad, los atributos pueden ser encontrados en sustantivos y adjetivos. Para los atributos derivados son importantes para cualquier etiqueta. La relación entre los atributos que puede ser también identificada, al (Rumbaugh, 1991): No modelar objetos como atributos. No modelar identificadores de atributos que son nombrados como identificadores, por ejemplo el ID de una persona. Modelar relaciones entre atributos. No modelar valores internos (estado) de un objeto. No modelar detalles específicos.
Actividad 1.3.2	Los escenarios se pueden identificar con las señales, entradas, decisiones, interrupciones, transiciones y acciones, para o de los usuarios. Los eventos con efectos sobre el flujo de control, deben ser agrupados con un mismo nombre, lo mismo cuando los valores de los parámetros son diferentes.

Actividad 1.4	La Construcción de la modelación, comienza listando las entradas y las salidas de los valores del sistema, los cuales son parámetros de los eventos entre el sistema y la salida. Para mostrar los valores de salida se crea un diagrama de flujo. Pude ser creado a diferentes niveles de detalle, estas pueden ser (Rumbaugh, 1991): Restricciones entre objetos al mismo tiempo. Restricciones entre objetos de la misma clase en diferente tiempo. Precondiciones. Postcondiciones. El último paso para la construcción de un modelo funcional, es la especificación de la optimización de un criterio.
Actividad 1.5	Para mejorar los modelos se adicionan operaciones en el modelado de los objetos. Las operaciones se pueden encontrar tomando en cuenta lo siguiente (Rumbaugh, 1991): Eventos. Estados de Acción y Actividades. Funciones del diagrama de flujo. Operaciones adicionales. Cuando el análisis esta completo los modelos deben ser verificados nuevamente. Esto se hace rescribiendo los requerimientos de los modelos y ver si son consistentes con los requerimientos iniciales.
Actividad 2	Construcción del Diseño del sistema
Actividad 2.1	Organización del sistema en subsistemas, el primer diseño es dividir el sistema en pequeños componentes, nombrando a los subsistemas. Un subsistema es un paquete de clases, eventos y restricciones, para los cuales es importante tener una pequeña interfase con otros subsistemas. Los subsistemas pueden ser identificados por los servicios, los grupos de funciones relacionados para un propósito en común (Por ejemplo aritmético o entrada y salida).

Tabla 2.4 Continuación de la tabla de la metodología de OMT

Esta metodología tiene su potencial en el análisis y su debilidad en el área del diseño. Básicamente construye tres modelos y finalmente verifica estos. Organiza el sistema en subsistemas.

2.3.3 Metodología de Jacobson OOSE

La Ingeniería de Software Orientada a Objetos (Object-Oriented Software Engineering (OOSE)), es una metodología implementada por Ivar Jacobson.

2.3.3.1 Definición

La OOSE es la combinación de dos técnicas diferentes, las cuales han sido usadas por largo tiempo. La primera técnica es la programación orientada a objetos, la cual fue desarrollada en 1960 y pronto pareció ser usada en muchas áreas de aplicación (Jacobson,1995). De la programación orientada, OOSE únicamente usa conceptos de encapsulación, herencia y relación entre clases e instancias.

2.3.3.2 Características

También la OOSE es llamado el "caso de uso aproximado". En este acercamiento, un modelo de caso de uso sirve como modelo central, para el cual otros modelos son derivados (Jacobson,1995). Un modelo de caso de uso describe la completa funcionalidad de un sistema, y son la base en el análisis, la construcción y la prueba. Lo más importante en el análisis es entender al sistema de acuerdo a la función de sus requerimientos.

Las operaciones de los objetos en el panorama interno de los objetos, es descrito durante el análisis (Jacobson, 1995). Una operación que puede ser ejecutada por una entidad de un objeto, puede incluir:

- Creación y borrado de la entidad de objetos.
- Recuperación de información.
- Comportamiento que debe de cambiar si la entidad del objeto es cambiada.

2.3.3.3 Descripción general de la Metodología

La técnica utilizada dentro de esta metodología, hace mención de tres puntos dentro de los cuales hay ciertos componentes, que finalmente es el desarrollo del proyecto (Jacobson,1995). Este modelo delimita el sistema y define la funcionalidad, como se muestra en la tabla 3.5.

Funcionalidad	
Parte 1	Función 1: Modelo de caso de uso, cual describe actores. Los actores definen roles que los usuarios pueden jugar en un intercambio de información con el sistema y el caso de uso representa la funcionalidad dentro del sistema. Principal problema del modelo de objetos, es desarrollar un punto de vista lógico para poder crearlo. Se debe de hacer una lista de sustantivos, soportada para especificar el caso de uso (Jacobson,1995).
Parte 2	Función 2: El análisis del modelo estructural (requerimientos del modelo) es para modelar los tres tipos de objetos: interfase de objetos, entidad de los objetos y control de los objetos. El comportamiento que es modelado en los casos de uso es repartido entre los objetos en el análisis del modelo (Jacobson,1995).
Parte 3	Función 3: El diseño del modelo puede referir el análisis y puede adaptar la implementación del ambiente. La interfase de objetos y la semántica de operaciones son definidas, y las decisiones puede ser hechas con el programador (Jacobson,1995).

Tabla 2.5 Tabla de funcionalidad de la metodología de Jacobson

Es un proceso organizado para la construcción industrial de software. El proceso de diseño está guiado por casos de uso, una técnica que se basa el entendimiento de un sistema en la forma en la cual es usado. Define el modelo de análisis, identificando objetos entidad, de interfaz y de control; independientes del ambiente de implementación. Toda la funcionalidad que es dependiente del entorno del sistema se expresa en objetos de interfaz. Cada objeto de interfaz traduce acciones de los actores en eventos dentro del sistema y traducir los eventos del sistema en algo visible por el actor.

2.3.4 Metodología Jackson - JSD

El Sistema de Desarrollo Jackson (JSD), fue desarrollada por Michael Jackson entre 1972 y 1974. Las especificaciones de JSD consisten principalmente de una distribución de redes de procesos, que se comunican mediante el envió de mensajes.

2.3.4.1 Definición

JSD define una serie de eventos, se define sistemáticamente, y forman las bases para definir los datos y las salidas (JSP & JSD, 1989).

La implementación a veces envuelve la reeconfiguración o transformación de la red en un número pequeño de procesos virtuales. La metodología Jackson identifica pocos objetos, hace pocas consideraciones acerca del mundo real. Este tipo de metodología es muy compleja ya que hace énfasis en el pseudocódigo (JSP & JSD, 1989). Hay confusión en las diferencias entre atributos y operaciones, y muchas veces omite atributos. JSD es complicada y difícil de entender en su totalidad. Una de las razones de la complejidad de JSD, es el fuerte uso que hace del pseudocódigo; los modelos gráficos son más fáciles de entender.

2.3.4.2 Características

La importancia de la metodología es archivar una descomposición del desarrollo de tareas para una clase de problemas. Otro punto importante es el tener puntos de revisión del desarrollo (JSP & JSD, 1989). Los datos y la programación estructurada deben tener la propiedad de ser fáciles de leer y de escribir.

Para JSD los aspectos importantes son (JSP & JSD, 1989):

1) Distribución de procesos secuenciales en red.
2) Cada proceso puede contener sus propios datos locales.
3) Los procesos se comunican leyendo y escribiendo mensajes, o leyendo únicamente accesos de un dato a otro.
4) Usualmente la única dirección de conexión en una red es entre modelos.

2.3.4.3 Descripción general

Para el desarrollo de software mediante el empleo de JSD, se tienen seis pasos secuenciales a seguir (JSP & JSD, 1989):

i. Paso de acciones de entidades.
ii. Paso de estructura de entidades.
iii. Paso de modelo inicial.
iv. Paso de función.
v. Paso de temporización del sistema.
vi. Paso de implementación.

El método consta de cuatro etapas (JSP & JSD, 1989). En las dos primeras se construye la estructura del programa, que es, la especificación de los patrones de secuencia, de selección e de iteración de componentes que define los posibles caminos del flujo de control. La tercera y cuarta etapas se completa el programa y se verifica que la estructura del programa sea correcta para el programa.

Para verificar que el programa sea correcto se tiene que (JSP & JSD, 1989):

i. Dibujar la estructura de diagramas para cada dato de entrada y salida.
ii. Fusionar la estructura de datos de diagramas en una sola estructura de diagrama.
iii. Hacer una lista de operaciones ejecutables del lenguaje de programación.
iv. Convertir el programa desde la representación gráfica en un formato de texto.

En cuanto a la estructura del programa: Se debe fusionar la estructura de datos, primero se definen las correspondencias entre los componentes de las diferentes estructuras de datos. Dos componentes corresponden entre si (JSP & JSD, 1989):

i. Existe el mismo número de cada componente en cualquier programa.
ii. Existe una relación funcional entre los componentes.
iii. La relación funcional es una a una, entre los pares de las instancias de los componentes, los pares ocurren entre los componentes.

La metodología de Jackson es una técnica que modela el mundo real con un enfoque en parte orientado a objetos, donde se utiliza reglas basadas en las estructuras de datos. Sin embargo, son esencialmente métodos de descomposición en donde las estructuras de datos son utilizados como soporte de descomposición. Se puede decir que JSD es una metodología parcialmente orientada a objetos que hace menos énfasis en la parte funcional y comienza el análisis del sistema con un proceso de modelación.

2.3.5 Metodología de Yourdon

Edward Yourdon es autor de una metodología de análisis y diseño estructurado, que se conoce por su nombre, publicó numerosos libros sobre temas de informática y dirigió, hasta 1986, su propia empresa de consultoría (Yourdon, 1976). Entre los trabajos en los que pudo poner a prueba su capacidad, se cuenta la dirección del diseño de software para la edición de The New York Times.

2.3.5.1 Definición

La importancia fundamental del método de Coad y Yourdon es su descripción breve y concisa, así como el uso de textos generales como fuentes para las definiciones; de modo que las definiciones se enmarcan dentro del sentido común y reducen el empleo de modismos. La debilidad principal del método es su notación compleja, la cual es difícil de utilizar sin el apoyo de una herramienta.

3.3.5.2 Características

Dentro de la metodología de Yourdon, se tienen tres herramientas de modelación importantes.

a) Diagramas de flujo.
b) Diagramas entidad relación.
c) Diagramas de transición (state-transition).

Los diagramas de flujo ilustran la función que el sistema debe de desarrollar (Yourdon, 1976). Los diagramas entidad-relación son los que nos muestran como están relacionados los datos; y quien depende de quien.

Finalmente los diagramas de transición, como su nombre lo indican, son los diagramas que van a mostrar y marcar el tiempo en que se lleva acabo una operación, cada uno de estos diagramas podrá ser documentado. El hecho de que podamos ver estos modelos visualmente, son de gran ayuda para los usuarios ya que son fáciles de leer. En este tipo de análisis para la modelación de sistemas son muy importantes tanto los datos como los procesos. Dentro de los procesos el analista puede verificar con el usuario, cuales deberán ser los resultados finales requeridos por el usuario.

2.3.5.3 Descripción General de la Metodología

Los procesos son mostrados gráficamente en forma de círculos, la comunicación entre los procesos y los usuarios se realiza por medio de semi-curvas (Yourdon, 1976).

Una de las partes importantes dentro de el análisis y desarrollo de la metodología de Yourdon es el ciclo de vida de un proyecto (project life cicle).

El objetivo básico, como se explica en la metodología, es el ciclo de vida y se divide en tres partes (Yourdon, 1976). Ver la tabal 3.6 donde se explican estas partes, que son actividades, consistencia y control.

Objetivos

Actividades	Definir las actividades que tiene que hacer el sistema en el desarrollo del proyecto.
Consistencia	El segundo objetivo es, introducir consistencia entre todos los proyectos desarrollados dentro de la misma organización
Control	Y por último se tiene que verificar los puntos del manejo de control. Esto quiere decir que los analistas deben prever un tiempo de terminación del proyecto, en esto influyen las tomas de decisiones, debe llevar un control de cuando es necesario volver hacer una revisión y cuando seguir a delante. Todo proyecto a traviesa la fase de análisis, diseño e implementación.

Tabla 2.6 Tabla de los objetivos de la metodología de Coad & Yourdon

En la metodología Yourdon, la importancia que tiene el proyecto del ciclo de vida para un analista, remarca que no sólo es importante el hecho de manejar herramientas para la modelación de un proyecto, también es importante los métodos que utilicemos para el buen desarrollo de este.

Dentro de la estructura del ciclo de vida, se tienen nueve actividades y tres terminales (Yourdon, 1976). Las actividades las se dividen de la siguiente forma mostrada en la tabla 3.7.

Actividades

Actividad 1	La supervivencia que Identifica las responsabilidades de cada usuario.
Actividad 2	Análisis del sistema.
Actividad 3	Diseño.
Actividad 4	Implementación.
Actividad 5	Aceptación de las pruebas de generación.
Actividad 6	Calidad.
Actividad 7	Descripción de los procedimientos.
Actividad 8	Conversión de la Base de Datos
Actividad 9	Instalación

Tabla 2.7 Tabla de las actividades de la metodología Yourdon

El diseño provee un método organizado rompiendo el problema original en pequeños problemas que podemos analizar profundamente.

La colección de procedimientos y conceptos generalmente incrementara la productividad y efectividad del proceso. Además, esta metodología combina servicios e información, e incrementa la modularidad. Las estructuras de control y datos pueden ser definidas en una manera integral.

2.3.6 Metodología Lenguaje de Modelación Unificado (UML)

UML son las siglas en inglés que significa Lenguaje de Modelación Unificado. Fue desarrollado para simplificar y consolidar un número extenso de metodologías orientadas a objetos. En 1996, OMG (Object Management Group) propuso un estándar para las metodologías orientadas a objetos. Booch, Jacobson y Rumbaugh comenzaron a trabajar con las metodologías y eventualmente todo el trabajo termino por desarrollar UML.

2.3.6.1 Definición

Este lenguaje nos permite hacer una visualización, construcción y documentar los diagramas que ocupemos en el diseño de software. UML se usa para la construcción, diseño y entendimiento de sistemas (Rumbaugh. et al, 1998).

2.3.6.2 Características

Se introduce el término unificado, porque dentro de este se encierran acepciones importantes, tales como:

A. A través de la historia de métodos, es decir, la combinación de conceptos de varios métodos de objetos orientados.

B. El siguiente es que el ciclo de vida de este término se refiere a que los diferentes conceptos y las notaciones, pueden emplearse en diferentes escenarios.

C. Con la implementación de los lenguajes y plataformas, que incluye diferentes lenguajes de programación, base de datos, a diferencia de lo que encontramos con los otros tipos de modelación, es que UML incluye onceptos semánticos, ciclos de vida, notaciones, etc.

D. Se cuenta con ambientes estáticos y dinámicos. Cuando se habla del ambiente estático se refiere a cuales son los objetos se consideran en el diagrama y las relación entre ellos, es decir el envío y recepción de mensajes entre ellos. En el ambiente dinámico se refiere a cual es el tiempo que se llevaran los objetos en realizar las operaciones y cuanto es el tiempo que tardan en finalizar sus tareas en conjunto.

E. Una de las herramientas importantes es que podemos generar código.

F. Cuenta con una estructura visual donde se puede hacer la representación de las personas que forman parte del sistema, así como de los objetos. Los objetos representan, cualquier cosa física o conceptual, por ejemplo libros, imágenes, automóviles etc.

2.3.6.3 Descripción General

UML trabaja con modelos por separado, se tienen los diagramas de clase, de colaboración, de secuencia, etc. Cada uno de estos diagramas pertenece a una estructura que al final se colocará en paquetes que ayudan a entender como trabaja el sistema, y al final todos tendrán una relación.

El propósito de UML es que todos los usuarios que modelan en algún lenguaje lo puedan usar (Rumbaugh. et al, 1998). La intención de esta unificación de los tres tipos de modelación OMT, Booch y Rumbaugh, es que sea un proceso completo de desarrollo. Se puede concluir que la construcción de UML, con todos sus diagramas, reglas y conceptos se agrupo en los siguientes panoramas: estructura estática, comportamiento dinámico, la construcción de la implementación, organización del modelo y por último los mecanismos.

Una de las ventajas que tenemos con UML, es que podemos hacer anotaciones para especificar el porque de ese elemento en el diagrama. Esto se debe a que si hay varios modeladores pueda haber una mejor comunicación en la revisión de los diseños. Anteriormente se planteo un panorama general de lo que contenía UML. Para tener una mejor idea de cómo se compone totalmente UML, dentro del área estructural se tienen siete diagramas elementales que se muestran en la tabla 3.8 (Rumbaugh. et al, 1998):

Diagramas

Diagramas de Casos de Uso	Los diagramas de Casos de Uso son dialogo entre el actor y el sistema, y contienen los siguiente elementos: ➢ actores ➢ asociaciones ➢ extensiones ➢ incluyentes ➢ generalización de los uso de caso.
Diagramas de Clase	Las clases son grupo de objetos con propiedades y comportamientos en común. Los diagramas de clases son creados para representar una vista de todas las clases en el modelo. Las clases con características en común se agrupan en paquetes, que es lo que llamamos bibliotecas, se componen a su vez de: ➢ clases ➢ asociaciones ➢ generalización ➢ dependencia ➢ realización ➢ interfaces.
Diagramas Secuencia	Un diagrama de secuencia muestra la interacción de los objetos y las clases envueltas en escenarios y la secuencia de mensajes entre los objetos, y se componen de: ➢ objetos ➢ líneas de tiempo ➢ fechas (representación del envió de mensajes)

Diagramas de Estados	Los diagramas de estados coordinan los mensajes que puedan recibir y mandar un objeto. Los estados son una condición durante la vida de un objeto durante la cual se satisfacen algunas condiciones, se desarrolla alguna acción, o se espera por un evento. El estado de un de un objeto se puede caracterizar por el valor de uno o mas atributos de una clase (Terry Quatrini, 1999). Estos diagramas se componen de: ➢ Estados de transición ➢ Estado inicial ➢ Estado final
Diagramas de Componentes	Un componente es la representación física de un archivo como puede ser en java como archivo.java. Las clases son mapeadas a un paquete quedando esta fusión como un componente, y se representan por: ➢ Componentes ➢ Líneas de unión
Diagramas de Colaboración	Estos diagramas muestran la interacción de los objetos organizado alrededor de los objetos y la relación entre cada uno, se componen de: ➢ Objetos (Dibujados en forma de rectángulos) ➢ Líneas de conexión ➢ Textos (Mensajes) ➢ Fechas (Apuntando del que envía el mensaje al que lo recibe).
Diagrama de Actividades	Estos diagramas presentan la parte dinámica del sistema, los diagramas de flujo muestran las actividades y el control en el sistema, también muestran que actividades se pueden realizar en paralelo, y cualquier camino a través del flujo de actividades (Terry Quatrini, ...). Contiene actividades, transiciones entre las actividades y puntos de decisión, se componen de: ➢ Actividad ➢ Transición ➢ Decisión ➢ Barras de sincronización

Tabla 2.8 Tabla de Diagramas principales en UML

UML es una forma de modelar y diseñar con un gran potencial, ya que contiene a los tres más grandes desarrolladores en la programación orientada a objetos, también cuenta con una herramienta que es Rational Rose con la cual se desarrolla un sistema hasta generar la estructura del programa.

2.4 Herramientas para el desarrollo de Objetos Orientados

2.4.1 Definición

CASE: Computer-Aided Software Engineering. Es una colección de herramientas que soportan el desarrollo del software en cada escenario. Las herramientas CASE son programas que nos ayudan en el análisis, diseño y soporte. En nuestro caso se usa Rational Rose que permitirá desarrollar el sistema propuesto.

2.4.2 Características de las herramientas CASE

i. Orientación Grafica.
ii. Fácil de usar.
iii. Descomposición de Procesos.

Se utilizan los siguientes parámetros para la evaluación de una herramienta CASE, como se muestra en la siguiente tabla:

Casos de Uso	Es un usuario amigable, fácil de usar, flexible, adaptable a las necesidades del usuario.
Robustez	Consistencia, integridad, buena sintaxis, buena semántica.
Funcionalidad	Eficiencia para soporte de los métodos.
Facilidad de Inserción	Para incorporar las herramientas en el ambiente de trabajo. El esfuerzo necesario para aprender la herramienta y usarla.

Tabla 2.9 Tabla de parámetros de evaluación de las herramientas case

Pero el entorno CASE, en sí mismo, necesita otros componentes. Un conjunto de servicios de portabilidad constituyen un puente entre las herramientas CASE y su marco de integración, así como la arquitectura de entorno. El marco de integración es un conjunto de programas especializados que permite a cada herramienta CASE comunicarse con las demás.

Por ejemplo para crear una base de datos de proyectos y mostrar una apariencia homogénea al usuario final. Los servicios de portabilidad que permiten las herramientas CASE y su marco de integración pueden migrar a través de diferentes plataformas hardware y sistemas operativos, sin grandes esfuerzos de adaptación.

La principal ventaja de la utilización de una herramienta CASE, es un incremento de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo además de la propia herramienta.

Otra de las ventajas de la herramienta CASE es que esta muy bien documentada no solo a nivel de ayuda, sino con ejemplos y tutoriales. La ayuda se despliega ya sea por temas y subtemas así como también por orden alfabético. Este tipo de herramientas contiene pequeñas ilustraciones junto con definiciones del tema buscado.

2.4.3 Ejemplo de herramientas CASE

1. Rational Rose
2. Visio System Architect
3. WinA&D
4. ASCENT

CAPITULO 3: APLICACIÓN DE UML PARA EL DESARROLLO DEL SISTEMA

3.1 Definición del sistema a modelar

Introducción

Actualmente se observa el gran beneficio en el área de Ciencias de la Tierra que se obtiene de la adquisición y manejo de imágenes satelitales. Con las imágenes los investigadores interpretan y analizan grandes extensiones de agua para conocer sus características. Debido la importancia en nuestro medio de aprovechar las imágenes para realizar los diversos estudios y dado el elevado costo y tiempo de adquisición. Son datos Radiométricos Satelitales para el estudio de los procesos oceánicos y de climatología de la temperatura de la superficie de los mares de México y aguas internacionales adyacentes. Dentro de las necesidades a cubrir la aplicación es que despliegue y almacene en una PC las imágenes procesadas y no procesadas, además de proporcionar ciertos recursos para realizar más procesos asociados a estas imágenes.

Para desarrollar la aplicación se realizara un análisis completo de los requerimientos del usuario, un factor importante para el desarrollo del sistema es reducir los tiempos de despliegue de las imágenes y los tiempos de acceso al conjunto de datos, con el propósito de acelerar la visualización de los mismos bajo los mismos pasos definidos por el lenguaje UML.

3.1.1 Objetivos del Sistema

Los objetivos principales del proyecto son:

i. Aplicación de UML para el diseño e implementación de un sistema para en procesamiento de imágenes satelitales.
ii. Manejo de imágenes, aplicando algún proceso.

3.1.2 Antecedentes

A fines de 1995 se instaló en el Instituto de Geografía (IG) de la UNAM un sistema de adquisición y manejo de imágenes de satélite. Los datos que se reciben en este sistema son de los sensores AVHRR (radiométricos avanzados de muy alta resolución) montados en los satélites NOAA-12, NOAA-14, NOAA-15 y NOAA-16, los cuales circundan la Tierra cada dos horas, en órbitas heliosincrónicas polares a 850 Km. de altura. Estos radiómetros miden 5 bandas espectrales de radiación (una visible, una en el infrarrojo cercano y tres en el infrarrojo-térmico), tiene una resolución espacial optima, de 1.1 Km. y cubren un área de 2800 Km. de ancho (en dirección transversal a la órbita del satélite) por casi 5400 Km. de largo, en cada paso por la zona de recepción de la antena. Cada satélite ve la misma escena terrestre dos veces al día. Se tiene un banco de datos o base de datos de imágenes satelitales, y a partir de este banco se desarrollará una serie de herramientas para los usuarios del Laboratorio de Oceanografía Física (LOF).

En 1999 se inició el análisis y el desarrollo de la base de imágenes satelitales (BITSMEX) en el LOF. Actualmente la Oceanografía satelital esta en pleno desarrollo y ha demostrado aptitud para contribuir al conocimiento del océano.

Características de una imagen

Actualmente las imágenes forman parte del área de investigación del procesamiento digital. La demanda del manejo de imágenes se ha incrementado en años recientes, gracias al manejo y mejoramiento de las impresiones y otras aplicaciones como TERASCAN. La razón para el interés de imágenes digitales es clara: la representación de imágenes en forma digital permite información visual para ser manipulada fácilmente de diversas formas útiles y originales. En un sistema de recepción vía satélite es necesario tomar en cuenta varios aspectos y parámetros como:

i. Hora de paso: Hora en que empieza la captura de la imagen desde el satélite.
ii. Satélite: Se cuentan actualmente con cinco satélites NOAA-12, NOAA-14, NOAA-15, NOAA-16 y NOAA-17.

iii. Área geográfica: FALTA
iv. La cantidad de líneas: Los satélites capturan un área, se define por 6,000 líneas horizontales y 2,400 verticales aproximadamente.
v. Sun elev: Es necesario conocer cual es el ángulo del sol con respecto a la antena en donde se reciben las imágenes.
vi. Sat elev: Es otro parámetro importante, y hace referencia al ángulo del satélite con respecto a la antena en donde se reciben las imágenes.

3.1.3 Requerimientos

Para desarrollar esta o cualquier otra aplicación es necesario en primer lugar determinar las necesidades del usuario, es decir, contar con un estudio previo de cuales son los objetivos que se pretenden lograr, por medio de una entrevista con el usuario o los usuarios, se llega a establecer y a concluir cuales serán los puntos para el análisis y desarrollo de la aplicación. Así, la interfaz entre la computadora y el usuario deben ser lo más comprensible y sencilla posible, utilizando herramientas CASE que faciliten el desarrollo del proyecto, uno u otro puede ser manejado por cualquier usuario del laboratorio.

De acuerdo al análisis y preguntas a los usuarios del laboratorio se puede resumir que los requerimientos de la aplicación a desarrollar, son cubiertos con las etapas, que se define a continuación:

a) Inicio: Después de contar con la información, lo más importante es el acceso a la misma, de nada serviría contar con los datos más exactos o precisos si esto se encuentran dispersos y perdidos o en un caso menos drástico, compilados dentro de una caja a la cual no se puede acceder de una maneja ágil lo cual impida su correcto análisis o tratamiento.

Es por ello que la facilidad con que el usuario pueda tener contacto con la aplicación y fácil manejo toma relevancia haciendo de este punto uno de las más sobresalientes. En el menú se presentan las opciones a trabajar con las imágenes, lo básico es:

i. Abrir: Esta opción ejecuta una pantalla auxiliar, cuyo objetivo es la elección y carga de una imagen.
ii. Salir : Esta opción, cierra la aplicación y regresa a Windows.
iii. Imprimir.
iv. Guardar.

b) Imagen: El amplio mapa de imágenes proporciona información basta y general, dentro de la cual hay información a detalle, que puede pasar desapercibida para los ojos del analista y con esto dejar a un lado cuadros importantes, al momento de obtener conclusiones o de formular algún replanteamiento sobre el estudio del comportamiento del universo en estudio y por ello en el análisis de imágenes. Y buscar ampliar la visión del intérprete, así como proporcionar información específica de una sub-área de interés que permita llevar a una nueva dimensión al marco de observación.

i. Recorte: Una vez seleccionada la imagen es posible que sólo se requiera estudiar una pequeña porción de esta, con esta opción se podrá seleccionar una región de la imagen.

ii. Histograma: Muestra una gráfica probabilística que proporciona la distribución de cada valor de los pixeles dentro de una imagen. Ver figura 1

iii. Zoom in: Maximizar por pasos la Imagen.

iv. Zoom out: Minimizar por pasos la imagen.

v. Grid: Malla de referencia geográfica.

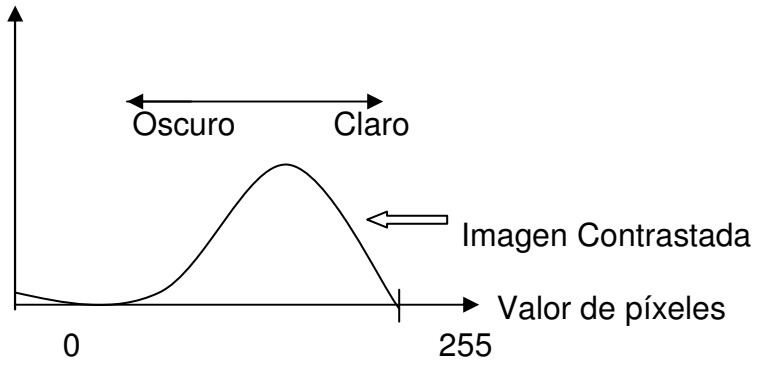


Fig. 3.1 Gráfica del histograma

c) Procesos: La información adquirida y representada dentro de una imagen tiene más de una cara, el experimentar en un sólo plano puede llegar a limitar la alimentación de datos del mundo de estudio hacia el sistema de análisis, de esto se deriva la importancia de promover un escalón hacia arriba en la escalara de procedimientos de estudio. Dejando a un lado el abstencionismo de inferir conforme la información de primer plano llega a convertirse en un reporte superficial, y debido a esto es importante adentrarse en los diferentes planos y representaciones de esta información buscando obtener mayor provecho de la misma.

i. Rotar: Se utilizan para rotar, reducir o agrandar una imagen, es decir una modificación de la imagen.
ii. Realce: Sirve para contrastar la imagen, que es una manipulación del histograma.
iii. Segmentación:
iv. Binarizacion:

d) Ayuda: Es importante tener una guía que proporcione una pronta respuesta a nuestras dudas y preguntas acerca del contenido del programa. Por esto se agregan estas dos opciones.

i. Acerca de
ii. Contenido

La ventaja de las imágenes de tipo satelital es que se tiene una gran cobertura. Se deben tener ciertos requisitos indispensables como la ausencia de nubes y la inclinación del satélite. La información básica en la captura de las imágenes en el laboratorio es la siguiente: para cada satélite el año, mes, día y hora de captura. A continuación se muestra en la siguiente tabla un resumen de los requerimientos del sistema a desarrollar.

Requerimientos del sistema para el procesamiento de imágenes satelitales	
1.- Inicio	i. Abrir ii. Guardar iii. Salir iv. Imprimir
2.- Imagen	i. Recorte ii. Histograma iii. Zoom in iv. Zoom out v. Grid
3.- Procesos	i. Realce ii. Binarización iii. Segmentación iv. Rotación
4.- Ayuda	i. Acerca de ii. Contenido

Tabla 3.2 Tabla de Requerimientos

3.2 Metodología UML empleando la herramienta Rational Rose

La metodología CASE es un enfoque estructurado para los sistemas de ingeniería en un entorno de procesamientos de datos. Se compone de un grupo de etapas, tareas y técnicas, que permiten cumplir con todas las etapas del ciclo de vida de un sistema (Quatrini, 1999).

Rational Rose esta diseñada para proveer el desarrollo del software con una completa visualización para el desarrollo de soluciones eficientes en negocios. Esta herramienta provee la capacidad de (Quatrini, 1999):

I. Identificar y diseñar los objetos.
II. Partición de los servicios a través de los modelos
III. Diseñar como los componentes serán distribuidos a través de la red.
IV. Genera código directamente de los modelos.
V. Utiliza ingeniería inversa para crear modelos de los componentes existentes y las aplicaciones.
VI. Facilita la sincronización de los modelos con los códigos.

Tabla 3.3 Diseño

3.2.1 Definir las vistas de los objetos de usuario

Con esta actividad se busca identificar los atributos de un objeto que el usuario requiere para ejecutar una tarea en particular. Cada vista debe tener un conjunto de atributos y acciones. Se considera importante ya que esto permite disminuir la información a mostrar en las diversas ventanas, dependiendo de una tarea específica, haciendo más fácil a las mismas.

3.2.2 Definir las ventanas

En esta actividad se busca definir la apariencia y comportamiento de las ventanas principales; definiendo las acciones y dependencia de las mismas, seleccionando controles para representar los atributos. Se considera importante ya que se están definiendo las ventanas que serán parte de la interfaz y como será la navegación entre ellas. La actividad debe expresar los escenarios como acciones sobre las ventanas.

3.2.3 Diseño

Una vez identificados los actores para el sistema. El siguiente paso es identificar como interactúa el actor en el sistema. Con la identificación de los actores se crea el funcionamiento de estos en el sistema.

En el caso del sistema a diseñar se define un sólo actor usuario LOF, el cual es el interesado en procesar imágenes satelitales.

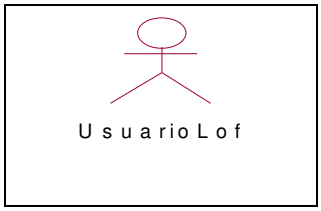


Fig. 3.4 Ejemplo de Usuario

3.2.3.1 Encontrando Casos de Uso

Descripción de las actividades del usuario LOF en el sistema:

- Seleccionar una imagen
- Realizar un proceso sobre la imagen
- Salir, Imprimir o guardar una imagen

Los casos de uso como ya mencionamos es el dialogo entre el actor y el sistema. Acontinuacion se definen los casos de uso del sistema:

a) Inicio

En la Fig.3 Muestra el diagrama de **Caso de Uso de Inicio**. De lado derecho se puede ver que los casos de uso se van integrando al usuario Lof. De esta forma empezamos a crear el panorama de los casos de Uso.

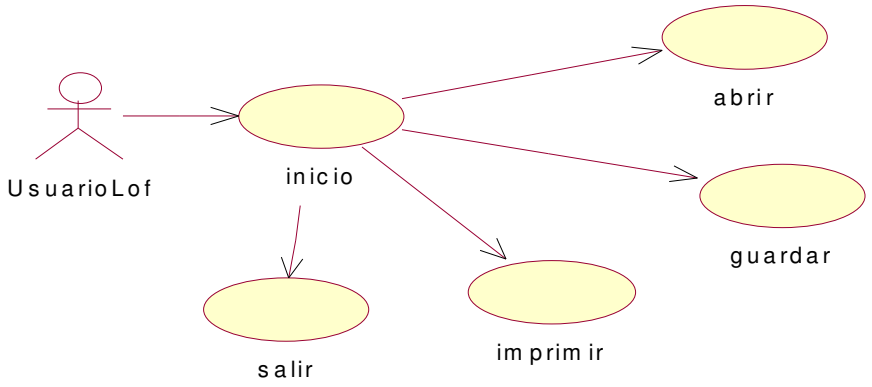


Fig. 3.5 Caso de Uso Inicio

b) Imagen

En la Fig. 4 se muestra el Diagrama caso de uso **Imagen**. Se contempla cinco opciones: Recorte, Histograma, Zoom in, Zoom out, Grid

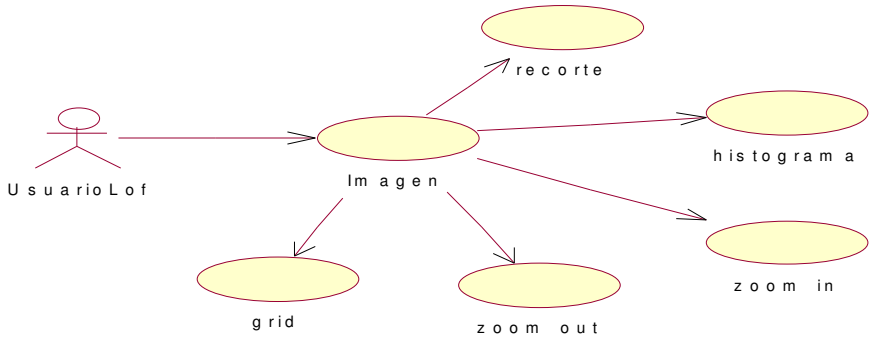


Fig. 3.6 Caso de Uso Imagen

c) Proceso

El **Caso de Uso Procesos** mostrado en la Fig. 5 , contiene 4 opciones: Realce, Binarización, Segmentación y Rotación.

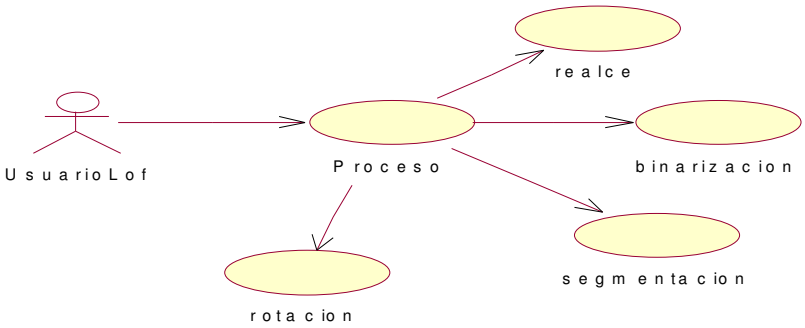


Fig. 3.7 Caso de Uso Proceso

d) Ayuda

La Fig. 6 muestra el **Case de Uso Ayuda** en el cual tenemos 3 opciones: Acerca de y Contenido.

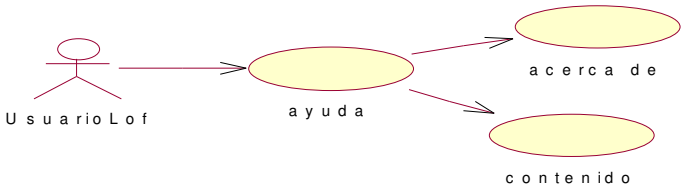


Fig. 3.8 Caso de Uso Ayuda

La Fig. 6 muestra el **Case de Ayuda,** en el cual tenemos dos referencias acerca de y contenido.

3.2.3.2 Definiendo los Diagramas de Secuencia

Los Diagramas de Secuencia muestran los objetos de interacción en la secuencia de tiempo (Quatrini, 1999). Como se menciono anteriormente cada objeto tiene una línea de tiempo representada por una línea horizontal.

La secuencia es la siguiente: Encontramos al sistema, seleccionamos un listado el sistema busca estas listas y despliega los formatos. Se muestra el ejemplo del Diagrama de Secuencia, en el Anexo A se muestra el diagrama completo.

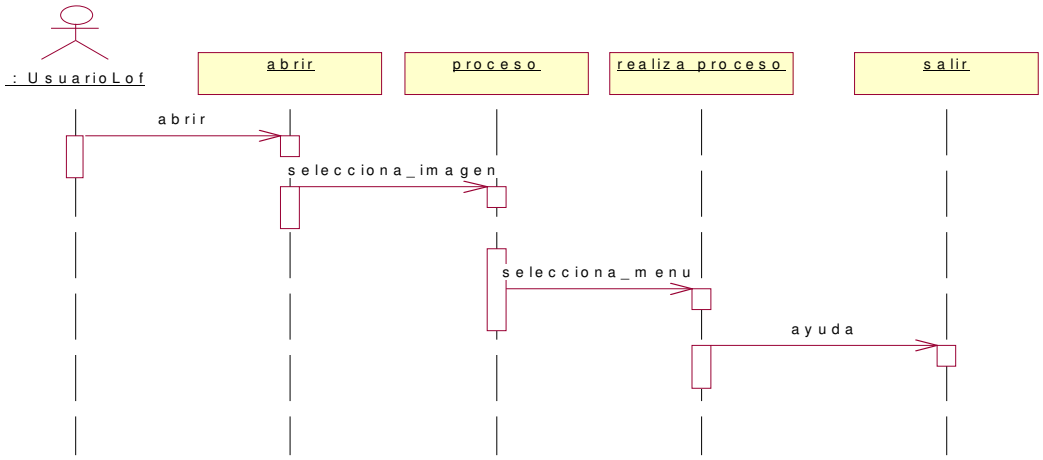


Fig. 3.9 Diagramas de secuencia

3.2.3.3 Diagramas de Colaboración

Un Diagrama de Colaboración es una interacción de diagrama que presenta la secuencia de mensajes que implementan una operación o una transacción. Estos diagramas pueden contener simples clases de instancias.

Cada diagrama provee una vista de la interacción o estructura de la relación que ocurre entre los objetos y los objetos con identidades iguales en el modelo.

Para crear un diagrama de Colaboración se obtiene de la información contenida del diagrama de Secuencia. Los diagrama de colaboración contiene iconos representan objetos. El diagrama completo se muestra en el anexo A.

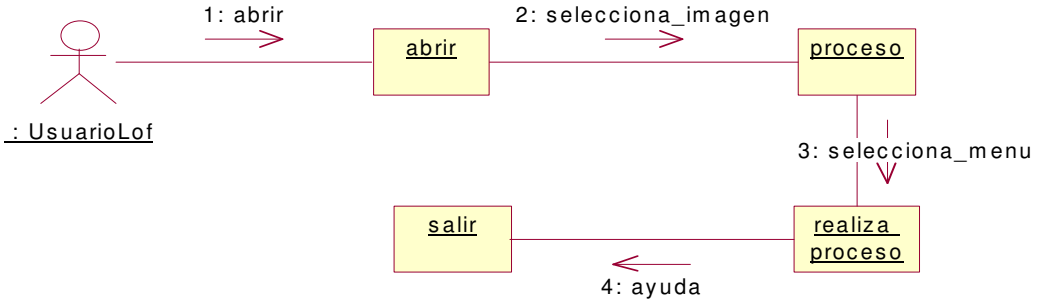


Fig. 3.10 Diagramas de colaboración

3.2.3.4 Encontrando Clases

La definición de clases es un paso importante y es posterior a la definición del caso, una vez que dentro de nuestro ciclo de trabajo hayamos definido cual son nuestros casos de uso se procede a realizar la tarea de definir las clases. En una sección previa dentro de este trabajo se menciono que la clase se entiende como la descripción de un grupo de objetos los cuales cuentan con propiedades o características en común. Las clases también funcionan como la plataforma para dar vida y crear los objetos.

Las imágenes son objetos que pertenecen a las imágenes procesadas o no procesadas. Cada imagen tiene un atributo y acceder a estas operaciones específicas se hace mediante la utilización de herramientas diseñadas para este fin como lo son las opciones de las consultas.

Lo que hace diferente a cada una de estas imágenes es que dentro de ellas siempre vamos a encontrar perfiles únicos que no se repiten dos veces, la recolección de estas imágenes se llevo acabo en periodos de tiempo previamente establecidos y diferidos dando por resultado que cada una de ellas fuera capturada en diferentes días y en diferentes horarios.

Metodología para encontrar clases

i. El cuerpo de una clase contiene una serie de responsabilidades que definen el comportamiento de los objetos.
ii. Las operaciones pueden ser creadas independientemente de los diagramas, ya que algunas veces no todos los escenarios son descritos o representados. Algunas clases no son fundamentales
iii. Cada atributo esta definido por los objetos. Cada clase define que tiene un valor. Por ejemplo, cada imagen tiene el atributo nombre, mes en que fue tomada, año, tamaño de imagen y un identificador. Los atributos como las operaciones deben ser escritas con letras minúsculas.
iv. Cada clase será definida con estereotipos.
v. Durante el análisis se definieron tres estereotipos. Ya que son útiles para definir las clases requeridas durante el desarrollo

De acuerdo a lo relacionado con UML las clases se clasifican como se muestra en la siguiente figura:

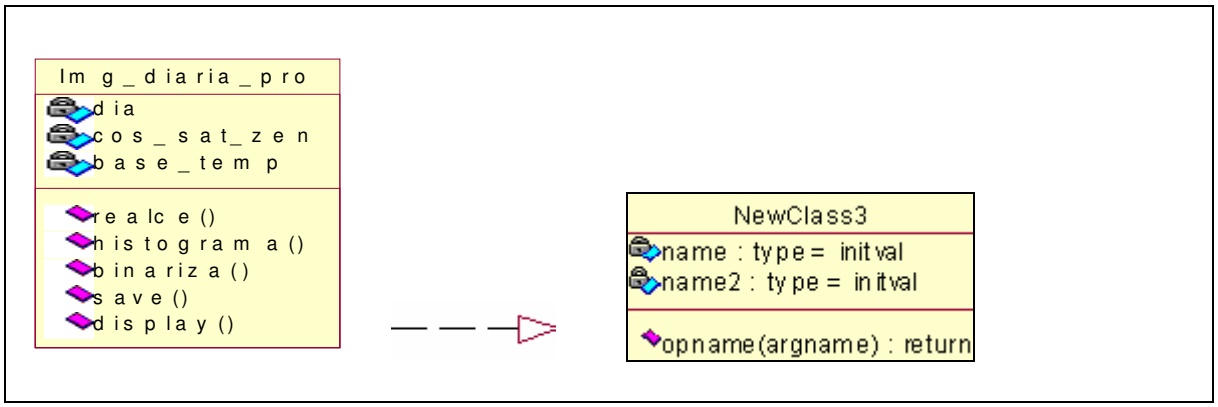


Fig. 3.11 representación de una clase

El Diagrama de clase esta constituido por la relación que existe entre cada grupo de los objetos. Es uno de los diagramas más importantes dentro de UML, ya que es el que nos proporciona el código. En el anexo A se muestra el diagrama de clases completo, del sistema.

3.2.3.5 Descripción de pasos para Generar la estructura del sistema

Uno de los diagramas importantes en Rational Rose es el diagrama de componentes, este nos permite generar código. El tipo de archivo que se obtiene no es siempre el mismo, este llega a presentar variantes las cuales van a depender del lenguaje que se utilice a lo largo del desarrollo.

Cada componente esta asignado a un lenguaje (Quatrini, 1999). Para la obtención de código fue necesario mapear las clases del diagrama de clases a el diagrama de componentes. El mapeo de clases es uno a uno, es decir una clase corresponde únicamente a un componente (Quatrini, 1999).

Los componentes generados para el desarrollo del diseño se representan de la siguiente forma. En el Anexo A se muestra el diagrama completo.



Fig 3.12 Componentes para generar código

3.2.3.6 Reportes obtenidos a partir de la generación de código

Este comando esta diseñado para generar un diccionario de datos o información de un modelo usando como medio de reporte a Microsoft World OLE

(Rational Rose software). La utilización de un modelo también comprende saber las características del mismo y por ello es muy valioso poder acceder a un comando que nos despliegue la información relevante al modelo usado.

Cuando seleccionamos en Rational Rose Documentación de Reportes, se despliega automáticamente la siguiente información (El diagrama completo se muestra en el anexo):

Logical View Report

Unified Modeling Language Syntax

Fig. 3.13 Presentación para generar código

Administrador

Private Attributes:
name : Name
usuarioID : Number
Public Operations:
open_system (argname :) : return
edit_menu2 (argname :) : return

Fig. 3.14 Ejemplo de Atributos

3.3 Programa Final

En Rational Rose cada componente representa el código de esa clase. A cada componente le asignamos un lenguaje.

Las clases las mapeamos en los componentes, el mapeo uno a uno facilita el entendimiento del código. Cada clase tiene su propia cabecera y un archivo .ccp (Quatrini, 1999). Los DLLs son bibliotecas, únicamente las bibliotecas puedes ser sujetas a cambios (Quatrini, 1999).

Dos archivos son generados para cada clase del encabezado (.h) y un archivo de especificación (.cpp).Una vez que los que las cabeceras son creados se les asigna un lenguaje. El lenguaje utilizado es JAVA, Rational Rose automáticamente asigna Java a los componentes.

Procedimiento para generar código:
Son seis los pasos a seguir en Rational Rose para generar código (Quatrini, 1999)

i. Crear las propiedades necesarias
ii. Crear los componentes
iii. Asignar código a los componentes
iv. Asignar clases a los componentes
v. Asignar propiedades a los elementos
vi. Seleccionar los componentes para generar código
vii. Evaluación de los Errores

Dentro de Rational Rose se tiene la propiedad de generar un LOG, este nos indica si tiene errores el sistema. Se muestra en el siguiente diagrama:

```
22:31:23|  Generating Java class: Imagen
22:31:25|
22:31:25|  [Generating Java]
22:31:27|  [Generating Java]
22:31:27|  Error: Img_lista -- name of public class and compilation unit must  match
22:32:12|  Generating Java class: Imagen
```

Fig. 3.15 Errores en el Programa

Este es un ejemplo del programa generado. En el Anexo B se presenta el código final

```
// Source file: img_infgeo.java
public class Img_infgeo {
    private int master;
    private int siglas_region;
    private int nombre;
    public Img_recorte m_Img_recorte;
```

Fig. 3.16 Ejemplo de Código

Las clases que se crean son las siguientes:



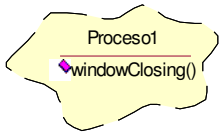
Fig. 3.17 Clases generadas por Java

CAPITULO 4: RESULTADOS

4.1 Cuadro Comparativo de las metodologías.

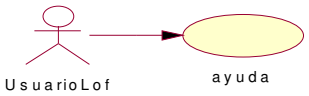
El siguiente cuadro comparativo muestra un resumen las diferentes metodologías estudiadas en el desarrollo de este trabajo. La finalidad es mostrar los elementos más notables como: historia, concepto descripción general, características, tipo de diagrama y una conclusión del la metodología.

Metodología	**Booch**
Historia 1991	Object-Oriented Design (OOD), Booch
Concepto	No es un proceso aislado. Interactúa con siguientes y precedentes en una especie de Bucle.
Descripción General	Es un leguaje para expresar modelos de sistemas, es un proceso que te guía para producir esos modelos y convertirlos en sistemas ejecutables.
Características	Metodología Booch usa los siguientes tipos de diagramas para describir las decisiones de análisis y diseño, tácticas y estratégicas, que deben ser hechas en la creación de un sistema orientado por objetos. 1. Diagrama de Clases. Consisten en un conjunto de clases y relaciones entre ellas. Puede contener clases, clases paramétricas, utilidades y metaclases. Los tipos de relaciones son asociaciones, contenencia, herencia, uso, instancias y metaclase. 2. Especificación de Clases. Es usado para capturar toda la información importante acerca de una clase en formato texto. 3. Diagrama de Categorías. Muestra clases agrupadas lógicamente bajo varias categorías 4. Diagramas de Transición de Estados. 5. Diagramas de Objetos. Muestra objetos en el sistema y su relación lógica. Pueden ser diagramas del escenario, donde se muestra cómo colaboran los objetos en cierta operación; o diagramas de instancia, que muestra la existencia de los objetos y las relaciones estructurales entre ellos. 6. Diagramas de Tiempo. Aumenta un diagrama de objetos con información acerca de eventos externos y tiempo de llegada de los mensajes. 7. Diagramas de módulos. Muestra la localización de objetos y clases en módulos del diseño físico de un sistema. 8. Subsistemas. Un subsistema es una agrupación de módulos, útil en modelos de gran escala. 9. Diagramas de procesos. Muestra la localización de los procesos en los distintos procesadores de un ambiente distribuido

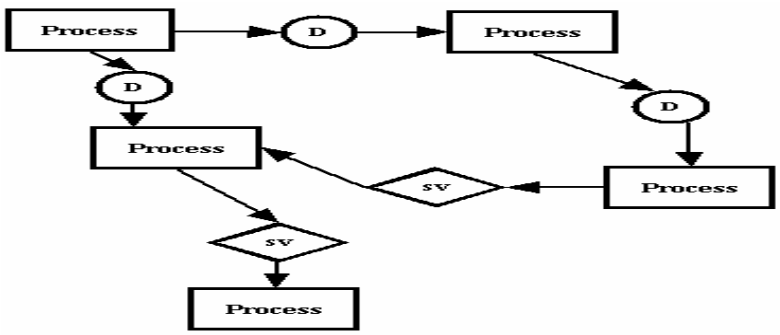
Metodología	**Booch**
Tipo de Diagrama	Este es un ejemplo diferenciado a las demás técnicas de modelación. 
Conclusión	Grady Booch es reconocido internacionalmente por la innovación de su trabajo en arquitectura de software, modelación e ingeniería de software.
Metodología	**Rumbaugh-OMT**
Historia 1991	Object Modeling Technique (OMT), Rumbaugh
Concepto	OMT hace un cubrimiento de las etapas de análisis, diseño e implementación definidas por la OMG, dejando sin cubrir el modelamiento estratégico. Esta metodología orientada a objeto muy difundida que se hace cargo de todo el ciclo de idea del software. Parte de la idea de utilizar los mismos conceptos y la misma notación a lo largo de todo ciclo de vida. Tiene una fase de diseño no muy compleja y se centra mucho en un buen análisis. Divide el ciclo de vida del software en cuatro fases consecutivas: análisis de objetos, diseño del sistema. Diseño de objetos e implementación.
Descripción General	El método OMT describe el Análisis, Diseño e Implementación. OMT divide el Análisis y Diseño en tres partes(Rumbaugh, 1991): Análisis: Construye un modelo del mundo real, a partir de un problema. Diseño del sistema: Diseño de toda la arquitectura de un sistema. Diseño de objetos: Refinamiento de la estructura de los objetos, a lado de la eficiencia de la implementación, y de los detalles que son añadidos a los objetos.

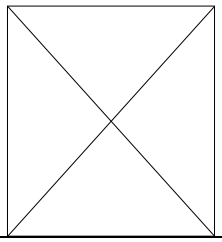
Metodología	**Rumbaugh-OMT**
Características	Las características de OMT se encuentran esencialmente en la identidad, la clasificación (el grupo de objetos en las clases), el polimorfismo y la herencia. ➢ Modelo de Objetos. Se define como un diagrama de objetos mas un diccionario de datos. El diagrama de objetos muestra clases y sus relaciones (generalización, agregación, asociación, instanciación). El diccionario de datos es el detalle de las clases en el diagrama de objetos ➢ Modelo dinámico. Se define como un conjunto de diagramas de estado mas un diagrama de Flujo de eventos Global. ➢ Modelo funcional. Es un diagrama de flujo con restricciones.
Tipo de Diagrama	
Conclusión	Es muy particular la forma en que OMT se distingue de las demás metodologías un ejemplo particular es el siguiente: Sistema- Rectángulo Clase- Caja con tres secciones Objeto – Rectángulo

Metodología	**Jacobson- OOSE**
Historia 1992	Object Oriented Software Engineering OOSE Ivar Jacobson
Concepto	OOSE es la combinación de dos diferentes técnicas, las cuales han sido usadas por largo tiempo. La primera técnica es programación orientada a objetos, la cual fue desarrollada en 1960 y pronto pareció ser usada en muchas áreas de aplicación (Jacobson, 1995).
Descripción General	Es un proceso organizado para la construcción industrial de software. Este proceso de diseño está guiado por casos de uso, una técnica que basa el entendimiento de un sistema en la forma en la cual es usado. Define el modelo de análisis, identificando objetos entidad, de interfaz y de control; independientes del ambiente de implementación.
Características	OOSE también llamado "Caso de uso aproximado". En este acercamiento, un modelo de Casa de Uso sirve como modelo central, para el cual otros modelos son derivados (Jacobson, 1995). Un modelo de Caso de uso describe la completa funcionalidad de un sistema. Los modelos de caso de uso son la base en el análisis, construcción y prueba. Lo más importante en el análisis es entender al sistema de acuerdo a la función de sus requerimientos.Objectory es un proceso organizado para la construcción industrial de software. Este proceso de diseño está guiado por casos de uso, una técnica que basa su centra el entendimiento de un sistema en la forma en la cual es usado. ➢ Modelo de Casos de Uso: Se basa en la descripción de elementos o usuarios externos al sistema (actores) y funcionalidad del sistema (casos de uso). ➢ Modelo de objetos: Representa la estructura estática de objetos. Puede contener objetos entidad, de interfaz y de control, entre otros tipos; y relaciones de herencia, conocido (una referencia estática), consiste de y comunicación. ➢ Diagrama de interacción. Muestran la secuencia de eventos entre paquetes u objetos necesarios para realizar un caso de uso. ➢ Diagrama de estado. Muestra los estados internos de un objeto complejo.

Metodología	**Jacobson- OOSE**
Características	Algunos de los conceptos más importantes de esta metodología son: ➢ Objeto Entidad. Representa información del sistema que debe sobrevivir cierto período de tiempo, por ejemplo, un caso de uso ➢ Objeto de Interfaz. Modela información y comportamiento que es dependiente de la interfaz actual del sistema. ➢ Objeto de Control. modela funcionalidad que no corresponde a ningún objeto en particular y que se presenta en algunos casos de uso. Estos objetos generalmente operan sobre varios objetos entidad, realizan algún algoritmo y retornan algún resultado a un objeto de interfaz. ➢ Paquete. Módulo que contiene código, traducible a un módulo en el lenguaje de implementación. ➢ Unidad. En pruebas, desde una clase hasta un subsistema
Tipo de Diagrama	
Conclusión	De la programación orientada, OOSE únicamente usa conceptos de encapsulación, herencia y relación entre clases e instancias. La segunda, los modelos conceptuales que son usadas para crear diferentes modelos del sistema u organización para ser analizada.

Metodología	**Jackson- JSD**
Historia	Sistema de Desarrollo Jackson (JSD) fue desarrollada por Michael Jackson entre 1972 y 1974. Las especificaciones de JSD consisten principalmente de una distribución de redes de procesos que se comunican mediante el envio de mensajes.
Concepto	Es un método de desarrollo del sistema que cubra el ciclo vital del software directamente o proporcionando un marco. El desarrollo del sistema de Jackson puede empezar con la etapa en un proyecto cuando hay solamente una declaración general de requisitos. El primer método de Jackson, programación estructurada Jackson (JSP), se utiliza para producir el código final. Desde el punto de vista técnico hay tres etapas importantes en el desarrollo, cada uno dividido en pasos y subpasos del sistema de Jackson. Desde el punto de vista de un líder hay un número de maneras de organizar este trabajo técnico. Primero describimos las tres etapas técnicas principales y en seguida discutimos el planeamiento del proyecto de JSD, la variación entre los planes, y las razones de elegir u otro. A continuación se muestran los Escenarios propuestos por Jackson: Etapa de modelación Etapa de red Escenario de implementación Proyecto y Planes
Descripción General	**JSD: Etapa de modelación** En la etapa que modela se hace una descripción de los aspectos del negocio o de la organización que el sistema será referido. La descripción debe analizar el proyecto, eligiendo cuál es relevante y no haciendo caso de cuál no es. La descripción modelo se escribe muy exacto. Esta precisión fuerza a hacer preguntas detalladas. La descripción modelo consiste en acciones, entidades y la información relacionada. Una acción es un acontecimiento, generalmente en la realidad externa, que es relevante al sistema y que ocurrencia debe registrar el sistema. Comenzamos el desarrollo del sistema de Jackson haciendo una lista de acciones con definiciones y cualidades asociadas. El resultado de la etapa que modela es un sistema de las tablas, de las definiciones y de los diagramas que describen: En términos del usuario exactamente qué sucede en la organización y qué tiene que ser registrada sobre lo que sucede. En términos de la puesta en práctica, el contenido de la base de datos, los apremios de la integridad y las reglas de la actualización.

Metodología	Jackson- JSD
Descripción General	**JSD: Etapa de Red** En la etapa de la red acumulamos una descripción exacta de lo que tiene que hacer el sistema, incluyendo las salidas que deben ser producidas y la manera el sistema es aparecer al usuario. Comenzamos esta red haciendo un programa para cada uno de las entidades que fue definida durante la etapa que modelaba. La red entonces es acumulada incremental agregando nuevos programas y conectándolos hasta la red existente. Los nuevos programas se agregan por las razones siguientes: Para recoger las entradas para las acciones, compruébelas para saber si hay errores, y páselos a los programas de la entidad. Para generar las entradas para las acciones que no corresponden a los acontecimientos externos. Tales acciones son substitutos para los acontecimientos verdaderos del mundo, quizás porque esos acontecimientos no pueden ser detectados. Para calcular y producir salidas. Los diagramas son apoyados por la información textual que describe el contenido de las secuencias de datos y de las conexiones del vector del estado.
Características	La importancia de la metodología es archivar una descomposición del desarrollo de tareas para una clase de problemas. Otro punto importante es el tener puntos de revisión del desarrollo (JSP & JSD, 1989). Los datos y la programación estructurada deben tener la propiedad de ser fáciles de leer y escribir.
Tipo de Diagrama	Diagrama de Red 
Conclusión	Los diagramas de la red demuestran la interacción entre los procesos. Se refiere a veces como diagramas de la especificación de sistema. Los diagramas de la estructura de la entidad ilustran las acciones tiempo – perdidas que las entidades se realizan dentro del sistema.

Metodología	**Yourdon & Coad**
Historia 1990	Object Oriented Analysis (OOA), Coad/Yourdon
Concepto	Object-oriented analysis (OOA) esta relacionado el desarrollo de Software de ingeniería de Software y las especificaciones que expresan un objeto dentro del modelo del sistema.
Descripción General	El análisis de Yourdon & Coad y el diseño orientados al objeto (OOA/OOD) es un método orientado al objeto que precede UML. Para dibujar un diagrama de Yourdon & Coad, cuenta con estos cinco pasos: 1. Encontrar las clases y los objetos 2. Identificar las estructuras 3. Definir los temas 4. Definir las cualidades 5. Definir los servicios
Características	El símbolo de clase de OOA, representando una clase, sus cualidades, y sus servicios. El símbolo de OOA Clase-y-Objeto • Notación de la estructura Generalización-Especialización • Notación de la estructura Entero-Parte • Notación del temas • Notación de atributos • Notación de la conexión de la instancia • Notación de servicio • Notación Diagrama Objeto Estado • Notación de Grafica de Servicio
Tipo de Diagrama	Diagrama Clase y Objeto - parte  Relación entero 
Conclusión	Los objetos y las clases son abstracciones de entidades con servicios y calidades exclusivos. La relación entre los objetos que contiene unos o mas otros objetos. Hay varios tipos de relaciones de la entero-arte incluyendo: montaje – parte, envase- contenido (gabinete- archivos).Las relaciones de la Generalización – Especialización refieren a las clases que heredan cualidades y servicios.

Metodología	**UML**
Historia 1994	Es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos.
Concepto	UML prescribe una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos. Previamente, un diseño orientado a objetos podría haber sido modelado con cualquiera de la docena de metodologías populares, causando a los revisores tener que aprender las semánticas y notaciones de la metodología empleada antes que intentar entender el diseño en sí. Ahora con UML, diseñadores diferentes modelando sistemas diferentes pueden sobradamente entender cada uno los diseños de los otros. El sistema debía se ser capaz de modelar no solo sistemas de software sino otro tipo de sistemas reales de la empresa, siempre utilizando los conceptos de orientación a objetos OO.
Descripción General	UML es un lenguaje estándar para la especificación, visualización contracción y documentación de los artefactos de software. UML presenta una colección de las mejores practicas de ingeniería que han probado tener éxito en la modelación el el largo y complejo de sistema.
Características	Los conceptos básicos por tipo de diagrama: • Diagrama de Estructura Estática • Diagrama de Casos de Uso • Diagrama de Secuencia • Diagrama de Colaboración • Diagrama de Estados • Diagrama de Actividades • Diagrama de Implementación Los conceptos avanzados por tipo de diagrama son: • Diagrama de Estructura Estática • Diagrama de Secuencia • Diagrama de Colaboración • Diagrama de Estados • Diagramas de Casos de Uso para modelar los procesos 'business'. • Diagramas de Secuencia para modelar el paso de mensajes entre objetos. • Diagramas de Colaboración para modelar interacciones entre objetos. • Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.

	• Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones. • Diagramas de Clases para modelar la estructura estática de las clases en el sistema.

Metodología	**UML**
Características	• Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema. • Diagramas de Componentes para modelar componentes. • Diagramas de Implementación para modelar la distribución del sistema.
Tipo de Diagrama	
Conclusión	UML no define procesos concretos que determine las fases de desarrollo de un sistema. UML es un método independiente del proceso. Los procesos de desarrollo deben ser definidos dentro del contexto donde se van a implementar los sistemas.

4.2 Programas obtenidos por Rational Rose

De acuerdo a los resultados obtenidos por Racional Rose.

I. // Source file: **ImgeProcessingTest.java**
II. // Source file: **HistoRV.java**
III. // Source file: **Object.java**
IV. // Source file: **Grid.java**

// Source file: **ImgeProcessingTest.java**

```
public final class ImageProcessingTest extends Object {
    ImageProcessingTest() {
   }
```

```
  /**
      @roseuid 427D10F201DB
     */
    public void main( argname) {
    }
}
```

// Source file: **HistoRV.java**

```
public final class HistoRV extends Frame {

    private imagen imageFuente;
    private int initAncho;
    private init initAlto;
    private imagen ImagenNueva;
    private init insertArriba;
    private init insertIzqda;
    private int[] histored = new init [256];
    private init[] histogreen = new init [256];
    private init[] histoblue = new init[256];

    HistoRV() {

    }

    /**  @roseuid 427D104E0117  */

    public return main( argname) {

    }

        /**   @roseuid 427D10550366 */

    public HistoRV( argname) {

    }

    /** @roseuid 427D105A00C4 */

    public return paint( argname) {

    }   }
```

```
// Source file: Object.java

public final class Object {

Object() {
    }
    /** @roseuid 427D10C802B7  */
    public void object( argname) {
    }
    /**  @roseuid 427D10CB03AB  */
    public return registerNatives( argname) {
    }
  /**  @roseuid 427D10CD03E0  */
    public return getclass( argname) {
    }
    /** @roseuid 427D10D00330 */
    public return hashcode( argname) {
    }
    /** @roseuid 427D10D3006E  */
    public return equals( argname) {
    }
    /**  @roseuid 427D10D501D9 */
    public return clone( argname) {
    }
    /**  @roseuid 427D10D80007 */
    public return toString( argname) {
    }
     /** @roseuid 427D10DD02CB */
```

```
 public return opname8( argname) {
    }
     /** @roseuid 427D13A400BB */
    public return notify( argname) {
    }
    /**  @roseuid 427D13BE025D */
    public return wait( argname) {
    }
}

// Source file: Grid.java
public final class Grid extends Object {
    private imagen imagenFuente;
    Grid() {
    }
    /**
       @roseuid 427D10E90097
     */
    public void main( argname) {
    }
}
```

CONCLUSIONES

Con el estudio de las diferentes metodologías podemos concluir que Booch se enfoca en las clases, las cuales siempre son agrupadas en varias categorías, a diferencia de la metodología de OMT en hace énfasis en el diseño del sistema y de los objetos. Esto quiere decir que para el sistema es importante la arquitectura y en los objetos la eficiencia de estos. Para OOSE la base principal no son los diagramas de clase sino los casos de uso: un caso de uso es la base principal en todo el desarrollo, debido a que involucra: análisis, construcción y prueba. En el análisis se centrara el entendimiento de los requerimientos del usuario, por lo que se plasmará de manera directa en el caso de uso.
Una diferencia muy marcada la encontramos en la metodología JSD ya que no se centra en los diagramas de clases y uso. Para Michael Jackson lo principal son los escenarios: Etapa de modelación, etapa de red, implementación y Proyecto - planes. Más que diagramas, son descripciones claras de las necesidades del proyecto. Visualizando las metodologías encontramos que Yourdaon & Coad es un método que precede a UML ya que encontramos en el desarrollo clases y objetos. Existe una variada selección de notaciones como por ejemplo: Notación de temas, atributos, servicio, conexión etc.

La aplicación de UML para desarrollo del sistema propuesto nos permitió visualizar en forma completa los procesos que integran el programa. En el diagrama de Casos de Usos se plasmo de manera precisa los actores y las actividades para la manipulación y procesamiento de imágenes satelitales. Por otra parte en el diagrama de clases se identifica la estructura del programa integrado por las clases definidas. Con lo anterior se muestra la importancia de modelar ya que se simplifica el desarrollo del sistema, en el cual se plasma la realidad y los requisitos a satisfacer a los usuarios del sistema.

Se puede concluir que el desarrollo de la aplicación para el procesamiento de imágenes satelitales fue elaborado aplicando la metodología UML con lo cual se cumple como objetivo general de la presente Tesis, así como sus objetivo particulares.

La herramienta utilizada para el desarrollo del software Racional Rose, solo se emplearon los elementos mas importantes de la modelación. Por lo cual es necesario hacer un estudio a futuro de las herramientas para el desarrollo del software.

ANEXO A

4.1 Introducción a UML

La notación UML, se ha convertido desde finales de los 90 en un estándar para modelar, con tecnología orientada a objetos, a todos aquellos elementos que configuran la arquitectura de un sistema de información. Y, por extensión, de los procesos de negocio de una organización, proyectos importantes de empresas, etc. De la misma manera que los planos de un arquitecto disponen de un esquema director a partir del cual levantamos un edificio, los diagramas UML suministran un modelo de referencia para formalizar los procesos, reglas de negocio, objetos y componentes de una organización. La interacción de todos estos elementos es una representación de nuestra realidad.

Nuestros límites para entender esta realidad están en nuestro lenguaje. El mundo es la suma total de lo que podemos definir, organizar y visualizar. Enseñar a utilizar un lenguaje formal siempre es problemático. Es así, necesario *mostrar* como este lenguaje, puede ser aplicado a la realidad, tal como la conocemos, y la compartimos con los demás.

4.2 Lenguaje UML

UML es un lenguaje estándar para la escritura de proyectos de software. UML es utilizado para visualizar, especificar, construir y documentar los componentes de un sistema. Este lenguaje muy expresivo, que abarca todos los panoramas necesarios para desarrollar y estructurar los sistemas.

UML es un lenguaje para: Visualización, Especificación, Construcción y Documentación de los sistemas (Booch.et.al, 2001). En la tabla 1 se explica cada uno de estos conceptos.

Visualización	Un modelo explícito facilita la comunicación. UML es mas que un gran número de símbolos gráficos.
Especificación	Los modelos de construcción que son precisos, no ambiguos y completos.
Construcción	Es posible mapear de un modelo de UML a un lenguaje de programación tal como Java, C++, Visual Basic. Los proyectos que son mejor expresados gráficamente son hechos en UML, mientras que los que son mejor expresados textualmente se hacen programando.
Documentación	Una buena organización de software produce todos los tipos de componentes para el código ejecutable. Requerimientos Arquitectura Diseño Código fuente Planes de proyecto

Tabla 4.1 Descripción de los conceptos en UML

Para aprender UML se requiere aprender tres elementos principalmente:

i. Bloques de construcción
ii. Reglas de cómo estos bloques deben ser relacionados
iii. Mecanismos que aplica todo el tiempo el lenguaje.

En la siguiente tabla se describen los elementos.

Bloques de construcción	Es necesario formar un modelo conceptual del lenguaje. El vocabulario de UML comprende tres tipos de bloques de construcción: ➢ Elementos ➢ Relaciones ➢ Diagramas
Reglas	Un modelo bien formado esta en armonía con los modelos relacionados. Las reglas semánticas a seguir son las siguientes: ➢ Nombres ➢ Alcance ➢ Visibilidad ➢ Integridad ➢ Ejecución
Mecanismos	Es posible mapear de un modelo de UML a un lenguaje de programación tal como Java, C++, Visual Basic. Los proyectos que son mejor expresados gráficamente son hechos en UML, mientras que los que son mejor expresados textualmente se hacen programando.

Tabla 4.2 Elementos en UML

4.3 Estructura de UML

4.3.1 Casos de Uso

Los Caso de uso son representaciones de la interacción que tiene un actor y un caso de uso. Por lo tanto un caso de uso se representa por medio de un ovalo que representa una de las tareas que debe realizar el sistema.

Esta es la representación gráfica de los elementos en los Caso de uso (Rational Rose, 1998):

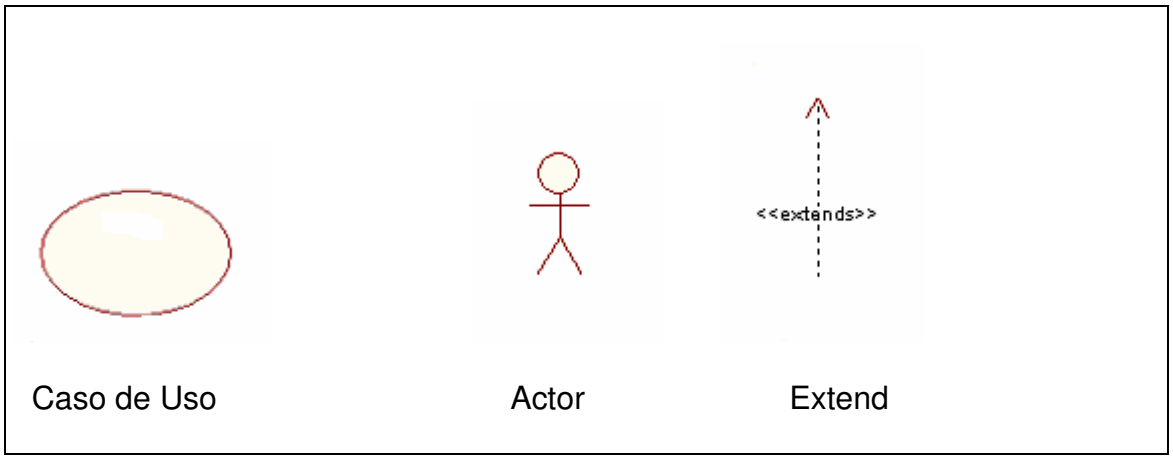


Fig. 4.3 Elementos de Casos de Uso

4.4 Clases

Las clases dentro de UML son las mas importantes, representan información ya procesada y analizada, de la cual se obtendrá el código para seguir desarrollando cualquier proyecto.

Una clase se compone de:

a) Atributos
b) Operaciones
c) Estereotipos
d) Asociaciones y Agregaciones entre clases
e) Multiplicidad
f) Clases abstractas

Una clase es la descripción de un grupo de objetos con estructura y comportamiento similar (Booch.et.al, 2001). El tiempo de ejecución provee una extensión, esto es, una instancia. En UML encontramos una notación especifica para declarar las clases y especificar sus propiedades. Una clase se identifica a través de un rectángulo, el nombre de la clase esta subrayado. En la Fig. 2 y Fig. 3 encontramos la representación física de una clase con sus respectivas divisiones.

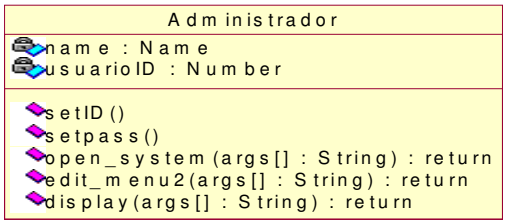


Fig. 4.4 Representación de una clase

Nombre de la clase y propiedades generales de la clase que incluye también el estereotipo.
Lista de atributos
Lista de operaciones

Fig. 4.5 Notación de una clase

a) Atributos

Los atributos se puede visualizar de tres formas;
(+) visibilidad publica
(#) protegida
(-) privada(OMG,1999)

Si no se marca algún tipo de visibilidad, significa que es publica o que simplemente esta indefinida. No es una regla que la clase deba llevar los compartimientos de abajo dentro de la clase, estos pueden ser suprimidos y no se hará referencia del error por no llevarlos. Si la clase es abstracta el nombre aparece en letras itálicas.

El propósito de una clase es la declaración de un conjunto de métodos, operaciones y atributos que describen la estructura y el comportamiento de los objetos (OMG,1999). Todos los objetos instanciados de una clase tendrán atributos.

Los atributos pueden o no llevar una descripción extra. Como por ejemplo, visibilidad, tipo y valores iniciales (Booch.et.al, 2001).

Nombre únicamente	origen
Nombre y visibilidad	+ origen
Nombre y tipo	origen : Punto
Nombre y un tipo complejo	head : *Item
Nombre visibilidad y tipo	name [0...1] : String
Nombre, tipo, valor inicial	origen : Punto = (0,0)
Nombre y propiedad	id: Entero { frozen }

b) Operaciones

Se puede especificar el parámetro, regreso de valores, concurrencia semántica y otras propiedades de cada operación (Booch et.al, 2000). Dentro del establecimiento de las clases en UML debemos señalar que es diferente una operación a un método.

C) Estereotipos

Dentro de las clases contamos con estereotipos que son usados para crear nuevos elementos. Representaran por lo tanto una subclase de una clase ya existente con los mismo atributo y relaciones pero con diferente intención (OMG,1999). Generalmente los estereotipos los representamos con un símbolo que es un circulo con una flecha indicando la dirección. Además lleva una palabra clave que es el nombre del estereotipo. Tenemos 5 tipos de estereotipos, van dentro de braquets angulares.

<< Boundary>>	Nos indica que es la representación con el exterior, típicamente son interfaces de control o pantallas.
<< Control>>	Nos indica que es el responsable en la secuencia de información.
<< Entity>>	Es una clase abstracta, como por ejemplo, productos ordenes de servicio.
<< Exception>>	Representa una excepción.
<< Interface>>	Especifica una operación externa visible.

Las siguientes figuras son la representación física de los estereotipos, como podemos observar Exception no esta representada, ya que no tiene una representación es solo un concepto.

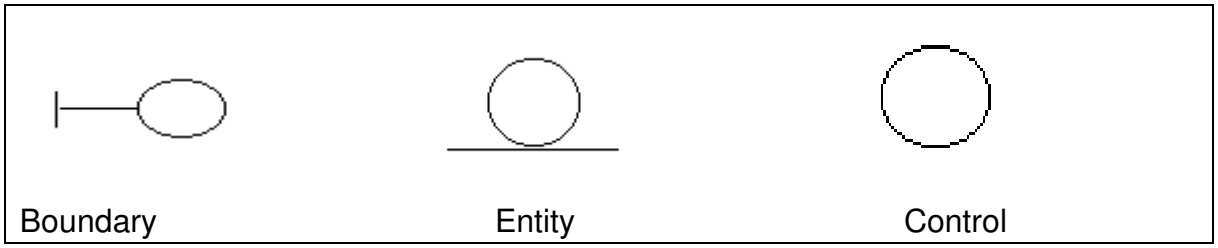


Fig. 4.6 Representación grafica de los estereotipos

e) Multiplicidad

Una vez establecida la asociación y la Agregación es necesario mostrar cuantos Objetos participan en la relación (Quatrini, 1999). Para Agregación y para Asociación es necesario que especifiquemos el número de instancias de una clase que se esta relacionando con la instancia de otra clase. Gráficamente veremos a la multiplicidad con números marcados en cada línea. De la siguiente forma:

1 Nos indica que exactamente uno
0...1 Que indica de cero a uno
1...* Uno o más objetos

En la siguiente figura se muestra la agregación.



Fig. 4.7 Diagrama de Agregación.

La siguiente figura muestra él numero de objetos que participan en esta relación es especifico. 0..4, muestra el rango de cero a cuatro cursos a impartir y el uno muestra un solo maestro para cada curso.

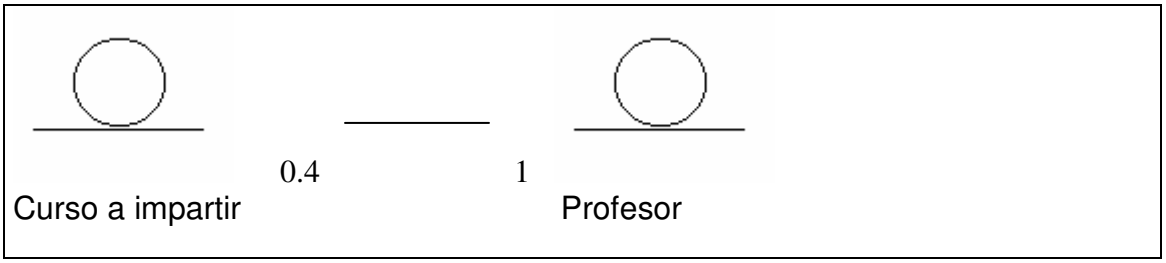


Fig. 4.8 Diagrama de multiplicidad

4.5 Diagramas básicos en UML

Un diagrama es una representación grafica de un conjunto de elementos (Booch.et.al, 2001).

Los diagramas hacen el sistema que se esta desarrollando mas comprensible y cercano a los objetos (Booch.et.al, 2001). Cada diagrama debe tener un nombre único con el contexto en que se encuentra y distinguirlo entre otros. No todas las veces se ocupan todos los diagramas estos dependerá del caso que se presente.

4.5.1 Diagrama de Casos de Uso

Comunicación entre el actor y sistema

Dentro de los diagramas Caso de Uso encontramos a los actores. Un Actor es la representación de algo o alguien que interactúa con el sistema (Quatrini, 2000).
Los modelos de los casos de uso se emplean para hacer un dialogo de comunicación entre el actor y el sistema, nos mostrara cuales son las capacidades del actor para poder actuar dentro del sistema. Por medio de los Casos de Uso podremos definir y manejar a nuestro sistema de una forma menos compleja y mejorando el rendimiento.

Dentro de los casos de Uso tenemos dos tipos de relación incluir (include) y extender (extend). Incluir es creado entre un nuevo caso de uso y el caso de Uso ya establecido, y además utiliza su funcionalidad. Extender es utilizado para un comportamiento opcional, comportamiento que corre bajo ciertas condiciones como trigger (para un movimiento o comportamiento posterior) (Booch.et.al, 2001).

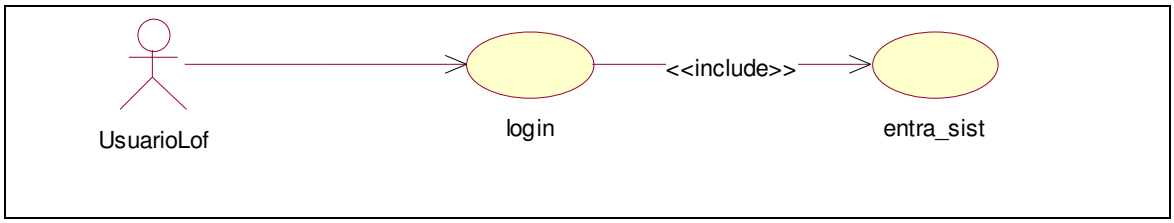


Fig. 4.9 Diagrama de Caso de Uso

4.5.2 Diagramas de Colaboración

Dentro de los Casos de Uso encontramos los Diagramas de " Realización de los Casos de Uso" estos diagramas se pueden representar gráficamente de dos formas, en Secuencia y en Colaboración. Los diagramas de Colaboración muestran la relación entre los objetos jugando diferentes papeles (OMG, 1999).

Es importante señalar que este tipo de diagrama no nos muestra el tiempo de separación entre la secuencia de las operaciones que se debe determinar y la secuencia de números.

4.5.3 Diagramas de Secuencia

En el diagrama de Secuencia se encuentran los actores, objetos, líneas punteadas verticalmente debajo de cada objeto y actores; las líneas punteadas que representa el tiempo el cual indica los eventos requeridos entre los objetos.

Los diagramas de secuencia nos ayudan a formar el tiempo de orden de los mensajes. La diferencia entre los diagramas de secuencia y los diagramas de colaboración es que en estos lo importante es la estructura de los objetos, como están organizados para recibir y enviar mensajes. Gráficamente un diagrama de colaboración es una colección de vértices y arcos (Booch et.al, 2001)

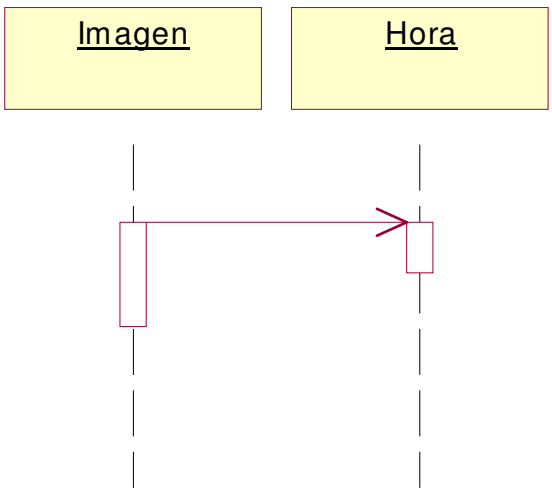


Fig. 4.10 Representación de un Diagrama de Secuencia

Para formar un diagrama de Secuencia, primero deberemos colocar los objetos que van a participar, seguido por los objetos que iniciaran las actividades o las interacciones y por último los mensajes entre estos. Los mensajes nos indicaran si se recibe o si se esta mandando un mensaje, gráficamente son flechas verticales que se dirigen a las líneas horizontales o también llamadas líneas de tiempo de vida. Los mensajes los colocamos de acuerdo al tiempo que se lleva, por ejemplo contaremos con 4 objetos, si se solicita el préstamo de un libro, el orden de colocación es el siguiente, el primer objeto será la persona que pide el libro prestado seguido por la persona de la biblioteca que prestara el libro, el tramite de préstamo y por ultimo el libro ya registrado como préstamo. Con tres mensajes, casi siempre los mensajes tiene dirección de izquierda a derecha. "Las líneas de los mensajes representan el tiempo visto por los objetos: el tiempo de vida de los objetos"(Booch et.al, 2001). Para ver la ejecución del tiempo, lo haremos de arriba hacia abajo en el diagrama. Dentro de los mensajes es valido que ellos mismos se manden mensajes.

Podemos concluir que la principal ventaja de realizar el diagrama de secuencia es que podemos representar el tiempo gráficamente . "UML nos permite representar las constantes de tiempo". Como ejemplo de representación entre dos mensajes que no deben tardar mas de 10 mircosegundos en realizar una operación, podemos representar la restricción de tiempo como {X-Y < 10 microsec} (Booch et.al, 2001).

Algunos de los mensajes se deberán mandar varias veces, mostrarlos cada vez que se requiera no es conveniente, principalmente porque se haría una cadena por lo tanto es imposible mandar mensajes de un diagrama a otro. "UML permite mostrar con un asterisco cuando un mensaje se necesita mandar muchas veces"(Stevens et.al, 2000)

Podemos señalar esto como en los siguientes ejemplos:
[x := 1...5] El menaje se mandara 5 veces
[x < 5] El mensaje se mandara varias veces, hasta que x tome el valor de 5
[item no se necontro] El mensaje se mandar varias veces hasta que el item sea encontrado. Debe de haber siempre una relación entre los casos de uso y los diagramas tanto de secuencia, colaboración y los de clase.

4.5.4 Diagramas de Actividades

Los diagramas de actividad representan la parte dinámica del sistema, que son diagramas de flujo, muestran el flujo de control de una actividad a otra, además podemos ver que actividades pueden realizarse al mismo tiempo.

Los diagramas de actividad contienen, actividades, transacciones entre actividades, puntos de decisión y barras de sincronización (Quatrini, 1999)

- Transición.- Son usadas para pasar el flujo de control entre una actividad y otra.
- sincronización.- Las barras de sincronización son condicionales los cuales muestra un punto de decisión.

4.5.5 Diagramas de estados

Muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro (Booch et.al, 2001). Dentro de los Diagramas de estados contamos con:

i. Estados
ii. Eventos
iii. Envio de mensajes

i. Estado

Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto esta esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos. Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado.

Se marcan también los estados iniciales y finales mediante los símbolos

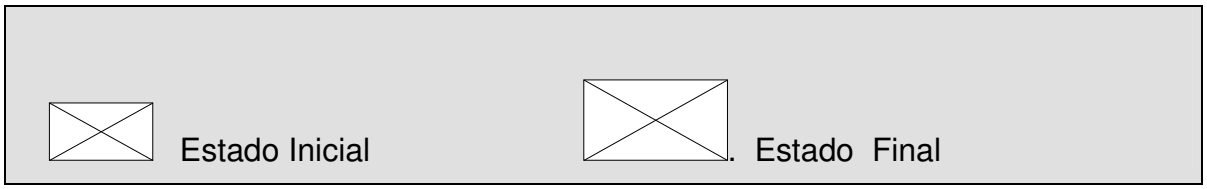


Fig. 4.11 Diagrama de estados

ii. Eventos

Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser por:

- Condición que toma el valor de verdadero o falso
- Recepción de una señal de otro objeto en el modelo
- Recepción de un mensaje
- Paso de cierto período de tiempo, después de entrar al estado o de cierta hora y fecha particular .

iii. Envío de mensajes

En envió de mensajes se ocupan dos términos: estados y eventos. Mostrando la transición de estados por medio de eventos, puede representarse el momento en en el cual se envían mensajes .

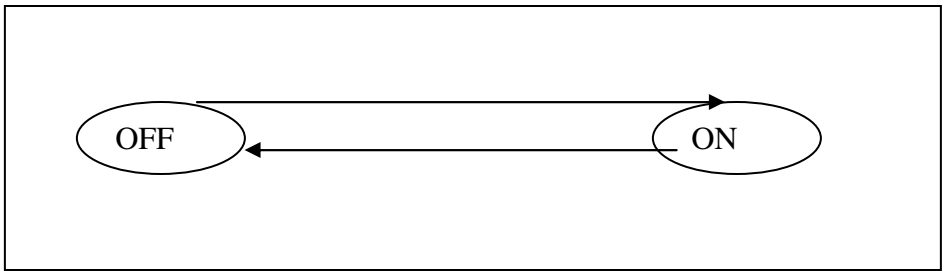


Fig. 4.12 Ejemplo de Diagrama de Estados

4.5.6 Componentes

Desde el punto de vista físico tenemos los diagramas de componentes que contiene una unidad de código que sirve como un bloque de construcción. Contiene lo que se desarrollo el paquete lógico. Los subprogramas contiene subrutinas o una sola subrutina. Las tareas contienen paquetes con fuentes de control independientes, si una clase se tiene que compilar de manera diferente entonces colocamos esta clase dentro de una tarea.

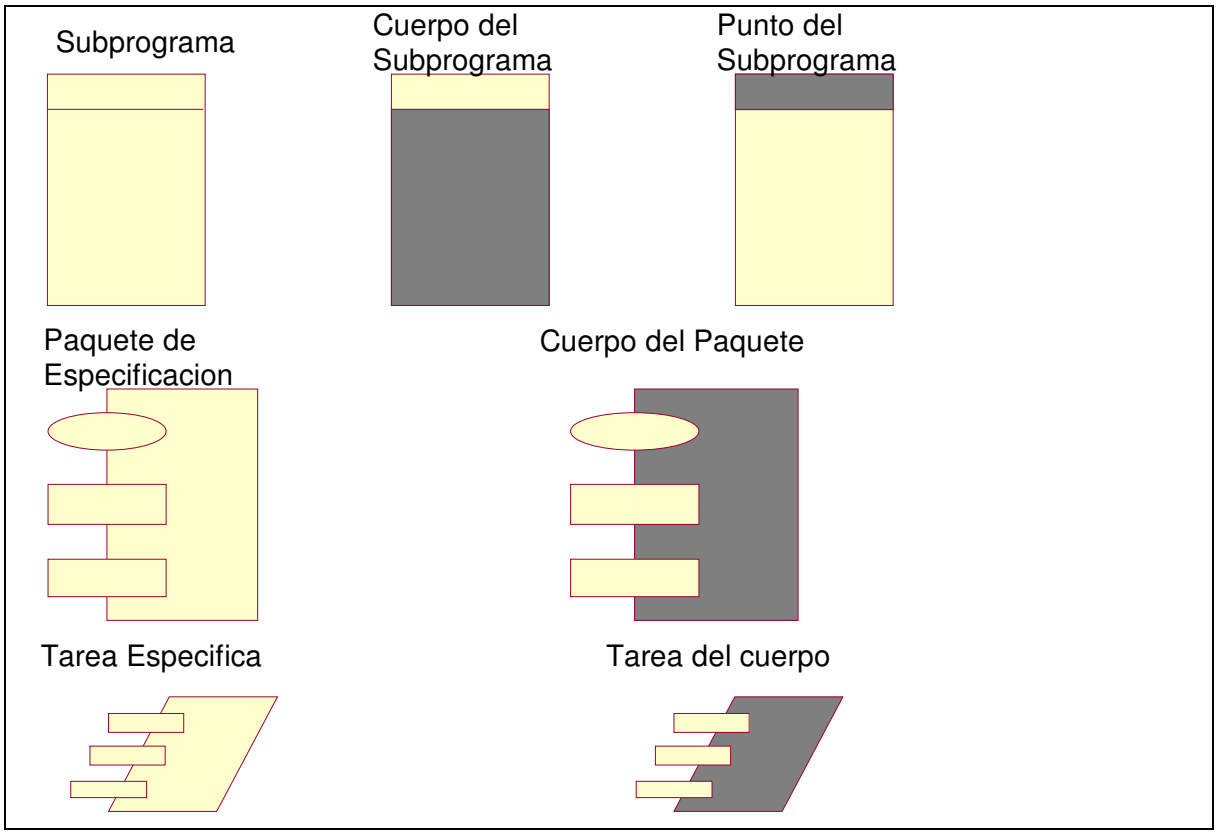


Fig.4.13 Diagrama de Componentes

4.5.7 Lenguajes utilizados en UML

Estos son los lenguajes en los que UML puede desarrollarse:

V. Custum

C#
C++
C++ Builder
Delphi
Java
Visual Basic

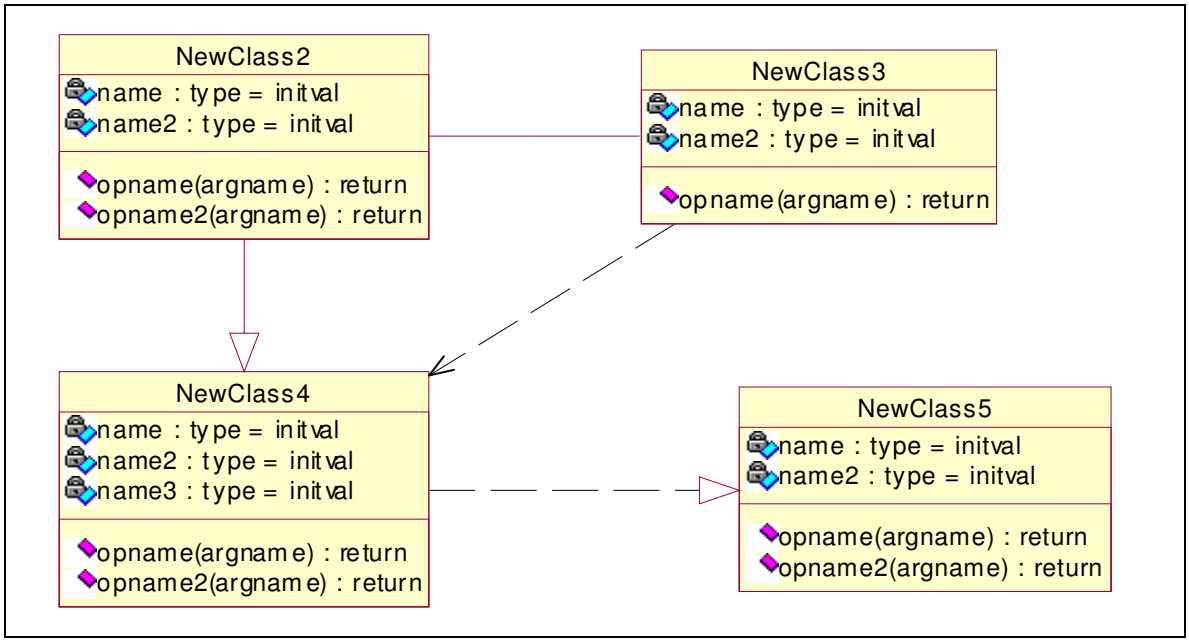


Fig. 4.14 Tabla Diagrama de Clases

ANEXOS B

A. Diagrama Caso de Uso

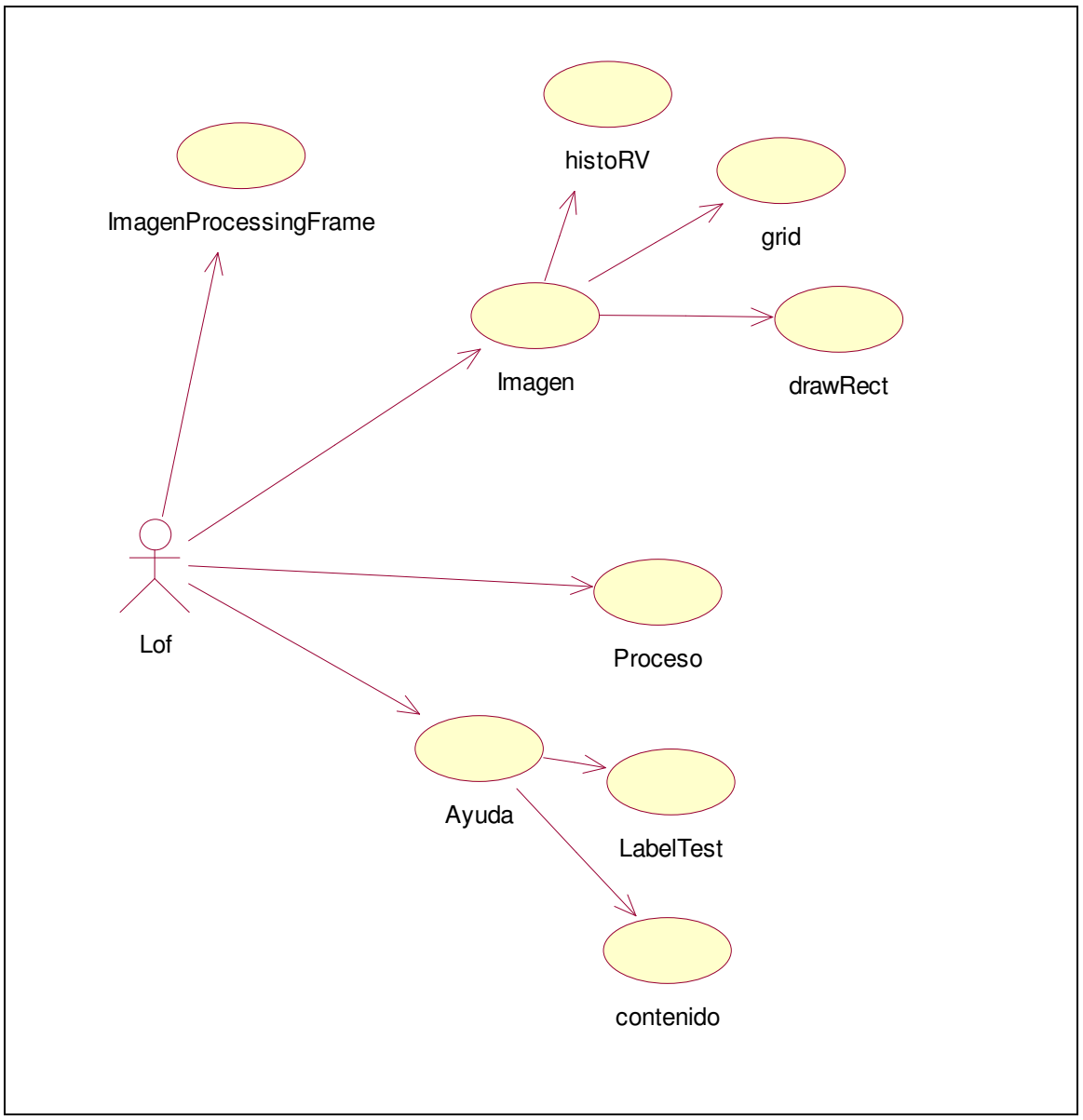


Fig. 4.15 Diagrama Caso de Usos

B. Diagrama se Secuencia

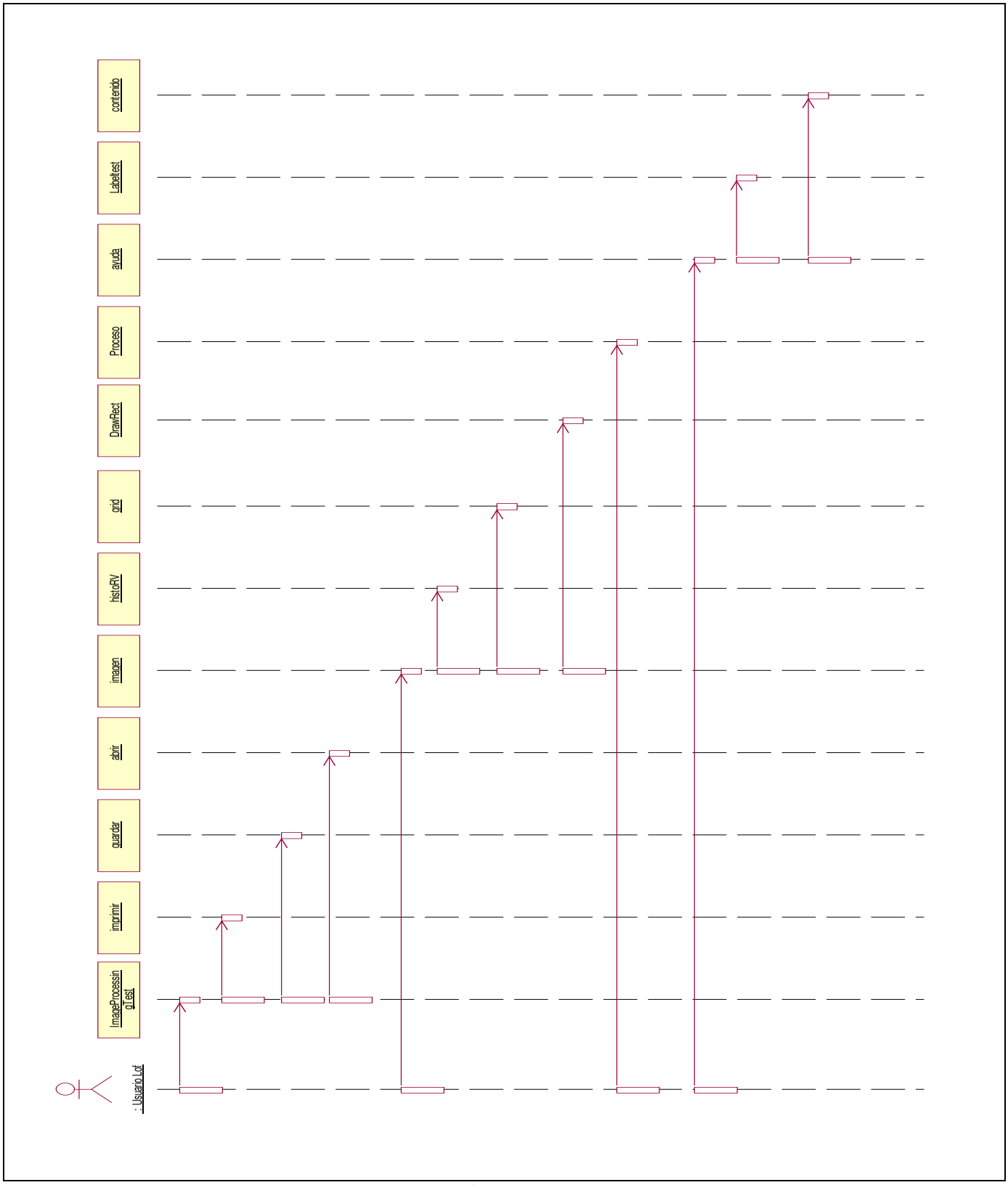


Fig. 4.16 Diagrama Caso de Usos

C. Diagrama de Colaboración

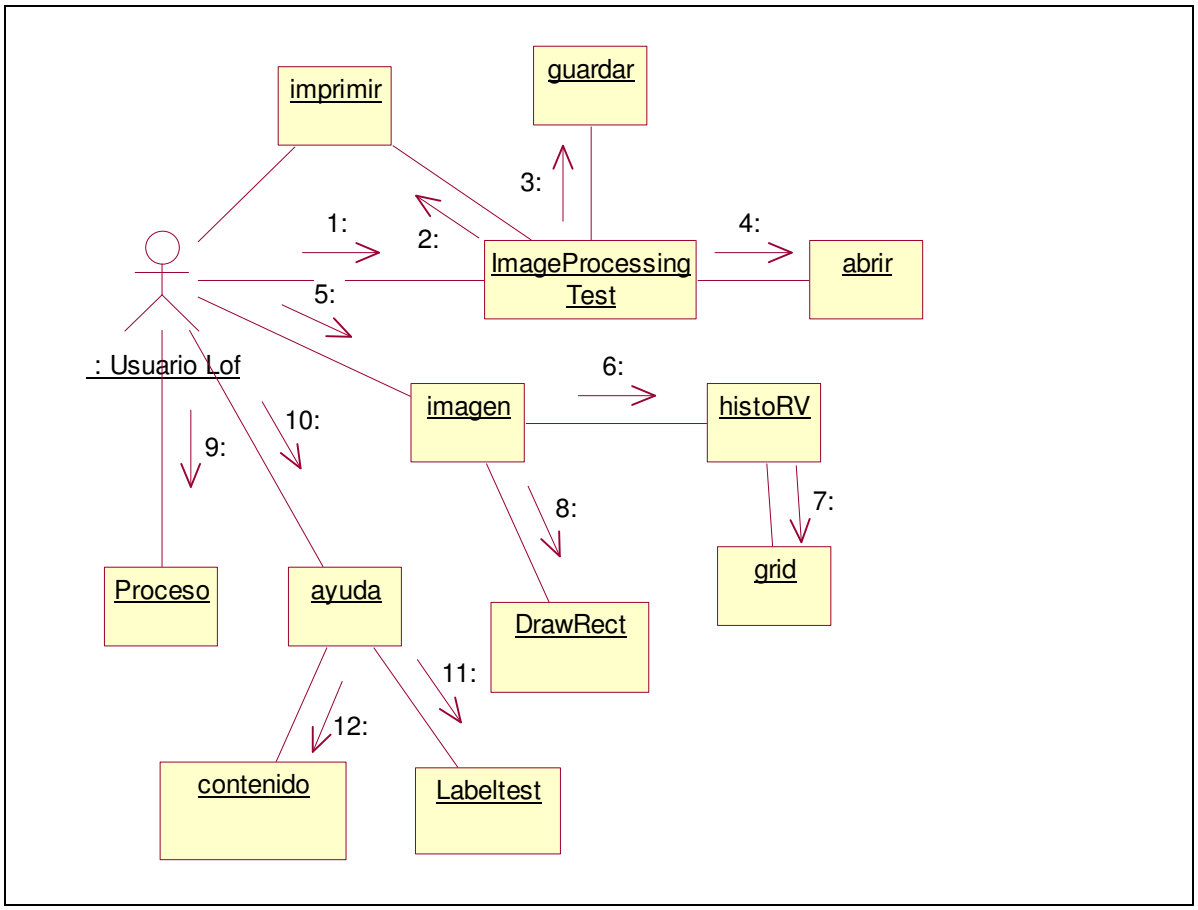


Fig. 4.17 Diagrama Caso de Usos

D. Diagrama de Clases

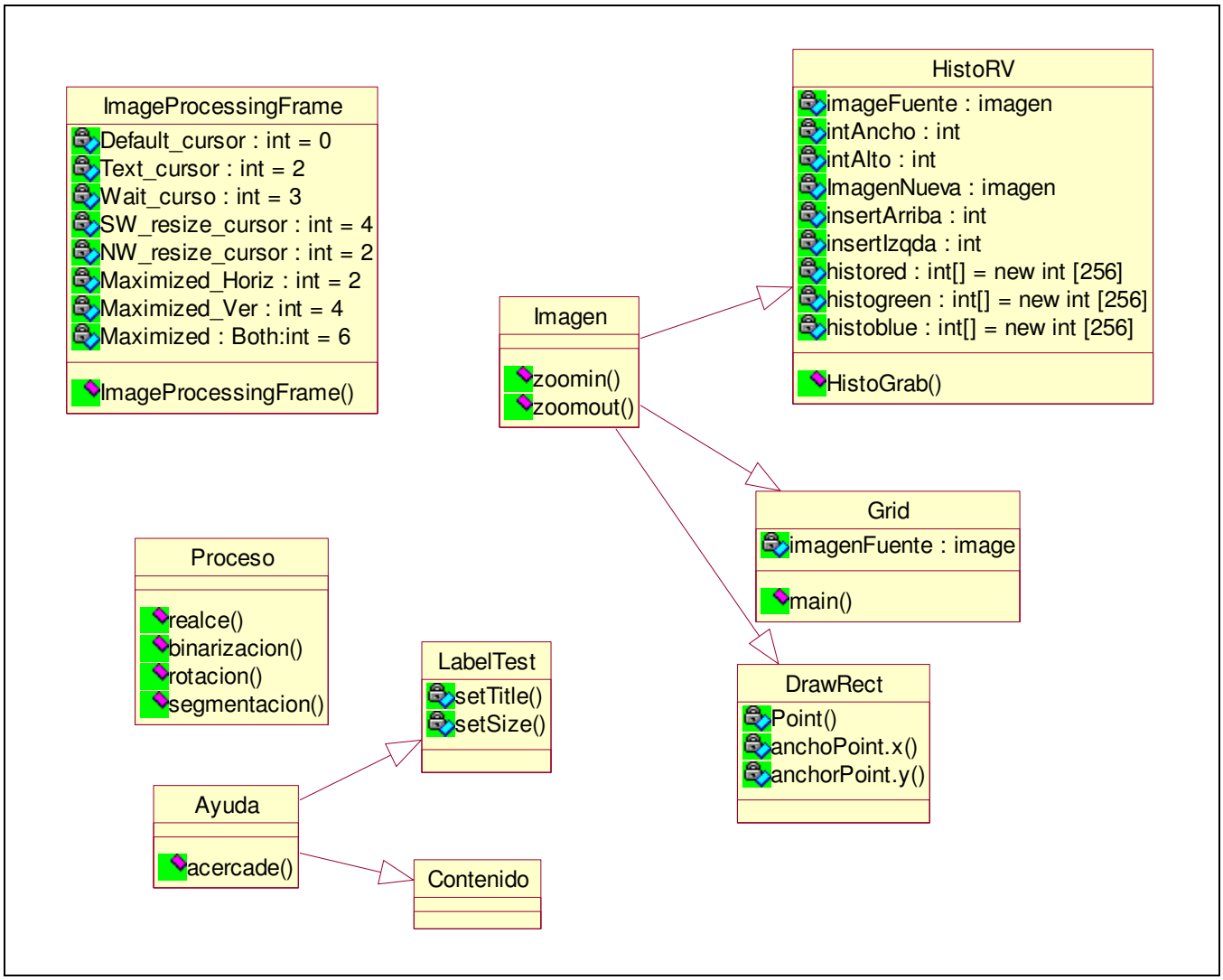


Fig. 4.18 Diagrama Caso de Usos

E. Componentes para generar código

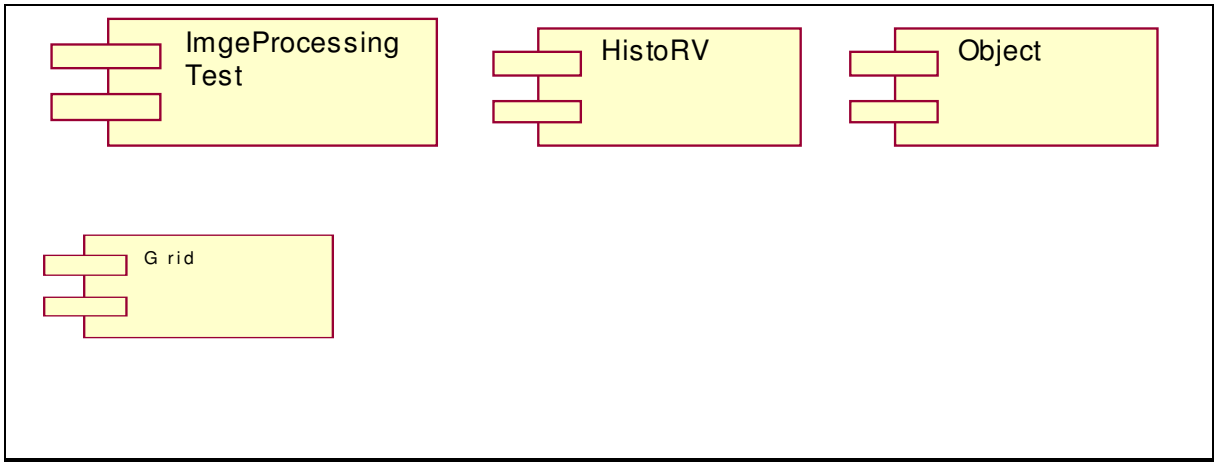


Fig. 4.19 Diagrama Caso de Usos

ANEXO C

En el Anexo B, se mostraran las pantallas resultado de la aplicación en Java. Como anteriormente se indico la aplicación cuenta con cuatro menús que a su vez tienen submenús en donde se podrán realizar las operaciones indicadas a las imágenes.

En el menú de inicio contamos con cuatro submenús, Open, Guardar, Imprimir y Exit.

Menu Inicio

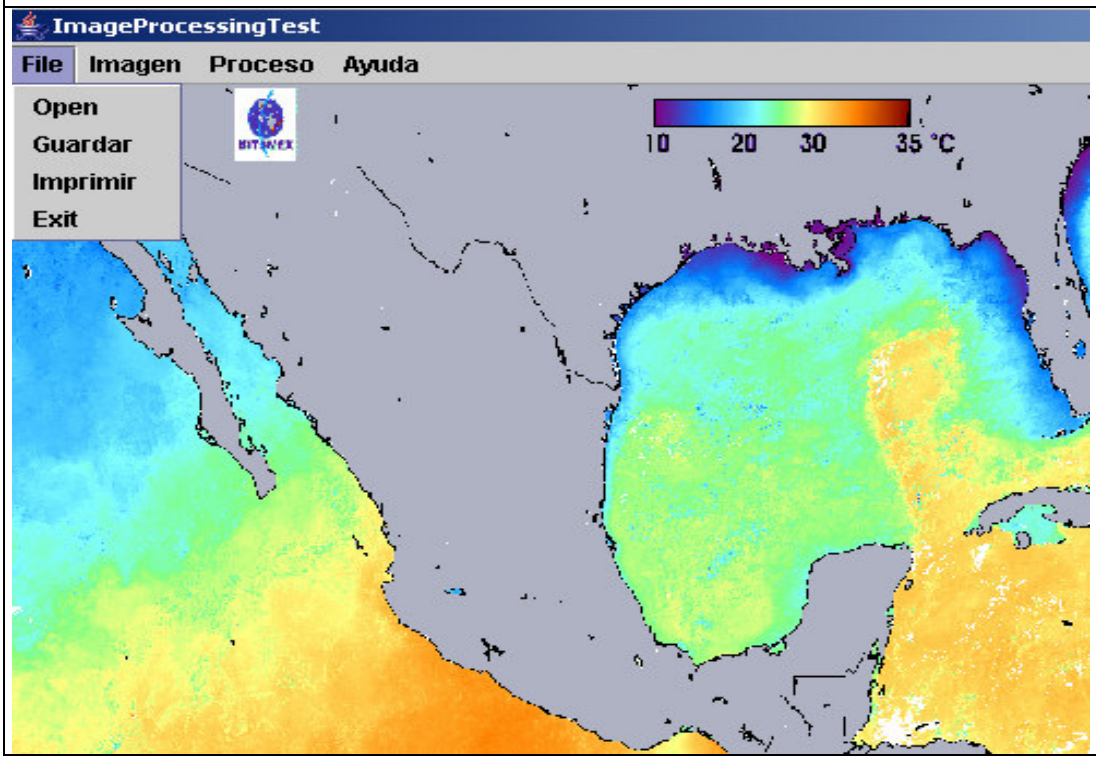


Fig. 4.18 Menú Inicio

La siguiente figura muestra muestra el submenú Open, el cual nos permite ver que podemos seleccionar cualquier carpeta.

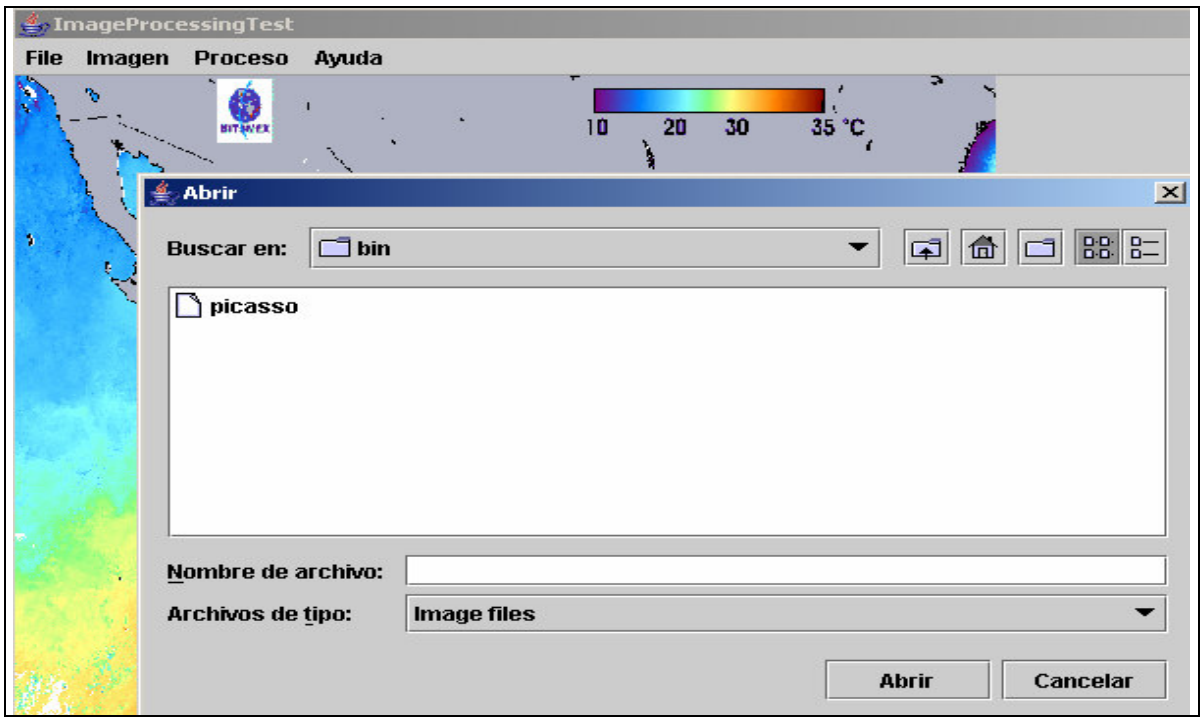


Fig. 4.19 Submenú Open

Menú Imagen cuenta con cinco submenús los cuales se muestran desplegados en la siguiente figura: Recorte, Histo, Zoomin, Zoomout y Grid.

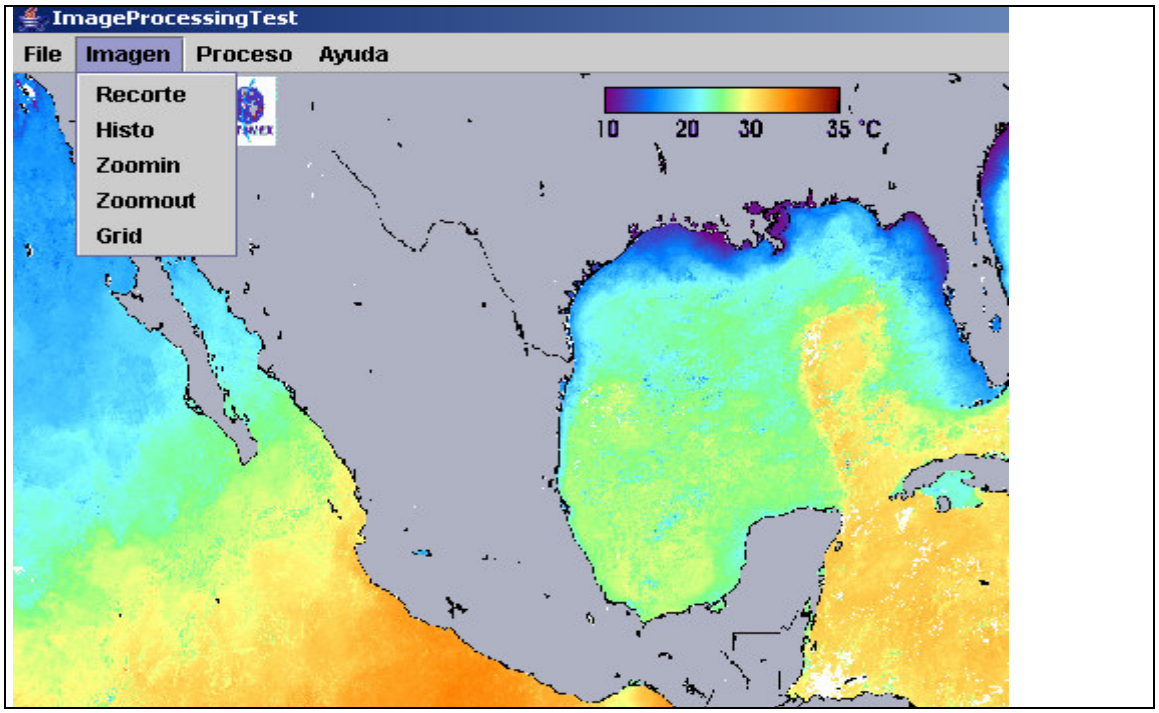


Fig. 4.20 Menú Imagen

Como tercer menú mostramos a Proceso en el cual se despliegan cuatro submenús: Ralce, Binarizacion, Segmentación y Rotación.

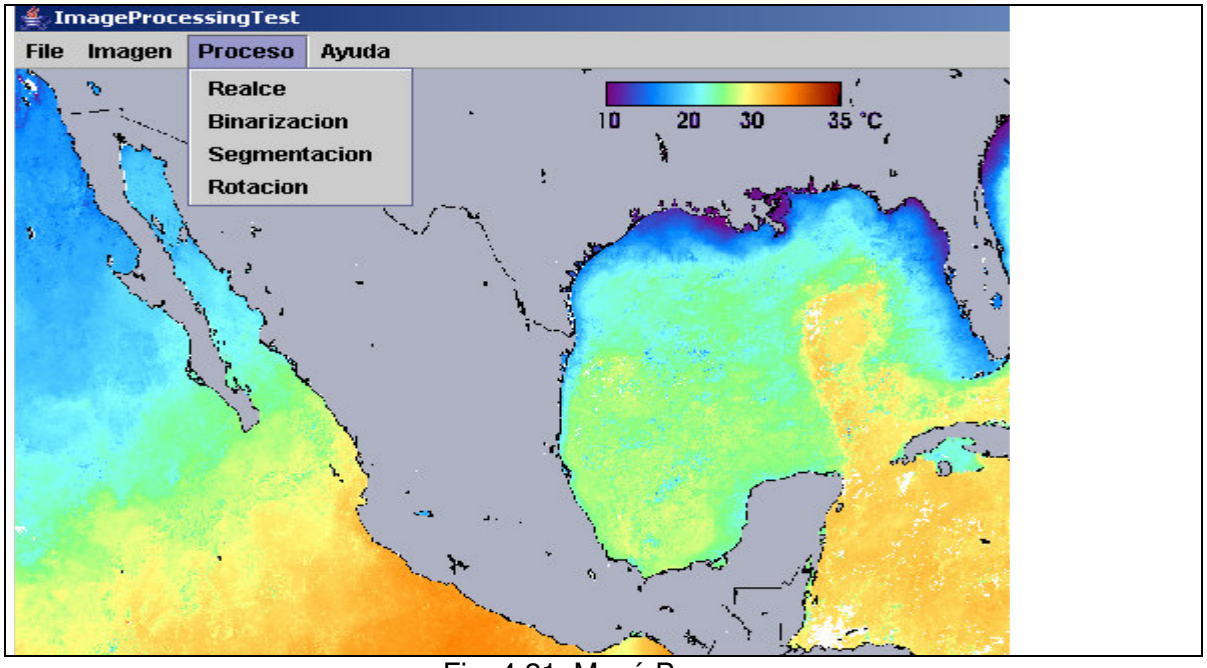


Fig. 4.21 Menú Proceso

Por ultimo tenemos el Menú Ayuda en el cual se presenta los submenús: Acerca de y Contenido.

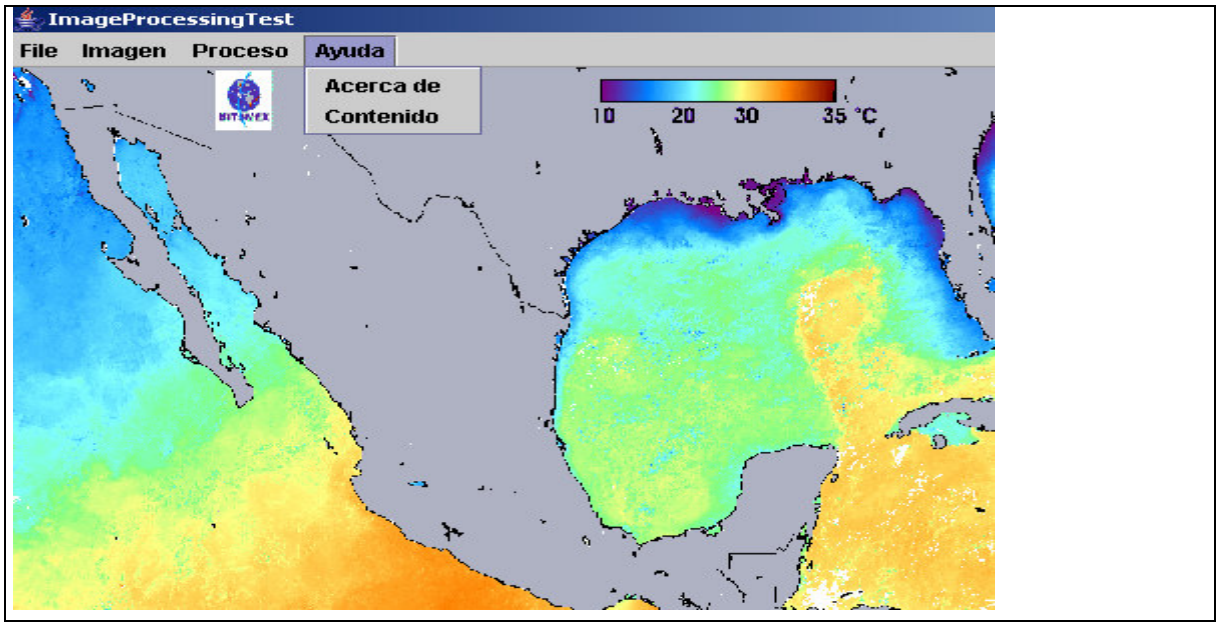


Fig. 4.22 Menú Ayuda

ANEXO D

De acuerdo a la los resultados obtenidos, por la programación en JAVA, se presenta el programa completo.

```
class ImageProcessingFrame extends JFrame
  implements ActionListener
 {
        public ImageProcessingFrame()
         {
          setTitle("ImageProcessingTest");
          setSize(900, 700);
          addWindowListener(new WindowAdapter()
                  {
                   public void windowClosing(WindowEvent e)
                          {
                            System.exit(0);
                          }
                  } );

          Container contentPane = getContentPane();
          panel = new ImageProcessingPanel();
          contentPane.add(panel, "Center");

          JMenu fileMenu = new JMenu("File");
          openItem = new JMenuItem("Open");
          openItem.addActionListener(this);
          fileMenu.add(openItem);

          guardarItem = new JMenuItem("Guardar");
          guardarItem.addActionListener(this);
          fileMenu.add(guardarItem);

          imprimirItem = new JMenuItem("Imprimir");
          imprimirItem.addActionListener(this);
          fileMenu.add(imprimirItem);

          exitItem = new JMenuItem("Exit");
          exitItem.addActionListener(this);
          fileMenu.add(exitItem);
```

```
JMenu imagenMenu = new JMenu("Imagen");

recorteItem = new JMenuItem("Recorte");
recorteItem.addActionListener(this);
imagenMenu.add(recorteItem);


histoItem = new JMenuItem("Histo");
histoItem.addActionListener(this);
imagenMenu.add(histoItem);

zoomItem = new JMenuItem("Zoomin");
zoomItem.addActionListener(this);
imagenMenu.add(zoomItem);

zoom1Item = new JMenuItem("Zoomout");
zoom1Item.addActionListener(this);
imagenMenu.add(zoom1Item);

gridItem = new JMenuItem("Grid");
gridItem.addActionListener(this);
imagenMenu.add(gridItem);


JMenu procesoMenu = new JMenu("Proceso");

realceItem = new JMenuItem("Realce");
realceItem.addActionListener(this);
procesoMenu.add(realceItem);

binarizacionItem = new JMenuItem("Binarizacion");
binarizacionItem.addActionListener(this);
procesoMenu.add(binarizacionItem);


segmentacionItem = new JMenuItem("Segmentacion");
segmentacionItem.addActionListener(this);
procesoMenu.add(segmentacionItem);

rotacionItem = new JMenuItem("Rotacion");
rotacionItem.addActionListener(this);
procesoMenu.add(rotacionItem);


JMenu ayudaMenu = new JMenu("Ayuda");

acercaItem = new JMenuItem("Acerca de");
acercaItem.addActionListener(this);
ayudaMenu.add(acercaItem);
```

```
        contenidoItem = new JMenuItem("Contenido");
        contenidoItem.addActionListener(this);
        ayudaMenu.add(contenidoItem);

        JMenuBar menuBar = new JMenuBar();


        menuBar.add(fileMenu);
        //menuBar.add(editMenu);
        menuBar.add(imagenMenu);
        menuBar.add(procesoMenu);
        //menuBar.add(refgeoMenu);
        menuBar.add(ayudaMenu);

       setJMenuBar(menuBar);
}

public void actionPerformed(ActionEvent evt)
      {
        Object source = evt.getSource();
        if (source == openItem)
        {
              JFileChooser chooser = new JFileChooser();
               chooser.setCurrentDirectory(new File("."));

               chooser.setFileFilter(new
                     javax.swing.filechooser.FileFilter()
                     {
                       public boolean accept(File f)
                        {
                              String name = f.getName().toLowerCase();
                               return name.endsWith(".jpg")
                                     || name.endsWith(".jpeg")
                                     || f.isDirectory();
                        }
                             public String getDescription()
                        {
                              return "Image files";
                        }
                     });
```

```
        int r = chooser.showOpenDialog(this);
        if(r == JFileChooser.APPROVE_OPTION)
        {
                 String name = chooser.getSelectedFile().getAbsolutePath();
                panel.loadImage(name);


        }
}
```

/** **para Guardar** */

```
if (source ==guardarItem)
{
       JFileChooser chooser = new JFileChooser();
        chooser.setCurrentDirectory(new File("."));


        chooser.setFileFilter(new
                javax.swing.filechooser.FileFilter()
                {
                  public boolean accept(File f)
                   {
                          String name = f.getName().toLowerCase();
                           return

                                   name.endsWith(".jpg")
                                || name.endsWith(".jpeg")
                                || f.isDirectory();
                   }
                        public String getDescription()
                   {
                           return "Image files";
                   }
                });

        int r = chooser.showSaveDialog(this);
        if(r == JFileChooser.APPROVE_OPTION)
        {
```

```
                String name = chooser.getSelectedFile().getAbsolutePath();
                File file = new File(name);
                panel.SaveImage(panel.getImage(),file);


            }
        }

        / ** Menu Inicio */
else if (source == recorteItem) panel.cut();
        else if (source == guardarItem) panel.guardar();
        else if (source == imprimirItem) panel.imprim();
        else if (source == exitItem) System.exit(0);

        / ** Menu Edit */

        else if (source == blurItem) panel.blur();
        else if (source == sharpenItem) panel.sharpen();
        else if (source == brightenItem) panel.brighten();
        else if (source == edgeDetectItem) panel.edgeDetect();
        else if (source == negativeItem) panel.negative();

        /** Menu Imagen */

        else if (source == recorteItem) panel.rotate();
        else if (source == histoItem)    panel.histo();
        else if (source == zoomItem) panel.zoomin();
        else if (source == zoom1Item) panel.zoomout();
        else if (source == gridItem) panel.grid();


        / ** Menu Proceso */

        else if (source == realceItem) panel.brighten();
        else if (source == binarizacionItem) panel.negative();
        else if (source == geoItem) panel.rotate();
        else if (source == segmentacionItem) panel.edgeDetect();
        else if (source == rotacionItem) panel.rotate();
```

```
            / ** Menu  Ayuda */

            else if (source == acercaItem) panel.ayuda();
            else if (source == contenidoItem) panel.Conten();

    }

    private ImageProcessingPanel panel;
    private JMenuItem openItem;
    private JMenuItem guardarItem;
    private JMenuItem imprimirItem;
    private JMenuItem exitItem;

    private JMenuItem blurItem;
    private JMenuItem sharpenItem;
    private JMenuItem brightenItem;
    private JMenuItem edgeDetectItem;
    private JMenuItem negativeItem;
    private JMenuItem rotateItem;

    private JMenuItem recorteItem;
    private JMenuItem histoItem;
    private JMenuItem zoomItem;
    private JMenuItem zoom1Item;
    private JMenuItem gridItem;

    private JMenuItem realceItem;
    private JMenuItem binarizacionItem;
    private JMenuItem geoItem;



    private JMenuItem segmentacionItem;
    private JMenuItem rotacionItem;

    private JMenuItem acercaItem;
    private JMenuItem contenidoItem;

}

class ImageProcessingPanel extends JPanel implements Printable
  {
        String pathImagen;
        Image imagenFuente;

        Point anchorPoint = new Point(0,0);

     DragRect dragRect = new DragRect();
```

```
        Int      RADIUS = 500;
        int      CENTER_X = 0;
        int      CENTER_Y = 0;


        public void paintComponent(Graphics g)
        {
          super.paintComponent(g);
                if (image != null)
                  g.drawImage(image,CENTER_X,CENTER_Y,RADIUS,RADIUS,
null);
        }

  public void loadImage(String name)
  {
          pathImagen = name;
          Image loadedImage = Toolkit.getDefaultToolkit().getImage(name);
          imagenFuente = loadedImage;
          MediaTracker tracker = new MediaTracker(this);
          tracker.addImage(loadedImage, 0);
          try {
                   tracker.waitForID(0);
                   }
          catch (InterruptedException e)
          {}
          image = new BufferedImage(loadedImage.getWidth(null),
          loadedImage.getHeight(null), BufferedImage.TYPE_INT_RGB);
          Graphics2D g2 = image.createGraphics();




          g2.drawImage(loadedImage, 0, 0, null);
          repaint();
  }

  private void filter(BufferedImageOp op)
  {
        BufferedImage                filteredImage                =                new
BufferedImage(image.getWidth(),image.getHeight(),image.getType());
          op.filter(image, filteredImage);
          image = filteredImage;
          repaint();
  }
```

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

public void guardar()
{
         }

         public void recorte()
                 {
                  }



public void blur()
{
        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 sharpen()
{
         float[] elements =
                {
                         0.0f, -1.0f, 0.0f,
                        -1.0f,  5.f, -1.0f,
                        0.0f, -1.0f, 0.0f
                };




        convolve(elements);
}
```

```
  void edgeDetect()
  {
        float[] elements =
                {
                        0.0f, -1.0f, 0.0f, -1.0f,  4.f, -1.0f, 0.0f, -1.0f, 0.0f
                };

         convolve(elements);
  }

  public void brighten()
  {
         float a = 1.5f;
         float b = -20.0f;
         RescaleOp op = new RescaleOp(a, b, null);
         filter(op);
  }

/ ** para llamar la clase que genera el histograma */
  public void histo()
         {
                HistoRV Histograma = new HistoRV(imagenFuente);

         }




/ ** para llamar la clase que genera el grid */
  public void grid()
         {

               IHM grid = new IHM(imagenFuente);
                 System.out.print("***********");


         }



/** llama la clase que genera la AYUDA (Acerca de) */
  public void ayuda()
         {
                 LabelTest ihm = new LabelTest();

         }
```

```
/ ** para llamar la clase que genera la AYUDA (Contenido) */
   public void Conten()
           {
              Contenido con = new Contenido();

           }

/ ** para el cut  */
  public void cut()
           {
                   LabelTest ihm = new LabelTest();

           }



/ ** para llamar la clase que genera IMPRIMIR */
  public void imprim()
           {
                 PrinterJob pj = PrinterJob.getPrinterJob();
                 pj.setPrintable(ImageProcessingPanel.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);
                   }
                   }
                 }



/ ** para llamar la clase que genera Zoom */
   public void zoomin()
           {

                 RADIUS += 25;
                 CENTER_X -= 12;
                 CENTER_Y -= 12;
                 repaint();

           }
```

```
/ **para llamar la clase que genera Zoom */
   public void zoomout()
           {
                   RADIUS -= 25;
                   CENTER_X += 12;
                   CENTER_Y += 12;
                   repaint();

           }

// *************************

  void negative()
  {
          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);
  }

  void rotate()
  {
          AffineTransform transform =
          AffineTransform.getRotateInstance(Math.toRadians(5),
           image.getWidth() / 2, image.getHeight() / 2);
          AffineTransformOp op = new AffineTransformOp(transform,
                  AffineTransformOp.TYPE_BILINEAR);
          filter(op);
  }


  /** metodo que imprime */

  public int print(Graphics g, PageFormat pf, int pageIndex) throws
java.awt.print.PrinterException {

        System.out.println("hasta aqui llego 3");
        if (pageIndex != 0) return NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D)g;
        g2.translate(pf.getImageableX(), pf.getImageableY());
        paint(g2);
        return PAGE_EXISTS;
  }
```

```
  /** método para guardar */

  public void SaveImage(Image origen, File name)
        {
                BufferedImage buffImage;
                JPEGImageEncoder encoder;
                float quality = 1.0f;
                try {
                buffImage =
                                        new BufferedImage(
                                                origen.getWidth(null),
                                                origen.getHeight(null),
                                                BufferedImage.TYPE_INT_RGB);

                AffineTransform tx = new AffineTransform();
                Graphics2D g2d = buffImage.createGraphics();
                                        g2d.drawImage(origen, tx, null);
                                        g2d.dispose();
                                OutputStream os;

/***** codificar la imagen en formato JPG y escribirla en un fichero****/

                                        os = new FileOutputStream(name);
                                        encoder = JPEGCodec.createJPEGEncoder(os);
                                        JPEGEncodeParam                 jep                 =
encoder.getDefaultJPEGEncodeParam(buffImage);
                                        jep.setQuality(quality, true);
                                        encoder.setJPEGEncodeParam(jep);
                                        encoder.encode(buffImage);
                                        os.close();

           }
       catch (IOException e) {
            }
           }

       public void mouseClicked(java.awt.event.MouseEvent mouseEvent) {
       }

       public void mouseDragged(java.awt.event.MouseEvent mouseEvent) {

            dragRect.setBounds(anchorPoint.x, anchorPoint.y,

                mouseEvent.getX() - anchorPoint.x,
```

```
            mouseEvent.getY() - anchorPoint.y );
        System.out.println(" ancho "+ mouseEvent.getX());
        System.out.println(" alto "+ mouseEvent.getY());
        dragRect.normalize();

        repaint();
    }

    public void mouseEntered(java.awt.event.MouseEvent mouseEvent) {
    }

    public void mouseExited(java.awt.event.MouseEvent mouseEvent) {
    }

    public void mouseMoved(java.awt.event.MouseEvent mouseEvent) {
    }

    public void mousePressed(java.awt.event.MouseEvent mouseEvent) {
        anchorPoint.x = mouseEvent.getX();
        anchorPoint.y = mouseEvent.getY();
        System.out.println(" X = "+ anchorPoint.x);
        System.out.println(" Y = "+ anchorPoint.y);
    }

    public void mouseReleased(java.awt.event.MouseEvent mouseEvent) {
    }


    private BufferedImage image;


/** return */

public BufferedImage getImage() {
        return image;
}

/** param image */

public void setImage(BufferedImage image) {
        this.image = image;
}

}
```

```
class DragRect extends Rectangle {
    void normalize() {
        if( width < 0 ) {
            x += width;
            width = -width;
        }
        if( height < 0 ) {
            y += height;
            height = -height;

        }
    }
}
```

HistoRV.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;

class HistoRV extends Frame {

  Image imagenFuente;                // Imagen cargada del disco
  int iniAncho;
  int iniAlto;
  Image imagenNueva;
  int insetArriba;
  int insetIzqda;
  int[] histored = new int[256];
  int[] histogreen = new int[256];
  int[] histoblue = new int[256];
  int max_histr = 0;
  int max_histg=  0;
  int max_histb = 0;


  public void main( String[] args ) {
    HistoRV obj = new HistoRV(imagenFuente);
    obj.repaint();

    }

  public HistoRV(Image imagen) {
     imagenFuente = imagen;

   // imagenFuente = Toolkit.getDefaultToolkit().getImage(pathImagen );
    MediaTracker tracker = new MediaTracker( this );
    tracker.addImage( imagenFuente,1 );


    try {
      if( !tracker.waitForID( 1,10000 ) ) {
        System.out.println( "Error en la carga de la imagen" );
        System.exit( 1 );
      }
    } catch( InterruptedException e ) {
      System.out.println( e );
      }
```

```
    iniAncho = imagenFuente.getWidth( this );
    iniAlto = imagenFuente.getHeight( this );

    this.setVisible( true );


    insetArriba = this.getInsets().top;
    insetIzqda = this.getInsets().left;

    this.setSize( insetIzqda+iniAncho+700,insetArriba+(2*iniAlto+300));
    this.setTitle( "ICMyL  Histograma " );
    this.setBackground( Color.lightGray );

    int[] pix = new int[iniAncho * iniAlto];
    try {
      PixelGrabber pgObj = new
PixelGrabber(imagenFuente,0,0,iniAncho,iniAlto,pix,0,iniAncho );
      if( pgObj.grabPixels() &&( (pgObj.getStatus() & ImageObserver.ALLBITS ) != 0
) ) {
       int c, r, g, b, max, hi;
        for( int i=0; i < (iniAncho*iniAlto); i++ ) {
              c = pix[i];
              r = (c&0xff0000)>>16;
              g = (c&0xff00)>>8;
              b = c&0xff;
              histored[r]++;
              histogreen[g]++;
              histoblue[b]++;



            }

          for (int i = 0; i < 256; i++)
                        {
                                    if (histored[i] > max_histr)
                                                max_histr = histored[i];
                                    if (histogreen[i] > max_histg)
                                                max_histg = histogreen[i];
                                    if (histoblue[i] > max_histb)
                                          max_histb = histoblue[i];
                        }
      }
```

```
      else {
        System.out.println( "Problemas al descomponer la imagen" );
        }
    } catch( InterruptedException e ) {
      System.out.println( e );
      }

    imagenNueva = this.createImage( new
MemoryImageSource(iniAncho,iniAlto,pix,0,iniAncho ) );

    this.addWindowListener(

      new WindowAdapter() {
        public void windowClosing( WindowEvent evt ) {
          System.exit( 0 );
        }
      }
    );
  }

  public void paint( Graphics g ) {
    if( imagenNueva != null ) {

        //*************
        Dimension d=size();
                int w=d.width;
                int h=d.height;
        int xr=(w-256)/8;
        int xg=(w-256)/2;
        int xb=(w-256)/1;
                int lastyr = h-h * histored[0]/max_histr;
                int lastyg = h-h * histogreen[0]/max_histg;
                int lastyb = h-h * histoblue[0]/max_histb;
                  for (int i=0; i<256; i++, xr++) {
                        int yr= h-h * histored[i]/max_histr;
                                g.setColor(new Color(i,i,i));
```

```
                                g.fillRect(xr,yr,1,h);
                                g.setColor(Color.red);
                                g.drawLine(xr-1,yr,xr,yr);

            lastyr=yr;
                  }
             for (int i=0; i<256; i++, xg++) {
                    int yg= h-h * histogreen[i]/max_histg;
                                g.setColor(new Color(i,i,i));
                                g.fillRect(xg,yg,1,h);
                                g.setColor(Color.green);
                           g.drawLine(xg-1,lastyg,xg,yg);
                        lastyg=yg;
                           }


                for (int i=0; i<256; i++, xb++) {
                        int yb= h-h * histoblue[i]/max_histb;
                                g.setColor(new Color(i,i,i));
                                g.fillRect(xb,yb,1,h);
                                g.setColor(Color.blue);
                           g.drawLine(xb-1,lastyb,xb,yb);

            lastyb=yb;

            this.setSize(700, 500);
        }



     //*************
 /*+   Dimension d = size();
     Color bg = getBackground();
     Color fg = getForeground();
     for(int i=0;i<256;i++){
     g.setColor(new Color(255, 0, 0));
     //para diburar las lineas
     g.drawLine(i,0,i,histored[i]/2);
     g.setColor(new Color(0, 255, 0));
     // Las barras color verde
     g.drawLine(i+300,0,i+300,histogreen[i]/300);
     g.setColor(new Color(0, 0, 255));
     // para dibujar el color azul
     g.drawLine(i+600,0,i+600,histoblue[i]/2);
     //this.setSize(400, 400);
     }   */
    }
   }
 }
```

Grid.java

```
import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
//import java.awt.Graphics.*;
//import java.awt.image.*;
//import javax.swing.*;
//import java.awt.geom.*;



// Este es un paquete nuevo del JDK 1.1

public class grid {
        Image imagenFuente;

   public void main( String args[] ) {
      IHM ihm = new IHM(imagenFuente);
       }
   }

 // Se crea una subclase de Frame para poder sobreescribir el método
// paint(), y presentar en pantalla las coordenadas donde se haya
// producido el click del ratón

class MiFrame extends Frame{
   int ratonX;
        int ratonY;
   Image imagenFuente; // esto lo agregue
   public void paint( Graphics g ) {
           int latitud = ratonX;
     int longitud = ratonY;

                // pantalla2= getSize();
                imagenFuente = getImagenFuente();
                g.drawImage(imagenFuente,0,0,null);


                  if (ratonX <= 612 & ratonY<=384){
                      latitud = 20;
```

```
                        longitud = 30;
                         }



                  else if (ratonX > 612 & ratonY<384){
                                          latitud = 200;
                                  longitud = 30;

                                  }

                                  else if (ratonX < 612 & ratonY>384){
                                                                  latitud = 400;
                                                          longitud = 30;

                                  }
                          else if (ratonX > 612 & ratonY>384){
                                                                  latitud = 500;
                                                          longitud = 30;

                                  }

                                  else {
}

        g.drawString( ""+latitud+"+grados, "+ratonY,ratonX,ratonY );
        for(int i=0;i<256;i++){
                g.setColor(Color.gray); //vertical
        g.drawLine(i*20,0,i*20,i+800);
        g.setColor(Color.gray); // horizontal
                g.drawLine(0,i*20,i+1100,i*20);

          }
        }

          /** return */
          public Image getImagenFuente() {
                return imagenFuente;
          }

          /** param image */
          public void setImagenFuente(Image image) {
                imagenFuente = image;
          }
    }

 //}
```

/ ********* **Esta clase se utiliza para instaciar un objeto de tipo interfaz de Usuario*******/

```
class IHM {
   public IHM(Image imagen) {
                Image imagenFuente = imagen;
                MiFrame ventana = new MiFrame();
        ventana.setImagenFuente(imagen);


       ventana.setSize( 1000,700 );
       ventana.setTitle( "Tutorial de Java, Eventos" );
       ventana.setVisible( true );
```

/***** **Se instancia y registra un objeto receptor de eventos para terminar la ejecución del programa cuando el usuario decida cerrar la ventana** *******/

```
     Proceso1 procesoVentana1 = new Proceso1();
      ventana.addWindowListener( procesoVentana1 );
```

/********** **Se instancia y registra un objeto receptor de eventos que será el encargado de procesas los eventos del ratón para determinar y presentar las coordenadas en las que se encuentra el cursor cuando el usuario pulsa el botón del ratón** ********/

```
       ProcesoRaton procesoRaton = new ProcesoRaton( ventana );
       ventana.addMouseListener( procesoRaton );
       System.out.println( imagenFuente );

       }
   }
```

/*******Esta clase Receptora monitoriza las pulsaciones de los botones del ratón y presenta las coordenadas en las que se ha producido el clic Se trata de una clase Adpater, luego solo se redefinen los métodos que resulten útiles para el objetivo de la aplicación********/

```
class ProcesoRaton extends MouseAdapter {
  MiFrame ventanaRef; // Referencia a la ventana
```

```
 /***** Constructor *****/

   ProcesoRaton( MiFrame ventana ) {

 /****** Guardamos una referencia a la ventana *****/

      ventanaRef = ventana;
      }

/**** Se sobreescribe el metodo mousePressed para determinar y  presentar
en pantalla las coordenadas del cursor cuando se pulsa el raton *******/


   public void mousePressed( MouseEvent evt ) {


 /***** Recoge las coordenadas X e Y de la posición del cursor y las almacena
en el objeto Frame ******/

      ventanaRef.ratonX = evt.getX();
      ventanaRef.ratonY = evt.getY();

 /****** Finalmente, presenta los valores de las coordenadas *******/

      ventanaRef.repaint();
      }
   }


/******* Este repector de eventos de la ventana se utiliza para concluir la
ejecución del programa cuando el usuario pulsa sobre el botón de cierre del
Frame ********/


class Proceso1 extends WindowAdapter {
   public void windowClosing( WindowEvent evt ) {
      System.exit( 0 );
      }
   }
```

BIBLIOGRAFIA

[1] Booch Grady. Rational Santa Clara, California. Object Oriented Análisis and Design. The Benjamin/ Cummings Publishing Company, Inc. Second Edition1994. pp 589.

[2] Booch Grady, James Rumbaugh & Ivar Jacobson, 2001, The Unified Modeling Language User Guide. Editorial Addison Wesley 2001. pp 482 .

[3] Bud Timothy, An Introduccion to object oriented programming, Addison Wesley Publishing Company, Oregon State University, 1991, pág.4

[4] Ceballos Francisco Javier. Programación Orientada a Objetos con C++. Editorial Computec. Segunda edición, 1998. pp 676.

[5] Deitel H.M y Deitel P.J. Cómo Programar en C/C++, Editorial Prentice Hall. Segunda edición, 1995. pp 906

[6] Fowler Martin & Kendall Scott. UML Distilled A brief guide to the Standard Object Modeling Language, Edit Addison-Wesley 2000 pp 183.

[7] Graham Ian , Object Oriented Methods,

[8] Harmon Paul & Mark Watson. Understanding UML The Developer´s Guide.1998. pp 367.

[9] Jacobson Ivar, Magnus Christeeson, Patrick Jonsson & Gunnar Overgaard. Object – Oriented Software Engineering. Edit Adison – Wesley. 1995. pp 528.

[10] Joyanes Aguilar Luis. Programación Orientada a Objetos. Osborne McGraw-Hill 1998. Segunda Edición. pp 895.

[11] JSP & JSD Cameron R. John. The Jackson approach to software development. IEEE Computer Society Press. Second Edition. 1989. pp 526.

[12] Martín James & James J Odell, Análisis y Diseño Orientado a Objetos, Prentice Hall. 1993.

[13] Pressman R.S Ingeniería del Software. Un enfoque práctico. Mc Graw-Hill. 1993. España. 824 pp.

[14] Quatrini Terry.Visual Modeling with Rational Rose 2000 And UML Addison-Wesley.1999. pp 256.

[15] Rumbaugh James, Michael Blaha, William Premerlani, Frederick Eddy. Object-Oriented Modeling and Design. Prentice Hall, 1991, pp 500

[16] Rumbaugh James, Ivar Jacobson & Grady Booch. The Unified Modeling Language Reference Manual. Edit Addison Wesley.1998. pp 550

[17] Stevens Perdita & Rob Pooley, 2000, Using UML SoftWare Engineering With objects and Components. Addison- Wesley. pp 256

[18] Voss Greg, Programación Orientada a Objetos,
Mac Graw Hill

[19] VV.AA. , Aprendiendo Programación Orientada a Objetos en 21 Lecciones. Prentice Hall Mexico. 2002. pp 111