

Universidad Nacional de Río Cuarto  
Facultad de Ciencias Exactas, Físico-Químicas y Naturales  
Departamento de Computación



---

**jTLC**  
Analizador de Cromatografía de Capa Fina

---

Trabajo Final (Cód. 1970)  
Carrera Licenciatura en Ciencias de la Computación

**Tardivo Cristian Adrián**  
DNI: 34.771.116  
**Baldani Sergio Ezequiel**  
DNI: 35.544.299

Director: **Dr. Renzo Degiovanni**

Co-Director: **Dr. Nazareno Aguirre**  
**Dr. Pablo Rossi**

Río Cuarto, 2016

## **Resumen**

Este trabajo implica el desarrollo de un software capaz de realizar un análisis de cromatografía, puntualmente TLC. TLC (Thin Layer Chromatography) o cromatografía de capa fina es una técnica cromatográfica que utiliza una placa vertical que consiste en una fase estacionaria polar adherida a una superficie sólida. A partir de esta placa se pueden obtener datos e información útil para el análisis de la composición química de las sustancias que se ponen a prueba.

El software desarrollado permite el análisis cromatográfico descripto, al trabajar con la placa mencionada y una foto o imagen se podrá llevar a cabo la técnica de cromatografía y analizar los componentes químicos. Al final del proceso se entrega un reporte con los resultados completos sobre los compuestos observados. El software permite a la vez el guardado de proyectos para retomar el análisis y una personalización de todos los parámetros, medidas y áreas a tener en cuenta. También cuenta con ayudas de guía constantes en el paso a paso que implementa el software, la posibilidad de insertar comentarios y de retroceder en los pasos ya llevados a cabo.

# Índice general

<b>1 Introducción</b>	<b>2</b>
1.1 Cromatografía . . . . .	4
<b>2 Conceptos Previos</b>	<b>5</b>
2.1 Programación Orientada a Objetos . . . . .	5
2.1.1 Java . . . . .	6
2.2 Patrones Arquitecturales . . . . .	7
2.2.1 MVC: Model-View-Controller . . . . .	8
2.3 Teoría de Cromatografía . . . . .	10
2.3.1 Teoría Principal . . . . .	10
2.3.2 Diferentes Usos del Análisis Cromatográfico . . . . .	11
2.4 Aplicación Específica: TLC . . . . .	12
2.5 Herramientas . . . . .	13
2.5.1 Entornos de Desarrollo Integrado . . . . .	13
2.5.1.1 NetBeans Java IDE . . . . .	13
<b>3 Análisis del problema</b>	<b>14</b>
3.1 TLC: Cromatografía en capa fina . . . . .	14
3.2 Creación de un nuevo proyecto . . . . .	15
3.3 Exploración de proyectos . . . . .	17
3.4 Exportación de datos . . . . .	17
3.5 Procesos destacados . . . . .	17
3.5.1 Búsqueda de puntos de corte . . . . .	18
3.5.2 Búsqueda de muestras . . . . .	18
3.5.3 Búsqueda de picos . . . . .	19
3.5.4 Integración del área . . . . .	20
3.5.5 Ejemplo del procesado de imágenes y resultados parciales	20
<b>4 Diseño, Implementación y Prueba</b>	<b>40</b>
4.1 Diseño General del Sistema . . . . .	40
4.2 Detalles de Implementación . . . . .	43
4.3 Prueba . . . . .	64
<b>5 Conclusión y Trabajos Futuros</b>	<b>65</b>

# Capítulo 1

## Introducción

La tecnología actual permite que se desarrolle grandes cantidad de programas destinados a diversos avances científicos, mientras que la ciencia se ha encargado a la vez de poner al desarrollo de software a su servicio. La existencia del software libre ha facilitado la relación entre la tecnología y la ciencia, ya que los programadores cuentan con una diversa cantidad de herramientas para aplicar las técnicas científicas. Si bien en muchos casos las técnicas aplicadas requieren de un hardware especializado para su finalidad, existen técnicas capaces de aplicarse con recursos más accesibles aunque a veces no se logren resultados con la precisión de técnicas con hardware específico.

Normalmente la creación de un programa se lleva a cabo a partir de integración de diferentes campos de conocimiento. Por ejemplo, si se desarrolla un software que trabaje con planillas de cálculo, lo más seguro es que se mezclen áreas como la computación (el desarrollador y sus conocimientos) con la economía, para tomar conocimiento de qué y cómo se deberían ejecutar ciertas acciones que este tipo de software requiere; lo mismo sucede en los casos de hacer un software con fines de diseño gráfico ya que se necesitan los conocimientos de un diseñador para poder nutrir de funciones útiles para este tipo de profesionales. El programador debe integrar sus conocimientos de algún modo para poder entender qué está haciendo y para qué, logrando así, al nutrirse de información, un software que cubra las necesidades del usuario final del software.

En la actualidad existen múltiples recursos tecnológicos para poder ejecutar análisis con mayor facilidad y precisión. A la vez es posible aplicar algoritmos que ayuden al usuario en el proceso de cualquier análisis. Es posible crear programas que guíen al usuario durante el trabajo y realizar tareas

con un alto nivel de detalle y precisión generando informes completos que aporten datos estadísticos en muchos estilos y formatos, aportando incluso imágenes.

El objetivo primordial de esta tesis es crear un software capaz de realizar el proceso de cromatografía en capa fina que se realiza en la Universidad Nacional de Río Cuarto (UNRC). Dicho proceso se complementa con el software de nombre Christhin. El nuevo software a desarrollar reemplazará a Christhin en el proceso, buscando brindar a los usuarios mayores facultades y beneficios, a fines de mejorar el estudio realizado. Puntualmente se trabaja en biodiésel y el análisis de cromatografía aporta un indicio sobre la pureza y calidad del mismo para pasar luego por un estudio de mayor precisión y análisis que requiere de hardware específico, con el que no se cuenta actualmente como un abundante recurso en el país. Por lo tanto es muy importante determinar mediante TLC cuáles son las muestras y componentes que debieran pasar por aquel proceso tan dificultoso. Mejorar la precisión del TLC, es a la vez mejorar las oportunidades de obtener un producto de calidad.

## 1.1. Cromatografía

La ciencia que se encarga de estudiar los diferentes componentes de una sustancia, es la química analítica. Una de las técnicas más usadas para separar los distintos componentes de una mezcla para su posterior estudio, es la cromatografía. Cuando se deja mover una mezcla sobre un soporte, como la tela o papel (y diversos materiales), los elementos de la mezcla son retenidos en la superficie del soporte de diferente manera y a diferentes velocidades. Al final terminan separándose. Un ejemplo un poco más cotidiano sobre esto, es si se derrama vino sobre un mantel, la mancha que se genera no es uniforme, sino que hay una zona donde predominarán tonos azules y otra con la tonalidad roja. Los pigmentos del vino se han separado de cromatográficamente.

Cromatografía es la técnica que permite separar los componentes de una mezcla y su análisis posterior, basado en que las distintas sustancias que forman los componentes de una mezcla se dejan arrastrar a diferentes velocidades sobre un soporte. Este método que permite la separación es físico.

Luego de esto se puede analizar de diferentes formas y se utiliza para lograr la separación de los componentes de una mezcla como para medir la proporción de cada elemento de la misma.

# Capítulo 2

## Conceptos Previos

### 2.1. Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) [5] es un paradigma de programación basado en el uso de clases<sup>1</sup> donde sus instancias son llamadas objetos. La idea principal en este paradigma es lograr crear, por medio del *Software*, un sistema basado en componentes independientes que interactúen entre ellos de la misma forma que lo hace el sistema concreto a modelar. La programación orientada a objetos se basa en diseñar interacciones entre objetos para crear aplicaciones y sistemas informáticos, usando diversas técnicas como la abstracción, la herencia, el polimorfismo, la cohesión, el acoplamiento y encapsulamiento. Un objeto busca modelar un elemento del mundo real. Cada objeto encapsula un propio estado mediante los datos de sus atributos, y su comportamiento definido mediante código. Los objetos se pueden organizar en jerarquías de clases por medio de herencia, facilitando así el rehuso y categorización de atributos en común. Brindan encapsulamiento y ocultamiento de información permitiendo así dividir un sistema estructurado en objetos modulares los cuales poseen sus propios datos (atributos) y son responsables de su propio comportamiento. Por medio de encapsulamiento los datos de un objeto son independientes a otros, por lo que los cambios efectuados en uno no deberían afectar al otro.

La programación orientada a objetos reta al desarrollador a ubicar los datos de tal manera que no sea directamente accesible por el resto del sistema,

---

<sup>1</sup>Plantillas para la creación de objetos que brinda los valores iniciales del estado (variables) y implementaciones de comportamiento (métodos).

sino que su acceso sea realizado por medio de llamadas a funciones especiales o métodos los cuales se incluyen juntos a los datos. Estos métodos actúan de intermediarios permitiendo el control de los datos dentro de los objetos.

Diseñar Software usando componentes modulares que soportan herencia (clases) tiene la intención de hacer mas fácil la reutilización de componentes existentes, como así también permitir definir fácilmente otros nuevos extendiendo ya existentes en subclases con comportamiento especializado.

Ademas de facilitar el diseño del sistema, la POO permite aplicar de manera mas sencilla patrones de diseño logrando así desarrollar soluciones mas sencillas y elegantes a problemas recurrentes.

### 2.1.1. Java

*Java* es un lenguaje de programación orientado a objetos y una plataforma de computación desarrollada por *Sun Microsystem* en 1995. Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases. Además ha sido diseñado para tener pocas dependencias de implementación en una plataforma específica permitiendo que el desarrollador pueda correr sus aplicaciones en cualquier dispositivo o sistema operativo. Para lograr esto *Java* corre sobre una maquina virtual (*Java Virtual Machine*). El código escrito por el programador se compila a un código intermedio conocido como *Java Bytecode* el cual es ejecutado en la máquina virtual (*JVM*), la cual lo interpreta y ejecuta. Además de esto, *Java* brinda bibliotecas adicionales para acceder a características de cada dispositivo (gráficos, *threads*, *network*, etc) de forma unificada.

Su sintaxis deriva en gran medida de C y C++ pero tiene menos características de bajo nivel que cualquiera de ellos. No permite sobrecarga de operadores ni ofrece herencia múltiple, posee un sistema automático de asignación y liberación de memoria (basado en un recolector de basura). Es un lenguaje estáticamente tipado, con *binding* dinámico.

*Java* se ha diseñado para trabajar en ambiente de redes y contienen una gran biblioteca de clases para la utilización del protocolo *TCP/IP*, incluyendo *HTTP* y *FTP*. El código *Java* se puede manipular a través de recursos *URL* con la misma facilidad que C y C++ utilizan recursos locales (archivos). Además de esto, *Java* soporta hilos múltiples, permitiendo desarrollar así aplicaciones concurrentes más ricas.

## 2.2. Patrones Arquitecturales

Un patrón arquitectural expresa un esquema fundamental para la organización estructural de un sistema de Software. Proporciona un conjunto predefinido de sub sistemas especificando sus responsabilidades, definiendo reglas y directrices para la organización de las relaciones y dependencias entre ellos [6]. Los patrones arquitecturales representan el nivel más alto dentro de los patrones de sistemas, estos ayudan al desarrollador a especificar la estructura fundamental de una aplicación y alcanzar una propiedad específica global del sistema, tales como adaptabilidad, fiabilidad, etc.

Estos patrones pueden agruparse en 4 categorías:

- *From Mud to Structure (Del barro a la estructura)*: los patrones en esta categoría se centran en evitar un mar de componentes u objetos, en particular, dan soporte a la descomposición estructurada de la tarea general del sistema en sub tareas. En esta categoría están los patrones como *Layers*, *Pipes and Filters* y *Blackboard*.
- *Distributed System (Sistemas distribuidos)*: dentro de esta categoría está el patrón *Broker* el cual provee una completa infraestructura para aplicaciones distribuidas.
- *Interactive Systems (Sistemas interactivos)*: los patrones *Model-View-Controller* y *Presentation-Abstraction-Control* permiten desarrollar sistemas basados en interacción humano-computadora de manera eficiente y ordenada.
- *Adaptable Systems (Sistemas adaptables)*: patrones como *The Reflection* y *Microkernel* permiten la generación de aplicaciones extensibles y de fácil adaptación a la evolución tecnológica y necesidades funcionales.

A pesar de esto, la gran mayoría de sistemas no pueden ser estructurados de acuerdo a solo un patrón arquitectural, por lo que deben ser descompuestos y agrupados en sub patrones implementando funcionalidades individuales para luego ser combinados en un todo, logrando así un *framework* estructural.

### 2.2.1. MVC: Model-View-Controller

El patrón arquitectural *Model-View-Controller* divide una aplicación interactiva en tres componentes básicos. El *modelo*, el cual contiene el núcleo funcional y los datos. La *vista* quien muestra la información al usuario y el *controlador*, que se encarga de manejar los *inputs* del usuario [6]. Juntos, las vistas y los controladores, comprenden la interfaz de usuario. Además un mecanismo de propagación de cambios asegura la consistencia entre la interfaz de usuario y el modelo del sistema. En la figura 2.1 se detalla la estructura general del patrón.

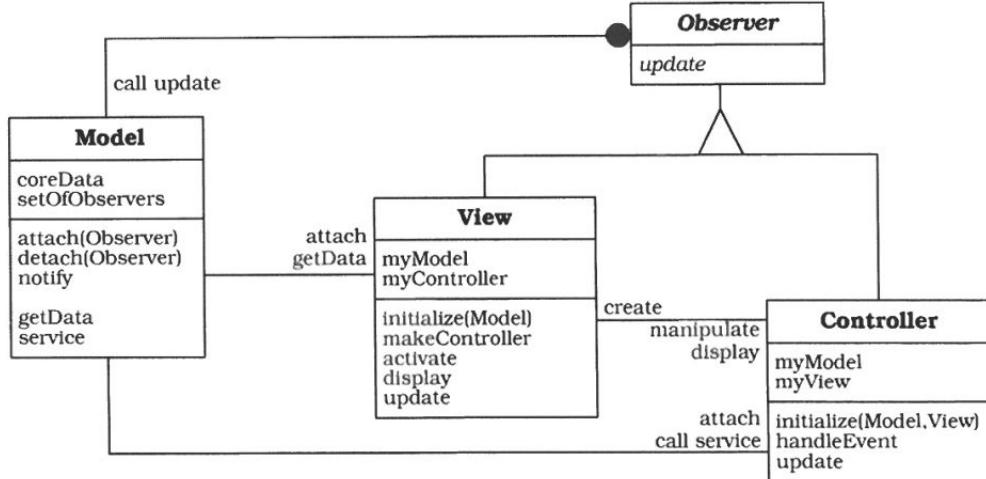


Figura 2.1: Estructura básica del patrón Model-View-Controller.

El modelo contiene el núcleo funcional de la aplicación, encapsula los datos y exporta procedimientos que realizan el procesamiento específico de la aplicación. Los controladores llaman a esos procedimiento en nombre del usuario. El modelo también provee funciones para el acceso a los datos que contiene, las cuales pueden ser utilizadas por los componentes de las vistas para adquirir los datos a mostrar [6]. El mecanismo de propagación de cambios mantiene registro de todos los componentes que dependen del modelo dado que las vistas y los controladores necesitan ser informados de los cambios ocurridos en él. Los cambios realizados en el estado del modelo disparan el mecanismo de propagación de cambios, siendo este mecanismo el único

*link* entre el modelo y los controladores de vistas. Las responsabilidades del modelo son: implementar el núcleo funcional de la aplicación, registrar las vistas y controladores dependientes de él y notificar los cambios sobre los datos que maneja.

Las vistas presentan la información al usuario, diferentes vistas pueden mostrar los mismos datos de formas alternativas. Cada vista define un procedimiento de actualización el cual es activado por el mecanismo de propagación de cambios. Cuando el procedimiento de actualización es llamado, la vista obtiene los datos del modelo y los muestra por pantalla. Durante la inicialización todas las vistas son asociadas con el modelo y registradas en el mecanismo de propagación de cambios. Cada vista utiliza sus propios controladores para lograr la interacción con el modelo y manipular la visualización de los datos [6]. Las responsabilidades de las vistas son: crear e inicializar el controlador asociado a ella, mostrar la información al usuario, implementar un procedimiento de actualización, obtener los datos del modelo, permitir la actualización (de ser necesario) de los datos mostrados.

Los controladores manejan los *inputs* del usuario como si fueran eventos. Se puede decir que cada controlador implemente un procedimiento de *event-handling* el cual es llamado en cada evento relevante a la vista. Estos eventos son transformados en requerimientos al modelo o a la vista asociada al controlador [6]. En el caso de que el comportamiento del controlador dependa del estado del modelo, este debe estar asociado con el mecanismo de propagación de cambios. Las responsabilidades de los controladores son: aceptar los *inputs* del usuario como eventos, transformar esos eventos en requerimientos a servicios del modelo o la vista, implementar un procedimiento de actualización de ser necesario.

En la figura 2.2 se detalla en un escenario en particular el comportamiento general del patrón. Primero el controlador recibe el *input* del usuario, procesa el evento y llama a un procedimiento del modelo el cual ejecuta el servicio requerido generando cambios en los datos que maneja. Luego el modelo notifica a todas las vistas y controladores, y registra los cambios por medio del mecanismo de propagación de cambios, llamando a los respectivos procedimientos de actualización. Seguido a esto las vistas solicitan al modelo los datos actualizados y refresca la información que muestra. Opcionalmente el modelo puede enviar datos al controlador para actualizar sus funcionalidades. La traza de ejecución finaliza retornando al controlador original.

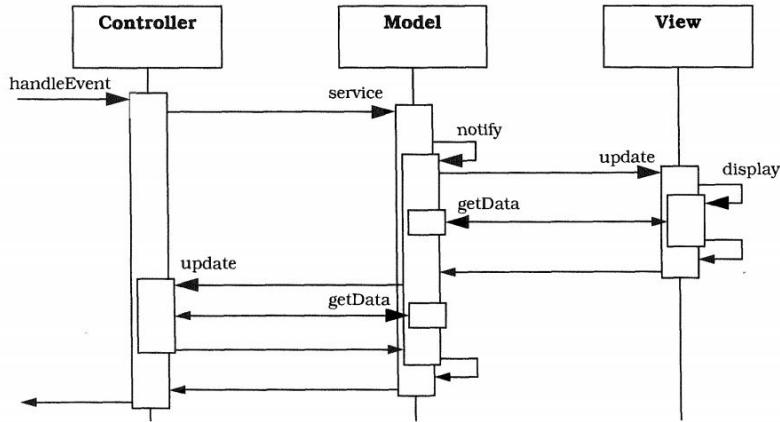


Figura 2.2: Diagrama de secuencia: comportamiento general MVC.

## 2.3. Teoría de Cromatografía

### 2.3.1. Teoría Principal

La ciencia que se encarga de estudiar los diferentes componentes de una sustancia es la química analítica. La técnica que más se utiliza para separar los componentes en una mezcla para que esta se analice, se llama cromatografía. La mezcla básicamente se trata sobre un soporte (papel o tela por ejemplo), dejando que la misma corra y según las diferentes velocidades de cada componente, los mismos se separan. El hecho de que los componentes se muevan a diferentes velocidades se debe a las diferentes fuerzas de adsorción. Y se dice adsorción cuando una sustancia se introduce dentro de la estructura de otra sustancia, un ejemplo claro de dicha definición se da al lograr que una esponja absorba agua y si cortamos la esponja en pedazos, se encontrará agua. En diferentes perspectivas, en la absorción no introduce la sustancia al volumen sino que solo se adhiere a la superficie, por ende la adsorción es un fenómeno superficial. Teniendo en cuenta los nuevos términos introducidos se puede decir que la cromatografía es una separación física de los componentes de una mezcla basado en la adsorción selectiva de los componentes de la mezcla al moverse por un soporte. El fin de la cromatografía es separar los componentes de una mezcla para medir la proporción de cada elemento. La técnica de la cromatografía se aplica en dos fases:

- Fase estática, la mezcla se coloca sobre un soporte fijo.
- Fase móvil, se mueve otra sustancia sobre la mezcla ya presente en la fase estática, en el soporte. Es durante esta fase donde se inicia el proceso de separación de componentes.

### 2.3.2. Diferentes Usos del Análisis Cromatográfico

La cromatografía tiene diferentes usos en todas sus clasificaciones:

- Metilación de ácidos grasos en el análisis de aceite.
- Análisis industriales.
- Análisis forenses bromatológicos.
- Análisis toxicológicos.
- Análisis clínicos.
- Análisis de contaminación ambiental.
- Separación de iones interferentes en análisis clásicos.
- Separación de iones de características similares.
- Desmineralización del agua.
- Preparación de disoluciones.
- Disolución de sustancias insolubles.
- Fraccionamiento de proteínas.
- Determinación de grado de pureza.
- Seguimiento de reacción.
- Determinación de aminoácidos o proteínas en una solución.
- Aplicación sobre alimentos y productos naturales.

## 2.4. Aplicación Específica: TLC

La cromatografía en capa fina (TLC) es un método de cromatografía en el plano. Se emplea una capa plana y delgada que a la vez es el soporte o que recubre una superficie (vidrio, plástico). La fase móvil se mueve a través de la fase estacionaria por capilaridad, es una técnica rápida, de buena resolución y más sensible que la cromatografía en papel. Se han reportados grandes avances en la producción de biodiesel por medio de reacciones de transesterificación <sup>2</sup> y operaciones de tratamiento de aceites destinadas al consumo humano. La técnica de TLC resulta de gran importancia para seguir las reacciones correspondientes y medir el nivel de producción. El objetivo principal de la implementación de la técnica de TLC es la determinación de la composición de los compuestos lipídicos que conforman la mezcla oleosa interviniente en la transesterificación para la producción de biodiesel. Las ventajas de seguir este procedimiento son la rapidez y el bajo costo de los ensayos experimentales en capa fina. La cromatografía en capa fina es una herramienta muy útil para controles de calidad y pureza de productos. A su vez, encuentra una extensa aplicación en laboratorios industriales. En lo que respecta al resultado final de aplicar la técnica de TLC, comparando el área de la mancha del estándar con la del analito, se puede hacer una estimación cuantitativa de la cantidad del componente presente. Los mejores resultados se obtienen cuando se raspa la mancha de la placa, se extrae el analito del sólido que forma la fase estacionaria, y se determina el analito por algún método físico o químico adecuado. Un tercer procedimiento consiste en utilizar un densímetro de barrido que puede medir la radiación emitida de la mancha por fluorescencia o reflexión. Actualmente, el análisis digital de la capa fina es la técnica más precisa y aplicada. Para el análisis cuantitativo de la corrida cromatográfica de TLC, se debe procesar la placa digitalmente por algún medio óptico-digital, como ser un scanner, y luego procesar la información con un software apropiado. El software *Scion Image* de la Empresa *Scion Corporation (ScionCorp, 2010)* ha mostrado un alto grado de desenvolvimiento para el procesamiento de imágenes obtenidas por TLC lo que permite obtener información sobre la intensidad de color de la imagen, y esto puede ser relacionando con la concentración de los compuestos lipídicos. Para el procesamiento de los datos obtenidos, el software *Matlab 7.8* de la

---

<sup>2</sup>La transesterificación es el proceso de intercambiar el grupo alcoxi de un alcohol. Es un proceso utilizado en la producción de biodiésel.

empresa *MathWorks* presenta un gran número de herramientas analíticas, lo que permite obtener la variación de la concentración de los compuestos lipídicos respecto al corrimiento mostrado en la placa TLC.

## 2.5. Herramientas

### 2.5.1. Entornos de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE) es una aplicación (*Software*) que brinda muchas facilidades a los programadores en el desarrollo de sistemas. Un IDE consiste normalmente de un editor de código, un compilador y/o intérprete, un depurador y hasta un constructor de interfaz gráfica. La mayoría de los IDEs modernos poseen herramientas de auto-completado inteligente de código, brindan soporte a diversos lenguajes de programación, poseen navegador de clases y paquetes, refactorización inteligente de código entre otros.

Los IDEs más complejos brindan la posibilidad de anexarle *Software* de terceros en forma de *plug-ins* brindando extensibilidad funcional como control de versiones, generadores de casos de pruebas, pruebas unitarias, analizadores de memoria, análisis de código en tiempo de ejecución, constructores de interfaz avanzados, etc.

#### 2.5.1.1. NetBeans Java IDE

*NetBeans* es un entorno de desarrollo integrado libre, hecho principalmente para programar en Java, aunque también soporta diversos lenguajes como *PHP*, *C/C++* y *HTML5* junto a *JavaScript* y *CSS*. *NetBeans* está escrito en Java y puede ser ejecutado en *Windows*, *OS X*, *Linux*, *Solaris* y cualquier otra plataforma que sea compatible con la *JVM* (*Java Virtual Machine*) [3]. Fue desarrollado inicialmente en el año 2000 por *Sun MicroSystem* como un proyecto de *Software* libre el cual continúa en desarrollo y crecimiento por una comunidad de desarrolladores junto a *Oracle Corporation*. *NetBeans* está diseñado en forma modular, por lo que cada función del sistema esta provista por un modulo en particular. De esta forma el usuario puede agregar sus propios módulos al sistema (respetando una API descrita por el IDE) o bien puede obtenerla del repositorio de *plug-ins online*.

# **Capítulo 3**

## **Análisis del problema**

### **3.1. TLC: Cromatografía en capa fina**

La cromatografía en capa fina es una técnica rápida muy utilizada en laboratorios de química orgánica, además permite: determinar el grado de pureza de un compuesto; comparar muestras, si dos muestras corren de forma idéntica, podrían ser iguales, de lo contrario son compuestos diferentes; realizar el seguimiento de una reacción, estudiando el comportamiento de reactivos y productos finales. Se toma la muestra y se deposita en un extremo de una lámina de plástico o aluminio que previamente debieron ser recubiertos por una fina capa de adsorbente. Dicha lámina se coloca en una cubeta cerrada junto con uno o varios disolventes mezclados. A medida que estos disolventes ascienden por la capilaridad a través del adsorbente, se produce el reparto diferencial de los productos que se hayan en el compuesto, entre el adsorbente y el disolvente. Existen en la actualidad dos adsorbentes que se utilizan en la mayoría de los experimentos, el gel de sílice ( $\text{SiO}_2$ ) y la alúmina ( $\text{Al}_2\text{O}_3$ ), ambas de carácter polar. La alúmina anhidra es el más activo de los dos, el que retiene con más fuerza los compuestos, se utiliza para separar compuestos relativamente apolares. El gel de sílice, por el contrario, se utiliza para separar sustancias más polares. El proceso de adsorción se debe a interacciones intermoleculares de tipo dipolo-dipolo o enlaces de hidrógeno entre el soluto y el adsorbente. El adsorbente debe ser inerte con las sustancias a analizar y no actuar como catalizador en reacciones de descomposición. El orden de elución de un compuesto se incrementa al aumentar la polaridad de la fase móvil o eluyente. El orden de elución de un compuesto

se incrementa al aumentar la polaridad de la fase móvil o eluyente. Se suelen utilizar disolventes con bajos puntos de ebullición y viscosidad, lo que les permite moverse con rapidez. Normalmente se emplea una mezcla de dos disolventes en proporción variable, mientras que la polaridad de la mezcla será el valor promediado en función de la cantidad de cada disolvente. El eluyente idóneo en cada caso se encuentra por método de ensayo-error. El desarrollo del experimento cromatográfico se realiza por lo general a través del método ascendente, lo cual se da al permitir que un eluyente ascienda por una placa casi en vertical, por la acción de la capilaridad. La cromatografía se realiza en una cubeta cromatográfica para conseguir la máxima saturación posible de la atmósfera de la cámara, las paredes se impregnán con el eluyente. El eluyente debe de colocarse un tiempo antes de iniciar el experimento, por lo que se acostumbra a colocarse con una hora previa, mientras que el desarrollo no suele llegar a los 30 minutos. Sin embargo la cromatografía cualitativa lleva apenas unos minutos, cuando la preparativa lleva horas. Una vez colocadas las placas, las mismas se desarrollan durante un tiempo prefijado o bien cuando alcanza una línea dibujada a una distancia fija del origen, distancia común a todas las muestras para estandarizar el valor RF (la distancia recorrida por el solvente). Las placas se pueden secar rápidamente con corrientes de aire caliente. La mejor posición de desarrollo para el componente es el punto medio entre el origen y el frente del eluyente, ya que permite separar las impurezas que se desplazan con mayor y menor velocidad. El frente del eluyente nunca debería tocar el borde de la placa.

## 3.2. Creación de un nuevo proyecto

- *Formulario inicial:* se completa un formulario en ventana de diálogo con los datos del proyecto: nombre, fecha de muestra, fecha del análisis y descripción.
- *Carga de la imagen:* se puede cargar la imagen a procesar a través de la exploración de archivos o bien con la moderna opción de drag & drop. La imagen no cuenta con un límite de tamaño pero bien se entiende que cuánto mayor sea, más precisión se obtendrá.
- *Corte de la imagen:* la imagen de muestra puede contener espacios extra en los bordes exteriores o imperfecciones que dificultarían el trabajo de

asistencia que ofrece el software, entonces el software hace una búsqueda de estas imperfecciones, detectándolas y apartandolas de la imagen a examinar, lo cual el usuario puede modificar si lo considera necesario.

- *Rotación de la imagen:* al igual que se remarcó antes, la posibilidad de la existencia de imperfecciones en el dibujo de la imagen, se puede dar el caso en el que la orientación no es la correcta o la deseada por el usuario, por ende se le brinda una herramienta de uso simple para que pueda alinear la muestra de modo tal que quede vertical cada uno de los experimentos.
- *Selección de muestras:* una vez que el usuario ha refinado los límites de la imagen de muestra el software la analiza y estima separando en diferentes áreas cada muestra interna por separado. Nuevamente se le brinda al usuario la posibilidad de personalizar la estimación e incluso agregar o eliminar áreas, agregar un nombre a cada muestra.
- *Selección de puntos especiales:* teniendo ya las muestras por separado se le puede determinar a cada una o de forma conjunta el límite superior e inferior de análisis, lo que naturalmente sería por ejemplo el punto de siembra como límite inferior y el frente del disolvente como límite superior.
- *Análisis de muestras:* en esta etapa del proceso se analizan individualmente las muestras en gráficos que representan a la muestra y delimitando las áreas correspondientes a los sectores más importantes de la muestra, donde se encuentra mayor concentración de componentes y visualmente se pueden ver manchas grandes o bien oscuras en relación al resto de la imagen de la muestra. Automáticamente el asistente del software seleccionará los picos del gráfico pero el usuario nuevamente puede dar corrección total a lo seleccionado por el sistema para poder proseguir. Esta etapa además ofrece un gráfico aparte que permite comparar las muestras en conjunto, con la selección particular sobre las muestras que se desea comparar.
- *Resultados de análisis:* esta etapa procesa cada muestra y por cada una se analizan los picos seleccionados. Los picos por muestra se enumeran y se detalla en pantalla la información obtenida: nombre del pico, límites, altura, superficie, superficie relativa y línea base. Se destaca además que

en cada etapa descripta se le permite al usuario añadir comentarios por cada muestra seleccionada.

- *Reportes:* se realiza una recopilación de la información obtenida en las etapas anteriores, acerca de la imagen original y sobre la que se trabaja, del mismo modo se muestra sobre cada muestra por separado y el análisis respectivo llevado a cabo, junto con los comentarios y la información recolectada.

### 3.3. Exploración de proyectos

El software permite la exploración de proyectos dentro del directorio configurado para buscar proyectos previamente guardados. Una vez seleccionada la opción se muestra una galería de imágenes con información debajo de cada una que corresponde a nombre y descripción del proyecto guardado, mientras que la imagen es la misma que subió originalmente el usuario para analizar.

### 3.4. Exportación de datos

En el proceso que se lleva a cabo en el software se permite exportar información, por ejemplo las imágenes originales a tratar y las procesadas por el sistema (debido al asistente y la previa aprobación del usuario), la información sobre la media muestral y los datos en crudo, permitiendo discriminar los datos en muestras individuales. Del mismo modo, llegando al final del proceso se le permite exportar el análisis completo en diferentes formatos como csv, pdf, odt o html.

### 3.5. Procesos destacados

En esta sección se hará una breve descripción sobre los procesos destacados del proyecto, sobre todo se hará énfasis en la asistencia que provee el software al usuario, las búsquedas automáticas y los resultados calculados. A fines de facilitar la compresión del proceso se aclararán ciertos procesos que serán frecuentemente nombrados.

- *Blur gaussiano:* efecto que se aplica para suavizar la imagen.

- *Escala de grises*: efecto que se aplica sobre la imagen y cambia sus valores de colores a escala de grises.
- *Inversión de colores*: efecto de imagen que se aplica para invertir los colores.
- *Threshold*: efecto que se aplica para monocromatizar los colores.
- *Cálculo de la media*: la imagen queda codificada en colores y se calcula un promedio de valores en base a si se necesita por orientación horizontal o vertical.

### 3.5.1. Búsqueda de puntos de corte

Como proceso previo se aplican los efectos de imagen en el siguiente orden: blur gaussiano, escala de grises, inversión de colores y threshold. Se realiza un cálculo de la media horizontal y vertical. A partir de cada media se busca una codificación de color que permite determinar el inicio de imagen de la muestra. Por ejemplo, suponga que se tiene escala de colores del 0 al 10 dentro de la media vertical, sabiendo que el inicio de la lista será el margen superior de la imagen. bajo el criterio que a partir de una codificación de 5 o más se puede determinar que la imagen real de la muestra ha iniciado. Entonces teniendo la siguiente media: 0000011222222233344555... es posible determinar que aproximadamente después de los primeros 20 puntos de la imagen, inicia la muestra real. Por lo tanto los primeros 20 puntos estarían sobrando y se diría que a partir del punto 20 se estima un punto de corte. De forma análoga se tratan el resto de los puntos de corte, el vertical inferior y los horizontales.

### 3.5.2. Búsqueda de muestras

Como proceso previo se aplican los efectos de imagen en el siguiente orden: blur gaussiano, escala de grises, inversión de colores y threshold. Se codifica la imagen desde una perspectiva vertical en ceros y unos para determinar las zonas con color, esto permite delimitar las zonas verticales donde existen muestras. A continuación se mostrará el ejemplo de una codificación y una interpretación de la misma.

```

  -----
0 0 0 1 0 0 0 0 1 0
0 0 1 1 0 0 0 0 1 0
0 0 0 0 0 0 1 0 0 0
  -----
1 0 1 1 0 0 1 0 1 0

```

Debajo de la línea de puntos se aprecia el resultado de la codificación. Entonces podrían determinarse cuatro áreas de muestra según el ejemplo, generadas por las subsecuencias de unos en el resultado. De este modo se toman los índices correspondientes con los que se determinan las áreas sobre la imagen real.

### 3.5.3. Búsqueda de picos

Como proceso previo se aplican los efectos de imagen en el siguiente orden: blur gaussiano, escala de grises e inversión de colores. Luego de aplicar los filtros, se realiza el siguiente proceso en cada muestra: se realiza el cómputo de la media desde una perspectiva horizontal y con esta lista de valores que representa a la parte más larga de la imagen que se considera la muestra se buscarán los picos. Dichos picos tendrán un alto mínimo para considerarse como tal. En un principio se realiza la búsqueda de un pico considerando el máximo de la lista que no se ha analizado. En base a este máximo se hace una búsqueda por izquierda y por derecha de un punto de inflexión, es decir, si se busca por izquierda del máximo, los valores decrecen con cierto ángulo de decrecimiento pero llegará el momento en que este ángulo se invierta o bien cobre estabilidad en sus valores (esto se analiza en base al gradiente obtenido entre los puntos), se puede decir que allí hay un punto de inflexión. Luego se realiza el mismo proceso hacia el lado derecho del máximo, se guardan los valores de puntos de inflexión y el área que se encuentra entre dichos puntos (que contienen al máximo del pico) se considera como área en la que se realizarán los cálculos futuros del análisis, áreas de integración. Dentro del análisis de áreas se realizan correcciones para evitar áreas superpuestas considerando un punto medio que separe a ambas. Como parte de la búsqueda de picos se realiza además un cálculo extra en el que se determina la línea base de cada área, la cual está definida por la recta que se forma entre cada punto de inflexión pero sin tener en cuenta aquellos puntos que corten la línea de la curva.

### 3.5.4. Integración del área

Una vez obtenidos los picos y las líneas base correspondiente se procede a realizar la integración del área sumando el área que hay bajo la función de la curva pero teniendo en cuenta además el área que está por encima de la línea base.

### 3.5.5. Ejemplo del procesado de imágenes y resultados parciales

A continuación se presentará un ejemplo en el que se podrá ver la imagen fuente del experimento y el proceso que recibe antes de llegar al resultado final.

- *Imagen fuente*



Figura 3.1: Imagen fuente.

- *Búsqueda de puntos de corte*

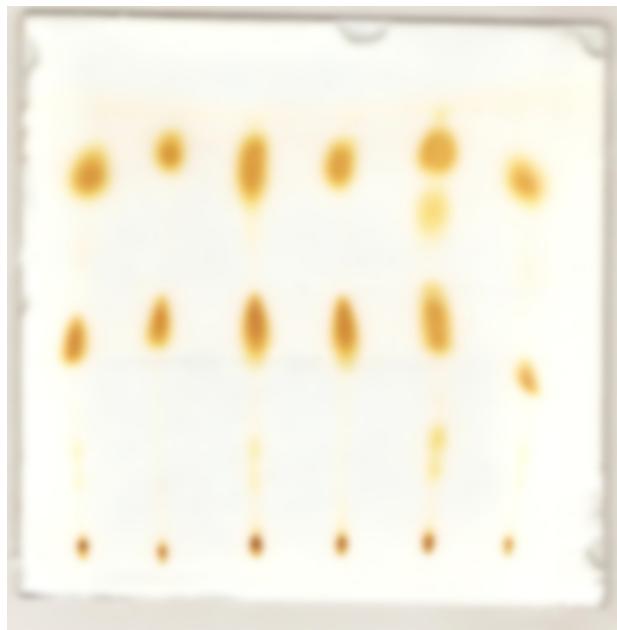


Figura 3.2: Imagen fuente con efecto blur aplicado.

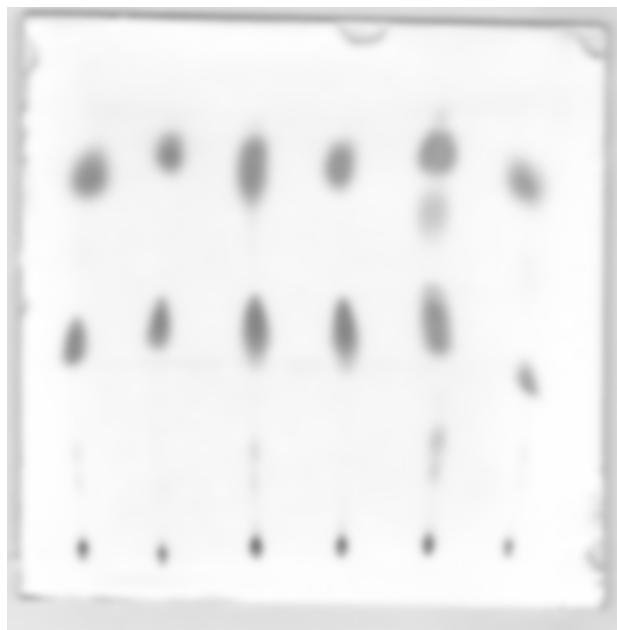


Figura 3.3: Imagen blur con efecto escala de grises aplicado.

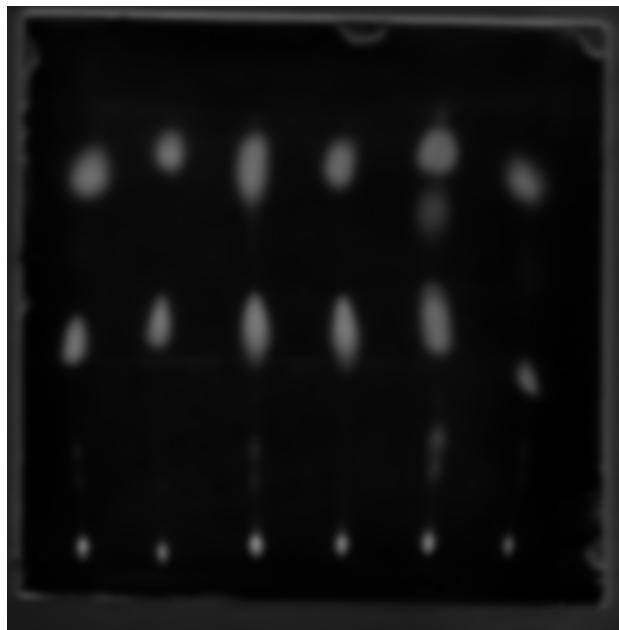


Figura 3.4: Imágen blur-gray con efecto colores invertidos aplicado.

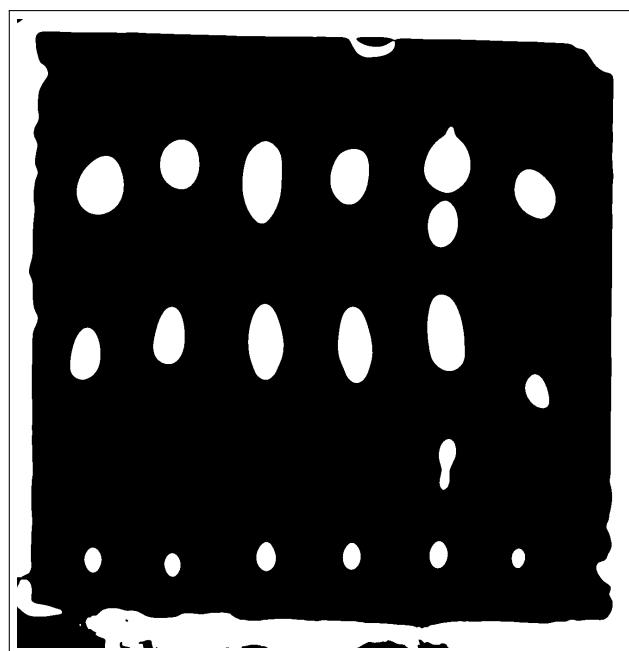


Figura 3.5: Imágen blur-gray-invert con efecto threshold aplicado.

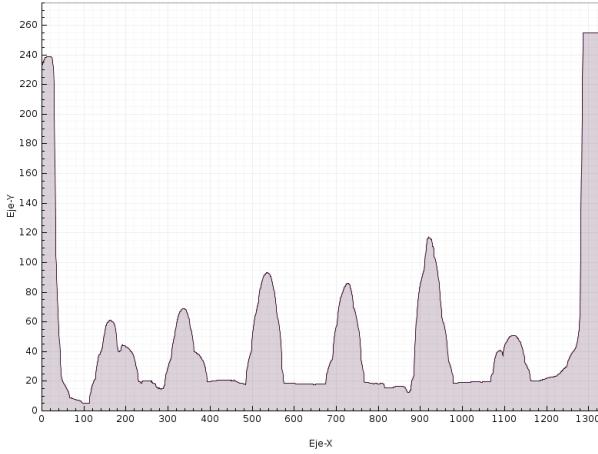


Figura 3.6: Plot de media sobre imagen con efectos en eje X.

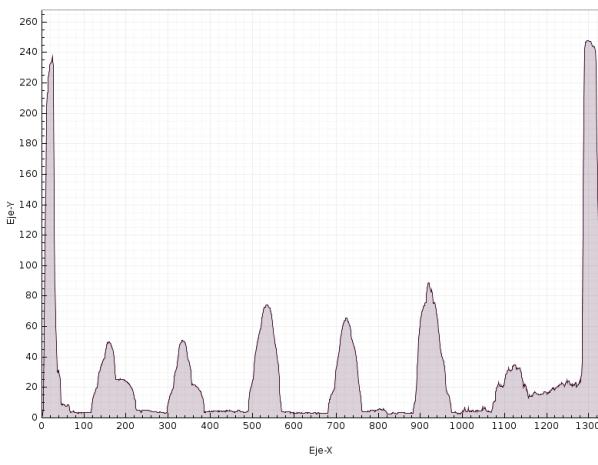


Figura 3.7: Plot de media sobre imagen con efectos (sin blur) en eje X.

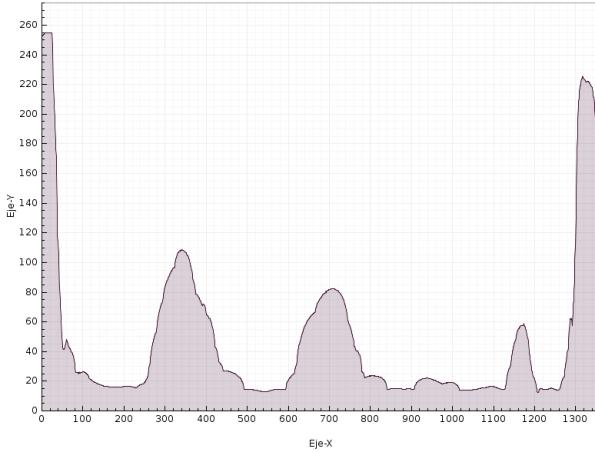


Figura 3.8: Plot de media sobre imagen con efectos en eje Y.

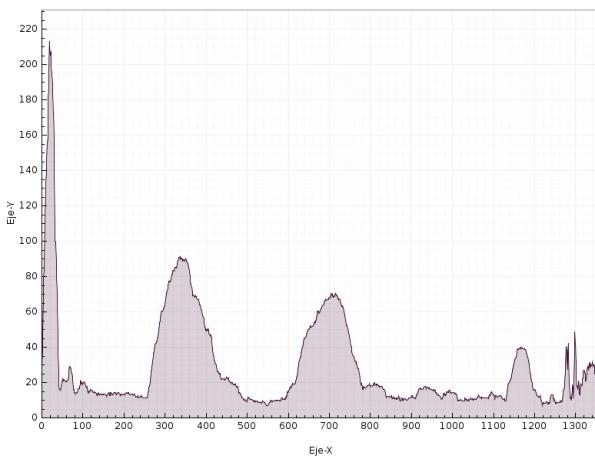


Figura 3.9: Plot de media sobre imagen con efectos (sin blur) en eje Y.

Puntos de corte encontrados:

top: (73, 135)

bottom: (1216, 1261)

- *Búsqueda de muestras*



Figura 3.10: Imágen fuente con efecto blur aplicado.



Figura 3.11: Imágen blur con efecto escala de grises aplicado.

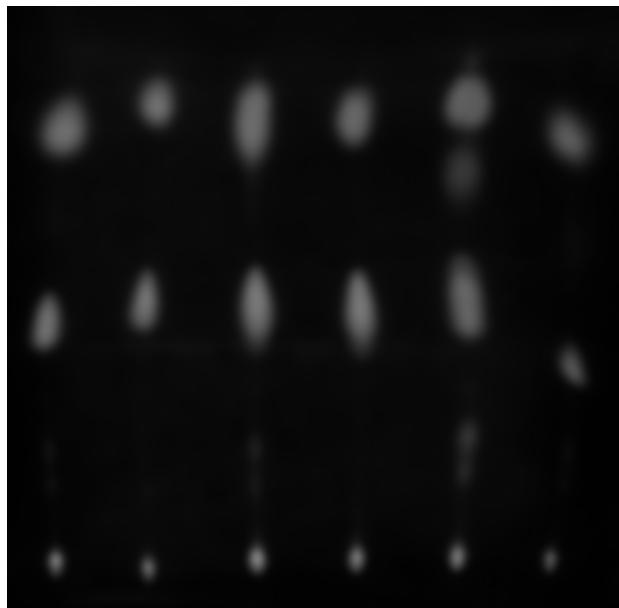


Figura 3.12: Imágen blur-gray con efecto colores invertidos aplicado.

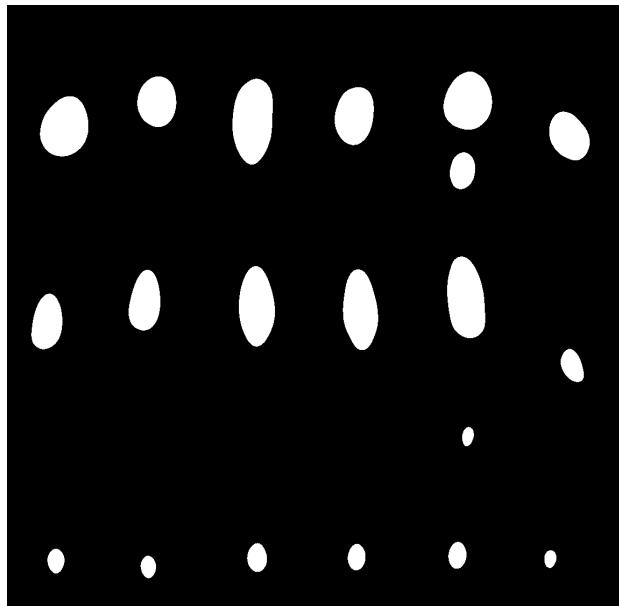


Figura 3.13: Imágen blur-gray-invert con efecto threshold aplicado.

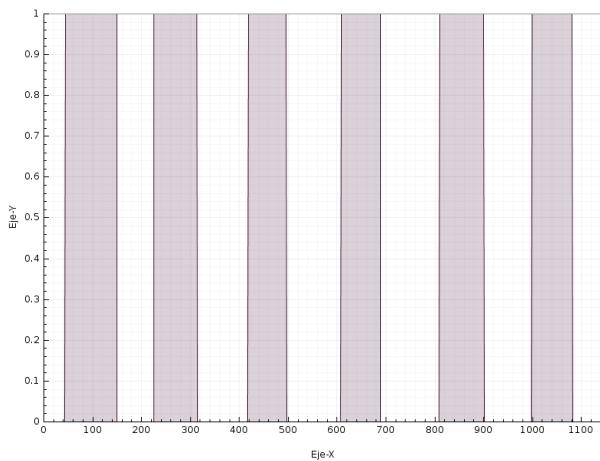


Figura 3.14: Plot de media sobre imagen con efectos en eje Y.

Muestras encontradas (puntos sobre eje-x):

- (45, 150)
- (226, 314)
- (419, 497)
- (609, 689)
- (811, 901)
- (999, 1082)

- *Búsqueda de picos (muestra 1)*

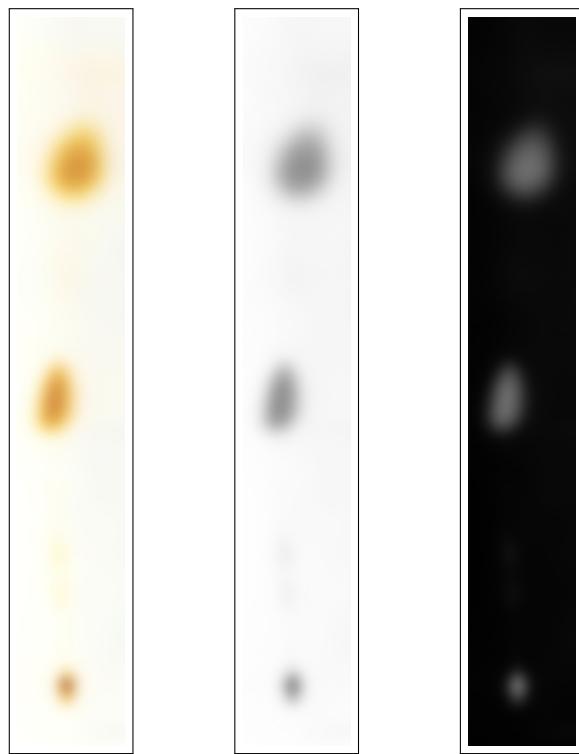


Figura 3.15: Aplicación de filtros sobre imagen de la muestra

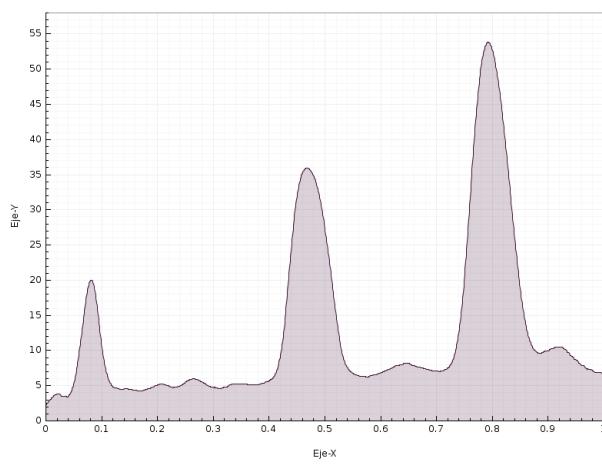


Figura 3.16: Plot de media sobre imagen con efectos en eje X.

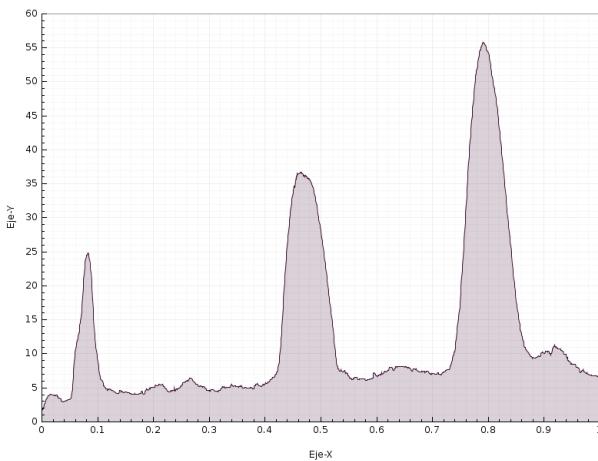


Figura 3.17: Plot de media sobre imagen con efectos (sin blur) en eje X.

Picos encontrados:

- (0.42933333, 0.53333336)
- (0.73422223, 0.8702222)

Picos encontrados (sin blur):

- (0.43733335, 0.47111112)
- (0.47111112, 0.52177775)
- (0.72, 0.86755556)

- *Búsqueda de picos (muestra 2)*

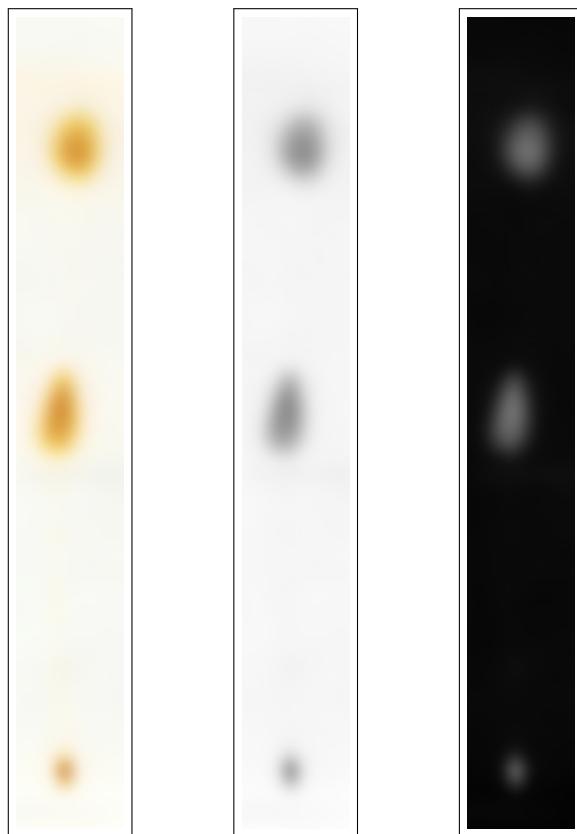


Figura 3.18: Aplicación de filtros sobre imagen de la muestra

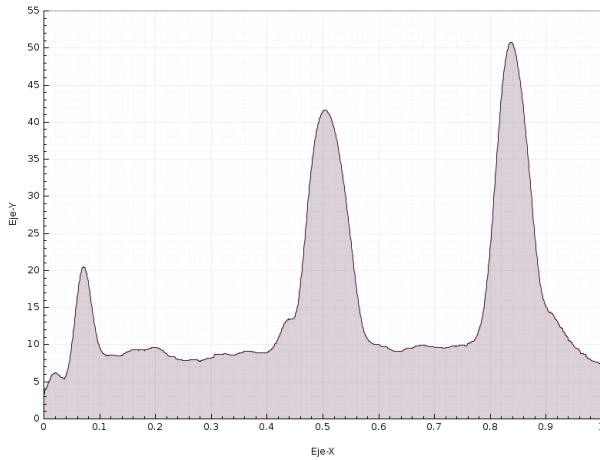


Figura 3.19: Plot de media sobre imagen con efectos en eje X.

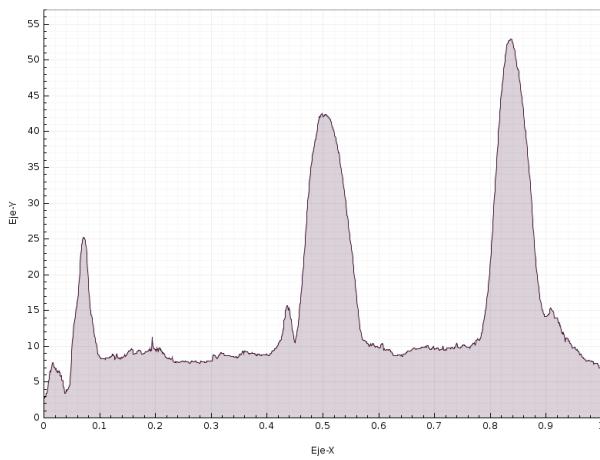


Figura 3.20: Plot de media sobre imagen con efectos (sin blur) en eje X.

Picos encontrados:

- (0.4231111, 0.5742222)
- (0.7911111, 0.97422224)

Picos encontrados (sin blur):

- (0.46222222, 0.5004445)
- (0.5004445, 0.55644447)
- (0.77511114, 0.8977778)

- *Búsqueda de picos (muestra 3)*

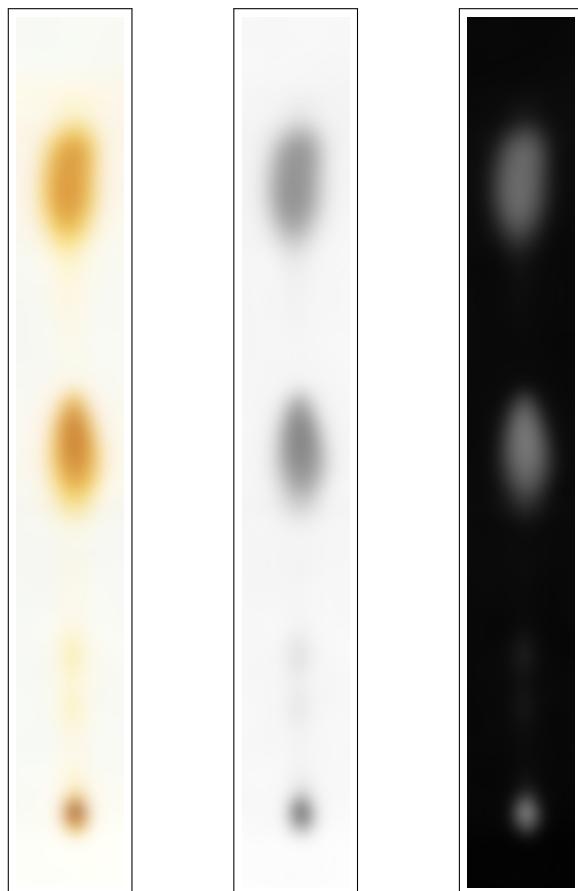


Figura 3.21: Aplicación de filtros sobre imagen de la muestra

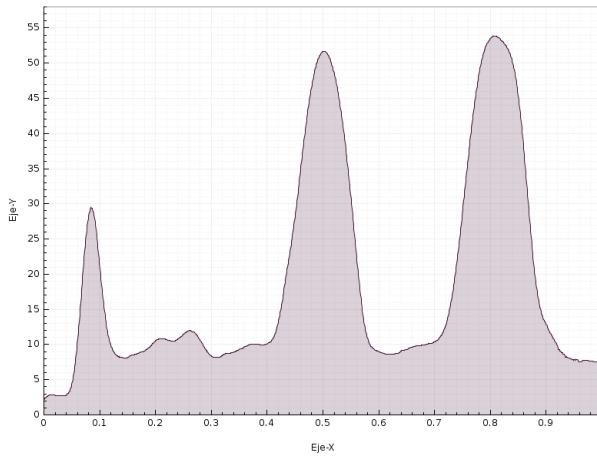


Figura 3.22: Plot de media sobre imagen con efectos en eje X.

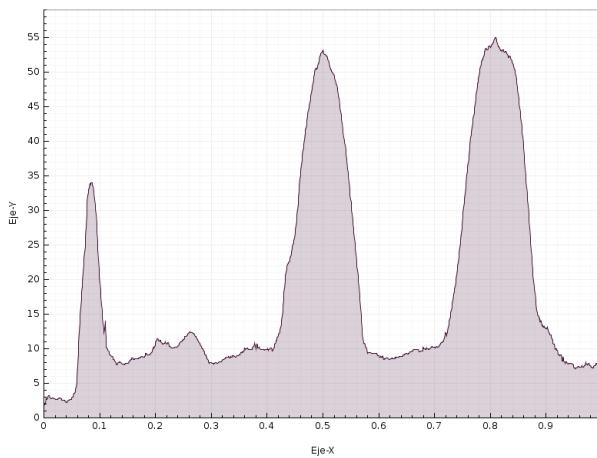


Figura 3.23: Plot de media sobre imagen con efectos (sin blur) en eje X.

Picos encontrados:

(0.4328889, 0.57688886)  
(0.7351111, 0.9208889)

Picos encontrados (sin blur):

(0.7457778, 0.8008889)

- *Búsqueda de picos (muestra 4)*

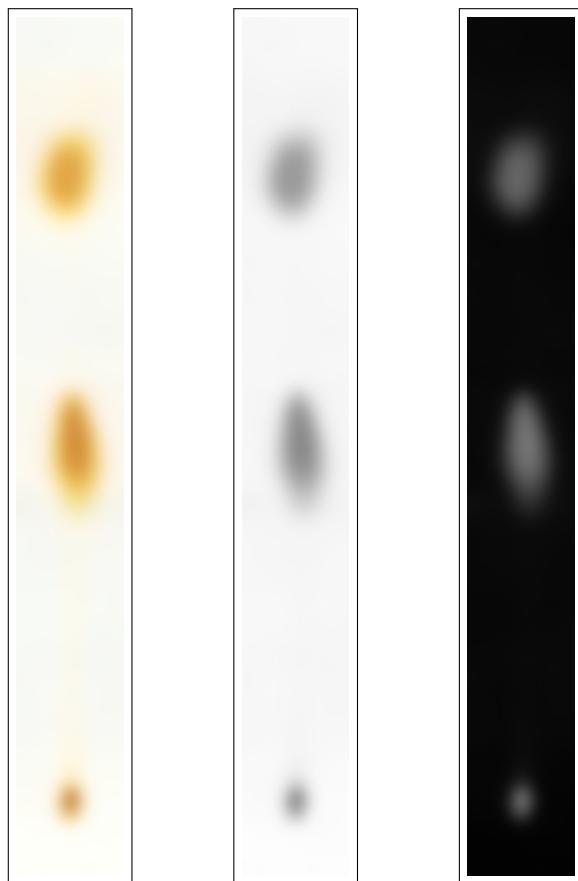


Figura 3.24: Aplicación de filtros sobre imagen de la muestra

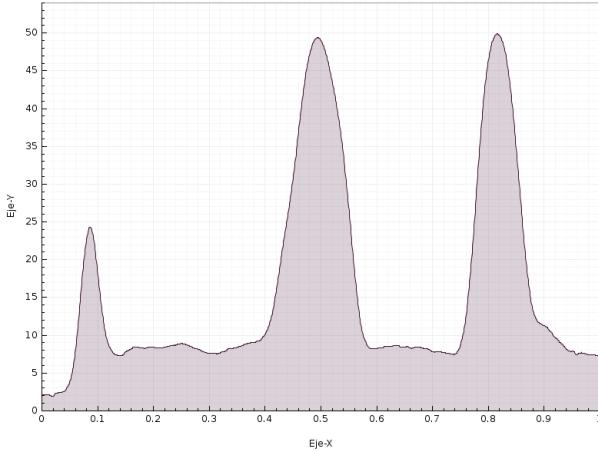


Figura 3.25: Plot de media sobre imagen con efectos en eje X.

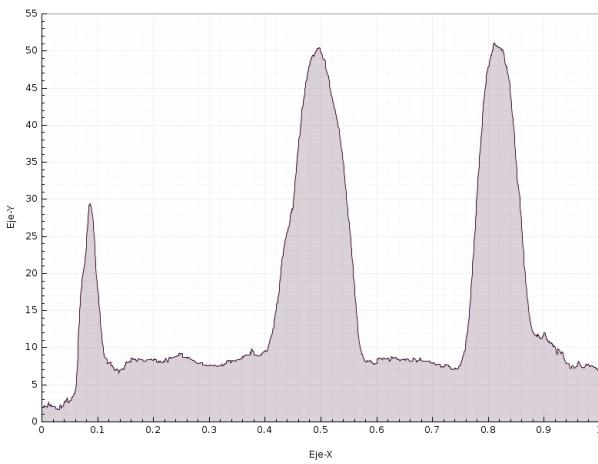


Figura 3.26: Plot de media sobre imagen con efectos (sin blur) en eje X.

Picos encontrados:

(0.424, 0.5786667)

(0.7493333, 0.8791111)

Picos encontrados (sin blur):

(0.408, 0.576)

(0.74755555, 0.88622224)

- *Búsqueda de picos (muestra 5)*

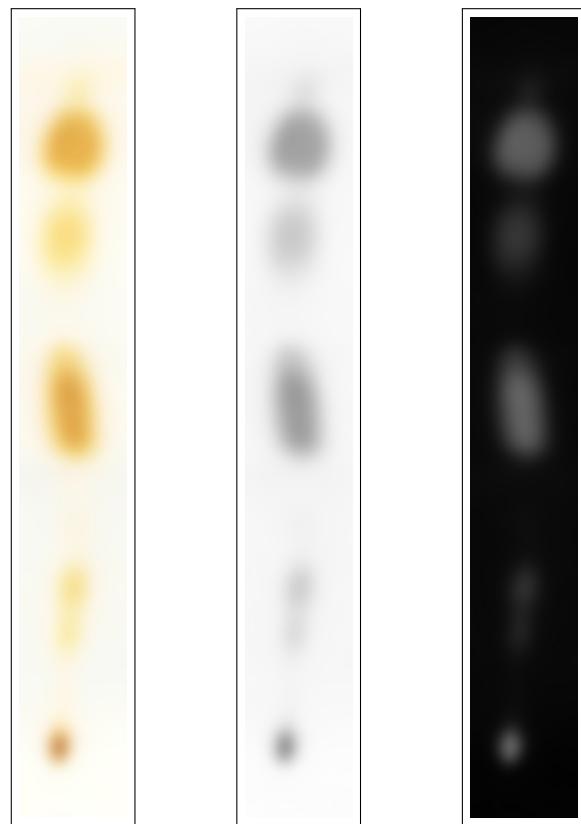


Figura 3.27: Aplicación de filtros sobre imagen de la muestra

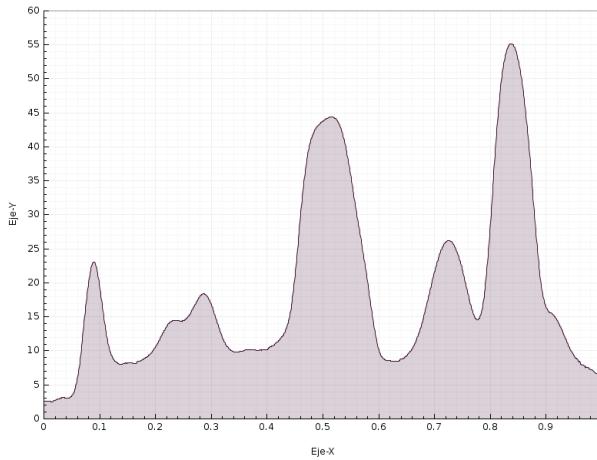


Figura 3.28: Plot de media sobre imagen con efectos en eje X.

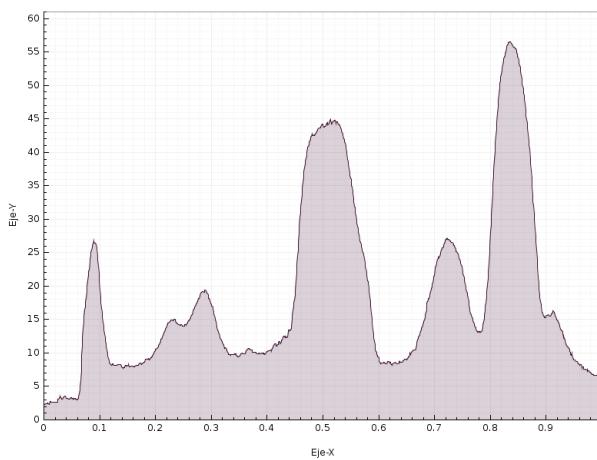


Figura 3.29: Plot de media sobre imagen con efectos (sin blur) en eje X.

Picos encontrados:

(0.4471111, 0.5982222)

(0.7831111, 0.9342222)

Picos encontrados (sin blur):

(0.7857778, 0.8968889)

- *Búsqueda de picos (muestra 6)*

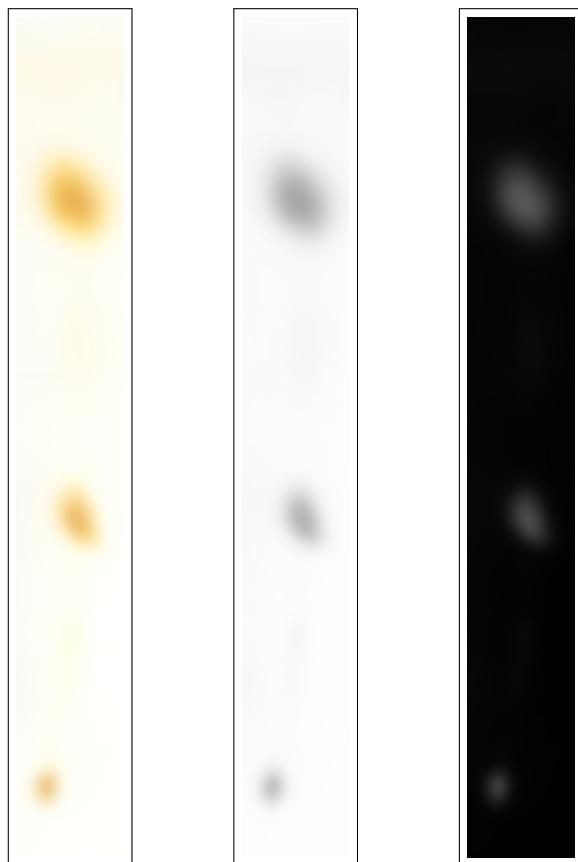


Figura 3.30: Aplicación de filtros sobre imagen de la muestra

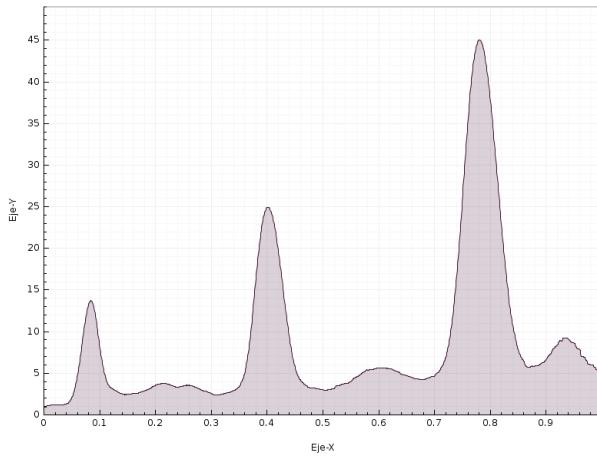


Figura 3.31: Plot de media sobre imagen con efectos en eje X.

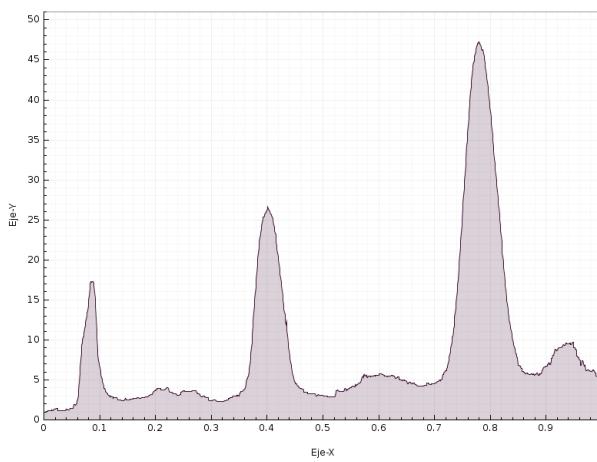


Figura 3.32: Plot de media sobre imagen con efectos (sin blur) en eje X.

Picos encontrados:  
(0.7271111, 0.8622222)

Picos encontrados (sin blur):  
(0.72977775, 0.8613333)

# Capítulo 4

## Diseño, Implementación y Prueba

En este capítulo se presentan los detalles de diseño e implementación del sistema, empezando con un simple diagrama de paquetes general hasta llegar a explicar los detalles internos de su desarrollo.

### 4.1. Diseño General del Sistema

Como diseño general del sistema se denota la agrupación de los componentes en paquetes según su comportamiento y funcionalidad.

El sistema se divide en 4 paquetes básicos:

- *jtlc.core*: contiene los componentes que forman la lógica del sistema, los modelos (Experimento, Muestra y Pico), las unidades que permiten llevar a cabo el proceso (Procesado de imágenes y Análisis de datos), la generación de reportes y el almacenamiento en disco de los experimentos realizados.
- *jtlc.assets*: engloba los recursos básicos del sistema, los elementos gráficos, las plantillas de reportes, los documentos de información al usuario y los paquetes de idiomas.
- *jtlc.view*: agrupa todos los componentes que forman las diferentes interfaces gráficas, la vista principal, los cuadros de diálogo, los paneles

de cada etapa del proceso y los componentes personalizados. También contiene los diferentes DTO (Data Transfer Object) que permiten el intercambio de información entre los componentes gráficos y los controladores.

- *jtlc.main*: incluye los componentes principales del sistema, las estructuras globales que se utilizan a lo largo de todo el sistema, el controlador principal que maneja las vistas y la tarea que se encarga de iniciar la aplicación.

En el siguiente diagrama de paquetes [7] (figura 4.1) se detallan los paquetes que componen el sistema y su asociación.

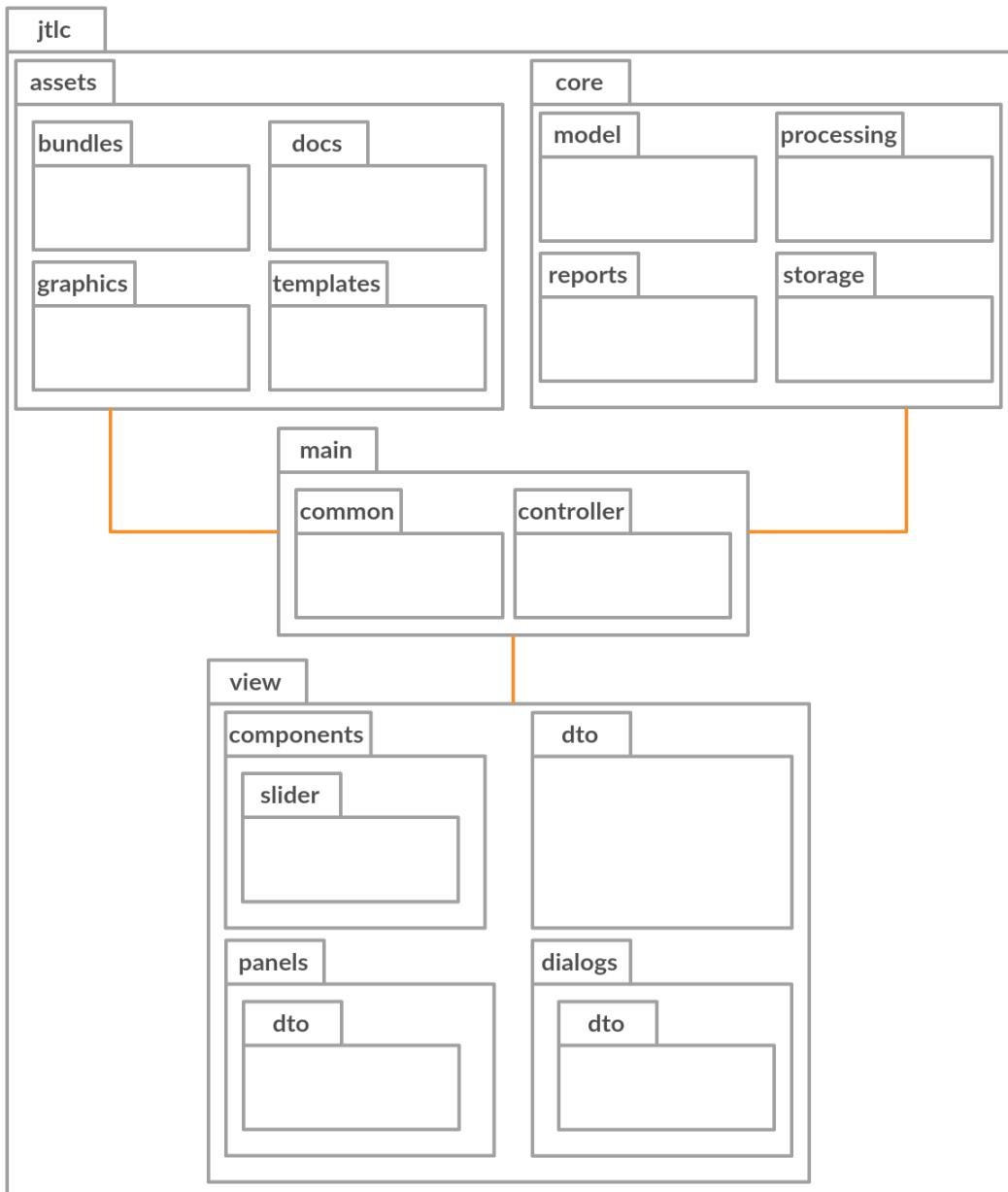


Figura 4.1: Diagrama de paquetes general del sistema.

## 4.2. Detalles de Implementación

Como se detalló anteriormente, el sistema esta implementado en *Java*, siguiendo la idea de diseño definida por el patrón arquitectural *MVC (Model-View-Controller)* destacando los 3 componentes principales Modelo, Vista y Controlador sumados a los componentes que permiten procesar las imágenes, realizar el análisis de las muestras, la generación de informes, el almacenamiento de proyectos entre otros.

El núcleo o *core* (paquete *jtlc.core*) del sistema esta compuesto por los paquetes *Model*, *Processing*, *Reports* y *Storage*. Estos módulos brindan las herramientas básicas que permiten el análisis de las muestras y la generación de reportes.

El modelo está compuesto por 3 clases: *Experiment*, *Sample* y *Peak* pertenecientes al paquete *jtlc.core.model* detallado en la figura 4.2. La clase *Experiment* define la estructura básica que posee un proyecto realizado en jTLC. Almacena los datos principales del proyecto, la imagen principal del experimento, la cual contiene las muestras a analizar; el par de puntos de cortes sobre la imagen principal, los cuales permiten recortar partes no deseadas de la imagen; el ángulo de rotación de la imagen; los ejes de inversión, que definen una combinación de ejes X - Y que permiten invertir vertical/horizontalmente la imagen; los comentarios, el título y la descripción del proyecto, la fecha de creación y análisis, entre otros. Es la estructura básica del modelo, permite almacenar, obtener y actualizar los diferentes campos/atributos del experimento. La clase *Sample* implementa las estructuras de datos necesarias para almacenar la información de una muestra en particular, como: los límites de la muestra, la imagen recortada, el frente solvente, el punto de siembra, la media muestral, la superficie total de los picos, los comentarios y una lista de *Peak*. *Peak* define un pico de la media muestral de cada *Sample*, almacena la posición, el nombre, los límites, la linea de base, la superficie (absoluta y relativa), el valor máximo dentro del pico junto a su posición y por último la altura del mismo.

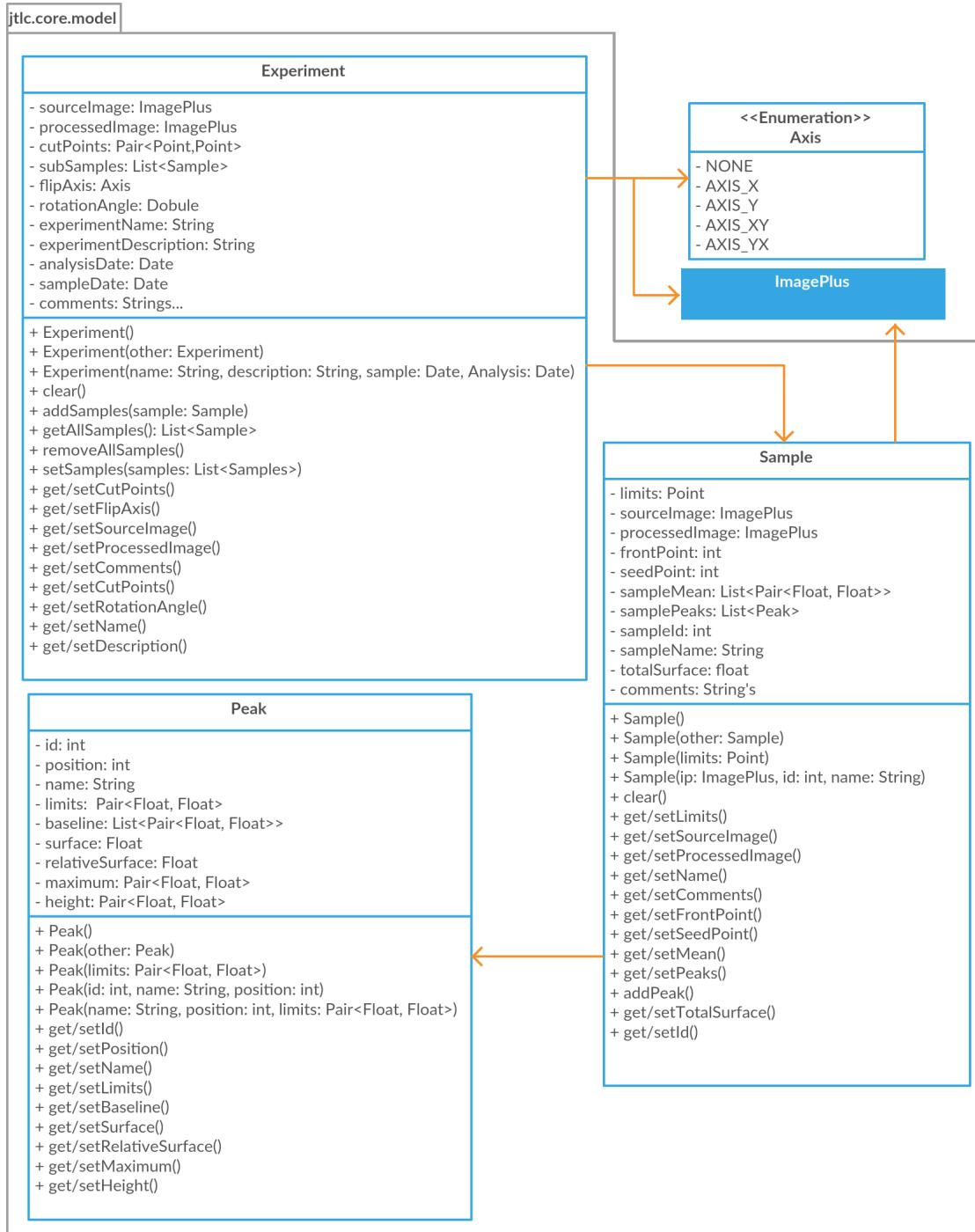


Figura 4.2: Diagrama de Clases paquete `jtlc.core.model`.

El análisis de las muestras se logra gracias a las funcionalidades brindadas por las clases *ImageProcessing* y *AnalisisProcessing*, pertenecientes al paquete *jtlc.core.processing* el cual se detalla en la figura 4.3. La clase *ImageProcessing* brinda métodos que permiten manipular fácilmente las imágenes cargadas, se basa en la librería *ImageJ* [1] para operar de manera unificada los diferentes formatos de imágenes, aplicar filtros, realizar operaciones y modificaciones, leer la matriz de píxeles, entre otros. *AnalisisProcessing* utiliza las funcionalidades provistas por la clase anterior para llevar a cabo el procesamiento de las imágenes que contienen las muestras a analizar. Facilita la búsqueda de los puntos de cortes, los límites de cada muestra individual, permite computar la media muestral, buscar picos, buscar la linea de base de cada pico, validar áreas de integración, integrar los picos, calcular los máximos locales, la altura del pico, entre otros.

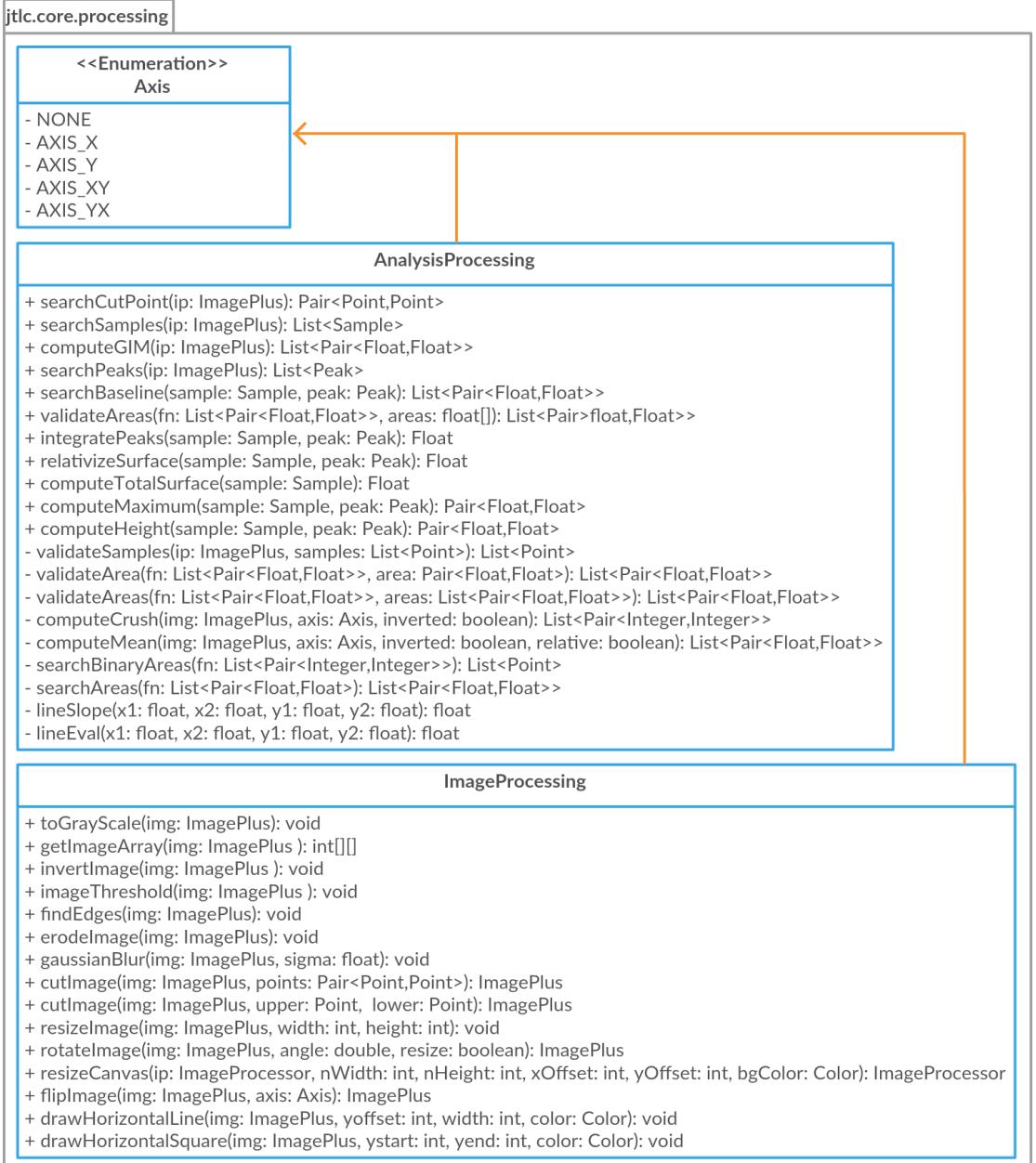


Figura 4.3: Diagrama de Clases paquete *jtlc.core.processing*.

Para la generación de reportes se utilizan las clases *Reporter* y *Template* pertenecientes al paquete *jtlc.core.reports* (figura 4.4). *Reporter* se encarga de generar la estructura básica de un reporte, se basa en un *Template* sobre el cual va llenando los diferentes campos con los resultados del proceso de análisis. Utiliza un template en formato ODT el cual es fácilmente accedido gracias a la clase *Template*, permite generar reportes en diferentes idiomas, según el idioma actual de la aplicación, así como también, permite exportar los reportes en formatos ODT, PDF, HTML, CSV y TXT.

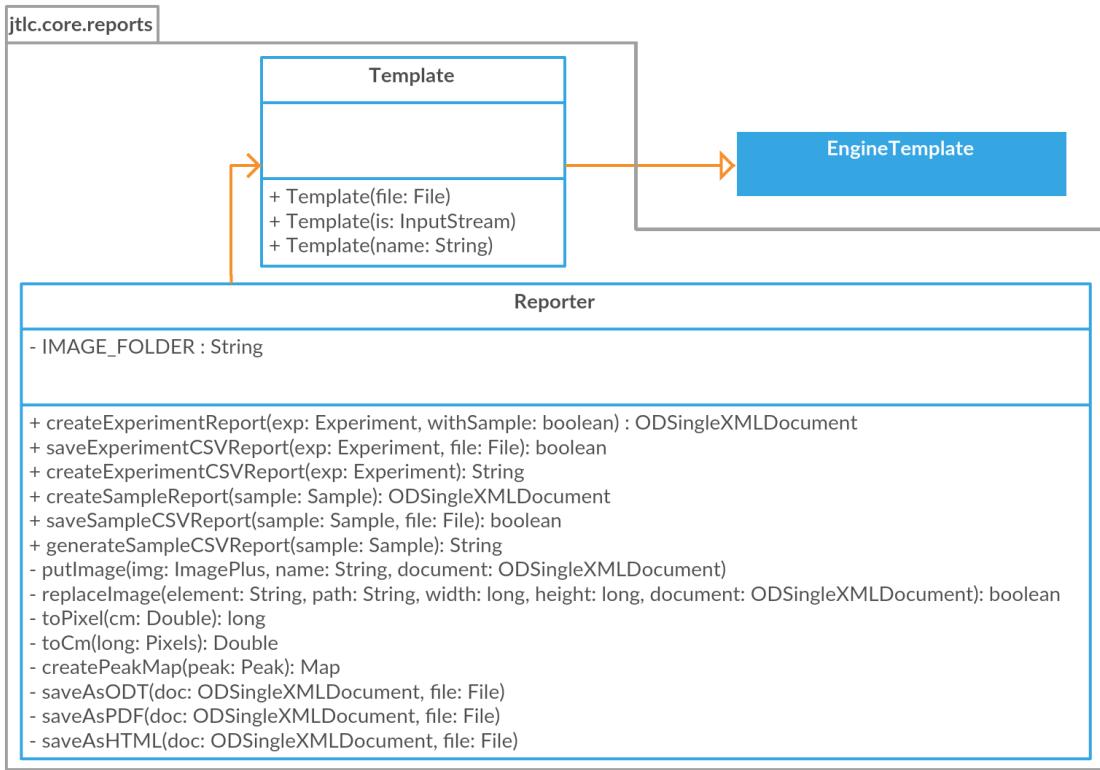


Figura 4.4: Diagrama de Clases paquete *jtlc.core.reports*.

El almacenamiento de los proyectos se logra gracias a las clases pertenecientes al paquete *jtlc.core.storage* (figura 4.5). El paquete esta compuesto por las clases *ImageStore*, *ModelSaver*, *ModelLoader*. La clase *ImageStore* brinda métodos que permiten fácilmente cargar imágenes desde el sistema de archivos, así como también permite almacenarlas en diferentes formatos, ya sea en disco o en memoria para luego ser procesadas nuevamente. La clase

*ModelSaver* permite guardar los datos del experimento actual en diferentes formatos, brinda la posibilidad de guardar datos en archivos de texto o almacenar todo el proyecto en un formato comprimido (*ZIP*), el cual incluye un archivo *XML* con los datos y resultados del experimento, las distintas imágenes procesadas y generadas durante el análisis así como la media muestral de cada muestra individual en el experimento. La clase *ModelLoader* brinda funcionalidades que permiten cargar en el sistema experimentos previamente guardados usando el módulo anterior, permite también leer una lista de proyectos dentro de una carpeta del sistema, cargando todos los proyectos que allí se encuentren. El proceso de carga inicia por leer el archivo comprimido (formato *ZIP* con extensión *.jtlc*), primero lee el archivo *XML* donde se almacenan todos los datos del proyecto y donde se encuentran las rutas relativas a las imágenes del proyecto a cargar, generando así un experimento que puede ser nuevamente cargado en el sistema para continuar su procesado o revisar los resultados previamente obtenidos.

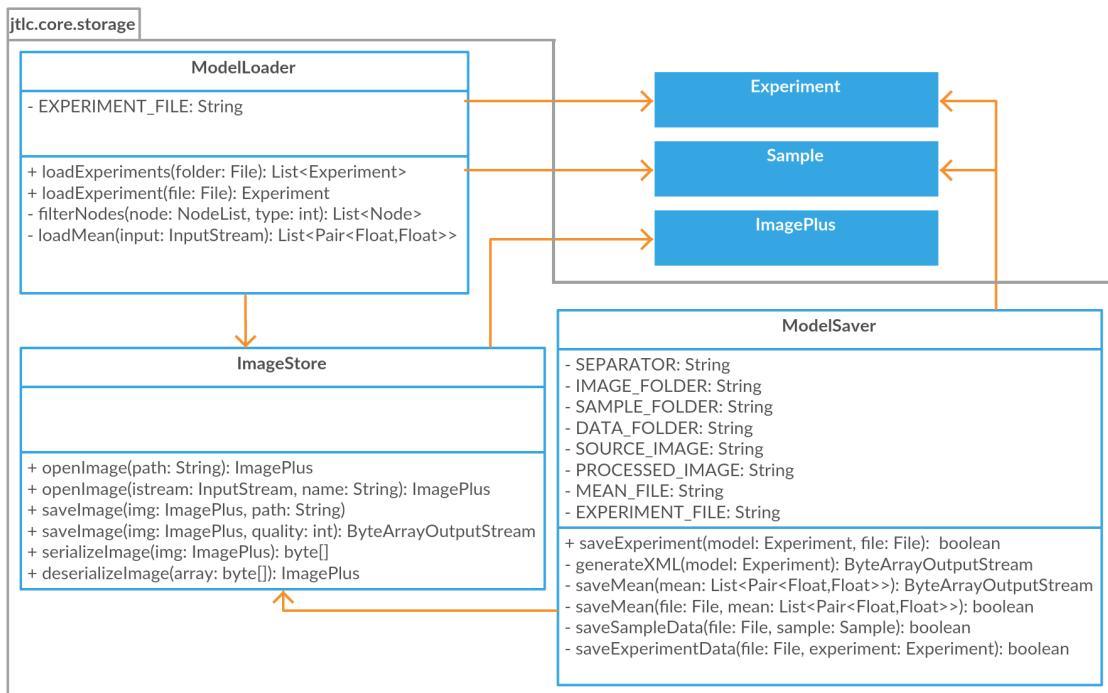


Figura 4.5: Diagrama de Clases paquete *jtlc.core.storage*.

El paquete *jtlc.assets* agrupa los recursos del sistema, contiene los elementos gráficos o íconos, los *templates* (plantillas) para la generación de reportes, los documentos de ayudas, las licencias así como los recursos de idiomas. Concentra toda su funcionalidad de la clase principal *Assets* la cual permite cargar fácilmente la iconografía del sistema, los textos de cada elemento según el idioma seleccionado, implementa la actualización de los textos durante el cambio de idiomas, permite cargar los *templates* para generar los reportes, cargar los documentos de información, entre otros. En la figura 4.6 se detalla el contenido general del paquete, con énfasis sobre la clase *Assets*

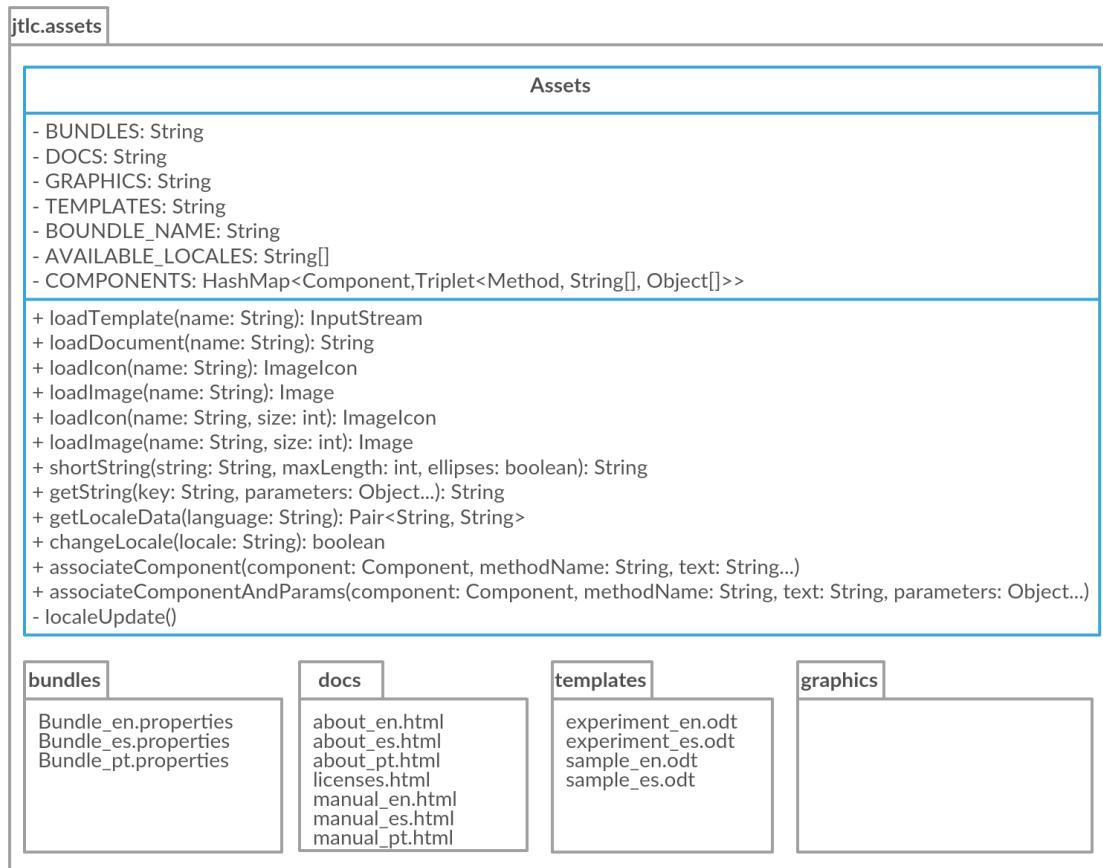


Figura 4.6: Diagrama de Clases paquete *jtlc.assets*.

La interfaz gráfica esta formada por las clases contenidas en el paquete *jtlc.view* (figura 4.8), el cual agrupa todos los componentes que forman las diferentes ventanas y paneles que permiten interactuar con el sistema. La vista principal de la aplicación esta implementada en la clase *MainView*, la cual utiliza los elementos provistos por los subpaquetes dentro de *jtlc.view* extendiendo de la clase *Observable* lo que le permite fácilmente comunicar los cambios ocurridos al controlador (*Observer*) por medio de notificaciones simples. El subpaquete *dto* define las estructuras básicas que permiten compartir información entre las vistas y los controladores (*DTO (Data Transfer Object)*), define la clase abstracta *AbstractDTO* la cual brinda una estructura común a todos los *DTO's* del sistema y permite marcar fácilmente si hubieron cambios durante el proceso de una vista o panel. El subpaquete *dialogs* contiene los diferentes cuadros de diálogos utilizados en el sistema, como así también los *DTO's* necesarios para comunicar los datos entre los diálogos y el controlador. Define una interfaz *IDialog* común a todos los cuadros de diálogo que permite obtener los resultados de manera simple y transparente. En la figura 4.7 se resume el paquete de forma general. Las diferentes ventanas o diálogos son:

- *ProjectDialog*: Este diálogo se utiliza al momento de crear un nuevo proyecto, permite introducir los datos básicos del experimento. Da como resultado un *ProjectDTO* conteniendo los datos ingresados.
- *SettingsDialog*: Permite visualizar y establecer las configuraciones del sistema. Toma como entrada un *SettingsDTO* con la configuración actual y da como resultado un objeto del mismo tipo contiendo la nueva configuración en el caso que se registren cambios en la misma.
- *InfoDialog*: Muestra la información del proyecto actual, permite cambiar los datos (editar) del mismo. Su entrada es un *InfoDTO* con los datos actuales del proyecto y su resultado es un objeto del mismo tipo con los nuevos datos en el caso que se realice algún cambio.
- *ImageExportDialog*: Permite exportar imágenes del experimento al sistema de archivos, así como también redimensionar la imagen al momento del guardado. Su entrada es un objeto *ImageExportDTO* conteniendo información sobre la imagen a exportar (tamaño, vista previa, etc) y su resultado es un objeto del mismo tipo conteniendo el tamaño de exportación deseado.

- *TextPanelDialog*: Este cuadro de diálogo se encarga de mostrar un área de texto plano o en formato *HTML*. Su entradas son: el texto a mostrar, el título de la ventana y el ícono de la misma. No da como resultado ningún valor.

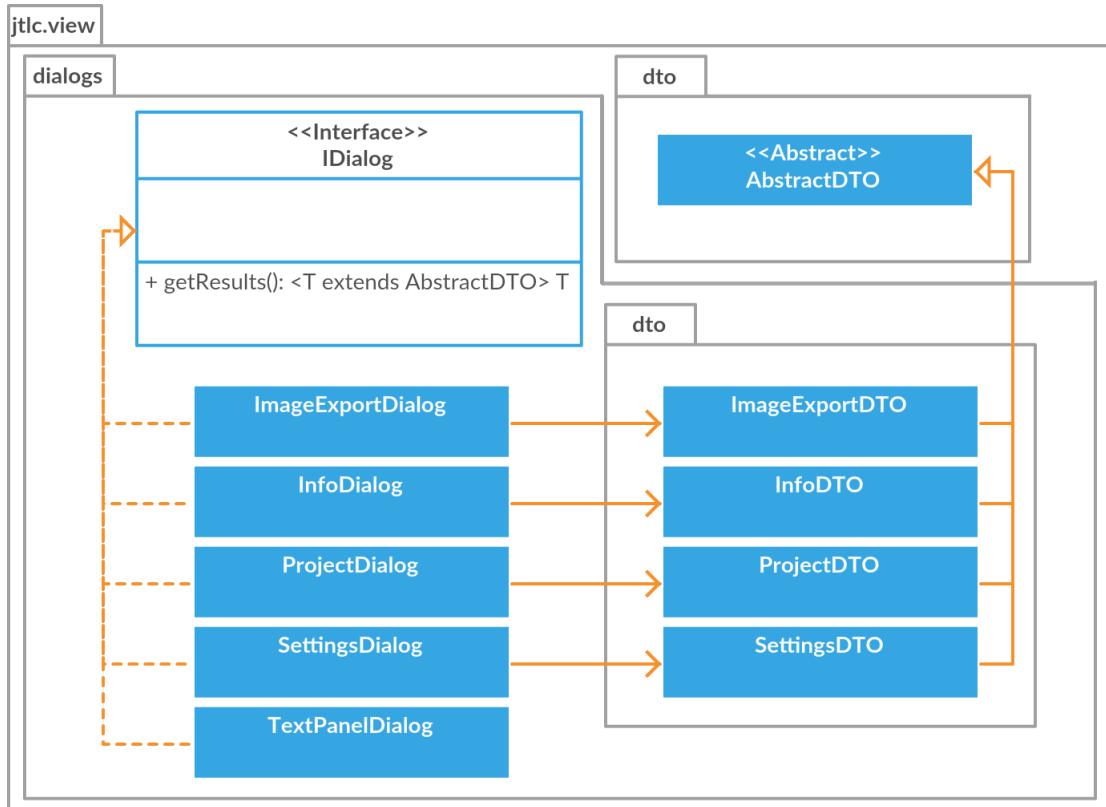


Figura 4.7: Diagrama de Clases paquete *jtlc.view.dialogs*.

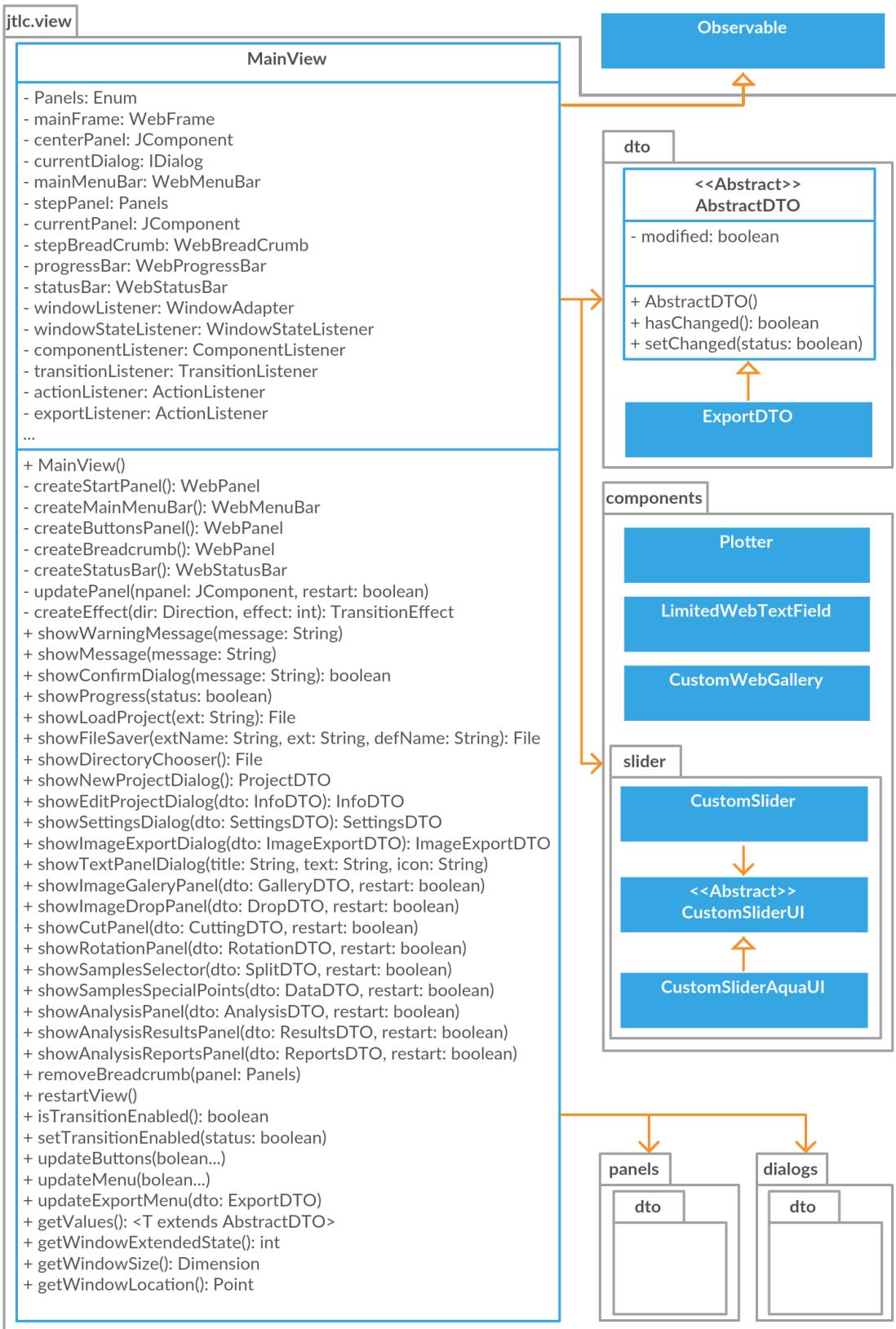


Figura 4.8: Diagrama de Clases paquete `jtlc.view`.

El subpaquete *panels* contiene los paneles principales del sistema que se utilizan durante el análisis de las muestras. Define una interfaz común a todos los paneles que facilita la interacción con los controladores. Dentro se define el subpaquete *dto* el cual contiene todos los *dto's* que permiten la interacción con los diferentes paneles, los cuales se basan en la clase *AbstractDTO* para generalizar y facilitar su uso. En la figura 4.9 se resume el paquete en forma general. Los diferentes paneles son:

- *GalleryPanel*: Es el encargado de mostrar la galería de proyectos al momento de explorar un directorio. Utiliza la clase *CustomWebGallery* en su implementación y así permitir elegir un proyecto para continuar a través de una galería de imágenes. Su entrada es un *GalleryDTO* conteniendo la lista de posibles experimentos a cargar y su resultado es el mismo objeto conteniendo la información sobre el proyecto elegido.
- *DropPanel*: Este panel permite cargar la imagen inicial del proyecto, es el primer panel visible durante el proceso del experimento. Implementa un área *Drag & Drop* que facilita la carga de imágenes con un simple arrastrar y soltar, también permite explorar el sistema de archivos buscando manualmente la imagen deseada a través de un selector de archivos estándar. Su entrada es un *DropDTO* conteniendo la imagen inicial a mostrar y los comentarios previamente guardados, su resultado es un objeto del mismo tipo conteniendo la imagen cargada y los nuevos comentarios ingresados.
- *CutPanel*: Permite recortar áreas de la imagen del experimento. Muestra la imagen de fondo y dos guías (una horizontal y otra vertical) que definen un recuadro sobre la misma delimitando el área final de la imagen recortada. Su entrada es un *CuttingDTO* conteniendo la imagen a procesar y un par de puntos que definen el área previamente establecida por el asistente de procesado. Su resultado es un objeto del mismo tipo conteniendo la nueva área seleccionada en caso de que haya ocurrido algún cambio, ademas posee información sobre los comentarios realizados en este proceso.
- *RotationPanel*: El panel de rotación permite realizar transformaciones básicas sobre la imagen del experimento. Brinda la posibilidad de rotar la imagen de manera arbitraria cualquier cantidad de grados, invertir la imagen tanto sea horizontal como verticalmente o realizar rotaciones

a escuadra. Su entrada es un *RotationDTO* conteniendo la imagen a procesar, los datos de rotación e inversión previamente establecidos, su resultado es un objeto del mismo tipo conteniendo las nuevas transformaciones aplicadas de haber ocurrido junto a los comentarios realizados en esta etapa del proceso.

- *SplitPanel*: Es el encargado de separar las muestras en imágenes individuales. Permite seleccionar manualmente a través de líneas de guía los puntos donde empieza y termina cada muestra a analizar. Su entrada es un *SplitDTO* conteniendo la imagen a procesar y los puntos de separación previamente calculados por el asistente de proceso, su resultado es un objeto del mismo tipo conteniendo los nuevos puntos de separación en el caso que el usuario haya realizado algún cambio junto a los comentarios realizados en este proceso.
- *DataPanel*: Este panel permite ingresar los datos específicos de cada muestra individual. Permite seleccionar el punto de siembra y el frente solvente de cada una, de manera global, es decir, el mismo para todas las muestras, o de manera individual. A su vez permite establecer el nombre de la muestra, comentarios individuales para cada una de ellas o comentarios globales a toda la etapa del proceso. Su entrada es un *DataDTO* conteniendo las imágenes de cada muestra individual, y los datos previamente cargados de cada una de ellas. Su resultado es un objeto del mismo tipo conteniendo los nuevos datos cargados de ser necesario.
- *AnalysisPanel*: En el panel de análisis se realiza el análisis de cada muestra a partir del gráfico de la media muestral obtenida durante el proceso. Permite seleccionar los picos individuales mediante un *Slider* sobre el gráfico o sobre la imagen fuente. También permite visualizar la comparación de la media de cada muestra y realizar comentarios individuales a cada una de ellas. La entrada de este panel es un *AnalysisDTO* conteniendo la media de cada muestra, los límites de cada pico calculados por el asistente de proceso, los comentarios previamente realizados y los datos de cada muestra. Su resultado es un objeto del mismo tipo conteniendo los nuevos comentarios y los nuevos límites de cada pico.
- *ResultsPanel*: En este panel se muestran los resultados obtenidos durante el proceso de análisis junto a los datos de cada muestra. Se de-

tallan los datos de cada pico, su superficie total y relativa, los límites, la altura, el valor máximo, la línea de base, entre otros. Su entrada es un *ResultsDTO* contenido los resultados de cada muestra y sus datos, su salida es un objeto del mismo tipo contenido los comentarios realizados en esta etapa. También permite establecer el nombre a cada pico individualmente para cada muestra procesada.

- *ReportsPanel*: El panel de Reportes muestra los reportes con los datos de las muestras y los resultados del análisis así como también los datos del experimento. Su entrada es un *ReportsDTO* contenido los datos a mostrar. No produce ningún resultado específico.

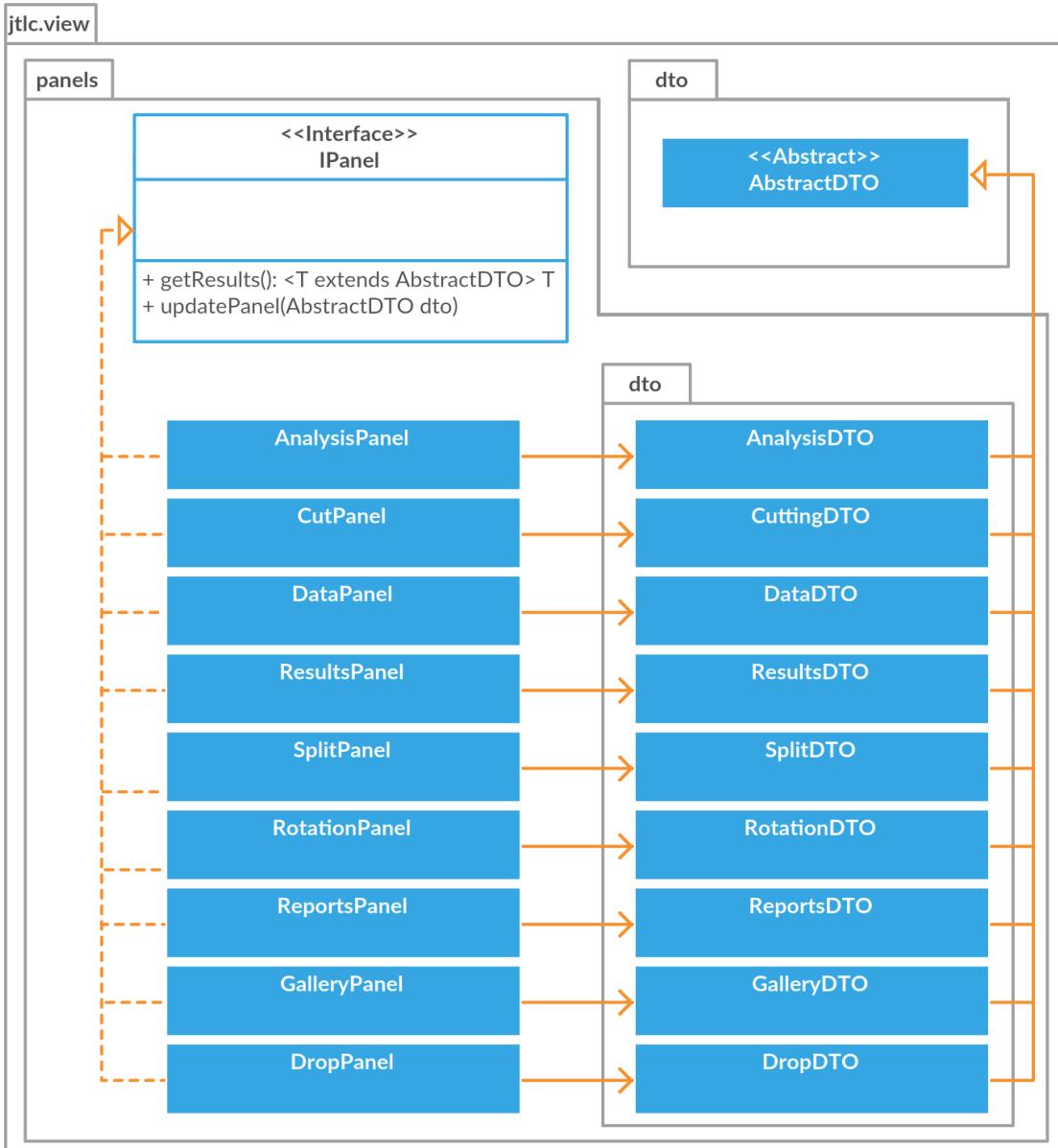
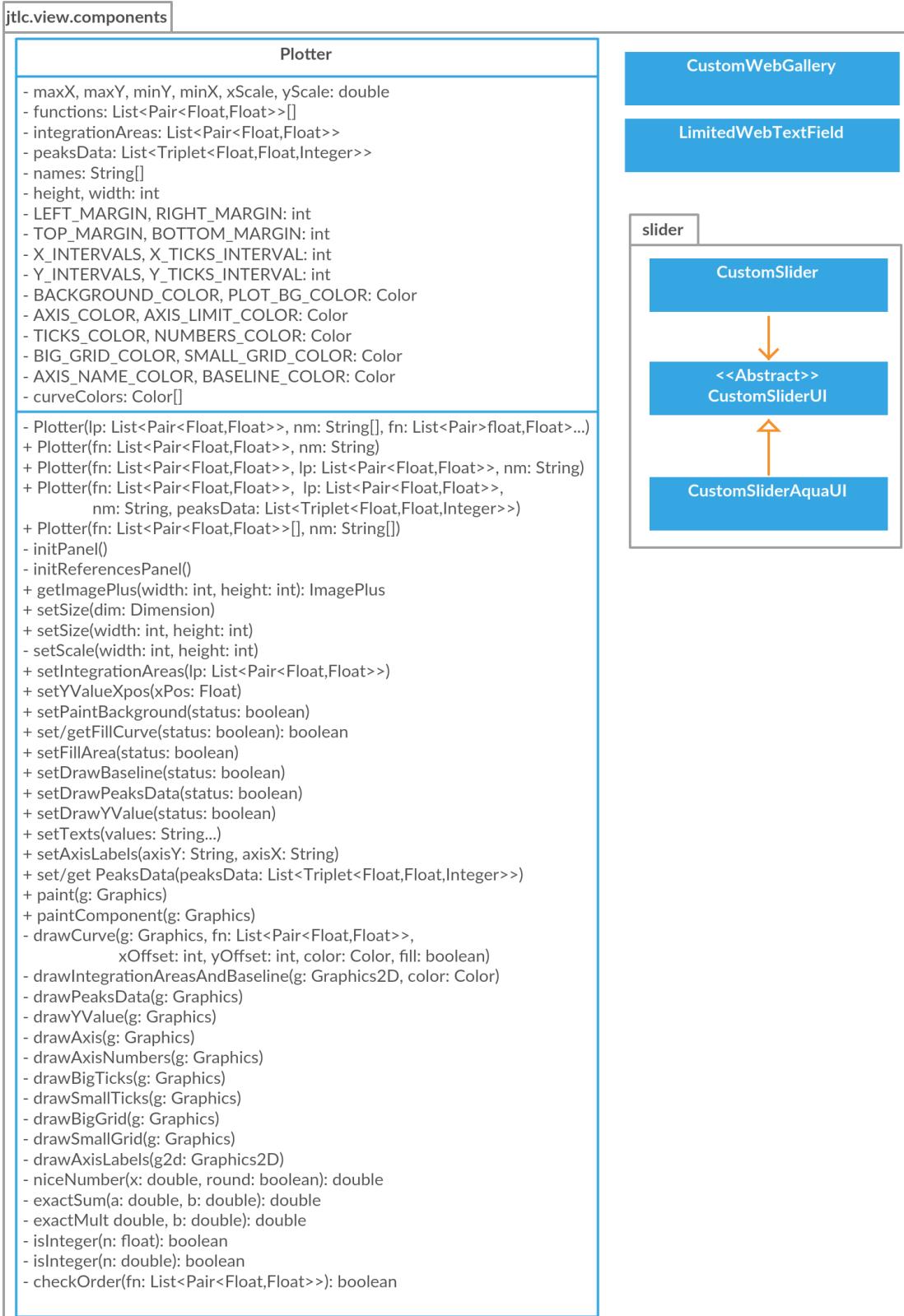


Figura 4.9: Diagrama de Clases paquete *jtlc.view.panels*.

Por último el paquete *components* contiene los componentes personalizados del sistema que se utilizan a lo largo de diferentes paneles, principalmente los *Sliders* que permiten delimitar áreas y el *Plotter* que se encargar de graficar la curva resultante de la media muestral de cada muestra analizada junto a la información básica de los picos, la línea de base y el área de integración. Los *Sliders* están implementados por la clase *CustomSlider* la cual se basa en la librería *MultiThumbSlider* parte del paquete *JavaGraphics* (<https://javagraphics.java.net/>) provista por *Jeremy Wood* agregando detalles personalizados que brindan funcionalidad extra como las líneas de guía, emparejar dos *thumbs*, dibujar el área de selección o recuadro entre dos *thumbs*, etc. El *Plotter* desarrolla un *JComponent* personalizado el cual a partir de gráficos 2D se encarga de dibujar o realizar un plot de un conjunto de valores en X e Y. En esta aplicación se utiliza para dibujar la media muestral de cada muestra junto al área de integración de cada pico, su línea de base y otros datos relacionados. En la figura 4.10 se detalla la implementación del paquete.

Figura 4.10: Diagrama de Clases paquete `jtlc.view.panels`.

El paquete *jtlc.main* (figura 4.11) contiene los principales componentes que permiten inicializar y controlar la ejecución de la aplicación, se subdivide en tres paquetes *main*, *common* y *controller*. El primero contiene la clase principal que se encarga de la ejecución del sistema, inicializar las vistas y crear el controlador principal de la aplicación. El subpaquete *common* contiene las estructuras básicas que son utilizadas a lo largo de todo el sistema, la clase *Point* que implementa un par de *Integers*, la clase *Pair* que implementa un par genérico de dos elementos, la clase *Triplet* la cual extiende *Pair* agregando un elemento mas al conjunto y por último la clase *Settings* que permite guardar y cargar la configuración previa del sistema o crear la configuración por defecto en el caso que sea la primera ejecución, usando un patrón *Singleton* para su cometido. El subpaquete *controller* (figura 4.12) contiene las clases que implementan el control de la aplicación: *Action* y *Controller*. La primera define una Anotación *Java* que permite identificar una acción a ejecutar dentro del controlador a través de un *String* de comando. El *Controller* se encarga de controlar el progreso de la aplicación durante el proceso de análisis así como también manejar la acciones de los diferentes cuadros de diálogo. Implementa un *Observer* el cual esta escuchando a los cambios o notificaciones producidas por las vistas. Define un método *update* el cual es ejecutado cuando ocurre un cambio o en respuesta a una determinada acción. Este método se encarga de derivar la responsabilidad de la respuesta a los diferentes servicios brindados por la clase a través de las anotaciones definidas en cada uno de ellos, busca entre todos los métodos definidos en la clase cual es el que corresponde ejecutar al comando requerido según la acción definida por la anotación *Action* del método. Para mejorar en eficiencia, al momento de inicializar el controlador, se crea también un *HashMap* contenido la acción definida por la anotación del método como clave y una referencia al mismo método como valor, de esta forma al momento de responder a una determinada acción solo se debe obtener a partir del mapa cual es el método que corresponde a dicha acción. Las acciones definidas por el controlador son:

- *NEW\_PROJECT*: Ejecuta el método *newProject* el cual muestra el cuadro de diálogo para crear un nuevo proyecto, inicializa las estructuras del nuevo proyecto con los datos ingresados, muestra el primer paso del proceso.
- *LOAD\_PROJECT*: Ejecuta el método *loadProject* el cual muestra el cuadro de diálogo para cargar un proyecto previamente guardado, con-

tinua cargando el proyecto seleccionado mostrando el primer paso del proceso.

- ***EXPLORE\_PROJECTS***: Ejecuta el método *exploreProjects* mostrando el cuadro de diálogo de selección de directorio, muestra la galería de proyectos contenidos en el directorio.
- ***SAVE\_AS\_PROJECT***: Ejecuta el método *saveProject*, muestra el cuadro de diálogo para seleccionar o crear un archivo, escribe los datos del experimento actual al archivo seleccionado usando los servicios brindados por *ModelSaver*.
- ***SAVE\_PROJECT***: Ejecuta el método *overwriteProject*, disponible en el caso que exista un archivo previo del proyecto actual. Sobrescribe el archivo con los nuevos datos del proyecto.
- ***EXIT\_SYSTEM***: Ejecuta el método *exitSystem*, el cual finaliza la instancia actual de la aplicación.
- ***EDIT\_PROJECT***: Ejecuta el método *editProject* el cual muestra el cuadro de diálogo que permite editar los datos básicos del proyecto y actualiza el proyecto actual con la nueva información.
- ***EDIT\_SETTINGS***: Ejecuta el método *editSettings* el cual muestra el cuadro de diálogo que permite cambiar la configuración actual del sistema, para luego ser aplicada a la instancia de ejecución corriente.
- ***NEXT\_STEP***: Ejecuta el método *nextStep* el cual según el paso actual en el proceso de análisis avanza hacia el siguiente paso calculando los datos necesarios y actualizando la vista acorde al paso actual. Los casos de ejecución son:
  - ***EXPLORE\_PROJECTS***: procesa los resultados de la exploración de proyectos (*processExploreProjects*), actualiza la vista al paso de carga de imagen del experimento.
  - ***LOAD\_IMAGE***: procesa la carga de imagen (*processImageDrop*) y calcula los puntos de recortes asistidos, actualiza la vista al paso de recorte de la imagen del experimento.

- *CUT\_IMAGE*: procesa el recorte de imagen (*processImageCutting*), actualiza la vista al paso de rotación de la imagen del experimento.
- *ROTATE\_IMAGE*: procesa la rotación de la imagen (*processImageRotation*) y calcula los puntos de separación de muestras automáticos, actualiza la vista al paso de selección de muestras individuales.
- *SAMPLES\_SELECT*: procesa la selección de muestras individuales (*processSamplesSplit*), actualiza la vista al paso de carga de los puntos especiales de cada muestra en el experimento.
- *SPECIAL\_POINTS*: procesa la selección de puntos especiales de la muestra (*processSpecialPointsSelection*) y calcula los picos de cada muestra junto a la línea de base de cada uno de ellos, actualiza la vista al paso de análisis de las muestras.
- *ANALIZE\_SAMPLES*: procesa la selección de picos de cada muestra (*processSamplesAnalysis*) y calcula el área de cada pico, el área total, máximos, altura, entre otros para cada muestra, actualiza la vista al paso de análisis de los resultados del análisis de las muestras.
- *SAMPLES\_ANALYSIS\_RESULTS*: procesa los resultados del análisis de las muestras y los picos (*processSamplesAnalysisResults*), actualiza la vista al último paso donde se visualizan los reportes del proceso de análisis del experimento.
- *PREV\_STEP*: Ejecuta el método *previousStep* el cual se encarga de retroceder en uno el paso de análisis actual, deshaciendo los cambios producidos en el último paso de ser necesario.
- *RESTART\_STEP*: Ejecuta el método *restartStep* el cual reinicializa el paso actual con los valores por defecto del experimento o por los previamente calculados por el asistente del proceso o por un análisis anterior en el caso de estar continuando con un proyecto previo.
- *ABOUT\_JTL*: Ejecuta el método *showAbout* el cual muestra el cuadro de diálogo con la información del sistema.
- *SHOW\_HELP*: Ejecuta el método *showHelp* el cual muestra el cuadro de diálogo con la ayuda de uso del sistema.

- **LICENSES**: Ejecuta el método *showLicenses* el cual muestra el cuadro de diálogo con las licencias de uso del sistema.
- **EXPORT\_DATA**: Ejecuta el método *export* el cual a partir de los parámetros de entrada identifica los datos que el usuario desea exportar, genera dicha información y la escribe en un archivo en disco según el fichero seleccionado por el usuario a través del diálogo de selección de archivos. Se basa en los métodos *exportData*, *exportMean*, *exportImage* y *exportReport* para lograr su cometido. *exportData* permite exportar los datos básicos del proyecto a un archivo de texto plano. *exportMean* permite exportar la media muestral de una muestra específica a un archivo de texto plano. *exportImage* permite exportar las diferentes imágenes del proyecto a un archivo de imagen. *exportReport* permite exportar los diferentes reportes del análisis de las muestras en archivos de tipo *ODT*, *PDF* y *HTML*.

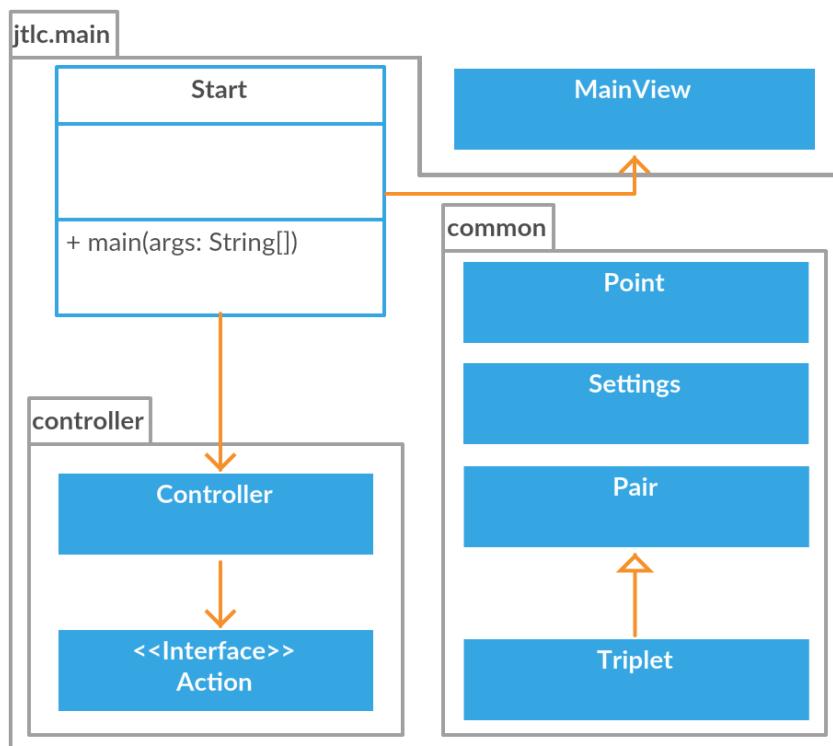


Figura 4.11: Diagrama de Clases paquete *jtlc.main*.

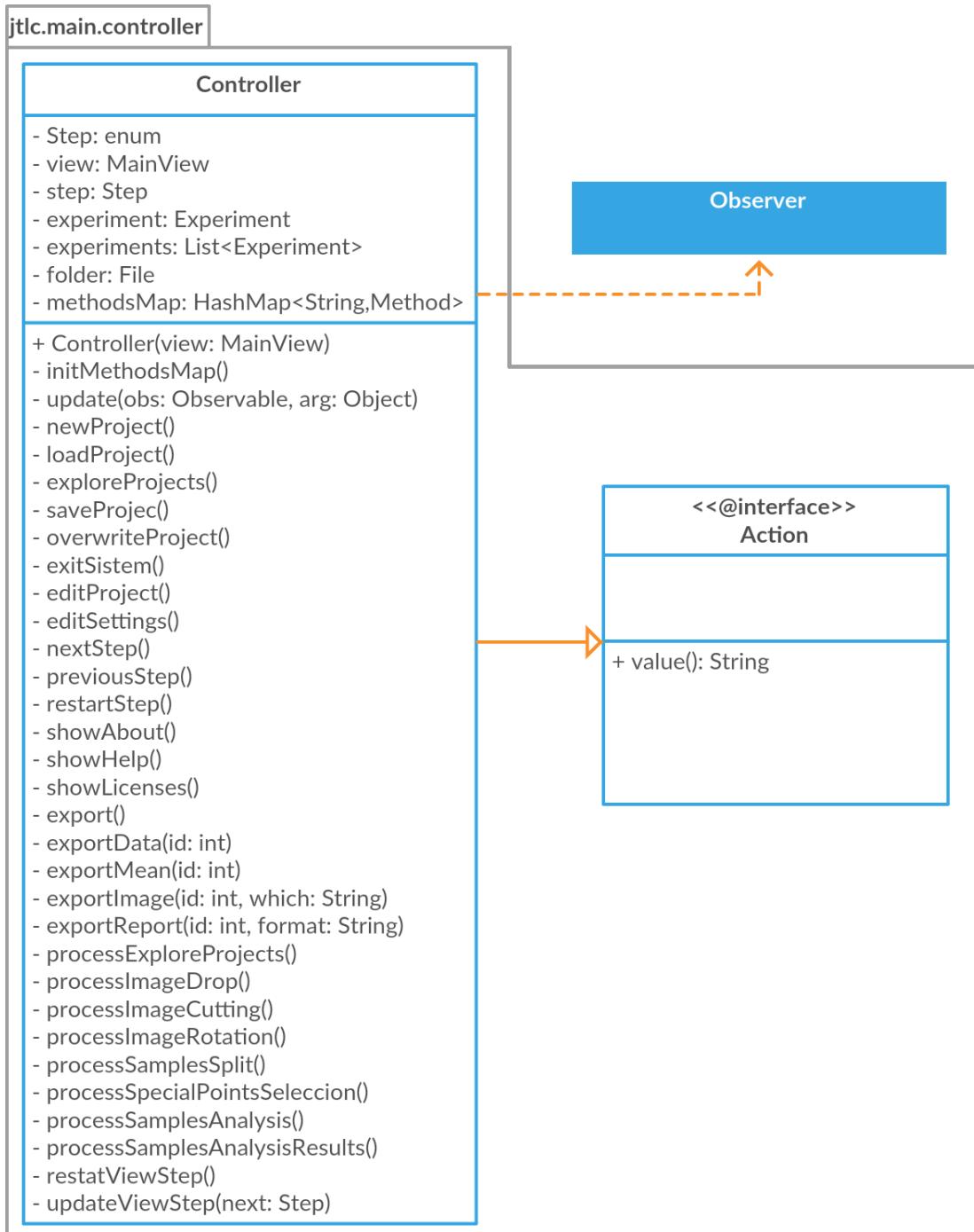


Figura 4.12: Diagrama de Clases paquete *jtlc.main.controller*.

### 4.3. Prueba

La prueba del sistema no se realizó siguiendo ningún mecanismo sistemático de testing (como por ejemplo, testing de unidad usando JUnit), sólo se realizaron pruebas *Ad hoc* sin planificación ni documentación. Estas pruebas fueron llevadas a cabo corriendo el sistema, probando diferentes escenarios observando el comportamiento del mismo y contrastándolo con el esperado. En este caso se realizaron análisis de muestras que fueron previamente analizadas usando otro *software* y se compararon los resultados obtenidos con nuestro sistema contra los resultados obtenidos a partir de otro *software* de análisis cromatográfico observando resultados bastante similares entre ambos con una tolerancia al error aceptable. Dada la naturaleza gráfica del problema y el asistente del proceso de análisis, gran parte de las pruebas estuvieron concentradas en las funcionalidades provistas por las vistas y paneles los cuales fueron desarrolladas testeando su comportamiento y resultados, principalmente la alineación de los *sliders* con las imágenes y el *plotter* para garantizar la precisión de los resultados y de las áreas seleccionadas. Algunos problemas que surgieron a partir de las pruebas fueron de control de datos faltantes, por ejemplo, al momento de seleccionar las muestras individuales ocurría un error si se intentaba continuar el proceso sin haber seleccionado ninguna muestra, el cual fue fácilmente solucionado comprobando que la cantidad de muestras seleccionadas sea mayor a 0, mostrando una advertencia en caso de no ser así sin poder continuar el proceso.

A partir de una versión estable del sistema se desarrollo una etapa importante de las pruebas que consistieron en la aprobación del funcionamiento del sistema por parte del experto en el área *Pablo Rossi* el cual junto a su equipo de trabajo actualmente utilizan la aplicación en un ambiente de producción, ellos son los principales usuarios que reportaran posibles errores en el sistema o mejoras a desarrollar.

# Capítulo 5

## Conclusión y Trabajos Futuros

En este trabajo se ha llevado a cabo el desarrollo de un Analizador cromatográfico para muestras de *Biodiésel*. Este sistema permite analizar varias muestras procesadas del producto contenidas en una única imagen al mismo tiempo, brindando herramientas básicas que permiten recortar y rotar la imagen, seleccionar las muestras individualmente de manera sencilla y visualmente amigable, definir los datos individuales de cada muestra (punto de siembra y frente solvente), generar la media muestral y analizar los picos de cada una de ellas obteniendo los datos superficiales absolutos y relativos de cada uno. Como extras el sistema permite llevar la traza del experimento desarrollado, guardar y cargar muestras previamente analizadas, exportar los datos de las muestras para ser analizados usando otro *software*, entre otros. El desarrollo de este sistema implicó la utilización, integración y aplicación de diversos conceptos estudiados durante la carrera como patrones arquitecturales y de diseño, la utilización y manejo de diversas estructuras de datos, el manejo de archivos, análisis matemáticos y estadísticos, integración numérica y regla del trapecio, entre otros conceptos. En particular el desarrollo de este sistema implicó el análisis de imágenes para lo cual se aplicaron diversos conceptos básicos de la computación gráfica y el procesado de imágenes.

Si bien, el desarrollo de este trabajo cumplió con las expectativas iniciales, durante su ejecución se hicieron mas que evidentes ciertas mejoras a incluir en un trabajo futuro. Algunas de estas serían:

- *Subpantalla de carga:* ciertos procesos del sistema pueden demorar algunos segundos dependiendo del ordenador y del tamaño de la imagen y durante esos segundos no existe indicador que resalte que el software

está realizando un proceso, por lo que se puede generar cierta confusión en el usuario. Naturalmente se suele utilizar una pantalla que recubre la vista y se indica que hay una tarea en proceso, una vez finalizada, se vuelve a mostrar la vista.

# Bibliografía

- [1] ImageJ, <https://imagej.nih.gov/ij/>
- [2] Java, <http://www.java.com>
- [3] NetBeans, <https://netbeans.org/>
- [4] JUnit, <http://junit.org/>
- [5] Bertrand Meyer (Abril 13, 1997). Object-Oriented Software Construction. Prentice Hall, 2nd edition.
- [6] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal (Agosto 16, 1996). Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Siemens AG, Alemania.
- [7] Grady Booch, James Rumbaugh, Ivar Jacobson, El lenguaje unificado de modelado, 2da edicion.