Universidad de las Ciencias Informáticas

Facultad 6

Título: Software para el Procesamiento de Imágenes de Microarray.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Servando Díaz Travieso

Adair Suárez Suárez

Tutores: Ing. Yunier René Pérez Valdés

Lic. Alexei Díaz Pérez.

Ciudad de la Habana, Julio de 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firma la presente a los __ días del mes de junio del año 2008.

Servando Díaz Travieso	Adair Suárez Suárez	Lic. Alexei Díaz Pérez
Autor	Autor	Tutor

Yunier René Pérez Valdés

Tutor

DATOS DE CONTACTO

TUTOR:

Lic. Alexei Díaz Pérez, Pinar del Río (1980). Licenciado en Física Nuclear en el INSTEC en el 2004. Actualmente profesor de la Universidad de las Ciencias Informáticas de la Facultad 6 en la que imparte clases de dos asignaturas, jefe del colectivo de Física de dicha facultad, líder del Proyecto Imágenes Microarray.

E-mail: alexeidp@uci.cu

Ing. Yunier René Pérez Valdés, Ciudad de la Habana (1982). Ingeniero en Ciencias Informáticas, graduado en la Universidad de las Ciencias Informáticas en el año 2007 con segundo perfil en Bioinformática. Actualmente profesor de las asignaturas de Introducción a la Programación y Programación 1 en la Facultad 6 de dicha institución. Arquitecto del proyecto GRAphTOol y líder del módulo de visualización molecular.

E-mail: yperezval@uci.cu

AGRADECIMIENTOS

Agradecer a la Revolución que me ha dado esta oportunidad de superarme, a los profesores que me apoyaron y los que no, a mi familia, Lali, Servando, Yari, Deli, Mila y Pongo, a los amigos que me han soportado todo este tiempo.

Servando

Quiero agradecer primero que todo a nuestro comandante en jefe, líder indiscutible de nuestra revolución sin la cual no hubiese sido posible mi presencia en esta universidad hoy, a mis hermanos, a mi papá, mi abuelo, mis tías, a mi novia Roxana, a mis suegros, a Lali que hizo realidad mi sueño, a los profesores que me han apoyado, a mis amigos y al físico.

Adair

A mi familia con todo el amor que se pueda llevar dentro, Lali, Servando, Deli y Yari.

Servando

A mi familia y en especial a mi hermano Alain

Adair

RESUMEN

La investigación previamente desarrollada como tesis de grado de la ingeniera Yenisbel Valle Garrido centró su atención en mejorar la eficiencia del procesamiento y cuantificación de imágenes de *microarray* en Cuba. Esta necesidad dio paso a la implementación de una aplicación de software para el procesamiento de imágenes a partir de la técnica de *microarray*, concebida particularmente para la obtención de valores de intensidades que soportan determinado nivel investigativo y que constituyó el objetivo fundamental de este trabajo. Se realizó un análisis crítico de los procesos de filtrado y procesamiento de imágenes digitales a usar, se fundamentan los conocimientos biológicos y matemáticos necesarios, y a partir de un diseño previamente realizado se implementa una propuesta de software, logrando que cumpla todos los requisitos funcionales y posea una interfaz grafica amigable.

PALABRAS CLAVE: *Microarray, Spot, Imágenes, Bioinformática, Cuantificación*

INDICE

INTRODUCCIÓN

El desarrollo vertiginoso de la era post-genómica dependerá en gran medida de lo eficiente que sea la comunidad científica mundial en desarrollar y validar herramientas biológicas relacionadas con el estudio del genoma.

El genoma de los seres vivos es el conjunto de genes que se encuentran distribuidos en cromosomas. Los genes, a su vez, son secuencias de ADN que contienen toda la información necesaria para sintetizar las proteínas, moléculas esenciales para la vida que realizan prácticamente todas las funciones celulares.

Cuando un gen se «activa» para dar lugar a su proteína correspondiente, ese gen se expresa en esa célula. Es conocido que anomalías en la expresión de los genes pueden llevar a disfunciones celulares, provocando graves enfermedades como el cáncer, entre otras. Por lo tanto, la identificación de los genes desregulados es un paso importante para conocer las bases moleculares de muchas enfermedades de carácter genético.

Estudiar la relación gen-enfermedad no está basado en analizar un gen único y sus efectos, sino en analizar el comportamiento de miles de genes de forma simultánea. Estos sistemas, denominados genéricamente como matrices, *microarray* o *biochips*, están cambiando la forma de plantear los problemas y extraer conclusiones de los experimentos que ofrecen una foto compleja del conjunto del genoma.

Para el avance científico en el siglo XXI, los *microarray* son una tecnología versátil y de amplias posibilidades, que han proporcionado velocidad al desarrollo de la biología molecular y la ciencia en general. La cuantificación de la expresión génica, a través de imágenes de *microarray*, se introduce como una técnica de alto rendimiento para trabajos en el área de la genómica funcional.

Con el propósito de desarrollar estudios para la detección de enfermedades con implicación genética en la población cubana, el Centro de Ingeniería Genética y Biotecnología (CIGB), dispone para analizar los resultados de dichos estudios del software ArrayVision, inicialmente utilizado en el desarrollo de proyectos de investigación.[1]

Al incrementarse el número de estudios desarrollados y en consecuencia crecer de modo proporcional la cantidad de imágenes necesarias a procesar, el software ArrayVision no resultó eficiente ante los nuevos requerimientos, evidenciándose en:

- El análisis de la región seleccionada, resulta poco asequible para el investigador por estar descrito de modo complejo.
- Los resultados obtenidos deben ser organizados manualmente.
- Referente a la alineación de spots, el software exhibe una cierta distorsión geométrica, acarreando errores en los valores de intensidad asociados a los spots.

Basados en las contradicciones planteadas y debido a que existe una tesis de grado que constituyó análisis y diseño de una solución similar surge ahora la necesidad de implementar una aplicación informática que cumpla con las especificaciones contenidas en dicho documento. El software estará condicionado por el problema científico: ¿Cómo procesar y cuantificar imágenes de *microarray* en Cuba?

Se desea obtener un software de sello nacional, que beneficie los intereses de nuestro País lo cual contribuirá al estudio de la dotación genética para prevenir enfermedades de bases moleculares.

Esta investigación comprende como objeto de estudio el procesamiento y análisis asistido por computadora de imágenes de *microarray*, cuyo campo de acción esta enfocado al procesamiento y cuantificación asistido por computadora de la expresión genética en imágenes de *microarray.* [1]

Desde el inicio del proyecto se trazaron pautas que permitieran darle cumplimiento de manera eficiente, identificándose como objetivo general, Implementar una aplicación informática para procesar y cuantificar imágenes *microarray*. En correspondencia, se han desglosado los objetivos específicos siguientes:

- Actualizar el diseño previamente realizado.
- Implementar funcionalidades que permitan el procesamiento y cuantificación de imágenes *microarray*.

Adhiriéndose a los objetivos mencionados, se trazaron una serie de tareas que dan cumplimiento a los mismos:

- Revisión del estado del arte del procesamiento de imágenes *microarra*y.
- Revisión del diseño previo realizado.

- Adaptación del diseño a las nuevas condiciones de la implementación del software.
- Construcción de los artefactos propios de la implementación, que comprendan el procesamiento y cuantificación de imágenes.
- Implementación de una funcionalidad que permita cargar una imagen.
- Implementación de un algoritmo que permita grillar una imagen.
- Implementación de un algoritmo que permita crear una imagen binaria.
- Implementación de un algoritmo que permita extraer valores de intensidades de los pixeles.
- Implementación de una funcionalidad que permita mostrar los resultados obtenidos.

El presente trabajo cuenta de tres capítulos estructurados de la siguiente forma:

Capítulo 1 – Fundamentación teórica

- Incluye una explicación de las técnicas, algoritmos y metodologías en la que se basó la investigación para darle solución al problema.

Capítulo 2 - Diseño

- Refinamiento del diseño previamente realizado.

Capítulo 3 - Implementación

- Construcción de los componentes y clases necesarias, descripción de los diagramas que se generan propios de la implementación así como el tratamiento de errores, interfaz y la concepción de la ayuda.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

Con la revolución tecnológica ocurrida en las últimas décadas en el campo de la informática y la biomedicina se ha conformado la bioinformática como resultado del punto de encuentro de estas dos

ciencias en su acelerado avance. Tal vez uno de los hitos más importantes que puede ilustrar la trascendencia del encuentro entre estas dos ciencias, es el *Proyecto del Genoma Humano.* Se usa el poder de calculo y de almacenamiento de información de las computadoras para analizar y procesar el alud de información que generan las investigaciones genéticas modernas representando un gran avance en cuanto al tiempo empleado en conocer la identidad de nuestros genes. [1]

1.2. Características y evolución del microarray.

La investigación microbiológica a nivel molecular está viviendo un cambio espectacular en los últimos años, fundamentado en tres grandes descubrimientos. El primero de estos tuvo lugar en la década de los 60's cuando se comienza a incursionar en la inmovilización de proteínas, posteriormente en la década de los 70's se logra la inmovilización de ácidos nucleicos y posteriormente en la década de los 80's y 90's surgen los microarray (Fig. 1.1). En apenas una década se ha pasado de trabajos basados en el estudio de uno o unos pocos genes al análisis de los genomas en su totalidad. Desde que en 1995 se publicara el primer genoma completo de un ser vivo, *Haemophilus influenzae*, ya se han descrito unos 145 genomas completos (incluyendo los de 19 organismos eucariotas como el hombre o el ratón) y aproximadamente 600 están siendo secuenciados en la actualidad. En este sentido, se están desarrollando nuevas tecnologías para trabajar con las grandes cantidades de datos que generalmente se obtienen en el proceso de análisis de un genoma. La técnica de *Microarray* de ADN (MAs) está atrayendo un gran interés ya que permite monitorizar la actividad de un genoma completo mediante un simple experimento. Además, cada uno de estos experimentos con MAs puede suponer el manejo desde cientos hasta decenas de miles de genes, normalmente con decenas de muestras por gen. [2]

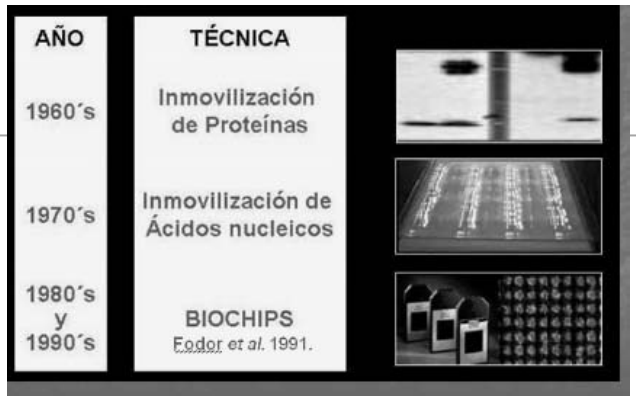


Figura 1.1. Evolución de la técnica *microarray.*

1.2.1. ¿Qué es un microarray?

Un concepto básico en *microarray* es el posicionamiento preciso en un soporte sólido de elementos que funcionen como detectores moleculares en altas densidades (Fig. 1.2). En la práctica, los *microarray* abarcan una amplia gama que puede tener diferentes soportes (membrana o vidrio) y diferentes moléculas que interaccionan en este medio. [1]

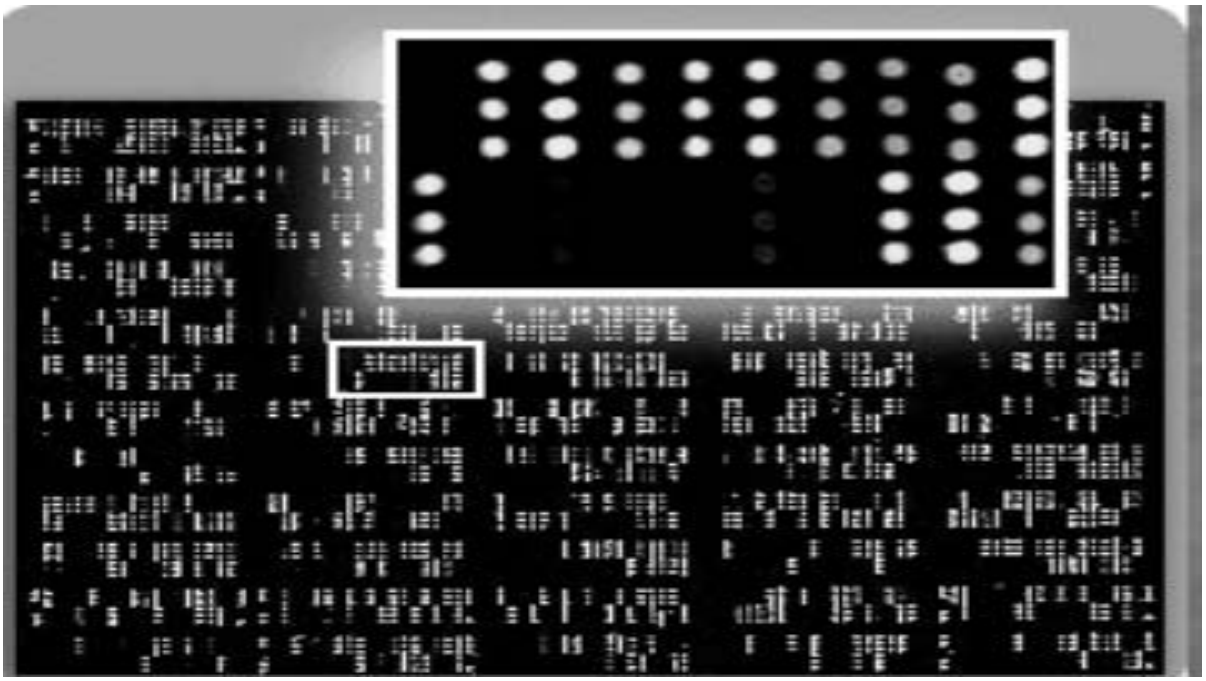


Figura 1.2. Imagen de *microarray* en un soporte de vidrio.

1.2.2. Construcción y funcionamiento de un microarray

La construcción de un *microarray c*onsiste en colocar en el mismo, cada uno de los genes cuyos niveles de expresión pueden ser cuantificados. Para ello se realizan cuatro procesos, proceso de la muestra, proceso del chip, proceso del chip con la muestra, y proceso de revelado, en el primer proceso se sintetiza (extracción-purificación, amplificación y marcaje) el material genético, en el segundo proceso se diseña y fabrica una capa de cristal, silicio o plástico, la cual presenta casillas que actúan a modo de tubo de ensayo. En el tercer proceso se hibrida, es decir, se insertan las muestras de material genético en las casillas previamente dichas y se eliminan todas las cadenas que no se han unido mediante lavados, sólo las moléculas que hibridan permanecerán en el *microarray* (Fig. 1.3), finalmente en el cuarto proceso se procede al revelado mediante un escáner óptico o con microscopía láser confocal. Luego, la imagen obtenida es procesada computacionalmente y se alcanza un patrón de intensidades en cada casilla.

En los estudios de *microarray* sólo habrá fluorescencia en los puntos del portaobjetos donde haya ocurrido hibridización y la intensidad de la fluorescencia detectada será proporcional al nivel de expresión del gen que se analiza. [1]

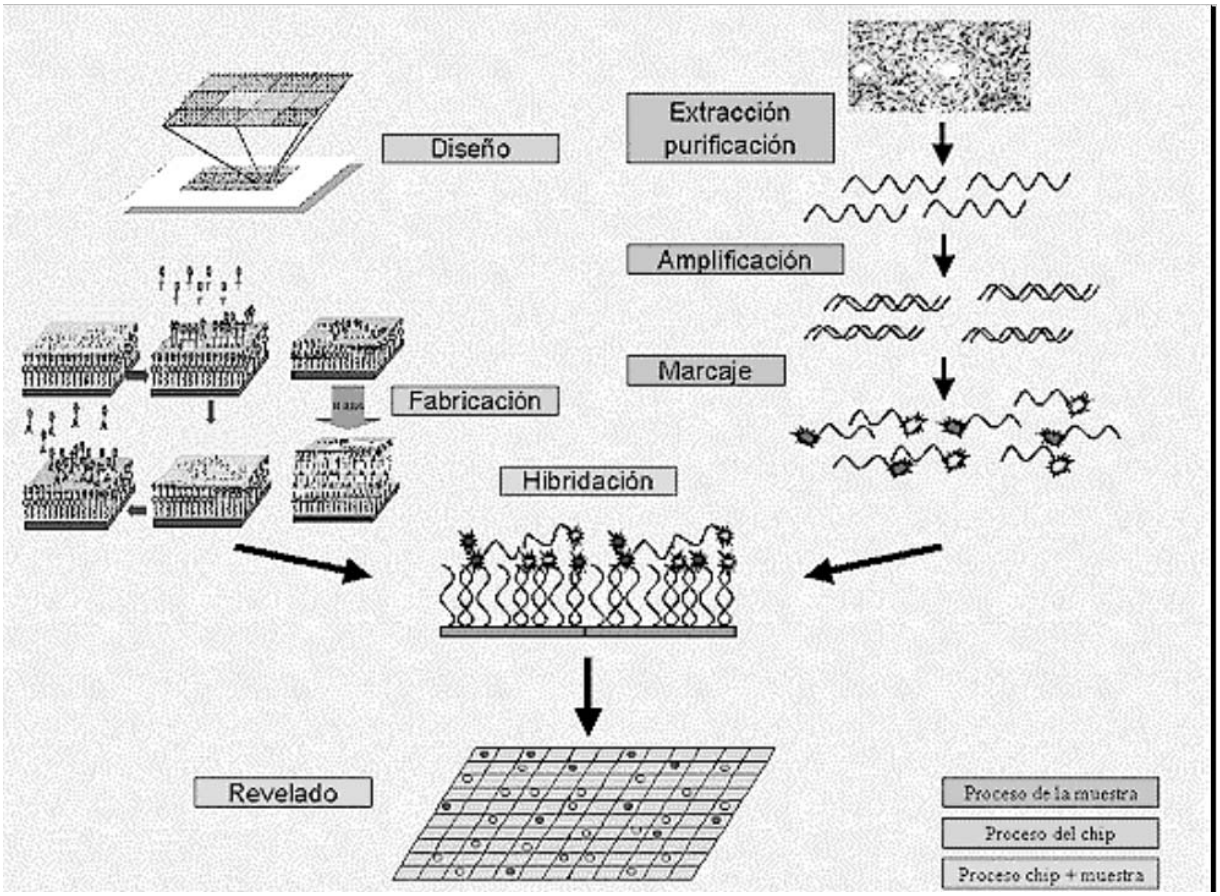


Figura 1.3. Representación de experimento básico de *microarray*.

1.2.3. ¿Qué tipo de *microarray* existen?

Para poder clasificar los *microarray* debemos dejar claro qué se entiende por blanco y qué se define como sonda. Existen controversias al respecto pero se tomará por blanco a la molécula libre y como sonda a la inmovilizada.

Existen diversos tipos de *microarray* según las sondas utilizadas que abarcan metodologías muy variadas, desde los más rudimentarios hasta los más sofisticados, algunos ejemplos de los tipos más comunes son:

- *Microarray* de proteínas

Las sondas son anticuerpos fijados a un portaobjetos de vidrio y los blancos son muestras de suero o tejido. Esta técnica se ve por el momento restringido por varios puntos, entre ellos podemos mencionar la dificultad de fabricar e inmovilizar estructuras *3-D* como son las proteínas y detectar interacciones de proteínas plegadas, sin olvidar mencionar que no se dispone aún de colorantes fluorescente que permitan cuantificar eficientemente a estas moléculas.

- *Microarray* de tejidos (TMA)

Esta técnica trata de resolver uno de los problemas principales y limitantes en el análisis molecular de tejidos: el tamaño limitado de la muestra. Se utiliza una aguja hueca para tomar muestras milimétricas de las regiones de interés de tejidos embebidos en parafina, en especial biopsias. Luego se depositan de manera ordenada en un nuevo bloque de parafina y se cortan con un micrótomo entre 100-500 veces y se reordenan sobre portaobjetos de vidrio donde se realizarán pruebas múltiples a nivel ADN, ARN y proteínas. Es una técnica costosa y poco práctica si se necesitan resultados inmediatos.

- *Microarray* de ADN

Esta técnica, muy usada en biología molecular y conocida en este ámbito con el calificativo de «*microarray*», consiste en la colección de fragmentos de ADN en una superficie sólida monitorizándose los niveles de expresión de ellos de forma simultánea. Las superficies empleadas para fijar el ADN son muy variables y pueden ser de vidrio, plástico e incluso de silicio. Este tipo de *microarray* es utilizado para identificar genes que producen ciertas enfermedades mediante la comparación de los niveles de expresión entre células sanas y células que están desarrollando otras enfermedades. Desde un punto de vista de longitud de la sonda usada, hay 2 tipos de *microarray* de ADN: *microarray* de cADN que presenta sondas de cientos de bases y de oligonucleótidos con sondas de 20-25 bases o 50-80.

- *Microarray* de oligonucleótidos

En los *microarray* de oligonucleótidos, las pruebas son designadas a partir de una secuencia conocida o un ARN predicho. Estos *microarray* dan estimaciones del nivel de expresión, pero por distintas condiciones no pueden ser observados en una misma matriz, por lo que por cada condición se ha de utilizar un chip.

- *Microarray de cADN*

En este tipo de *microarray* se hibrida el cADN (ADN copia) de dos condiciones que son marcados, cada uno de esas condiciones con dos fluoróforos diferentes. Las condiciones son mezcladas e hibridadas en el mismo *microarray*. Una vez realizado este primer paso se procede a escanear el resultado y a visualizar el mismo. De esta forma se pueden observar genes que se activan o se reprimen en distintas condiciones. En estos experimentos no se pueden observar niveles absolutos en la expresión. [1]

En el CIGB el *microarray* de ADN es la técnica usada actualmente.

1.2.4. Aplicaciones de los *microarray*

Las aplicaciones de esta tecnología fundamentalmente se localizan en el área biomédica, farmacéutica y clínica. Esta capacidad de análisis masivo y simultáneo es la que ha permitido un salto en las posibilidades que la técnica de *microarray* ofrece, no sólo cuantitativo (se analizan muchos más genes), sino cualitativo: el poder de la técnica radica en la posibilidad de aportar una visión molecular global de un determinado proceso patológico, por ejemplo de un tipo tumoral concreto. Este conocimiento global permite entrever el funcionamiento orquestado de los genes, como ellos y las proteínas para las que codifican, interaccionan entre sí en una circunstancia determinada, que es diferente entre un estado normal y uno patológico.

Los *microarray* se han aplicado al estudio de casi cualquier tipo de problema biológico, esto se evidencia en estudios de genes que se expresan diferencialmente y la búsqueda de terapias más efectivas contra el cáncer. Los estudios se realizan según las condiciones siguientes:

- Sanos/enfermos, tratados/no tratados.
- Clasificación molecular en enfermedades complejas.
- Identificación de genes característicos de una patología.
- Predicción de respuesta a un tratamiento.
- Detección de mutaciones y polimorfismos de un único gen (SNP).

Actualmente, la aplicación más frecuente de los *microarray* es en el análisis genético, pero en un futuro próximo sus aplicaciones serán mucho más amplias. [1]

1.2.5. Aplicaciones informáticas existentes para el análisis de imágenes de *microarray*

En la actualidad existe una gran variedad de soluciones informáticas destinadas al análisis de imágenes de *microarray,* ejemplo de esto son QuantArray 4.0 y Imagene 8.0. Estos softwares cumplen con los requerimientos necesarios para el procesamiento y cuantificación de las imágenes pero no es posible su adquisición debido a sus altos costos y la negativa de venta a nuestro país por las compañías distribuidoras debido al bloqueo económico impuesto por los Estados Unidos de América. Además, ya existe un antecedente en el que un software libre fue adquirido por una compañía transnacional.

1.2.5.1. QuantArray

QuantArray es un software desarrollado por Packard BioScence el cual se destaca por su gran capacidad de acoplarse a los distintos equipos utilizados para el proceso de obtención de la imagen, permitiendo hacer todo el procesamiento con un solo click, presenta una gran automatización en el proceso de grillado de la imagen, eliminando la necesidad de definir o editar manualmente las grillas, lo que influye en la velocidad de obtención de resultados, posee tres métodos de segmentación, los cuales son: segmentación por histograma, circulo variable y circulo fijo. El proceso de cuantificación de los resultados es completamente automático. El mayor inconveniente de este software es el alto precio de su licencia. [3]

1.2.5.2. Imagene

Imagene es un software desarrollado por BioDiscovery, este software se destaca por estar desarrollado para varios sitemas operativos (Mac, Windows y Linux). Presenta una gran automatización en el proceso de grillado de la imagen, eliminando la necesidad de definir o editar manualmente las grillas lo cual influye en la velocidad de obtención de resultados, los métodos de segmentación que presenta son completamente desarrollados segmentación por histograma, circulo variable, circulo fijo y forma variable. El proceso de cuantificación de los resultados es completamente automático. Los mayores inconvenientes de este software son su elevado costo de adquisición, así como, sus altos requerimientos. [4]

1.3. Imágenes digitales de un microarray

Una vez que se ha concluido la hibridación en un experimento de *microarray* el paso siguiente consiste en obtener una imagen digital que representa las intensidades de expresión de cada uno de los puntos (spots).

Una imagen digital en escala de grises es un matriz de MxN elementos numéricos cuyos valores posibles van del 0 (negro) al 255 (blanco), siendo este número la intensidad luminosa en el determinado punto o pixel (picture element), por convención el origen de la imagen se encuentra en el extremo izquierdo superior.

Una imagen digital a colores esta formada por 3 matrices de MxN elementos numéricos cuyos valores posibles van del 0 (negro) al 255 (blanco), siendo este número la intensidad luminosa en cada una de las bandas espectrales del RGB (Rojo, Verde, Azul), de cada punto o pixel, a diferencia de las imágenes en escala de grises, las imágenes a color requieren de la combinación de las 3 bandas de color, para representar el color de un pixel.

Por ejemplo, un determinado punto blanco de una imagen en escala de grises se describiría: P(x, y) = 255, sin embargo en una imagen a colores para describir el color del mismo punto se realizaría así: P(x, y) = (255, 255, 255), esto debido a que el (0, 0, 0) corresponde al negro absoluto y el (255, 255, 255) al blanco absoluto). [2]

En el caso de nuestro país las imágenes que se van a generar están expresadas en tonos de grises por la naturaleza del equipamiento con que cuentan nuestros polos científicos.

1.4. Tratamiento de imágenes. Procesamiento de imágenes de microarray

El tratamiento digital de imágenes comprende el procesamiento y el análisis de las mismas. El procesamiento está referido a la realización de transformaciones, la restauración y el mejoramiento de las imágenes. El análisis consiste en la extracción de propiedades y características de las imágenes así como la clasificación, identificación y reconocimiento de patrones. La importancia del procesamiento y análisis digital de imágenes se encuentra en el:

- Desarrollo de aplicaciones en biología y medicina de gran impacto social. El biólogo molecular puede presentar cuadros de moléculas y ganar conocimientos penetrando en su estructura.
- Incremento del potencial científico-técnico y de la capacitación de especialistas.
- Aumento del volumen de información para el procesamiento digital y análisis de imágenes con amplia repercusión social (TV, vídeo, medicina, biología, geología, etc.). [2]

1.4.1. Métodos de procesamiento

El procesamiento de las imágenes escaneadas de *microarray* puede ser dividido en tres tareas:

- Grillado: es el proceso de asignación de coordenadas a cada uno de los *spots.*
- Segmentación: clasifica los pixeles pertenecientes a los spots y los pertenecientes al fondo.
- Extracción de la intensidad: calcular para cada *Spot* un coeficiente de intensidad.

1.4.1.1. Grillado

La estructura básica de la imagen está determinada por la construcción del *microarray*. Refiriéndose a la cantidad de filas y columnas que tiene la matriz de spots (*grid), así como en cada subregión de la grid (subgrid)*. Para extraer las características de la *grid,* el software requiere de la información siguiente:

1. Cuántas *subgrids* tiene el arreglo en cada dirección (*x e y*).
2. Cuántos *spots* hay por *subgrids* en cada dirección (*x* e *y*).
3. Espaciado entre las filas y las columnas de la grid.

1.4.1.2. Segmentación

La segmentación de una imagen puede definirse como el proceso de particionar la imagen en diferentes regiones de diferentes propiedades. En un experimento de *microarray* la segmentación permite clasificar los *píxeles* en *foreground* o *background*, de manera que se puedan calcular intensidades como medidas de la abundancia de la secuencia transcripta para cada secuencia de ADN que se encuentre en la muestra. Todos los procedimientos de segmentación producen una máscara del *spot* (*spot mask*) que consiste en el conjunto de los *píxeles* donde se supone se encuentra un *spot*. Los métodos de segmentación pueden clasificarse en:

- Círculo fijo.
- Círculo variable.
- Forma variable.
- Histograma (basada en el umbral).

Segmentación con círculos fijos: Superpone un círculo del mismo tamaño a todos los *spots* de la imagen. Este método es fácil de implementar y funciona bien cuando todos los *spots* son circulares y

del mismo tamaño. Pero cuando los *spots* tienen forma variable, como ocurre en la mayoría de los *microarray*, tiende a dar resultados imprecisos.

Segmentación con círculos de tamaño variable: En este tipo de segmentación, el diámetro del círculo es estimado en forma separada para cada *spot*. Algunos software proveen la opción de ajustar a mano el diámetro de los círculos, *spot* por *spot* pero esto lleva mucho tiempo. Además en la práctica los *spots* rara vez son circulares y una máscara circular tendrá un mal ajuste. Las fuentes de no circularidad incluyen los procesos de la impresión o el post procesamiento de los portaobjetos luego de la impresión.

Forma variable: El crecimiento de regiones como usualmente se le conoce, es uno de lo métodos conceptualmente más simples para la segmentación; usado para extraer regiones de la imagen que están conectadas según cierto criterio predefinido. Este criterio puede estar basado en información de intensidades y/o bordes de la imagen. En su forma más simple, este método requiere un punto semilla (*seed point*) que es seleccionado manualmente por el usuario, y extrae todos los *píxeles* conectados a la semilla, que tengan el mismo valor de intensidad, es decir los *píxeles* adyacentes de amplitud similar se agrupan juntos para formar una región segmentada. Este método es poco implementado debido a su complejidad computacional, además la intervención del usuario en el proceso de selección puede incurrir en errores.

Segmentación por histograma (basada en umbral): Es el proceso de segmentación más simple. La mayor parte de las regiones de interés en una imagen están caracterizadas por una reflectividad o absorción de luz más o menos constante de su superficie (los *píxeles* de un *spot* tienen, aproximadamente, el mismo nivel de gris). La idea es obtener un umbral de brillo3 que separe el *spot* del fondo.

Dado el umbral T, la imagen umbralizada se obtiene mediante la expresión:

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq T \\ 0 & f(x,y) < T \end{cases}$$

Algoritmo de umbralización: recorrer todos los *píxeles* de la imagen; si su nivel de gris es mayor que el umbral es parte del *spot*, si es menor es parte del fondo. La umbralización es rápida computacionalmente.

La parte más difícil es encontrar el valor correcto del umbral.

La selección del umbral correcto es crucial en el proceso de segmentación por umbralización. La selección del umbral puede ser interactiva o puede ser el resultado de algún método de detección de umbral**,** el que se explica a continuación es el seleccionado para el desarrollo de la aplicación, no obstante se hace mención de otras elecciones de umbral:

- Segmentación umbralización *p*-cubrimiento.
- Segmentación por umbralización de histograma.
- Segmentación umbralización óptima.
- Segmentación umbralización multiespectral.
- Segmentación umbralización jerárquica.

Segmentación por umbralización de histograma: Dado que el histograma representa la distribución de probabilidad de que un *píxel* tenga un nivel de gris (u otra propiedad) dado y que es razonable suponer que un objeto tiene un rango de variación en la propiedad pequeño, es lógico suponer que un objeto de la imagen vendrá representado por un máximo local en el histograma. El proceso segmentación se convierte en la búsqueda de picos en el histograma. Este tipo de segmentación se implementa para buscar mínimos locales entre máximos locales significativos y utilizar el nivel de gris de esos mínimos como umbral de segmentación.

Esta estrategia de segmentación es sencilla. En primer lugar se localiza el mayor máximo local del histograma. A continuación se localiza el segundo mayor máximo local del histograma. Por último, se localiza el mínimo local entre ambos máximos y se utiliza su valor como valor para el umbral. [2]

En el trabajo se hará uso de la segmentación por umbral.

1.4.1.3. Extracción de la intensidad

A partir de los *píxeles* que representan los *spots* y su fondo, los softwares de procesamiento de imágenes deben calcular medidas numéricas para cada parte. A continuación describimos las medidas incluidas en ellos.

1. Media de la *spot*: media de las intensidades los *píxeles* que constituyen el *spot*.
2. Marca (*flag*), indica la calidad del *spot*.

La medida más importante es la intensidad del *spot* que puede elegirse entre la media y la mediana, pero esta última es la más recomendable. Intensidad del *spot*: cada valor de *píxel* en una imagen escaneada representa el nivel de hibridación en una posición del vidrio. La cantidad total de hibridación para una secuencia particular de ADN es proporcional a la fluorescencia total del *spot*. Por lo tanto, una medida natural de la intensidad del *spot* es la mediana de las intensidades de los *píxeles* dentro del *spot mask*.

1.5. Importancia del procesamiento y cuantificación de imágenes de microarray

Los programas *ArrayVision* [1], *Spotfinder* [5]*, ScanAlyze* [6], sirvieron como precedentes para el desarrollo de una aplicación propia, la cual permita de manera eficiente el procesamiento de toda información procedente de la investigación biológica plasmada en una imagen de *microarray*, así como la cuantificación rápida y automática de las imágenes mediante diferentes funciones matemáticas, métodos de análisis y procesamiento, con la finalidad de que sea útil en todos los centros científicos cubanos dedicados al estudio de la extracción de información de los genes, aunque en principio esté orientado a facilitar el trabajo de los investigadores del CIGB.

De manera general el procesamiento y cuantificación de imágenes de *microarray* ofrece una visión más global del estudio de los genes y además de la implicación en el aumento de la eficiencia del descubrimiento y desarrollo de nuevos fármacos, y la resistencia de enfermedades a drogas determinadas. [1]

1.6. Lenguajes de programación

En esta sección se especifican las características más significativas de los lenguajes de programación C++, C# y Java. Luego se fundamenta la elección de Java como lenguaje a utilizar para generar los artefactos software durante esta investigación.

1.6.1. Lenguaje de programación C++

C++ es un lenguaje de programación de propósito general inspirado en los lenguajes C y Simula67. Este lenguaje fue inventado por Bjarne Stroustrup en el año 1979 e inicialmente fue llamado "*C with classes*". El nombre actual, C++, fue adoptado en el año 1983 con el objetivo de indicar que el lenguaje era un "*C mejorado*". [3]

El lenguaje C++ soporta varios estilos de programación (e.g. procedural, orientado a objetos) e intenta ser tan eficiente y portable como lo es el lenguaje C. Además, permite la sobrecarga de los operadores, el trabajo con punteros, las directivas del preprocesador y los tipos genéricos, entre otras

funcionalidades. Por otro lado, el lenguaje soporta plenamente el polimorfismo y la herencia, destacándose la existencia de la herencia múltiple.

El lenguaje C++ es utilizado para desarrollar, prácticamente, sistemas de cualquier tipo (e.g. sistemas operativos, sistemas incrustados, aplicaciones). Algunos de ellos son los sistemas operativos Unix, Linux y Windows, los juegos Quake y Starcraft, y el programa de edición gráfica Photoshop. La principal dificultad que se encontró en este lenguaje es su portabilidad debido a que no puede generar ejecutables multiplataforma. [7]

1.6.2. Lenguaje de programación C#

C# es un lenguaje de programación orientado a objetos desarrollado por la compañía Microsoft como parte de la plataforma .NET. Su desarrollo fue guiado por Anders Hejlsberg y está basado en los lenguajes Delphi, Visual Basic, C++ y Java, entre otros.

El lenguaje C# ha sido diseñado para ser robusto, moderno y sencillo. El mismo promueve las prácticas sanas de programación (e.g. chequeo de tipos) y las reglas de globalización. En este lenguaje no existe el nivel de visibilidad global ni la herencia múltiple. Por otro lado, los punteros solo pueden ser utilizados en contextos no manejados y la memoria manejada no puede ser liberada explícitamente, además presenta un mecanismo robusto para el tratamiento de errores. Se destaca la existencia de tipos genéricos, la naturaleza compilado-interpretado del código del lenguaje (el código es llevado a una interfaz común del lenguaje (CLI) que luego es interpretado por el framework.net).

El lenguaje de programación C# es utilizado para desarrollar sistemas de gran diversidad. Sobresalen entre ellos las aplicaciones web, las aplicaciones de escritorio y los componentes reutilizables (toolkits y frameworks). La principal dificultad que presenta este lenguaje es que no es multiplataforma y no presenta para Linux un IDE de desarrollo aún lo suficientemente confiable para desarrollar aplicaciones robustas. [8]

1.6.3. Lenguaje de programación Java

Java es un lenguaje de programación de propósito general creado en la compañía Sun Microsystems y tiene su antecedente en el lenguaje “Oak”. Este último se comenzó a desarrollar en 1990 por James Gosling y Bill Jay, entre otros, con el objetivo de desarrollar sistemas incrustados. Sin embargo, en 1993, ante la explosión de las aplicaciones de Internet y el escaso mercado para los sistemas incrustados, el grupo decidió enfocar el lenguaje “Oak” hacia el desarrollo de sistemas distribuidos y sustituir el nombre “Oak” por la cadena “Java”.

El lenguaje Java es orientado a objetos y robusto. Entre sus características sobresalen, la ausencia de los punteros (la memoria es manejada automáticamente) y de las directivas del preprocesador, el soporte para la programación asincrónica y la utilización del enlace dinámico. Además, se destaca la naturaleza compilado-interpretado del código del lenguaje (el código Java es compilado a bytecodes y luego es interpretado por una máquina virtual). Este lenguaje oferta, también, un mecanismo robusto de manejo de excepciones. Tiene como característica que no permite la herencia múltiple.

Java se ha desarrollado unido a Internet. Es por ello que el lenguaje es utilizado, ampliamente, en el desarrollo de sistemas distribuidos (e.g. sistemas de gestión, aplicaciones web). Además, durante los últimos años, el lenguaje se ha utilizado crecientemente en el desarrollo de sistemas incrustados. [9]

1.6.4. Fundamentación de la elección

Debido a que este proyecto pertenece al polo de Bioinformática y es un estándar o paradigma del polo el desarrollo de software libre utilizando el sistema operativo Linux, además la potencia de las librerías JAI (Java Advance Imaging) de Sun para el procesamiento de imágenes, así como la posibilidad de crear un software multiplataforma bajo el Java Runtime 6.0, se seleccionó Java como el lenguaje indicado para la implementación de este proyecto.

1.7. Herramientas utilizadas.

En esta sección se describen las principales características de las herramientas Visual Paradigm for UML 6.0 y NetBeans IDE 6.0.

1.7.1. Visual Paradigm for UML 6.0

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Lista de características:

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento
- Modelado colaborativo con CVS y Subversion.

- Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI.
- Ingeniería de ida y vuelta.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, Esquemas XML, XML,.NET exe/dll, CORBA IDL
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso
- Diagramas EJB - Visualización de sistemas EJB.
- Generación de código y despliegue de EJB´s - Generación de beans para el desarrollo y despliegue de aplicaciones.
- Diagramas de flujo de datos.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.

- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- Importación y exportación de ficheros XMI.
- Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio.

- Editor de figuras.

1.7.2. NetBeans IDE 6.0

NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un entorno de desarrollo integrado (IDE) desarrollado usando la Plataforma NetBeans.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

Dicha plataforma es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones.

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas).
- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- Framework basado en asistentes (diálogos paso a paso).

1.8. Conclusiones

En este capítulo se abordaron los métodos y conceptos básicos de procesamiento de imágenes *microarray*, se justificó la utilización del lenguaje Java y UML así como el uso de las herramientas NetBeans IDE 6.0 y Visual Paradigm for UML 6.0 y sus características más significativas que justificaron su elección.

CAPÍTULO 2: Diseño

2.1. Introducción

Debido al cambio de plataforma donde se va a desarrollar el proyecto se hace necesario redefinir algunos aspectos del flujo de trabajo Diseño realizado anteriormente en la tesis de grado “Software Procesamiento de Imágenes de Microarray” de Yenisbel Valle Garrido, dado que en un inicio este fue concebido para el sistema operativo Windows, usando Matlab y Visual Studio.NET como herramientas de desarrollo y C# como lenguaje de programación. Teniendo en cuenta la necesidad de llevar el proyecto a software libre se decidió implementarlo sobre Linux, usando Java como lenguaje de programación y NetBeans como herramientas de desarrollo.

Para el desarrollo del flujo de trabajo Diseño, se tuvo como premisa fundamental la modelación del sistema, estructurando la información de forma tal, que soporte los requerimientos de mantenimiento, reusabilidad, escalabilidad y robustez. Primeramente se identificaron los objetos pertinentes, clasificados dentro de las clases en la granularidad correcta, se definen interfaces y establecen relaciones entre ellos. Luego se organizó la arquitectura del sistema de manera que contemplara las formas de acceder y procesar las imágenes de *microarray*.

2.2. Estilo arquitectónico

La arquitectura de software se entiende como «la organización fundamental de un sistema encarnada en sus componentes, las relaciones de los componentes con cada uno de los otros y con el entorno, y los principios que orientan su diseño y evolución».

Dentro de la amplia clasificación de estilos arquitectónicos se utilizó arquitectura en capas. Es relativamente sencillo reconocer por qué interesa una disposición en capas.

Las ventajas del estilo en capas son obvias. Primero que nada, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. [1]

Con este antecedente se propone una arquitectura en tres capas (Fig. 2.1), donde cada una se ocupa de un nivel del problema y debe tener poco acoplamiento con las demás de manera que el cambio en una capa, no altera en gran medida los cambios en la otra capa. Si se desea en un futuro incorporar una capa de presentación distinta, no debe alterar a la Capa Operacional ya desarrollada. En definitiva, cambiar la implementación de una capa, debe introducir los mínimos efectos en el resto de la aplicación.

- Capa Presentación: desarrolla todos los elementos que conforman la interfaz de la aplicación, facilitando el desarrollo del sistema y visualizando el resultado del procesamiento de imágenes de distintos modos, utilizando los componentes que provee el Java Runtime Environment; estos tienen correspondencia con funcionalidades de adquisición y/o visualización de imágenes y datos.

- Capa Operacional: encargada de manejar todos los datos del sistema y organizar la lógica del negocio, comprende todas las clases encargadas de realizar los algoritmos y métodos necesarios para la cuantificación de imágenes, denominadas clases controladoras; aparecen además las clases entidades, que definen algún tipo de información perdurable en el marco del procesamiento de dichas imágenes, esto garantiza el flujo de información de una capa a otra.

- Capa de Soporte: de manera general esta es una capa utilitaria, o sea incluye todo lo referente a librerías, componentes, *framework*, que den soporte a la Capa Operacional. En este caso usamos las bibliotecas JAI de Sun y JExcelAPI.

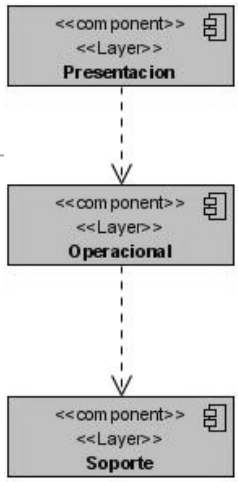


Figura 2.1: Arquitectura en 3 capas para el procesamiento de imágenes de *microarray*.

2.3. Clases del diseño

- Interfaces o vistas

Estas clases contienen los componentes necesarios para construir las vistas que se utilizan para la entrada de la imagen y para mostrar los resultados, del procesamiento y cuantificación, en términos de imágenes o de valores de intensidades asociadas a esta.

- Gestores

Estas clases contienen la lógica del negocio y manejan la mayoría de los algoritmos y operaciones que se realizan en el sistema para procesar y cuantificar la imagen. Se diseño al menos un gestor para cada uno de los procesos y subprocesos que se ejecutan en cada subsistema, a partir de la diversidad de estos.

- Entidades

Estas clases contienen la información necesaria del sistema en cuestión, la mayoría realizan operaciones simples sobre los datos que poseen. En ocasiones se complementan dando cierto nivel de complejidad en el subsistema correspondiente.

2.4. Realizaciones de Casos de Uso del diseño

Se refieren a continuación las realizaciones de casos de uso más importantes para la arquitectura del sistema, sobre la base de que estas realizaciones establecen una relación en la que se describe como se realiza un caso de uso especifico y como se implementa (o ejecuta) en términos de clases de diseño y sus objetos.

En este refinamiento del diseño se han eliminado dos casos de uso debido a que no representan una funcionalidad del software y no hacen ningún aporte al objetivo, dichos casos de usos son:

- ✓ Seleccionar Región de Interés
- ✓ Mostrar Imagen Cuantificada

El resto de los casos de uso se mantienen similares a los descritos en la tesis precedente a esta.

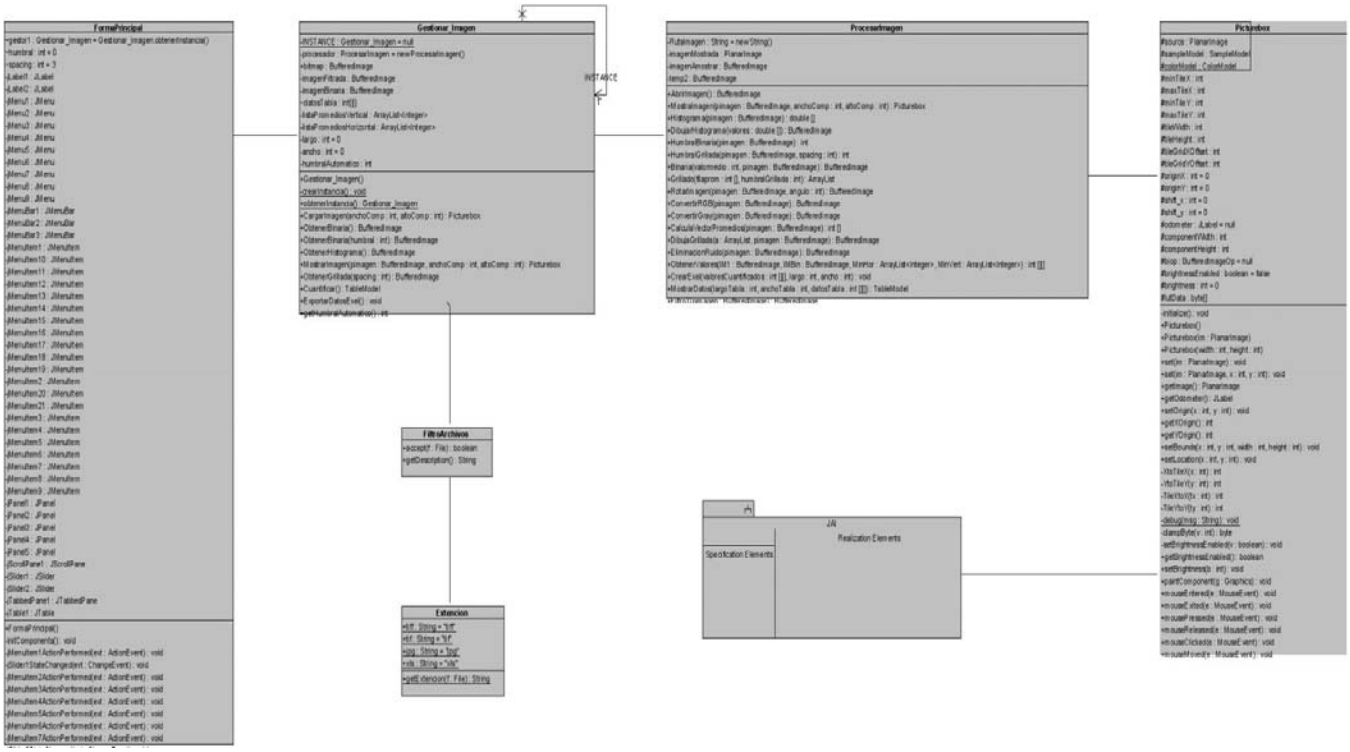


Figura 2.2. Diagrama de clases del diseño del caso de uso cargar imagen.

Diagramas de clases del diseño del resto de los casos de uso. **Ver Anexo 1**

2.5. Patrones de diseño: Singleton

Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, Christopher Alexander considera que «Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma». El Singleton es quizás el más sencillo de los patrones, también uno de los más conocidos y utilizados. Su propósito es asegurar que sólo exista una instancia de una clase y proporciona un punto de acceso global a esta instancia.

Este patrón se usa en la clase Gestionar_Imagen (Fig. 2.3), pues todo el flujo que se desarrolla para procesar y cuantificar la imagen de *microarray*, es propio de una imagen, es decir no se modifica o analiza más de una imagen a la vez, aunque esta clase cuenta con otras funcionalidades que por el momento no compete su explicación.

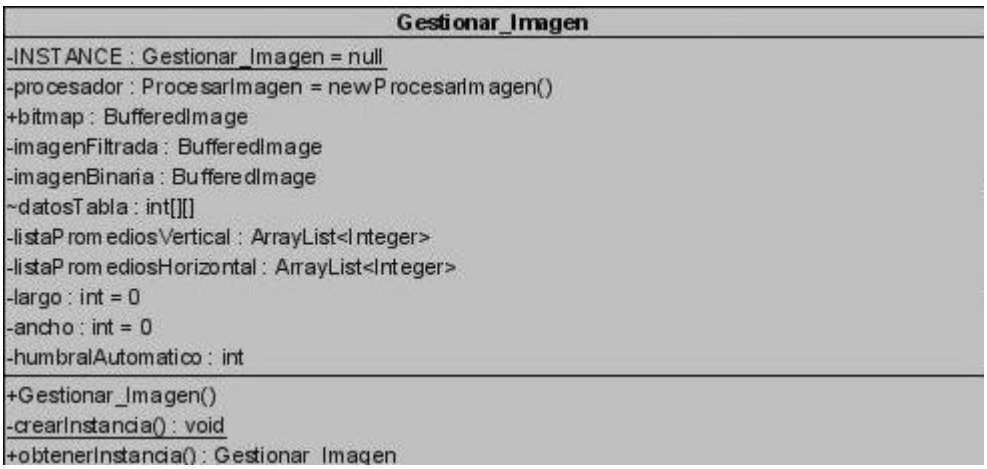


Figura 2.3. Representación del patrón «Singleton» en la clase imagen.

El funcionamiento de este patrón es muy sencillo y se reduce a los siguientes conceptos:

1. Ocultar los constructores de la clase Gestionar_Imagen, para que las otras clases que necesiten después de esta, no puedan crear instancias.

2. Declarar en la clase Gestionar_Imagen una variable miembro privada que contenga la referencia a la instancia única que se gestiona.

3. Proveer en la clase Gestionar_Imagen una función o propiedad que brinde acceso a la única instancia gestionada por esta clase; además, las clases que necesitan acceder a la instancia lo hacen a través de esta función o propiedad.

2.6. Componentes utilizados

En la implementación del software se utilizaron dos bibliotecas de componentes para el procesamiento de las imágenes y la persistencia de los resultados obtenidos.

2.6.1. Biblioteca JAI

Java Advanced Imaging (JAI) es una API de extensión de la plataforma Java que provee un conjunto de interfaces orientadas a objetos que complementan un modelo de programación simple y de alto nivel, permitiendo a los desarrolladores crear sus propias rutinas de manipulación de imágenes sin las restricciones adicionales de costo y licencia asociadas a los softwares comerciales de procesamiento de imágenes.

JAI se provee como descarga gratuita directamente desde la página de Sun Microsystems para plataformas Windows, Solaris, y Linux. Apple Computer provee una versión de la API para Mac OS X desde su sitio web disponible desde Mac OS X 10.3; Mac OS X 10.4 viene con el API pre instalado.

Mientras el API se provee en Java, otras implementaciones específicas para cada plataforma pueden usar la implementación pura de Java o proveer una implementación que se beneficia de las tecnologías nativas de la plataforma para proveer mejor rendimiento. [10]

2.6.2. Biblioteca JExcelAPI

Java Excel API es una biblioteca robusta y de código abierto que permite leer, escribir y modificar dinámicamente hojas de cálculo Excel.

Cualquier sistema operativo que pueda ejecutar la máquina virtual de Java puede procesar y entregar hojas de cálculo Excel. Debido a que la biblioteca está programada en Java puede ser invocada desde un servlet, ganando acceso a las hojas de cálculo Excel desde aplicaciones web.

JExcelAPI posee varias funcionalidades entre las que se encuentran:

- Lee datos de libros de trabajo de Excel 95, 97, 2000, XP y 2003.
- Lee y escribe fórmulas (versiones superiores a Excel 97).
- Genera hojas de cálculo en formato Excel 2000.
- Permite dar formato, a letras, números y fechas.
- Permite sombreado, bordeado y coloreado de celdas.
- Modifica libros de trabajo existentes.
- Esta internacionalizada, permitiendo el procesamiento de casi cualquier lenguaje, país, local o codificación de caracteres.
- Permite copiar diagramas.
- Permite inserción y copia de imágenes dentro de las hojas de cálculo. [11]

2.7. Conclusiones

Como resultado de este capítulo se redefinió un modelo de diseño que sirve de guía para la posterior implementación, con los artefactos necesarios para cumplir con algunas de las tareas trazadas inicialmente. Acorde a la línea de arquitectura definida, se identificaron los subsistemas y las clases correspondientes, ajustándose además el patrón de diseño Singleton a esta propuesta. El mayor peso de este capítulo recae en el modelo de diseño, caracterizándose el mismo por su adaptabilidad y lograr conducir a una implementación satisfactoria del sistema.

CAPÍTULO 3: Implementación

3.1. Introducción

Este capítulo trata acerca del flujo de trabajo Implementación representando los artefactos que la integran, como lo son los diagramas de componentes y el de despliegue.

3.2. Flujo de trabajo Implementación

Este flujo de trabajo comienza por lo que ha recibido de la etapa de diseño utilizándolo para la implementación del sistema en términos de componentes, en ella se convierten las clases del diseño en ficheros de código fuente, binarios, ejecutables, entre otros. La etapa está fuertemente enmarcada por el lenguaje de programación y compuesta por el diagrama de despliegue y por el de componentes.

3.3. Diagramas de componentes

Es un diagrama representado por un grafo de componentes software enlazados a través de relaciones de dependencia (compilación, ejecución).

Este diagrama muestra un grupo de elementos del modelo tales como componentes, subsistemas de implementación y sus relaciones. Se utilizan para modelar la vista estática de un sistema.

3.4. Diagrama de despliegue

El software ha sido diseñado para ser una aplicación de escritorio sin necesidad de conexión a algún servidor o de datos remotos, las imágenes se pueden cargar desde cualquier medio extraíble por lo que no es necesario hacer un diagrama de despliegue en este caso. El software funciona sobre una estación de trabajo completamente independiente.

3.5. Descripción de las soluciones.

3.5.1. Introducción

La solución está compuesta por 4 partes fundamentales que permiten obtener el resultado, estas partes son:

- Grillado.
- Segmentación.
- Cuantificación.
- Tabular y exportar valores.

3.5.2. Grillado

El grillado se obtiene a partir de la imagen inicial y su resultado son dos vectores que contienen las coordenadas donde se ubican las separaciones entre las columnas de spots. Para obtenerlos se extrae una señal horizontal y otra vertical de dicha imagen que contienen los promedios de intensidad de las columnas y filas de pixeles.

Figura 3.1. Representación de los máximos y mínimos de una columna de spots

Nótese que los máximos representan las columnas de spots dado que sus pixeles tienen mayor intensidad y los mínimos las separaciones entre estas.

El paso siguiente consiste en hallar un umbral automáticamente a partir del promedio entre la columna de pixeles de mayor intensidad y la de menor, el usuario también puede modificar el cociente de dicho promedio manualmente agregándole un factor de sensibilidad al proceso. Con este umbral se detectan las laderas de los máximos (subidas o bajadas) y se halla la coordenada que corresponde al punto medio entre estos, los cuales representan los mínimos que son a su vez la división entre las columnas de spots. [12]

Figura 3.2. Spots separados a partir de los puntos medios entre las laderas de intensidad de los spots

Estos valores nos ayudan en la cuantificación, debido a que usando las coordenadas separadoras horizontales y verticales podemos encerrar cada spot por separado en un recuadro.

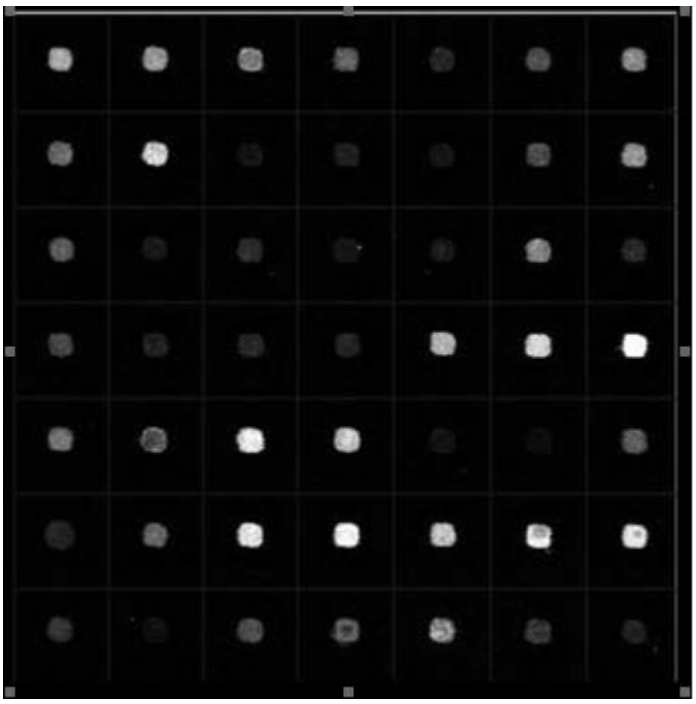


Figura 3.3. Imagen de *microarray* grillada.

A continuación se muestra el código con comentarios para su mejor comprensión:

Esta función se encarga de construir las señales constituidas por los promedios de las columnas de pixeles.

```
public int [] CalculaVectorPromedios(BufferedImage pimagen) //recibe como parámetro
{                                                  //la imagen original
    int ancho= pimagen.getWidth();
    int alto= pimagen.getHeight();
    int [] vectorPromedios= new int[ancho];
    int xxx=0;
    int aux1=0;     //se declaran las variables necesarias para contabilizar
                    //así como el vector resultado de la función
     for(int i=0;i<=ancho-1;i++)
     {
        xxx=0;      //ciclos para recorrer la imagen

        for(int j=0;j<=alto-1;j++)
         {  xxx+=(pimagen.getRGB(i, j))& 0xFF;  //se hace un “and” a nivel de bits
                                  //para acceder solo al ultimo byte de
         }                        //información del pixel.
          aux1=xxx;

          vectorPromedios[i]=aux1;   //se almacena el valor
```

```
    }
  return vectorPromedios;            //se retorna el vector
}
```

El siguiente código se encarga de calcular los vectores del grillado usando el umbral y los vectores promedios obtenidos con la función anterior:

```
public ArrayList Grillado(int[] filaprom, int humbralGrillada)   //recibe el umbral
  {                                                  //y los vectores

      ArrayList  minimos=new ArrayList(0);
       int caida=0;
       int subida=0;  //se declaran variables necesarias para el proceso
       int inicial=0;
       int finals=temp2.getHeight();
       boolean flag2=false;
       boolean flag1=true;
       minimos.add(inicial);   //el primer mínimo siempre será el inicio de la
                          //imagen, es decir: 0.
       for(int i=1;i<filaprom.length;i++ )//ciclo para recorrer
      {                               // el vector de promedios
          if((filaprom[i-1]<=humbralGrillada)&&  (filaprom[i]>=humbralGrillada)&&(flag1)) //condición para
             flag1=false;                        //detectar la primera subida
          else                                 //que se descarta usando un flag
          {

          if((filaprom[i-1]>=humbralGrillada)&& (filaprom[i]<=humbralGrillada)&&(!flag2))
          {
             caida=i;
             flag2=!flag2;    //condición que detecta la coordenada de cruce
          }                  //entre el umbral y la señal en bajada y la almacena

          if((filaprom[i-1]<=humbralGrillada)&& (filaprom[i]>=humbralGrillada)&&(flag2))
          {
             subida=i;
                minimos.add((int)(subida-(subida-caida)/2));
             flag2=!flag2;  //condición que detecta la coordenada de cruce          }
//entre el umbral y la señal en subida y ademas
          }                  //halla el punto medio entre las 2 coordenadas
       }
       minimos.add(finals-1); //el ultimo minimo se situa al final de la imagen
       return minimos;        //es decir, el largo de esta
    }
```

3.5.3. Segmentación

Para lograr la segmentación de la imagen se aplica la segmentación por umbralización global, para hallar el valor umbral se aplica el método de Otsu llamado así en honor a Nobuyuki Otsu que lo inventó en 1979, utiliza técnicas estadísticas, para resolver el problema. En concreto, se utiliza la variancia,

que es una medida de la dispersión de valores, en este caso se trata de la dispersión de los niveles de gris.

El método de Otsu calcula el valor umbral de forma que la dispersión dentro de cada segmento sea lo más pequeña posible, pero al mismo tiempo la dispersión sea lo más alta posible entre segmentos diferentes. Para ello se calcula el cociente entre ambas variancias y se busca un valor umbral para el que este cociente sea máximo. [13]

Exposición matemática

Como punto de partida tomamos dos segmentos de puntos ($K_0(t)$ y $K_1(t)$), que son definidos a partir del valor umbral *t*. *t* es la variable que buscamos, y los dos segmentos son el resultado deseado en la segmentación.

Sea $p(g)$ la probabilidad de ocurrencia del valor de gris 0 < g < G (G es el valor de gris máximo). Entonces la probabilidad de ocurrencia de los pixeles en los dos segmentos es:

K_0: $P_0(t) = \sum_{g=0}^{t} p(g)$ y K_1: $P_1(t) = \sum_{g=t+1}^{G} p(g) = 1 - P_0(t)$

Si tomamos dos segmentos (o sea un solo valor umbral) la suma de estas dos probabilidades dan evidentemente como resultado 1.

Si $\overline{g}$ es la media aritmética de los valores de gris en toda la imagen, y $\overline{g_0}$ y los $\overline{g_1}$ valores medios dentro de cada segmento, entonces se pueden calcular las variancias dentro de cada segmento como:

$\sigma_0^2(t) = \sum_{g=0}^{t} (g - \overline{g_0})^2 p(g)$ y $\sigma_1^2(t) = \sum_{g=t+1}^{G} (g - \overline{g_1})^2 p(g)$

La meta es mantener la variancia dentro de cada segmento lo más pequeña posible y conseguir que la variancia entre los dos segmentos sea lo más grande posible. Así obtenemos:

$$Q(t) = \frac{\sigma_{zw}^2(t)}{\sigma_{in}^2(t)}$$

La varianza entre los segmentos es:

$$\sigma_{zw}^2(t) = P_0(t) \cdot (\overline{g_0} - \overline{g})^2 + P_1(t) \cdot (\overline{g_1} - \overline{g})^2$$

La varianza dentro de los segmentos se obtiene de la suma de ambas:

$$\sigma_{in}^2(t) = P_0(t) \cdot \sigma_0^2(t) + P_1(t) \cdot \sigma_1^2(t)$$

El valor umbral *t* se elige de manera que el cociente *Q*(*t*) sea máximo. *Q*(*t*) es por lo tanto la medida buscada. De esta forma elegimos un valor umbral que optimiza los dos segmentos en términos de variancia. [14]

El método fue implementado en una función llamada *HumbralBinaria* a la que se le pasa como parámetro la imagen inicial y devuelve el umbral óptimo para segmentar la imagen por umbralización global. A continuación se muestra el código fuente con comentarios para su mejor comprensión.

```
public int HumbralBinaria(BufferedImage pimagen)// Histogram histograma)
 {
      double omega1=0;
      double omega2=0;
      double mu1=0;
      double mu2=0;
      double P[]=new double[256];
      double sigmaB2max=0;
      double sigmaB2=0;
      int humbralOptimo=0;
      int ancho= pimagen.getWidth();
      int alto= pimagen.getHeight();

      //Cálculo de la distribución de probabilidad de los niveles de gris

      for(int i=0;i<=ancho-1;i++)
      {
         for(int j=0;j<=alto-1;j++)
         {
            P[(pimagen.getRGB(i, j))& 0xFF]++;
         }
      }
      for(int i=0;i<=255;i++)
      {
         P[i]=P[i]/(ancho*alto);
      }
      double omega[]=new double[256];


//Cálculo del momento acumulado de orden cero y del de primer orden
```

```
            omega[0]=P[0];
        double mu[]=new double[256];
        for(int i=1;i<=255;i++)
        {   omega[i]=omega[i-1]+P[i];
            mu[i]=mu[i-1]+(i-1)*P[i];
        }

        //Cálculo del umbral óptimo según Otsu

        double mut=mu[255];    //Intensidad media total de la imagen
        for(int t=0;t<=255;t++){
            omega1=omega[t];
            omega2=1-omega1;
            if(omega1!=0&&omega2!=0)
            {   mu1=mu[t]/omega1;
                mu2=(mut-mu[t])/omega2;
                sigmaB2=omega1*((mu1-mut)*(mu1-mut))+omega2*((mu2-mut)*(mu2-mut));
                if(sigmaB2>sigmaB2max)
                {   sigmaB2max=sigmaB2;
                    humbralOptimo=t-1;
                }
            }
        }
    return humbralOptimo;
}
```

Con posterioridad y usando este umbral se recorre pixel a pixel la imagen original, a partir de esta, se genera una imagen binaria conformada por pixeles con valor 255 para los que presentan en la imagen original un valor de intensidad mayor que el umbral y 0 para los pixeles en la imagen original con valores de intensidad que estén por debajo del umbral (Fig. 3.4).

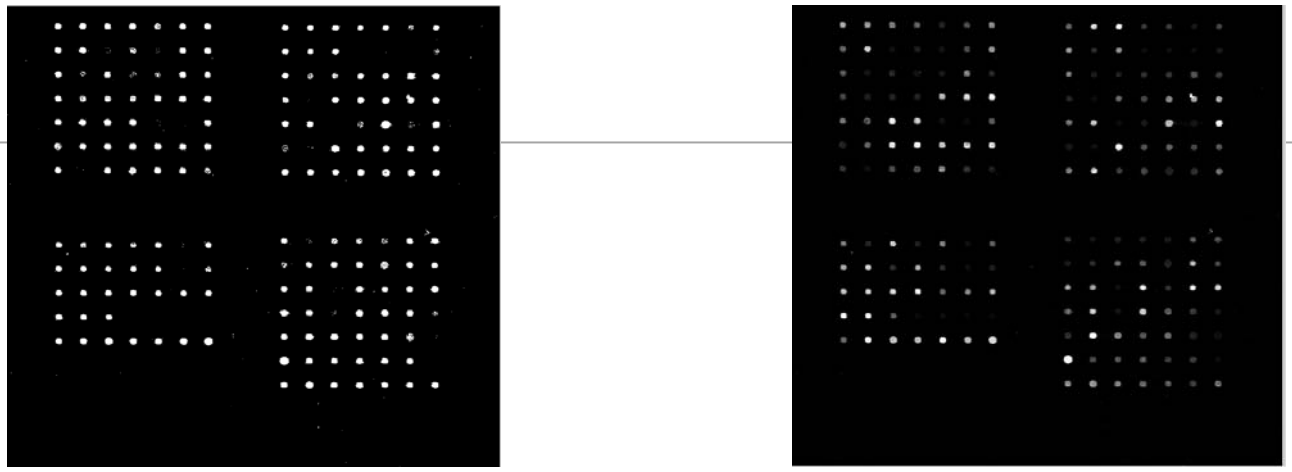


Figura 3.4: Imagen binaria (izquierda) creada a partir de la original (derecha) usando umbralización global según el método de Otsu.

3.5.4. Cuantificación

Una vez obtenida la imagen binaria y los vectores del grillado se procede a extraer los valores de los spots usando la imagen binaria para distinguir los pixeles pertenecientes a los spots o al fondo y los vectores para delimitarlos [15]. Para esta operación se usan cuatro ciclos anidados para recorrer la matriz de spots y los pixeles dentro de cada región de esa matriz comparando cada uno de ellos por su posición con los de la imagen binaria para distinguir los significativos. La función arroja como resultado una matriz de tipo entero de mxn sean m y n las dimensiones de la matriz de spots.

A continuación se muestra el código fuente con comentarios para su mayor comprensión.

```
public int[][] ObtenerValores(BufferedImage IM1, BufferedImage IMBin, ArrayList<Integer> MinHor,
ArrayList<Integer> MinVert)
{    int minVerSize= MinVert.size();
     int minHorSize= MinHor.size();
     int[][] valores = new int[minHorSize-1][ minVerSize- 1];
     int cont;
     int suma;
     int x=0; //Se declaran contadores para recorrer la matriz de spots
     int y=0;

   for (int i=0;i<minVerSize -1;i++)
     { //Ciclos para recorrer la matriz de spots
          x=0;
        for(int j=0;j<minHorSize-1;j++)
        {   cont=0;
            suma=0;
          for (int k =MinHor.get(i) ;k<MinHor.get(i+1); k++)
    //Ciclos para recorrer los fragmentos de la imagen que encierran cada spot
              for(int l =MinVert.get(j);l<MinVert.get(j+1);l++)
              {   if(((IMBin.getRGB(k, l))& 0xFF)==255)
                  {   cont++;
                      suma+=(IM1.getRGB(k, l))& 0xFF;
                  }
              }

            if(cont<64)
                 //Condición para chequear la existencia de cada spot
            {valores[x][y]=-1;
//se marcan los spots no existentes asignándole el valor -1
            }
            else
            {valores[x][y]=(int)suma/cont;
            }
          }x++;
        }
      y++;
```

```
    }
    return valores;   //Matriz de coeficientes de intensidad para cada Spot
}
```

3.5.5. Tabular y exportar valores

El siguiente método se encarga de mostrar los valores al usuario en una tabla sobre la ventana principal. Para este propósito se crea un modelo de tabla (TableModel) que se le pasa como parámetro a una tabla (jtable).

```
public TableModel MostrarDatos(int largoTabla,int anchoTabla,int [][] datosTabla)
{
      //se reciben como parámetros la matriz de valores a tabular y las dimensiones
         de esta
   TableModel a;
      a = new DefaultTableModel(largoTabla-1,anchoTabla-1);//se crea el TableModel

     for(int i =0; i<largoTabla-1;i++)
       for(int j =0; j<anchoTabla-1;j++)
       {
          a.setValueAt(String.valueOf(datosTabla[i][j]), i, j);
       }        //se le pasan los datos al TableModel usando dos ciclos anidados

      return a;    //El resultado es el TableModel modificado

}
```

Una vez mostrados los valores el usuario puede seleccionar exportarlos a formato XLS lo cual permite la siguiente función.

```
public void CrearExel(int[][] valoresCuantificados,int largo,int ancho)
{      //se reciben como parámetros la matriz de valores a tabular y las dimensiones de esta
```

```
   JFileChooser GuardarArchivo = new JFileChooser();//dialogo para cargar y guardar
                                        //archivos en NetBeans
 FiltroGuardar FiltroExel=new FiltroGuardar(); //filtro de extenciones
   GuardarArchivo.setFileFilter(FiltroExel);  //asignamos filtro
   String RutaGuardarImagen=new String();    //variable para la dirección

    GuardarArchivo.setApproveButtonText("Guardar Archivo");
                                        //texto del botón aceptar
    GuardarArchivo.setDialogTitle("Exportar Datos a Excel");
                                        //titulo de la ventana
     int returnVal = GuardarArchivo.showOpenDialog(GuardarArchivo);
                                        //Guardamos la decisión del usuario
     if (returnVal == JFileChooser.APPROVE_OPTION)
```

```
    {
        RutaGuardarImagen=GuardarArchivo.getSelectedFile().getPath();
    }       //si aceptó, guardamos la dirección donde se va a guardar
  try
  {
    WritableWorkbook workbook = Workbook.createWorkbook(new File(RutaGuardarImagen+".xls"));
                    //crea el archivo en la ubicación
    WritableSheet sheet = workbook.createSheet("Valores de Intensidad", 0);
    try
     {
       for(int i=0;i<largo-1;i++)
       {
         for(int j=0;j<ancho-1;j++)
         {
           jxl.write.Number media = new jxl.write.Number(i, j, valoresCuantificados[j][i]);
                 //se usan 2 ciclos para recorrer la matriz y la hoja de cálculo
                  //y copiar los valores de la primera a la segunda.
            sheet.addCell(media);
         }
       }
         workbook.write();     //se copia el archivo a disco duro

         workbook.close();
      }
      catch(jxl.write.WriteException e){};
  }
  catch(IOException e){};

}
```

3.5.6. Interfaz

Para diseñar la interfaz, se usó menú (jMenu) para mostrar las opciones disponibles debido a su facilidad de acceso así como el ahorro de espacio en la ventana principal, se usaron además un panel tabulado (tabbedPane) para mostrar los distintos resultados que va ofreciendo la aplicación debido a que posee varias pantallas o paneles cuyas vistas se pueden alternar y así posibilita mostrar varios resultados e imágenes sobre el mismo espacio que contribuye a la simplificación de la ventana principal. Se usó también una barra de desplazamiento (jSlider) para permitir al usuario entrar valores de umbral personalizado debido a su facilidad de uso. Se utilizó una tabla (jTable) para mostrar los valores resultantes debido a que es el componente que permite mostrarlos de forma tabulada y organizada.

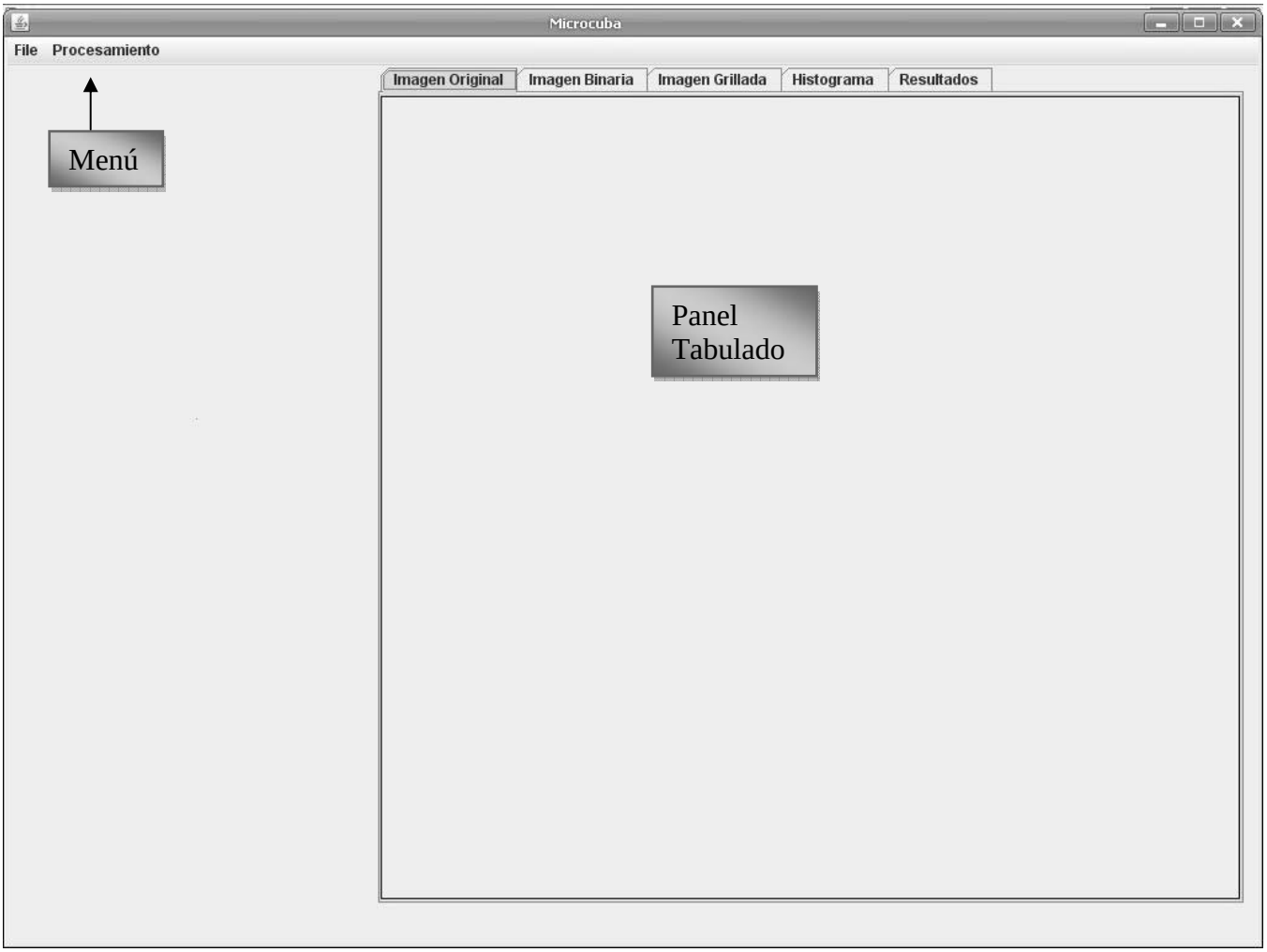


Figura 3.5. Ventana principal.

Figura 3.6. Ventana para seleccionar el archivo a cargar, nótese el filtro de extensiones.

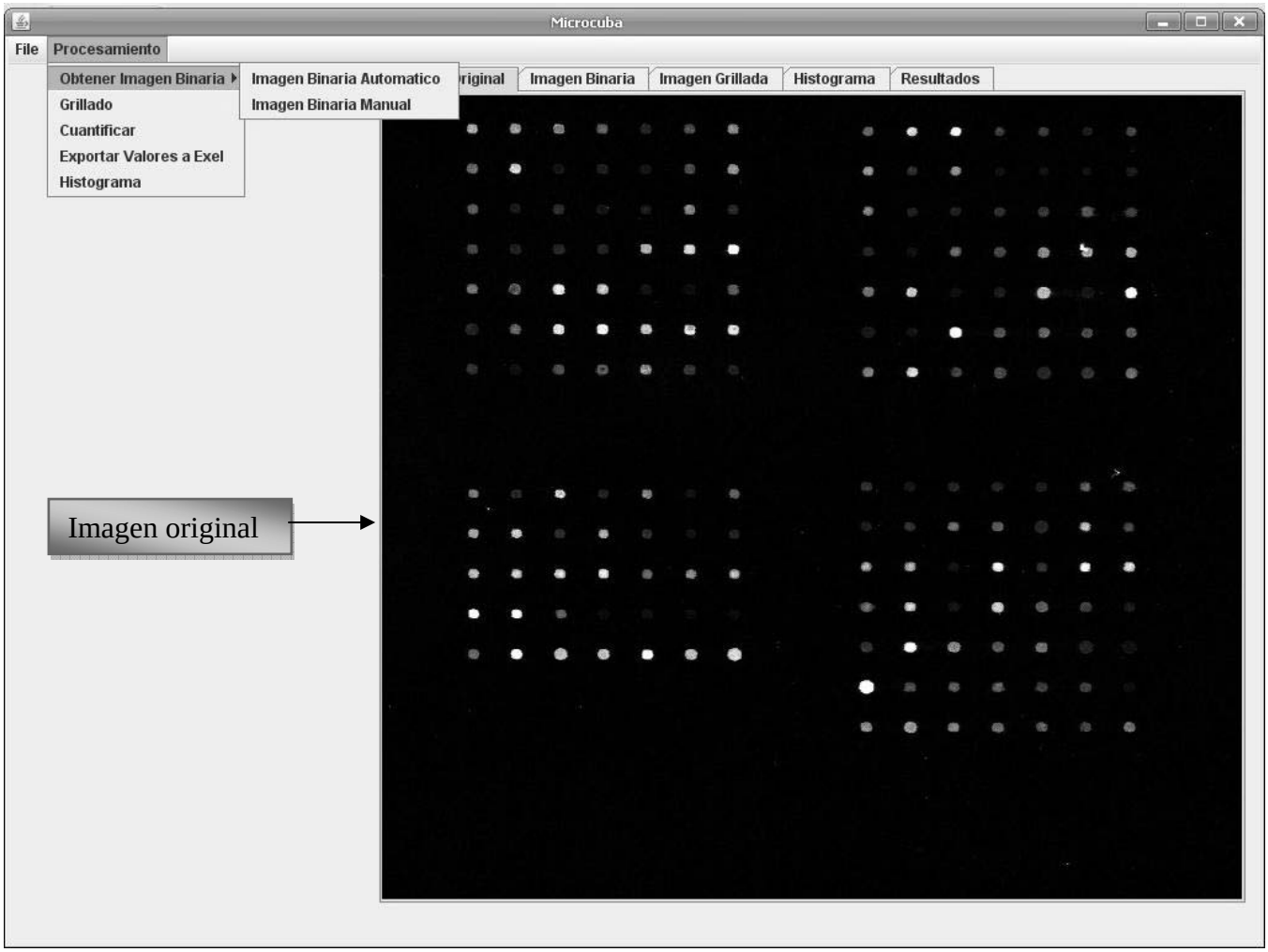


Figura 3.7. La imagen se encuentra cargada, se muestran opciones de segmentar por umbral automático o manual.

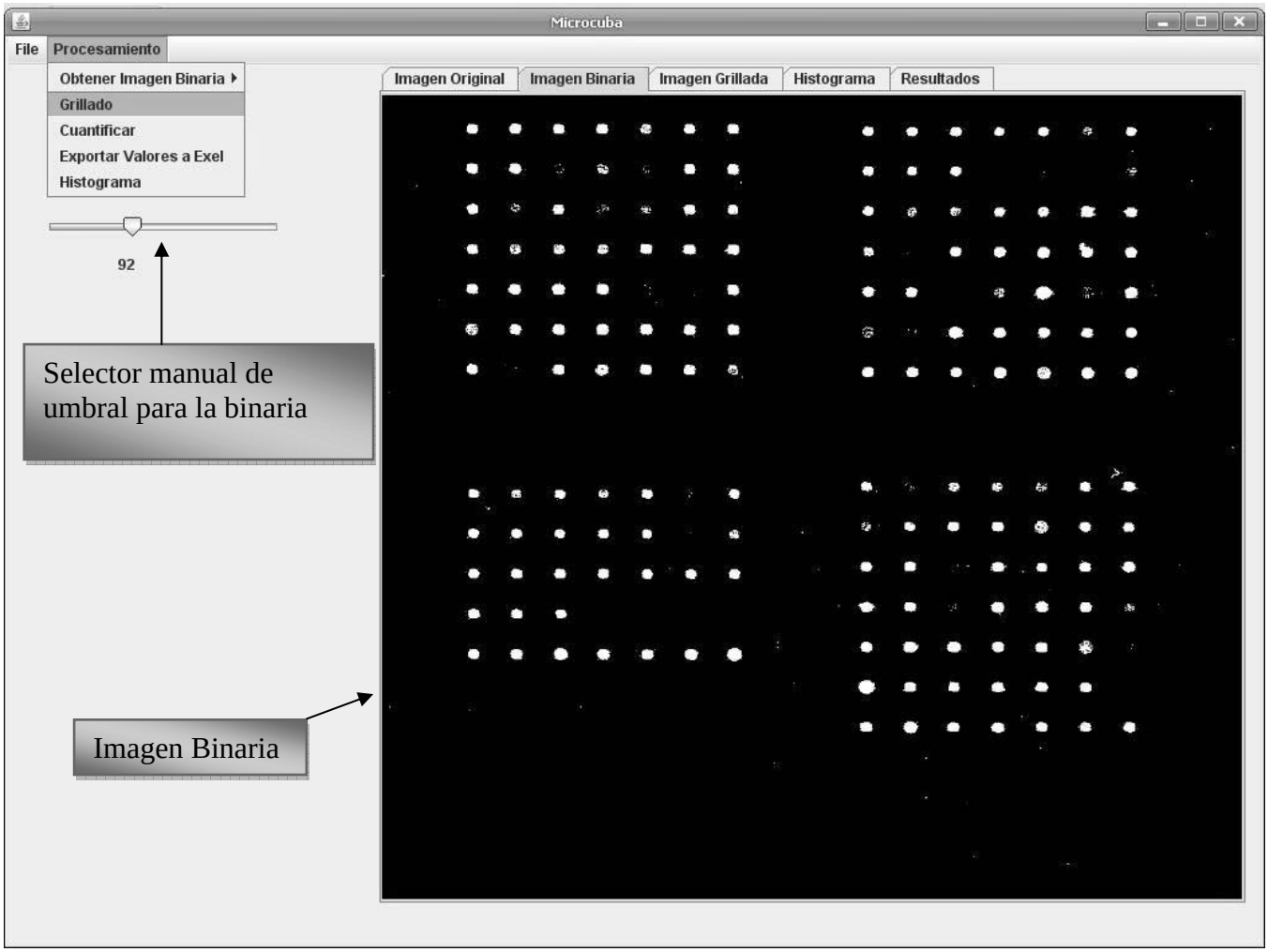


Figura 3.8. Se muestra la imagen binaria.

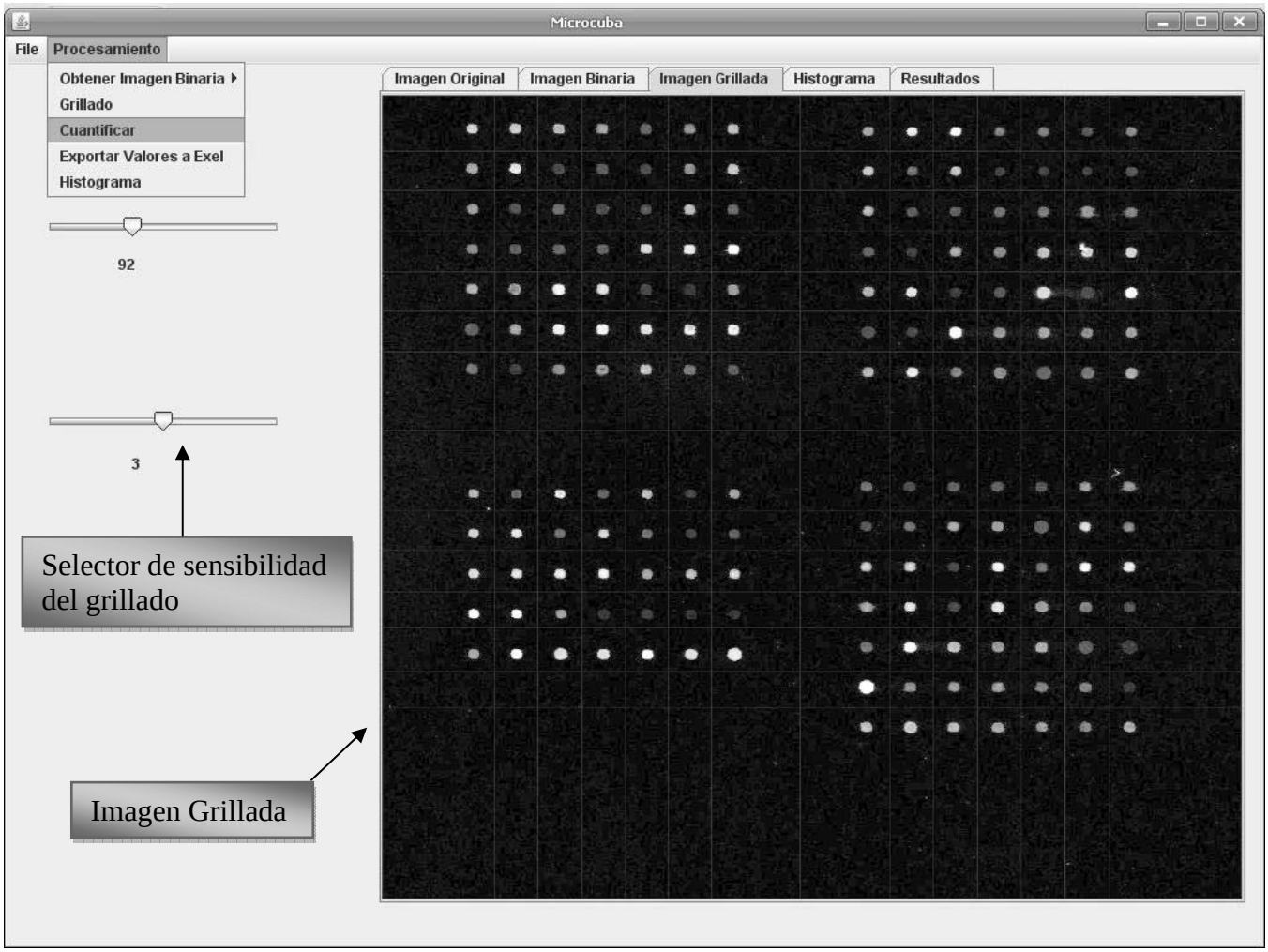


Figura 3.9. Se muestra imagen grillada así como el selector de sensibilidad del grillado.

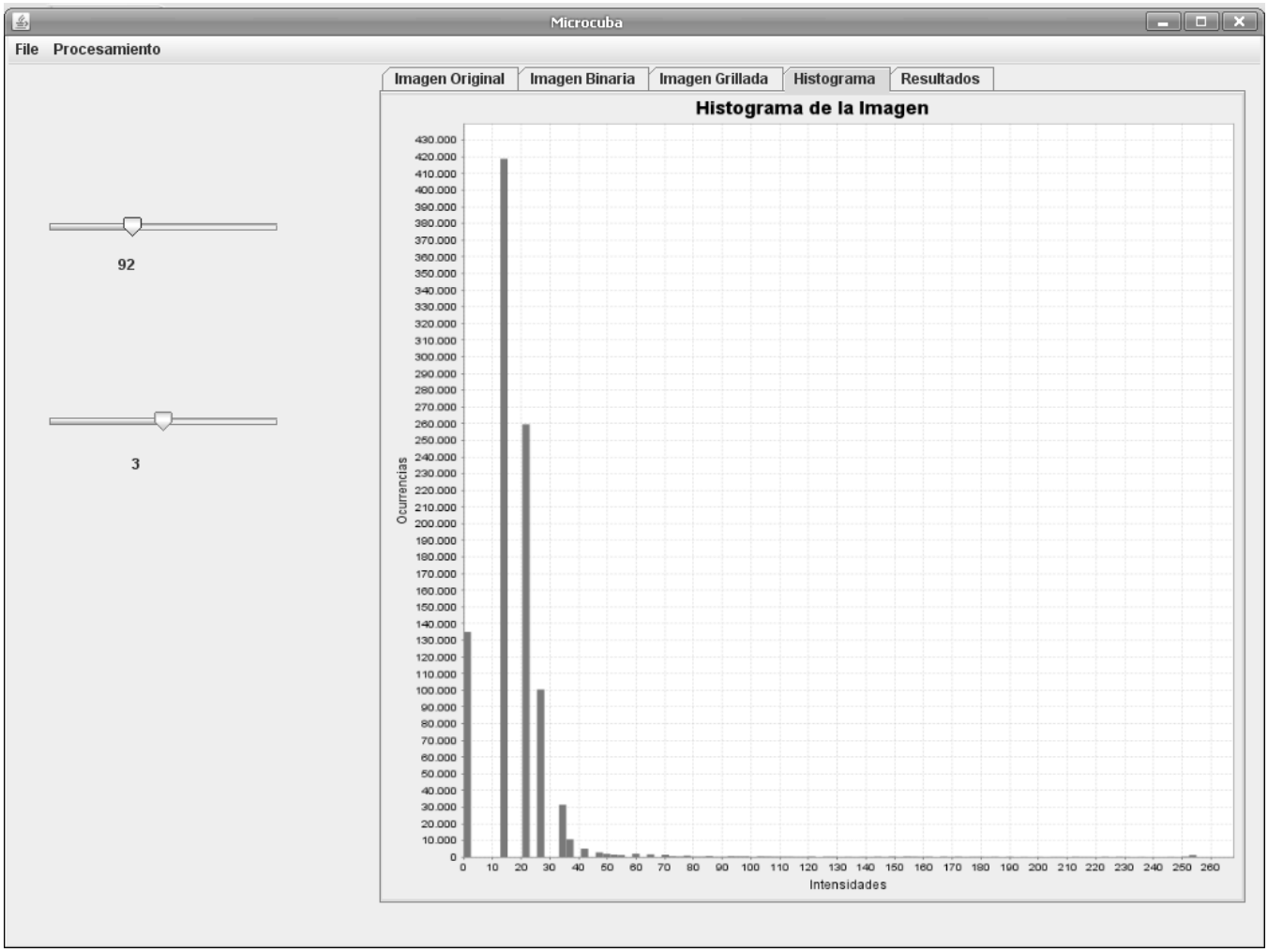
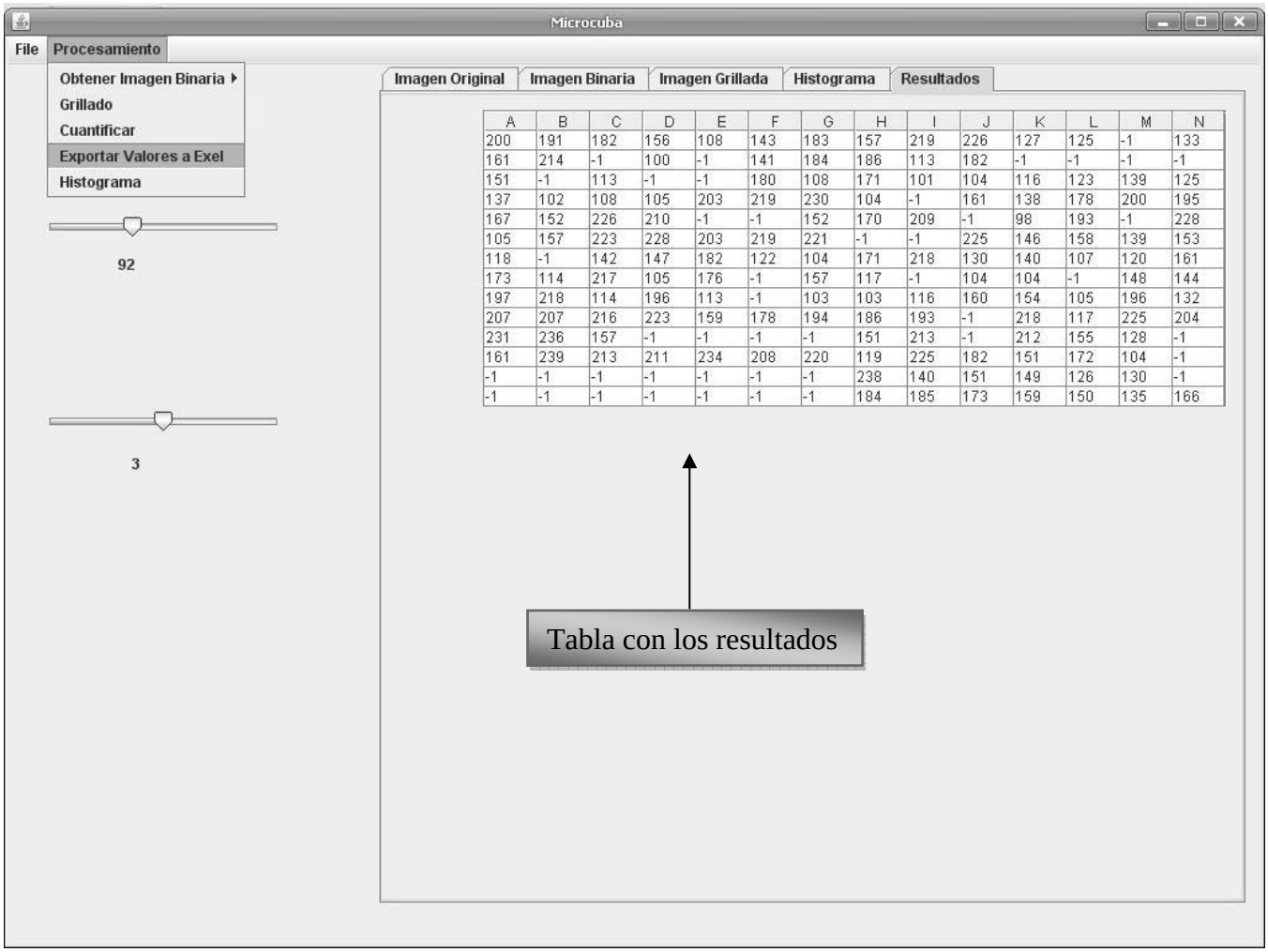


Figura 3.10. Vista del histograma de la imagen, se observa en la gráfica los valores de intensidad posibles en el eje X y la cantidad de ocurrencias en la imagen en el eje Y.



A	B	C	D	E	F	G	H	I	J	K	L	M	N
200	191	182	156	108	143	183	157	219	226	127	125	-1	133
161	214	-1	100	-1	141	184	186	113	182	-1	-1	-1	-1
151	-1	113	-1	-1	180	108	171	101	104	116	123	139	125
137	102	108	105	203	219	230	104	-1	161	138	178	200	195
167	152	226	210	-1	-1	152	170	209	-1	98	193	-1	228
105	157	223	228	203	219	221	-1	-1	225	146	158	139	153
118	-1	142	147	182	122	104	171	218	130	140	107	120	161
173	114	217	105	176	-1	157	117	-1	104	104	-1	148	144
197	218	114	196	113	-1	103	103	116	160	154	105	196	132
207	207	216	223	159	178	194	186	193	-1	218	117	225	204
231	236	157	-1	-1	-1	-1	151	213	-1	212	155	128	-1
161	239	213	211	234	208	220	119	225	182	151	172	104	-1
-1	-1	-1	-1	-1	-1	-1	238	140	151	149	126	130	-1
-1	-1	-1	-1	-1	-1	-1	184	185	173	159	150	135	166

Figura 3.11. Se muestran los resultados del proceso en una tabla, nótese que los valores -1 denotan los spots que no se manifestaron.

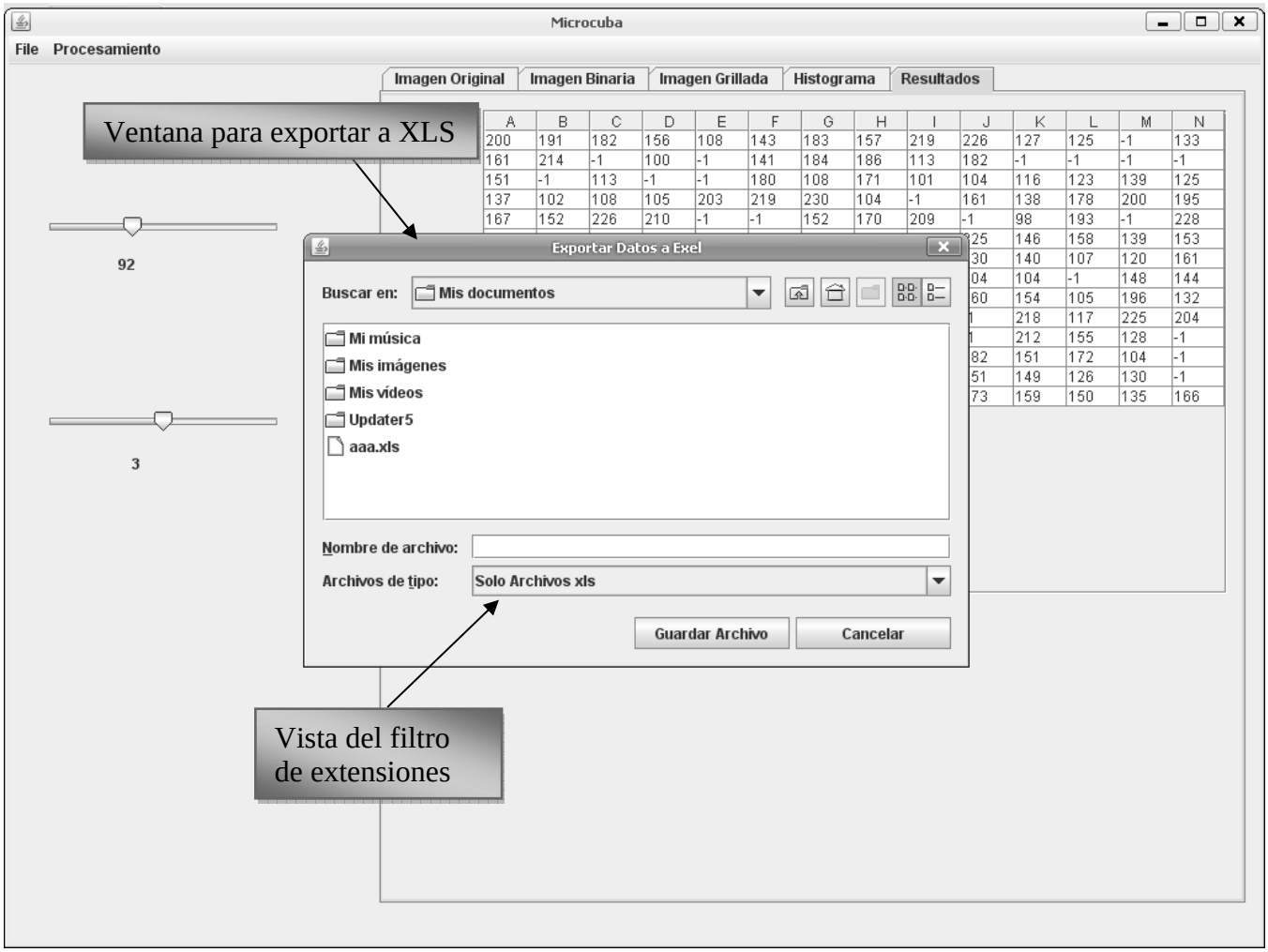


Figura 3.12. Ventana para exportar a .xls, nótese el filtro de extensiones.

3.6. Conclusiones

En este capitulo se mostró la implementación de las funcionalidades del software utilizando las herramientas seleccionadas y respetando las clases y métodos del diseño previamente realizado.

Se mostraron los algoritmos más significativos del software y se explicaron sus fundamentos matemáticos, lo cual conforma la primera entrega de la aplicación.

Conclusiones

- Se diseñó e implementó una aplicación multiplataforma para el proceso y cuantificación de imágenes *microarray* a partir de las bibliotecas JAI de Sun para el procesamiento de imágenes y JexcelAPI para el tabulado de los resultados.
- Se implementaron funcionalidades que permiten cargar y mostrar imágenes en Java.
- Se implementaron métodos que permiten hallar un umbral óptimo global basado en la imagen para generar una imagen binaria.
- Se implementaron métodos que permiten grillar la imagen, detectando así las posiciones de los spots.
- Se implementó una funcionalidad que permite mostrar los resultados en la aplicación.
- Se implementó un método que permite exportar estos resultados a formato XLS.
- Se implementó una funcionalidad que permite mostrar el histograma de la imagen.
- Se implementó una comunicación efectiva entre los métodos y funcionalidades que constituyen los pasos del procesamiento de imágenes *microarray*.

RECOMENDACIONES

A partir de las experiencias obtenidas en el desarrollo del trabajo y con vistas de lograr un aprovechamiento óptimo del resultado alcanzado se recomienda:

- Aplicar el software en todas las instituciones nacionales donde se apliquen técnicas de *microarray*.
- Incorporar técnicas de minería de datos para comparar resultados de diferentes estudios de imágenes *microarray* ya procesadas.

Referencias Bibliográficas

[1]. GARRIDO, Y. V. *Software para el Procesamiento de Imágenes de Microarray.* C.Habana, UCI, 2007. 101. p.

[2]. *Imaging Research Inc.* Disponible en: http://www.imagingresearch.com/products/ARV_details.asp

[3]. LUMONICS, G. *QuantArray Quantitative Microarray Analysis*, 2004. [Disponible en: http://www.bipl.ahc.umn.edu/quantarray.html.

[4]. Biodiscovery, Leading-Edge Microarray Analysis Software, 2008. [2008]. Disponible en: http://www.biodiscovery.com/index/imagene

[5]. *Spotfinder*. Dana-Farber Cancer Institute, 2005. [2008]. Disponible en: http://www.tm4.org/spotfinder.html

[6]. EISEN, M. *ScanAlyze*, Lawrence Berkeley National Laboratory, 2002. [2008]. Disponible en: http://rana.lbl.gov.

[7]. STROUSTRUP, B. *The C++ Programming Language*. 3rd. Addison-Wesley, 1997. p. 9780201889543

[8]. BUTOW, E. and T. RYAN. *C#: Your Visual Blueprint for Building .Net Applications*. Visual; Pap/Cdr edition, 2001. 352 p. 978-0764536014

[9]. DUQUE, R. G. *Una no tan breve historia de Java*, 2007. [2008]. Disponible en: http://mundogeek.net/archivos/2004/10/04/una-no-tan-breve-historia-de-java/

[10]. ---. *Java Advanced Imaging*, Wikipedia, The Free Encyclopedia., 2008. [2008]. Disponible en: http://en.wikipedia.org/w/index.php?title=Java_Advanced_Imaging&oldid=205787760

[11]. KHAN, A. *Java Excel API - A Java API to read, write, and modify Excel spreadsheets*, Sourceforge.net, 2008]. Disponible en: http://jexcelapi.sourceforge.net/

[12]. GONZÁLEZ, A. M. *Tratamiento Computacional de Imágenes*, 2005. [2008]. Disponible en: http://gva1.dec.usc.es/~antonio/docencia/20042005tci/teoria/similaridad/Similaridad.html

[13]. LIAO, P.-S.; T.-S. CHEN*, et al.* A Fast Algorithm for Multilevel Thresholding *Journal of Information Science and Engineering*, 2001: 713-727.

[14]. OTSU, N. *A threshold selection method from gray-level histogram*. *IEEE Transactions on*

System Man Cybernetics,, 1979. SMC-9: 62-66.

[15]. SCHENA, M.; D. SHALON*, et al.* Quantitative monitoring of gene expression patterns with a complementary DNA microarray, 2004: 270-467.

Bibliografía

[1]. *Imaging Research Inc.* Disponible en: http://www.imagingresearch.com/products/ARV_details.asp

[2]. *Spotfinder*. Dana-Farber Cancer Institute, 2005. [2008]. Disponible en: http://www.tm4.org/spotfinder.html

[3]. BIDARTE, U. *Adaptive Image Brightness and Contrast Enhancement Circuit for Real-Time Vision Systems*, Conf. on Industrial Technology, 2000. Vol. 1.

[4]. BUTOW, E. and T. RYAN. *C#: Your Visual Blueprint for Building .Net Applications*. Visual; Pap/Cdr edition, 2001. 352 p. 978-0764536014

[5]. CASTLEMAN, K. *Digital Imagen Processing*. 1996. p. *Prentice Hall.*

[6]. CHEN, S. *Contrast Enhancement using Recursive Mean-Separate Histogram Equalization for Scalable Brightness Preservation*. 2003. p. *Transactions on Consumer Electronics.*

[7]. CHOU, Y.-L. *Analisis Estadistico*. Nueva Editorial Interamericana, 1972. p.

[8]. CONTRIBUTORS, W. *C Sharp (programming language)*, Wikipedia, The Free Encyclopedia., 2007. [2008]. Disponible en: http://en.wikipedia.org/w/index.php?title=C_Sharp_%28programming_language%29&oldid=219445821

[9]. ---. *Java Advanced Imaging*, Wikipedia, The Free Encyclopedia., 2008. [2008]. Disponible en: http://en.wikipedia.org/w/index.php?title=Java_Advanced_Imaging&oldid=205787760

[10]. DAVEY, S. J.; S. B. COLEGROVE*, et al. Advanced Jindalee Tracker: Enhanced Peak Detector*. *Surveillance Systems Division Electronics and Surveillance Research Laboratory*, 1999.

[11]. DUQUE, R. G. *Una no tan breve historia de Java*, 2007. [2008]. Disponible en: http://mundogeek.net/archivos/2004/10/04/una-no-tan-breve-historia-de-java/

[12]. EISEN, M. *ScanAlyze*, Lawrence Berkeley National Laboratory, 2002. [2008]. Disponible en: http://rana.lbl.gov.

[13]. GARCÍA, J. G. *Análisis de Datos de Microarray de Expresión*, 2007. 9.

[14]. GARRIDO, Y. V. *Software para el Procesamiento de Imágenes de Microarray.*

C.Habana, UCI, 2007. 101. p.

[15]. GOH, K. H. *Automatic Video Contrast Enhancement*, Symposium on Consumer Electronics, 2004.

[16]. GONZÁLEZ, A. M. *Tratamiento Computacional de Imágenes*, 2005. [2008]. Disponible en: http://gva1.dec.usc.es/~antonio/docencia/20042005tci/teoria/similaridad/Similaridad.html

[17]. GONZALEZ, R. C. and R. E. WOODS. *Digital Image Processing*. 2. Prentice Hall, 2002. 793 p. 0201180758

[18]. J.R., P. *Practical Computer Vision using C*. 1993. p.

[19]. JONES. *Métodos Multi Grillas*. *Introducción a los LMGM*, 2005.

[20]. KELMANSKY, D. M. *Aspectos Estadísticos de Microarrays*, 2007. 21.

[21]. KHAN, A. *Java Excel API - A Java API to read, write, and modify Excel spreadsheets*, Sourceforge.net, 2008]. Disponible en: http://jexcelapi.sourceforge.net/

[22]. KIM, Y. *Contrast Enhancement using Brightness preserving Bi-histogram equalization*. 1997. p. *Transactions on Consumer Electronics.*

[23]. LIAO, P.-S.; T.-S. CHEN*, et al.* A Fast Algorithm for Multilevel Thresholding *Journal of Information Science and Engineering*, 2001: 713-727.

[24]. LUMONICS, G. *QuantArray Quantitative Microarray Analysis*, 2004. [Disponible en: http://www.bipl.ahc.umn.edu/quantarray.html.

[25]. OTSU, N. *A threshold selection method from gray-level histogram*. *IEEE Transactions on System Man Cybernetics,*, 1979. SMC-9: 62-66.

[26]. SCHENA, M.; D. SHALON*, et al.* Quantitative monitoring of gene expression patterns with a complementary DNA microarray, 2004: 270-467.

[27]. STROUSTRUP, B. *The C++ Programming Language*. 3rd. Addison-Wesley, 1997. p. 9780201889543

[28]. SUN MICROSYSTEMS, I. *The Java Tutorial*. ON-LINE, Sun MicroSystems, 1995.

[29]. ---. *Programming in Java Advanced Imaging*. ONLINE., Sun MicroSystems, 1999.

[30]. ZUIDERVELD, K. Contrast Limited Adaptive Histogram Equalization *Academic Press*, 1994.

Anexos

Anexo 1: Diagramas de Clases

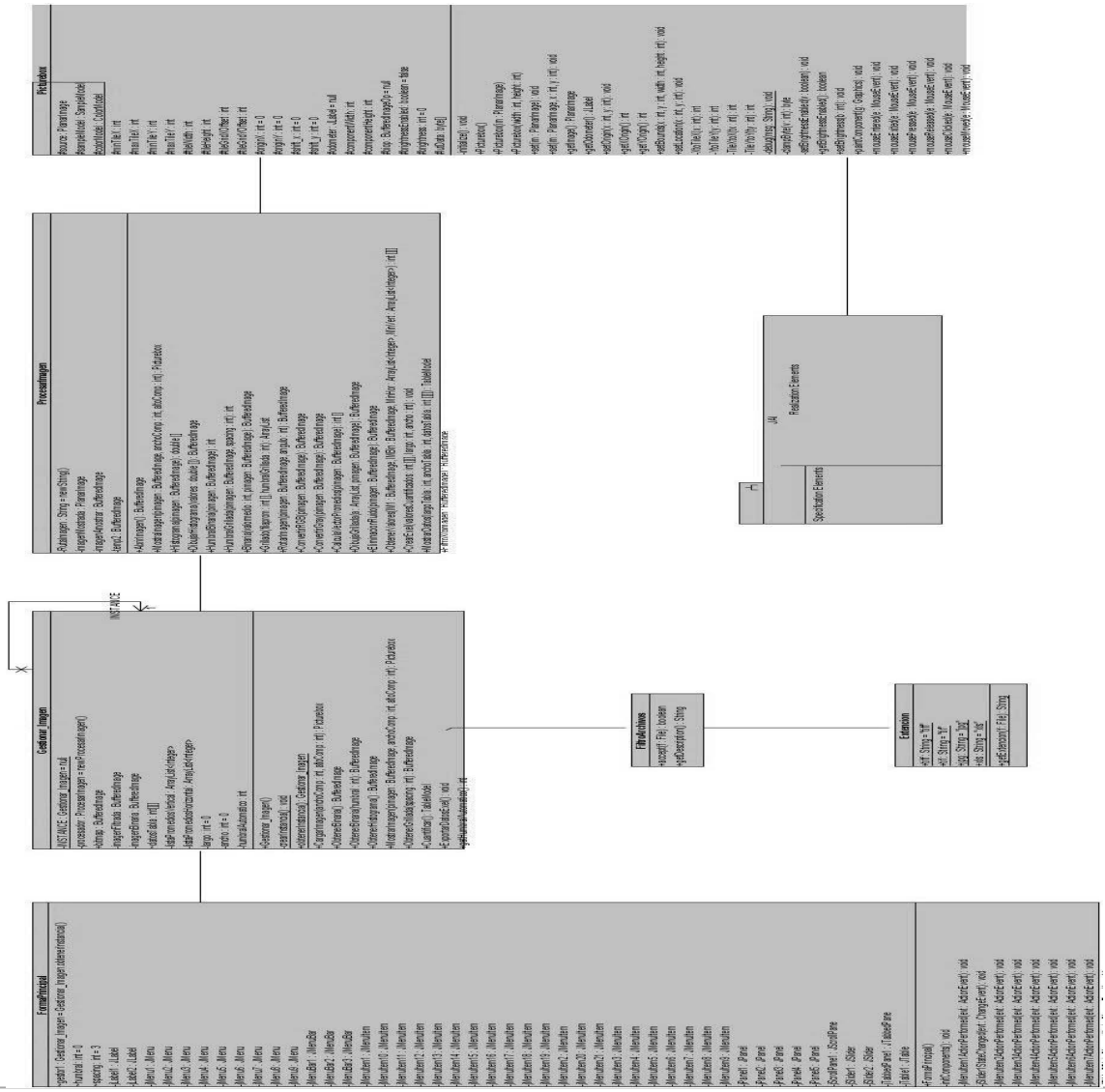


Figura A.1. Diagrama de clases del diseño del caso de uso “Cargar Imagen”

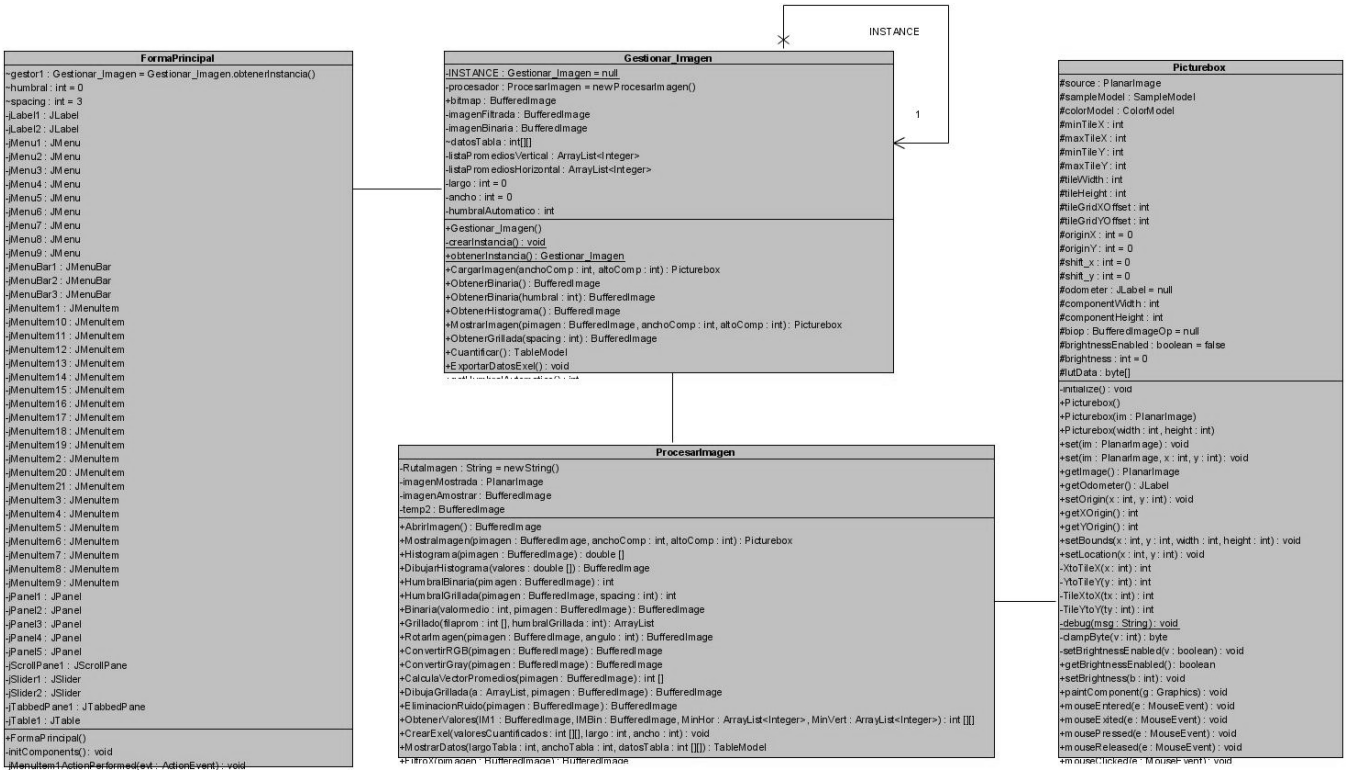


Figura A.2. Diagrama de clases del diseño del caso de uso “Grillar Imagen”

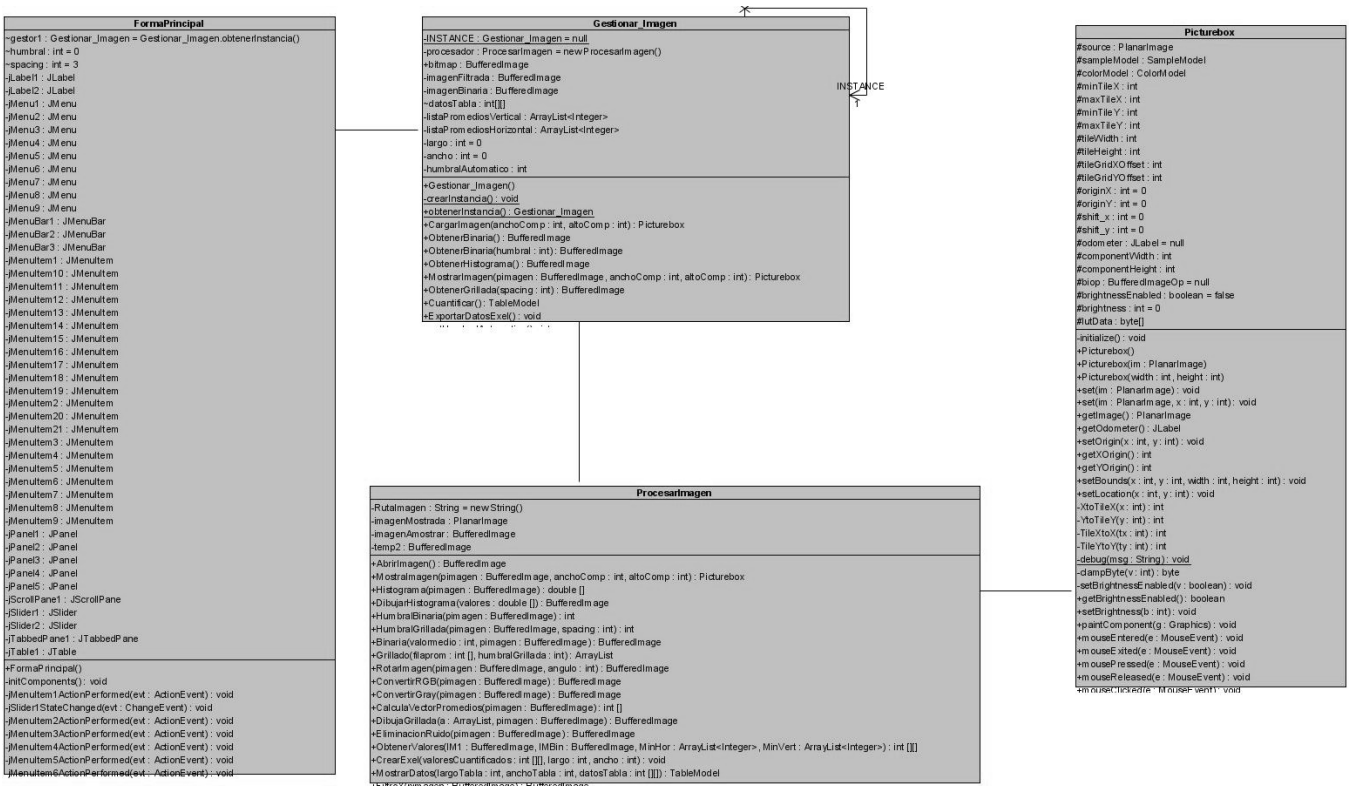


Figura A.3: Diagrama de clases del diseño del caso de uso “Segmentar Imagen”

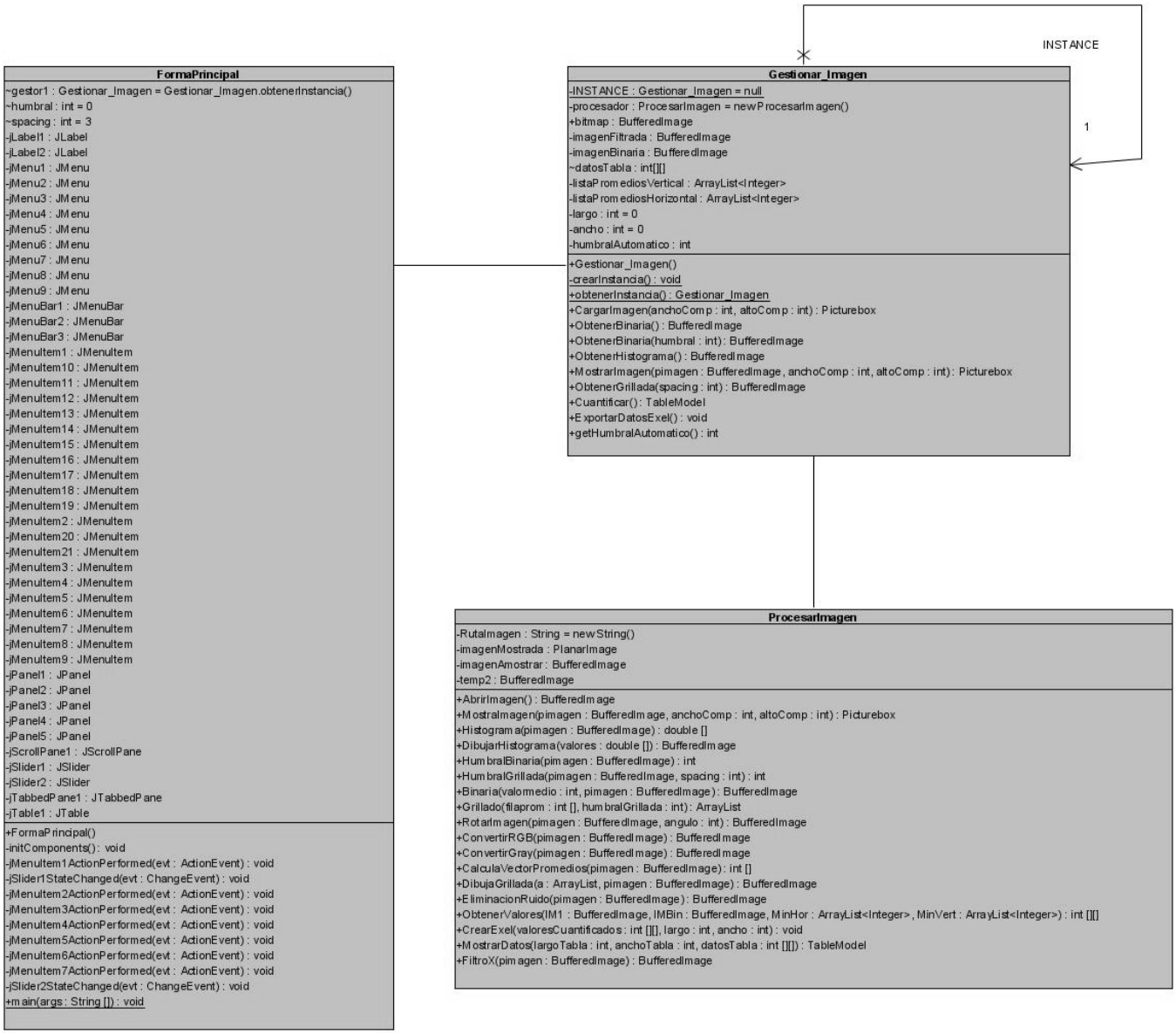


Figura A.4: Diagrama de clases del diseño del caso de uso “Cuantificar Valores”

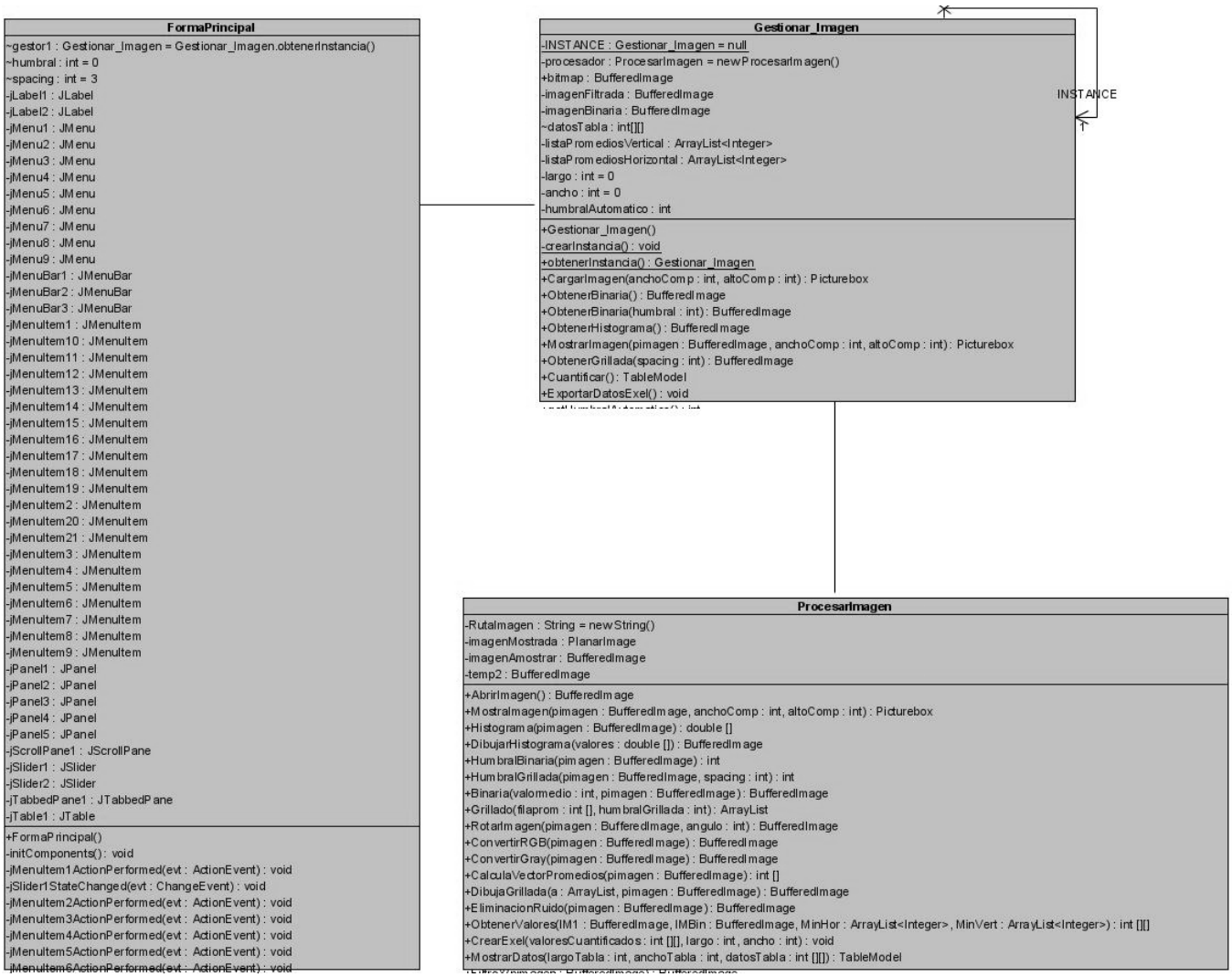


Figura A.5: Diagrama de clases del diseño del caso de uso “Tabular Valores”

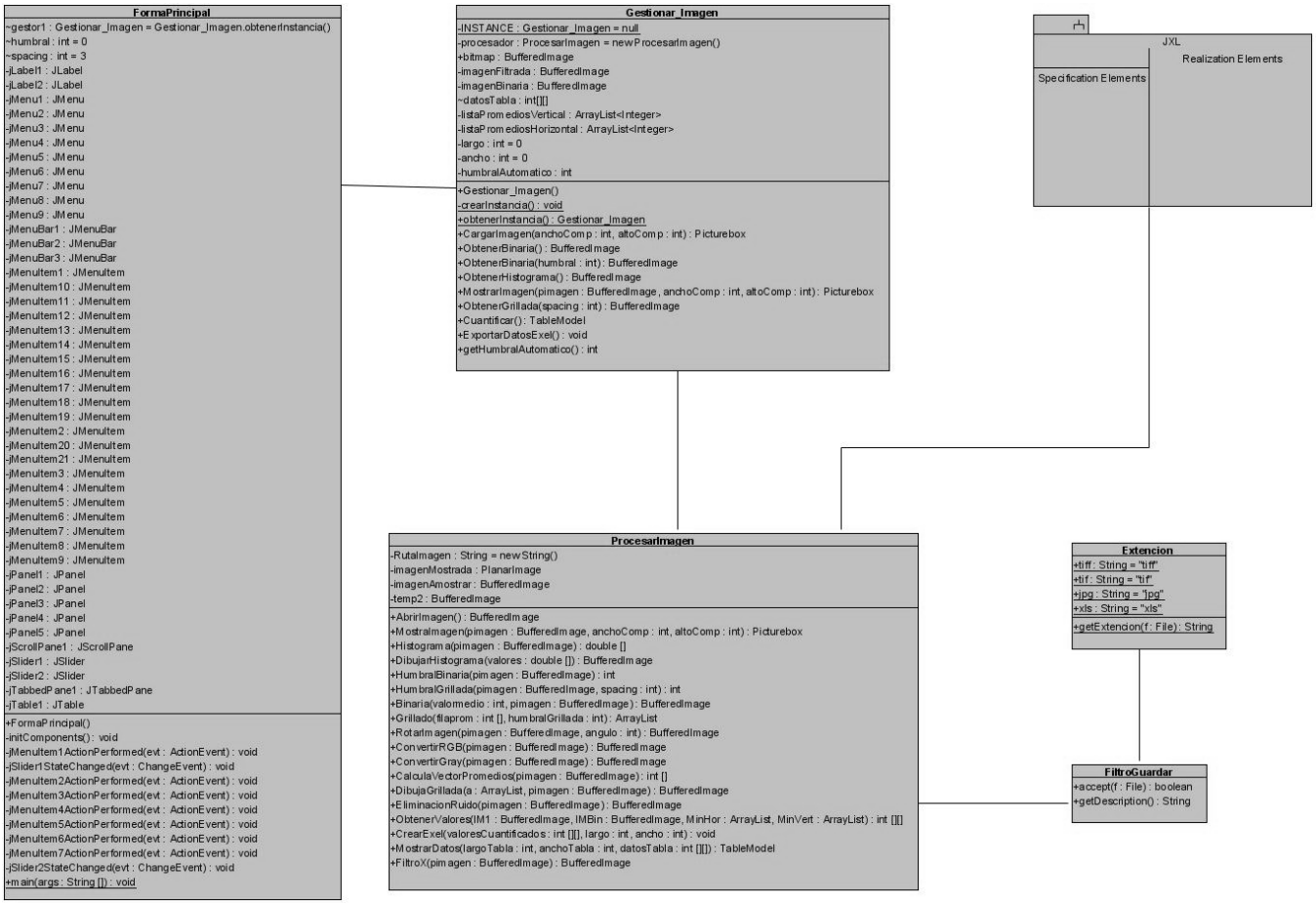


Figura A.6: Diagrama de clases del diseño del caso de uso “Exportar Valores”

Anexo 2: Descripción de las clases del diseño

Nombre: FormaPrincipal	
interfaz	
Atributo	**Tipo**
Gestor1	Gestionar_Imagen
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla A.1: Descripción de la clase FormaPrincipal

Nombre: Gestionar_Imagen	
Controladora	
Atributo	**Tipo**
procesador	Procesar_Imagen
bitmap	BufferedImage
imagenBinaria	Bufferedimage
ListaPromediosVertical	ArrayList<Integer>
ListaPromediosHorizontal	ArrayList<Integer>
humbralAutomatico	int
Para cada responsabilidad:	
Nombre:	crearInstancia()
Descripción:	Crea una instanci de la clase Gestionar_Imagen.
Nombre:	Cargar Imagen(JPanel panel)
Descripción	Muestra una imagen en un jpanel cargada desde un archivo.
Nombre:	ObtenerBinaria(int humbral)
Descripción:	Obtiene una imagen binaria a partir de un umbral.
Nombre:	ObtenerGrillada()
Descripción:	Obtiene una imagen grillada para mostrar
Nombre:	Cuantificar()
Descripción:	Extrae los coeficientes de intensidad por cada Spot y los guarda en un arreglo bidimensional.

Tabla A.2: Descripción de la clase Gestionar_Imagen

Nombre: ProcesarImagen	
controladora	
Atributo	**Tipo**
ElegirArchivo	JFileChooser
FiltroTiif	FiltroArchivos
RutaImagen	String
imagenMostrada	PlanarImage
imagenAmostrar	BufferedImage
temp2	BufferedImage
Para cada responsabilidad:	
Nombre:	AbrirImagen()
Descripción:	Convierte una imagen desde archivo a bufferedimage
Nombre:	MostraImagen(BufferedImage pimagen,JPanel panel)
Descripción:	Muestra una imagen en un jpanel
Nombre:	HumbralBinaria(BufferedImage pimagen)
Descripción:	Calcula el umbral para la posterior binarización
Nombre:	HumbralGrillada(BufferedImage pimagen)
Descripción:	Calcula el umbral para la posterior grillado
Nombre:	Binaria(int valormedio,BufferedImage pimagen)
Descripción:	Obtiene una imagen binaria a partir de un umbral y una imagen
Nombre:	Grillado(int[] filaprom, int humbralGrillada)
Descripción:	Calcula los vectores del grillado
Nombre:	RotarImagen(BufferedImage pimagen,int angulo)
Descripción:	Rota la imagen para el grillado
Nombre:	ConvertirRGB(BufferedImage pimagen)
Descripción:	Cambia el formato de la imagen a 24 bits
Nombre:	ConvertirGray(BufferedImage pimagen)
Descripción:	Convierte la imagen a formato de 8 bits
Nombre:	CalculaVectorPromedios(BufferedImage pimagen)
Descripción:	Calcula los vectores de los valores medios de las columnas de pixeles
Nombre:	DibujaGrillada(ArrayList a,BufferedImage pimagen)
Descripción:	Dibuja los separadores entre las columnas de spot sobre una imagen
Nombre:	ObtenerValores(BufferedImage IM1, BufferedImage
Descripción:	IMBin, ArrayList<Integer> MinHor, ArrayList<Integer> MinVert) Obtiene los valores promedio de cada Spot

Tabla A.3: Descripción de la clase Gestionar_Imagen

Anexo 3: Diagramas de Secuencia

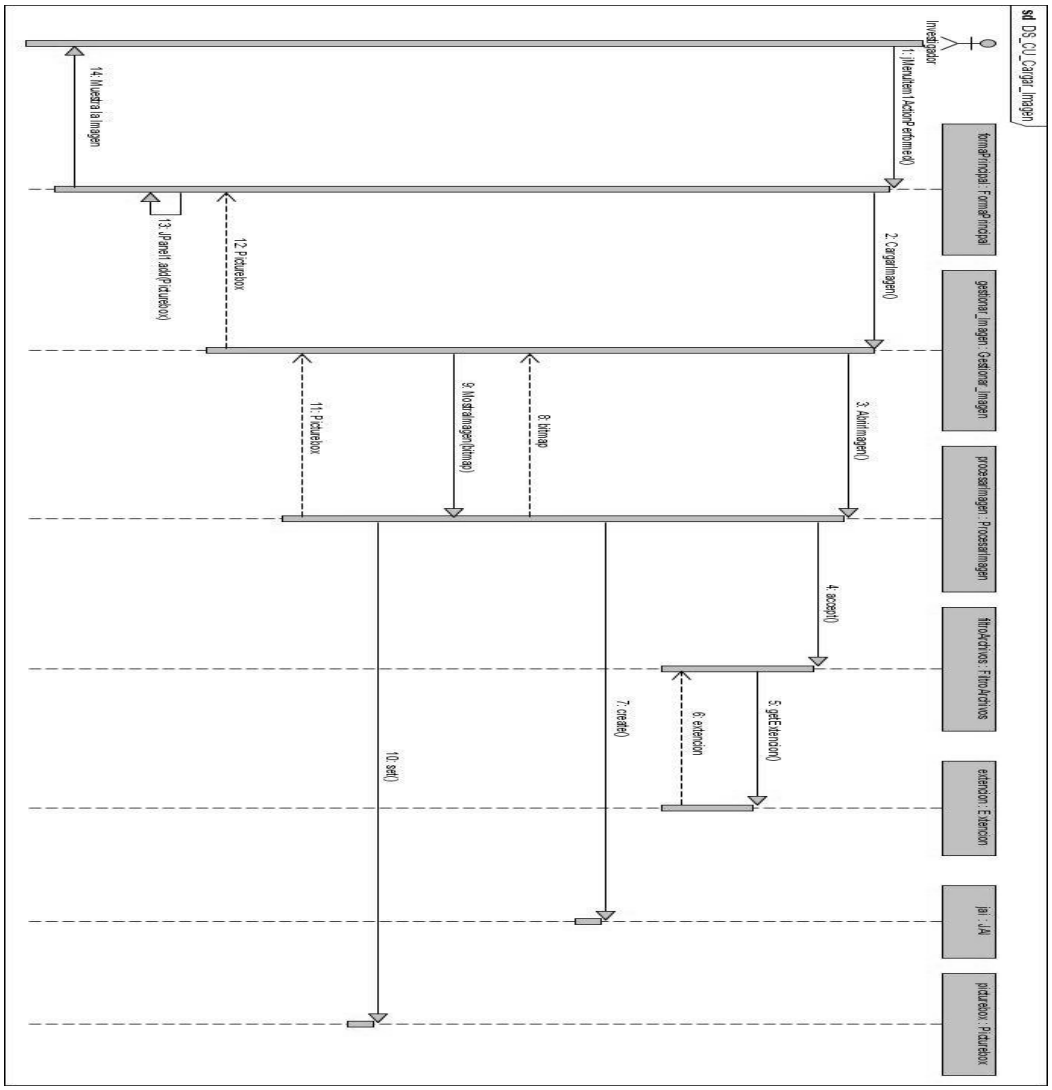


Figura A.7: Diagrama de secuencia del caso de uso “Cargar Imagen”

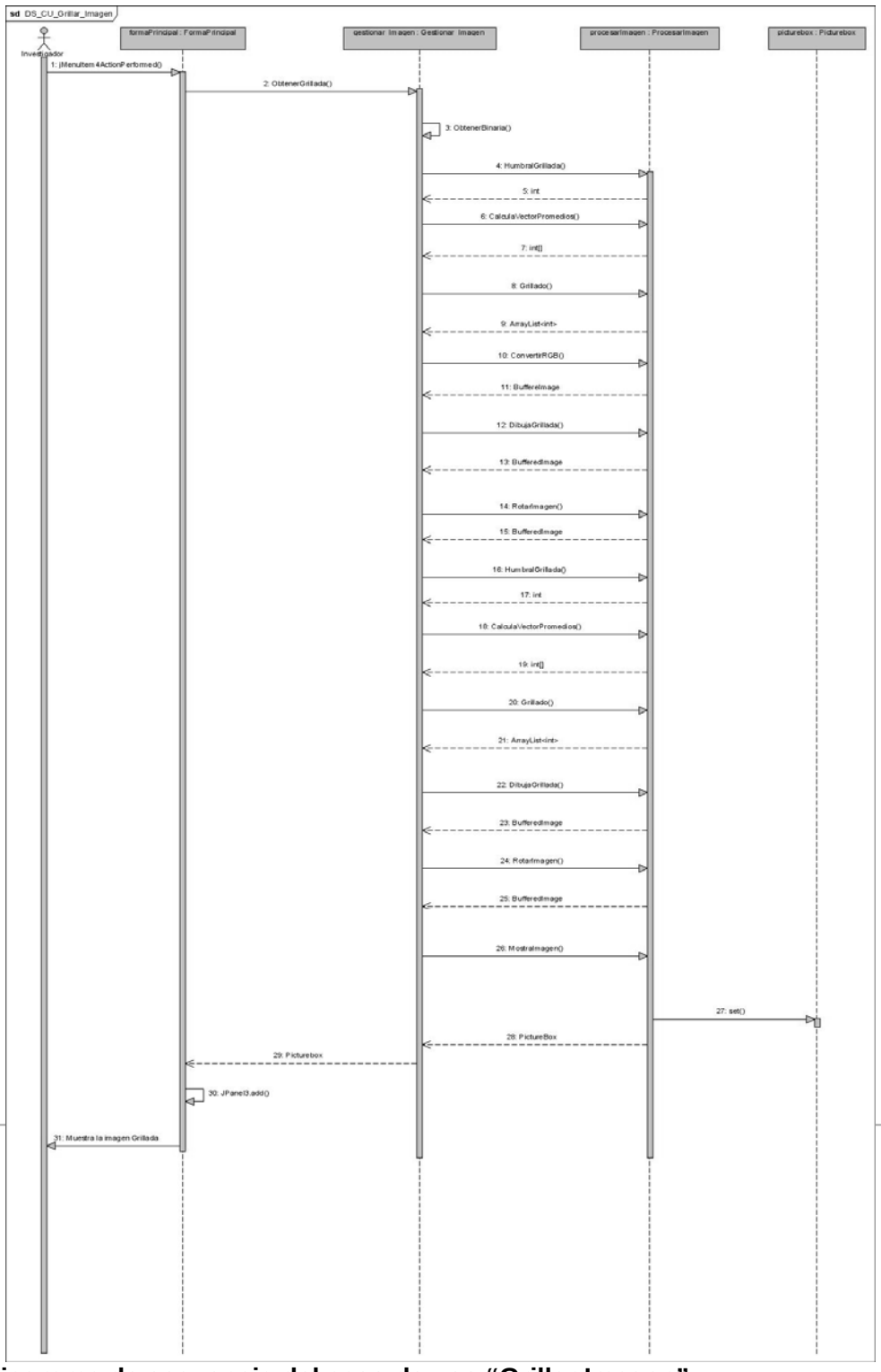


Figura A.8: Diagrama de secuencia del caso de uso “Grillar Imagen”

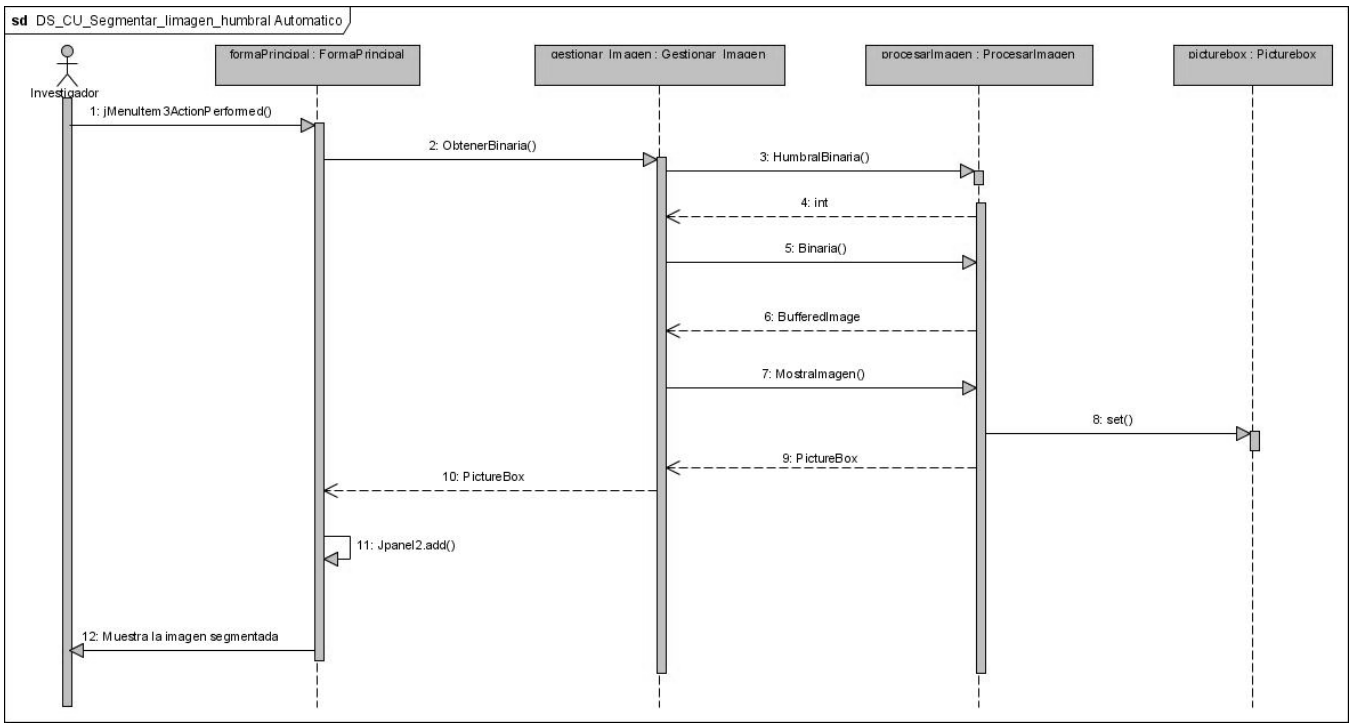


Figura A.9: Diagrama de secuencia del caso de uso "Segmentar imagen (automático)"

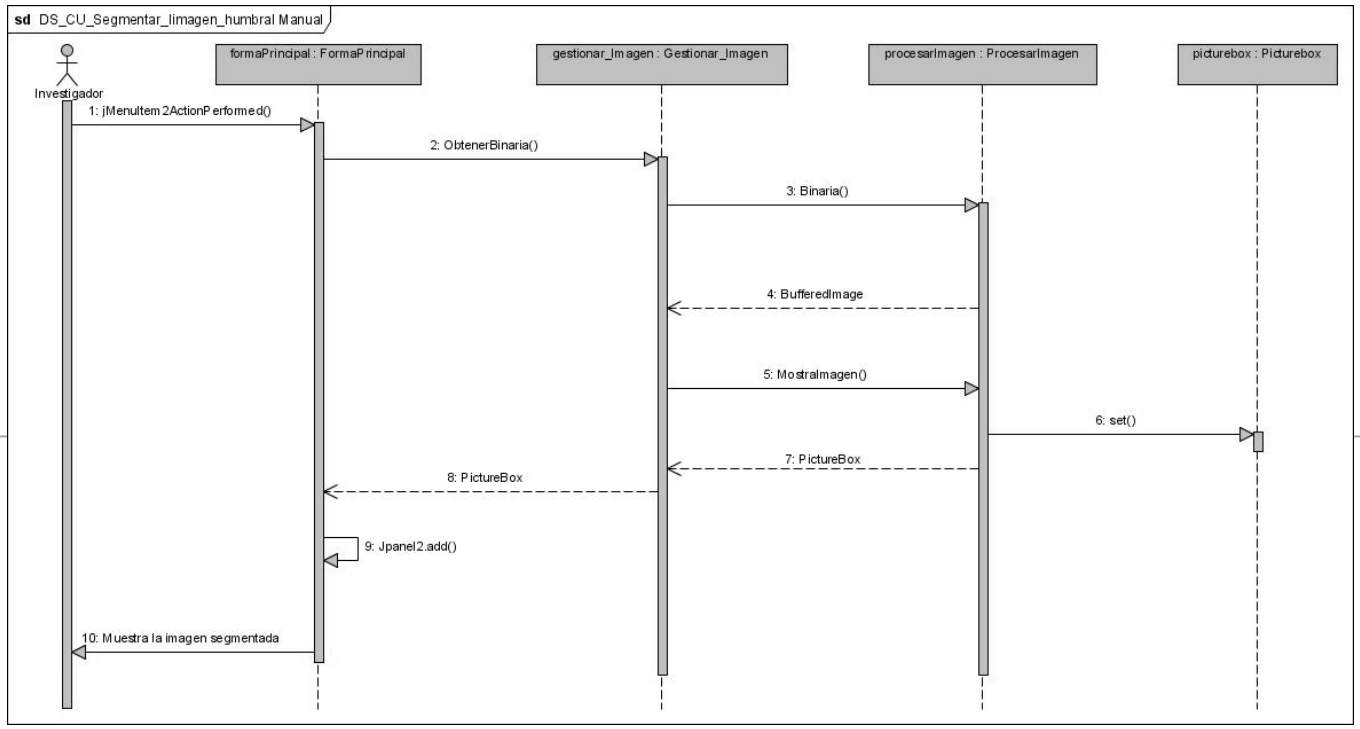


Figura A.10: Diagrama de secuencia del caso de uso "Segmentar Imagen (manual)"

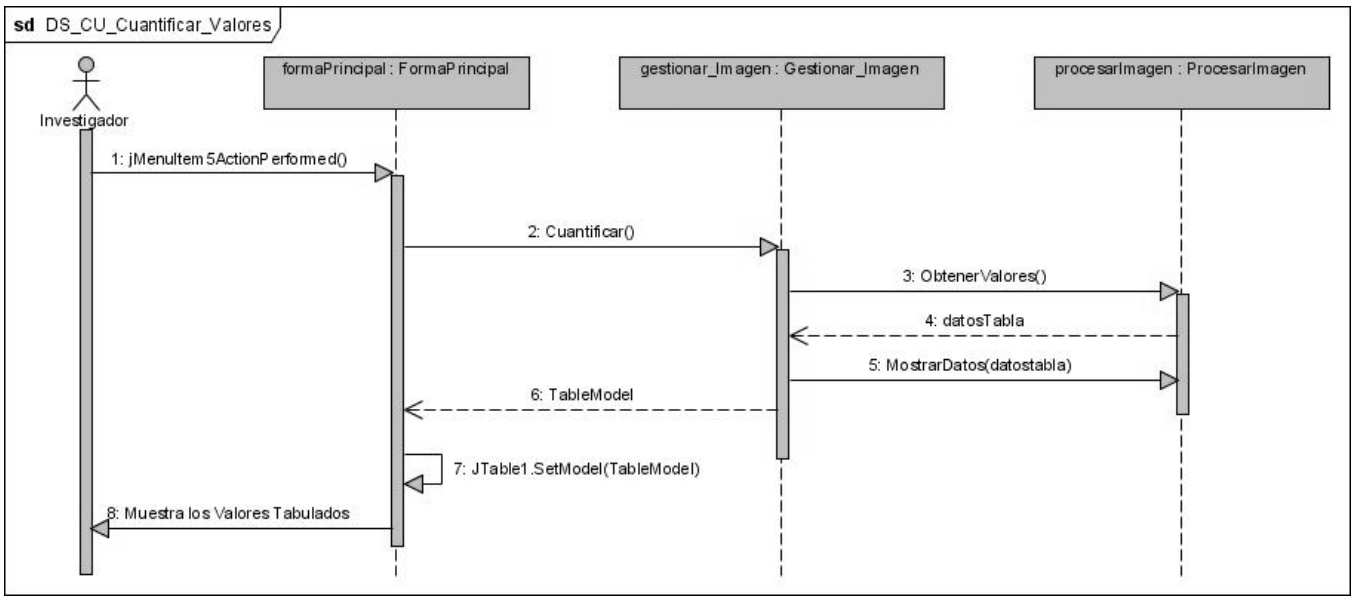


Figura A.11: Diagrama de secuencia del caso de uso "Cuantificar Valores"

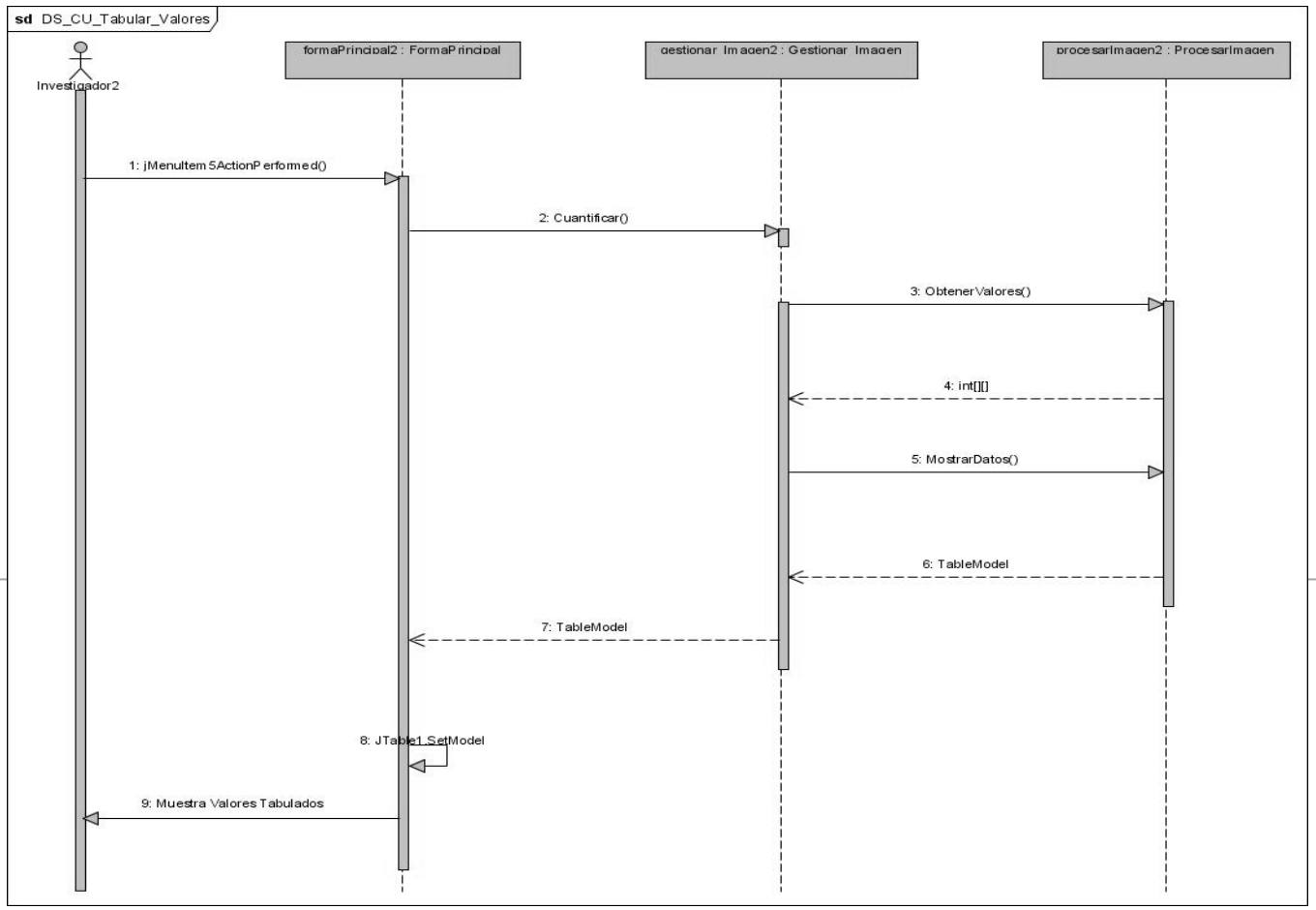


Figura A.12: Diagrama de secuencia del caso de uso "Tabular Valores"

Glosario de Términos

A

ADN: abreviatura del ácido desoxirribonucleico; es el componente químico primario de los cromosomas y el material en el que los genes están codificados.

B

Bytecodes: Tipo de código intermedio más abstracto que el código máquina.

C

cADN (ADN complementario): ADN sintetizado por transcriptasa inversa utilizando RNA como plantilla.

E

Extraer Intensidad: calcular para cada *spot* del *microarray* el valor de intensidad de fluorescencia.

F

Fluorescencia: La fluorescencia es la propiedad de una sustancia para emitir luz cuando es expuesta a radiaciones del tipo ultravioleta, rayos catódicos o rayos X.

G

Genoma: término que puede ser utilizado en más de un sentido, en dependencia del contexto, pero se refiere a todo el material genético de una célula u organismo.

Gen: segmento de ADN que sirve como unidad de información hereditaria. Coincide con los *spots* que representan la imagen de *microarray*.

Grillado: es el proceso de asignación de coordenadas a cada uno de los *spots*

H

Hibridización o hibridación: se refiere a uno de los tipos más comunes de técnicas usadas para estudiar el ADN y el ARN, empleando fragmentos de ADN o sondas de cadena sencilla marcada con una molécula.

I

Imagen de *microarray*: provee representaciones numéricas bidimensionales de las intensidades de fluorescencia. Cada número corresponde a la intensidad de un *píxel*, que conforman la imagen.

M

Microarray: método que permite estudiar cómo interactúan entre sí un gran número de genes. Tecnología para investigar los niveles de la expresión de millares de genes simultáneamente. mRNA(RNA mensajero): plantilla para la síntesis de proteínas. La secuencia del mRNA depende de la secuencia de una cadena complementaria de ADN.

Micrótomo: es un dispositivo mecánico que permite la elaboración de cortes finos de muestras (suelen ser tejidos con patologías) para su observación al microscopio.

S

Spot: los *spots* de los *microarray* son regiones fluorescentes localizadas sobre la superficie del sustrato. Segmentación: puede definirse como el proceso de particionar la imagen en diferentes regiones con diferentes propiedades. Permite clasificar los *píxeles* en foreground (*spot* de interés) o background (fondo).

SNP(polimorfismo de un solo nucleótido): Posición de la secuencia de ADN genómico que varía de un individuo a otro, es una variación en la secuencia de ADN que afecta a un solo nucleótido (adenina (A), timina (T), citosina (C) o guanina (G)) del genoma.

O

Oligo: el oligonucleótido es una secuencia corta de cadena única de ADN o ARN. Los oligos se usan a menudo como sondas para detectar ADN o ARN complementario.

P

Píxel: tiene una correspondencia biunívoca con una pequeña porción cuadrada de la imagen, es fácilmente visible si la imagen es ampliada en la pantalla de una computadora.

Procesamiento: el objetivo es extraer, para cada *spot* fijado en el *microarray*, una medida de su intensidad; el procesamiento de imágenes de *microarray* se puede dividir en 3 tareas Grillado, Segmentación, Extraer Intensidades.

Proteína: una molécula compuesta por una o más cadenas de aminoácidos. Las proteínas desempeñan actividades vitales en la célula.

PCR (Reacción en cadena de la polimerasa): es una técnica de biología molecular cuyo objetivo es obtener un gran número de copias de un fragmento de ADN particular, a partir de un mínimo.

U

Umbral: Cantidad mínima de señal que ha de estar presente para ser registrada por un sistema.

UML: Lenguaje unificado de modelado.