UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 7

Trabajo de Diploma para Optar por el Título de

Ingeniero en Ciencias Informáticas

Herramienta para la parametrización, ejecución y monitorización de experimentos con métodos de selección de atributos relevantes en microarreglos de ADN.

Autor: Yoandry González Castro

Tutor: MSc. Yasel Couce Sardiñas

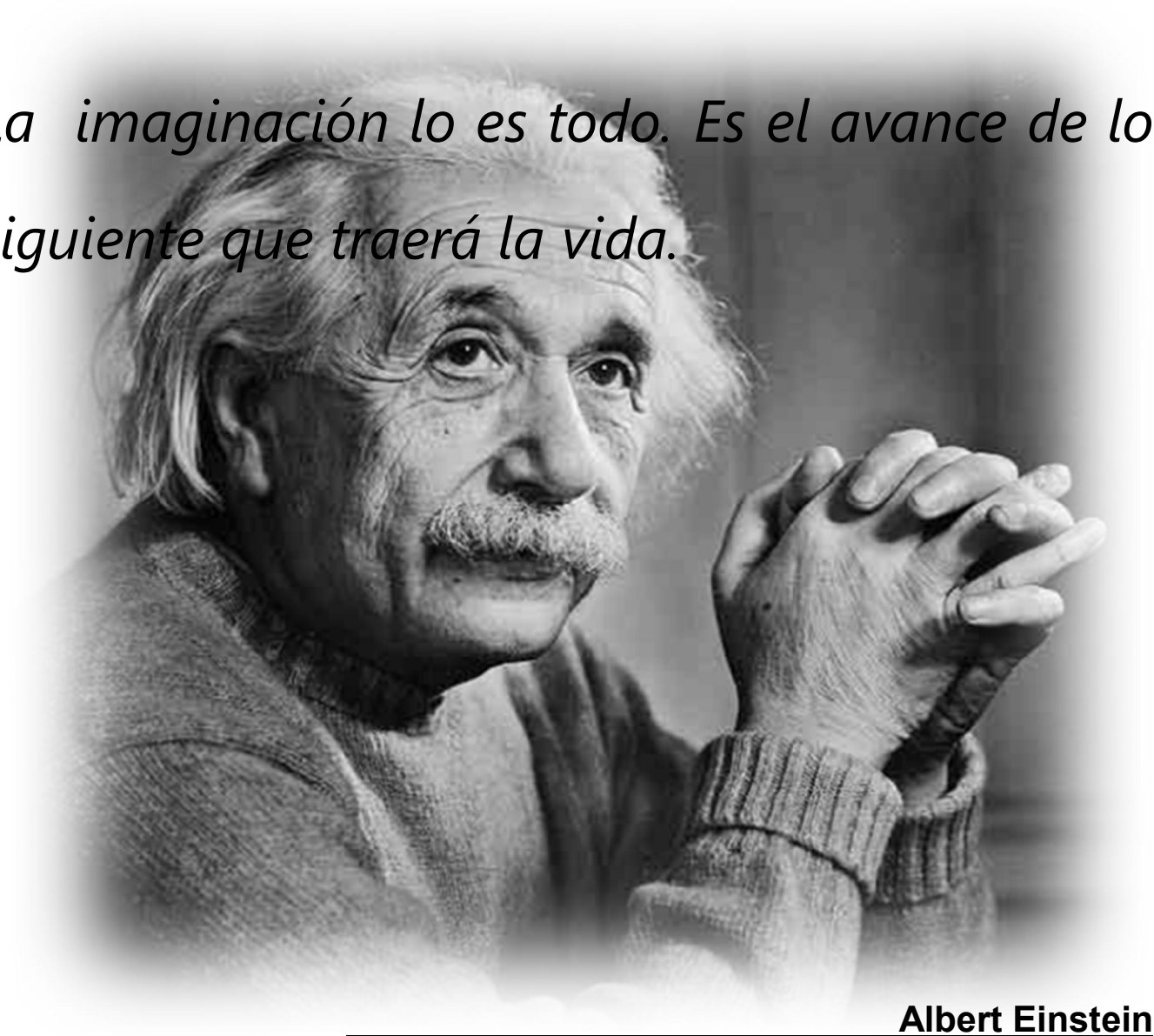
Consultante: MSc. Yovannys Sánchez Corales

La Habana, 27 de junio de 2013

"Año 55 de la Revolución"

Pensamiento

La imaginación lo es todo. Es el avance de lo siguiente que traerá la vida.



Albert Einstein

Datos de Contacto

Msc. Yasel Couce Sardiñas: Graduado de licenciado en Ciencias de la Computación en el año 2005, profesor asistente, se recibió como máster en Ingeniería del Software e Inteligencia Artificial por la Universidad de Málaga, España. Ha impartido las asignaturas de Sistemas Operativos, Seguridad Informática e Inteligencia Artificial. Actualmente se desempeña como aspirante a doctor por la Universidad de Málaga, España, en el tema de Selección de atributos relevantes en microarreglos de ADN para la aplicación a la recidiva de cáncer de mama.

Correo electrónico: yaselc@uci.cu

Agradecimientos

Muchísimas gracias:

A mis queridos padres por todo el sacrificio y paciencia durante el logro de esta meta.

A mi querida hermana por todo lo que significa para mí.

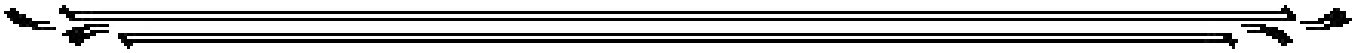
A la Revolución y a la Universidad por permitirme ser cada día mejor profesional, mejor persona.

Al tutor por toda la confianza depositada en mí.

A Fidel por su visión.

A todos los que he conocido, por estar,

¡Muchas gracias!



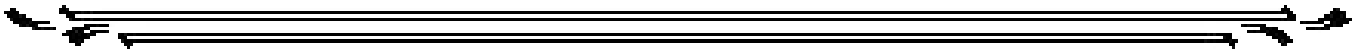
Dedicatoria

A mis queridos padres, Tania y José, por todo lo que hemos pasado juntos, y por todo el apoyo que me han brindado.

A mi hermanita Lisbeth por todas las peleas y por ser una de las personas más importantes en mi vida.

A mi familia, amigos y conocidos por sus enseñanzas e invaluable compañía, en especial a mis amigos Alexis y Héctor.

Yoandry González Castro



Resumen

La técnica de microarreglos de ADN permite el estudio simultáneo de miles de genes. De ahí su gran aplicación en diversos campos de la medicina, como es el caso del estudio de enfermedades tales como el cáncer. Esta técnica es propensa al ruido por lo que es necesario el uso de algoritmos computacionales para la extracción de información valiosa. Los resultados de estos algoritmos suelen tardar días durante el análisis de datos de interés investigativo, por lo que es necesario contar con mecanismos para controlar y monitorizar su ejecución.

Teniendo este hecho en cuenta, el presente trabajo de diploma pretende ayudar a controlar y monitorizar experimentos que utilicen algoritmos de selección de atributos relevantes en microarreglos de ADN. Para ello se planteó la construcción de un sistema para parametrizar, ejecutar y monitorizar dichos experimentos. Se utilizó la arquitectura basada en *plugins* haciendo uso de la plataforma .Net y Visual C# como lenguaje de programación.

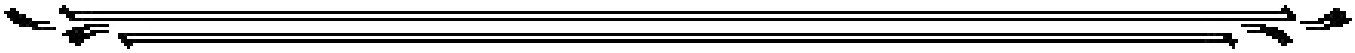
El sistema obtenido aporta como beneficio la posibilidad de realizar experimentos sobre datos de microarreglos de ADN de forma controlada, permitiendo a los investigadores centrarse en el análisis de los resultados.

Palabras claves: microarreglo de ADN, *plugin*, algoritmo, gen.

Índice

ÍNDICE

ÍNDICE



INTRODUCCIÓN

Desde hace varios años se ha hecho notable el inmenso impacto que tienen las Tecnologías de la información y la comunicación (TIC) en la vida del Hombre. La informática, en particular, es indispensable en el quehacer diario de la mayoría de los seres humanos, y en especial, en el científico-investigativo.

El proyecto Genoma Humano [1] impuso un cambio de paradigma dentro de la Genómica, se ha pasado de estudiar un sólo gen e ignorar el restante material genético a estudiar genomas completos. La diferencia está en la cantidad de información disponible. Anteriormente, los discretos descubrimientos en genética proporcionaban escasos datos para la investigación. Actualmente se cuentan con bases de datos formadas por billones de entradas dentro de las que se encuentran genomas completos. Este cambio de paradigma estuvo íntimamente relacionado con las TIC puesto que la única forma de manipular tan enormes cantidades de datos es disponiendo del *hardware* y el *software* necesario. De esta necesidad surge la bioinformática y con ella una gran oportunidad de aplicar herramientas y algoritmos computacionales con el objetivo final de entender la funcionalidad de ciertas células y la relación gen-enfermedad.

Dentro de las áreas de investigación de la bioinformática se destaca el área Análisis de Expresión Génica (AEG) [2]. Área que viene captando interés de científicos e investigadores por su aplicación y éxito en campos tales como el estudio del cáncer, enfermedades cardiovasculares, los tumores, efectos secundarios de la restricción calórica, y además en el diseño de fármacos. Dentro del AEG sobresale la técnica de microarreglos de ADN [3, 4, 5] porque permite analizar miles de genes simultáneamente, y hasta genomas completos, como es el caso del genoma humano.

Los experimentos realizados [6] con la técnica de microarreglos de ADN permiten el estudio eficaz de la relación gen-enfermedad como nunca antes se fue capaz, y esto se debe principalmente a que los experimentos tradicionales sólo son capaces de medir la expresión de uno o varios genes, lo cual representa una minúscula parte de los 20.500 genes humanos. Sin embargo, esta técnica es propensa al ruido, por lo que se entiende que los resultados obtenidos de aplicar la técnica pueden contener errores, por lo que es necesario desarrollar herramientas para eliminar, o al menos minimizar, este problema durante el análisis de los datos, siendo esta una de las tareas más importantes dentro de la bioinformática [7].

Durante el proceso de investigación, se realizan experimentos, los cuales arrojan información abundante y valiosa sobre el fenómeno en estudio. Información que, por lo general, resulta ser muy útil en futuras

investigaciones o revisiones de la misma, es por esta razón que contar con herramientas que aseguren la persistencia de la información, y que a la vez se mantengan funcionales en el tiempo, es una de las principales preocupaciones de científicos e investigadores debido a que le permiten dedicar poco tiempo a su desarrollo, además de rápido acceso al resultado de los experimentos, y en consecuencia poder centrarse en lo vital de la investigación, lo cual consiste en analizar los resultados de los experimentos.

El procesamiento de microarreglos de ADN [8, 9], para la selección de atributos relevantes que permitan predecir el desenlace de ciertas enfermedades, le añade una mayor complejidad a la realización de estos experimentos, fundamentalmente por la dimensión de los conjuntos de datos que se utilizan y por el número de ejecuciones distintas que requiere. Otro factor radica en la complejidad computacional de dicho procesamiento, lo que implica que sea necesario monitorizar en tiempo real la misma para detectar anomalías que pudieran prologar indefinidamente la ejecución del experimento, además de poder informar el progreso del mismo, dado que la ejecución suele estar en el orden de días.

Con el objetivo de incrementar la factibilidad de estudios sobre conjuntos de datos de microarreglos de ADN se propone como **problema a resolver:** ¿Cómo monitorizar la realización de experimentos con datos que provienen de aplicar la técnica de microarreglos de ADN?

Para dar respuesta a la situación planteada, el presente trabajo toma como **objeto de estudio,** la fase de análisis de resultados de la técnica de microarreglos de ADN, siendo el **campo de acción** la selección de atributos relevantes utilizando métodos filtros**.** Se concibió como **objetivo general** desarrollar una herramienta con interfaz gráfica para la parametrización, ejecución y monitorización de experimentos con métodos de selección de atributos relevantes en microarreglos de ADN.

Por tal razón se plantea la siguiente **idea a defender**: “Mediante la implementación de una plataforma sustentada en el uso de métodos filtros de selección de atributos relevantes de microarreglos de ADN, se facilitará la realización de experimentos sobre datos provenientes de microarreglos de ADN, mejorando el proceso de obtención de resultados.”

Para cumplir el objetivo propuesto se realizarán las siguientes tareas:

- Realización de la fundamentación teórica relacionada con los métodos de selección de atributos relevantes en microarreglos de ADN.

- Realización de un análisis crítico-valorativo de las aplicaciones existentes relacionadas con la extracción de atributos relevantes en microarreglos de ADN.
- Identificación del estado del arte de los métodos filtros de selección de atributos relevantes en microarreglos de ADN.
- Implementación del método de selección de atributos relevantes en microarreglos de ADN (MASSIVE).
- Implementación de una herramienta con interfaz gráfica para la parametrización, ejecución y monitorización de experimentos con microarreglos de ADN, basándose en los principios de buenas prácticas en el diseño e implementación de aplicaciones concurrentes.
- Validación de la propuesta implementada sobre la base de los resultados obtenidos por el autor del método utilizado.

Para la realización de esta investigación se siguió una estrategia descriptiva, donde se refleja lo más esencial y significativo, utilizándose los siguientes métodos investigativos:

Análisis-Síntesis: Este método fue utilizado en todo el proceso investigativo permitiendo descomponer todo el problema en varias partes que posibilitaron una mejor comprensión del mismo y luego buscar la interacción dialéctica que se establece entre dichas partes.

Histórico-Lógico: Fue de gran importancia para elaborar la fundamentación teórica de la investigación, permitiendo estudiar lo más relevante en el plano teórico acerca de la expresión génica, la técnica de microarreglos de ADN, métodos filtros para selección de atributos relevantes y herramientas que se utilizan para el desarrollo del trabajo.

Inductivo-Deductivo: Se utilizó para el planteamiento del objetivo y la extracción de las ideas fundamentales para la elaboración y fundamentación del trabajo de diploma.

Como resultado de la investigación se desarrollará una herramienta con interfaz gráfica que utilice métodos filtros de selección de atributos relevantes en microarreglos de ADN, la misma proveerá como aporte práctico una plataforma en la cual los investigadores puedan realizar experimentos, de los cuales podrán: visualizar cómo se comportan sus ejecuciones; comparar los resultados de distintas ejecuciones de un mismo experimento; guardar en un historial los resultados de experimentos anteriores, lo cual le permitirá llegar a conclusiones de forma rápida. En fin, se obtendrá una herramienta robusta que controle el proceso

de eliminación de ruido (atributos irrelevantes) que dificultan el análisis eficiente de los microarreglos de ADN, lo que asegura que los resultados obtenidos de los estudios alcancen un grado de confianza aceptable.

El documento está estructurado de la siguiente forma:

CAPÍTULO I. Fundamentación teórica de los sistemas de diseño de experimentos: El presente capítulo contiene los aspectos esenciales para entender el entorno del problema a resolver. Se describen los conceptos fundamentales asociados al dominio del tema, herramientas utilizadas y sistemas similares relacionados con la realización de experimentos.

CAPÍTULO 2. Desarrollo de la solución para el diseño de experimentos: se describe la propuesta del sistema utilizando los requerimientos funcionales y no funcionales identificados. Se puntualizan los aspectos relacionados al diseño de la solución propuesta, con el propósito de adquirir una comprensión de los requerimientos del sistema y se modelan los diagramas de clases del diseño. Se describen los estándares de diseño y codificación aplicados en la solución propuesta.

CAPÍTULO 3. Validación de la herramienta para el diseño de experimentos: se valida la propuesta implementada sobre la base de los resultados obtenidos por los autores de los métodos utilizados.

También contiene las **conclusiones** donde se realiza un breve resumen acerca del cumplimiento de los objetivos de la investigación; y se proponen **recomendaciones** para continuar desarrollando el objeto de la investigación.

Se incluye un **glosario de términos** que facilite el entendimiento del documento, en el cual se explican aquellos términos que podrían resultar nuevos o de diferente significado en el contexto de la investigación. Se exponen todos los materiales consultados y referenciados, quedando organizados en **referencias bibliográficas** y **la bibliografía,** según corresponda en cada caso.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se tratan temas relacionados con la técnica de microarreglos de ADN, el proceso de selección de atributos relevantes en microarreglos de ADN. Se mencionan los métodos filtros existentes para la selección de atributos relevantes en microarreglos de ADN. Además, se incluye una descripción de las herramientas y el lenguaje utilizado. Se realiza un análisis sobre los sistemas similares existentes relacionados con el procesamiento de microarreglos de ADN.

1.1. Técnica de microarreglos de ADN. Surgimiento y evolución

Gracias a la aplicación de la técnica de microarreglos de ADN se ha pasado de trabajos basados en el estudio de uno o unos pocos genes al análisis de genomas completos. Producto de los avances tecnológicos, la secuenciación de genomas se ha convertido en una actividad común, y desde que en 1995 se publicara el primer genoma completo de un ser vivo, *Haemophilus influenzae*, ya se han descrito unos cuantos miles de genomas completos (siendo el del hombre uno de los primeros), los cuales están disponibles en el sitio web del *National Center for Biotechnology Information* (NCBI) [10].

Definición de microarreglo de ADN

Los microarreglos de ADN son placas de vidrio, nylon o silicona, compuestas por pozos de 100-300mm, en las cuales cientos o miles de secuencias de genes se inmovilizan**.**

Tipos de microarreglos

Existen diversos tipos de microarreglos [11] según las sondas utilizadas que abarcan metodologías muy variadas, desde los más rudimentarios hasta los más sofisticados.

1.1.1 Aplicaciones de los microarreglos de ADN

Las aplicaciones de esta tecnología fundamentalmente se localizan en el área biomédica, farmacéutica, clínica y microbiológica.

Estos microarreglos se han aplicado en un gran número de estudios relacionados con problemas biológicos, prueba de ello son los interesantes resultados que se han obtenido [12, 13, 14]. Dentro de los cuales se tienen:

- Descubrimiento de tipos y/o subtipos de enfermedades como en el caso del cáncer de mama.
- Clasificación tumoral.
- Resistencia a quimioterapia.
- Identificación de marcadores tumorales específicos [15].
- Predicción del curso clínico y, potencialmente, mejora en el manejo del paciente con cáncer de pulmón.
- Predicción del comportamiento y evolución de pacientes con cáncer de colón.
- Predicción de respuesta a un tratamiento.
- Detección de mutaciones y polimorfismos de un único gen.
- Diagnóstico de anomalías congénitas.
- Mejor entendimiento de la obesidad.
- Desarrollo de medicamentos más efectivos a las enfermedades.
- **Aplicaciones en la microbiología médica:**
  - ✓ Investigación de la patogenia bacteriana.
  - ✓ Análisis de evolución bacteriana y epidemiología.
  - ✓ Estudio de los mecanismos de acción y de resistencia de los antibióticos.
  - ✓ Diagnóstico microbiológico de las enfermedades infecciosas.
  - ✓ Estudio epidemiológico molecular de ciertos microorganismos.

1.1.2 Funcionamiento de un microarreglo de ADN

Un experimento con este tipo microarreglo se basa en colocar en el mismo material genético conocido que funciona como sondas que luego se hibridan con ácidos nucleicos complementarios provenientes de la muestra en estudio, los cuales han sido marcados con material fluorescente. Luego de la hibridación se eliminan todas las cadenas que no se han unido mediante lavados. Hay que señalar que sólo habrá fluorescencia en los puntos donde haya ocurrido hibridación y la intensidad de la fluorescencia detectada será proporcional al nivel de expresión de los genes que se analizan.

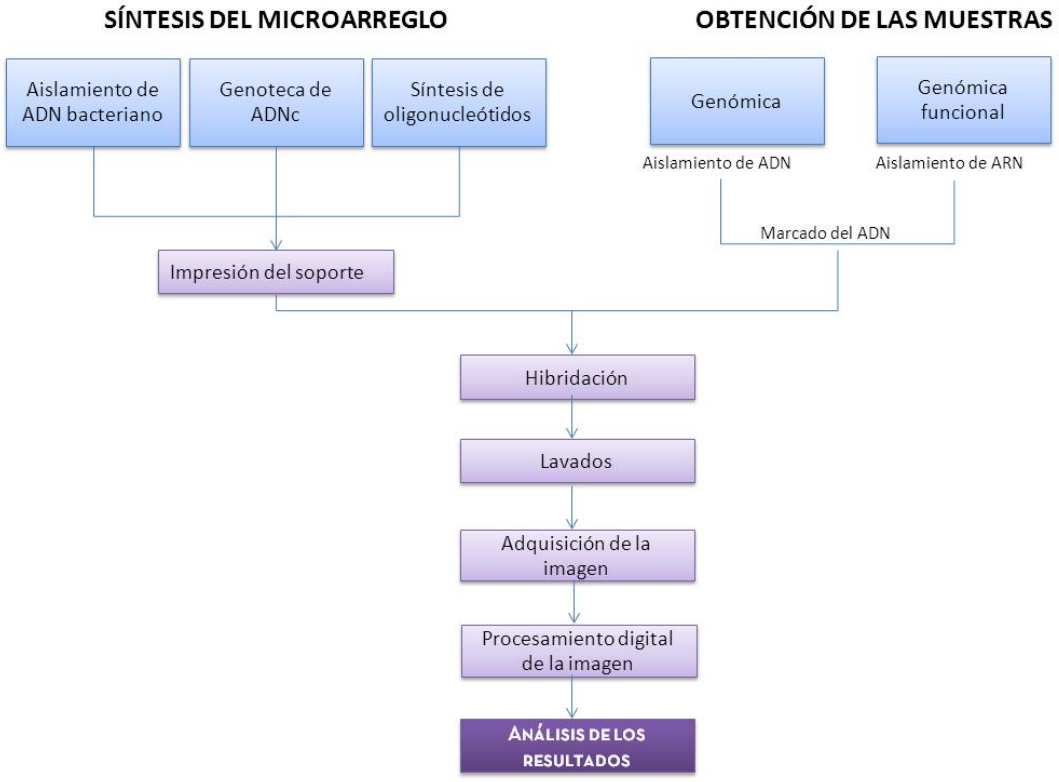


Figura 1. Esquema de la preparación de los microarreglos de ADN y del procesamiento de las muestras para su análisis

El patrón de hibridación es revelado mediante un escáner y la imagen resultante se convierte a valores numéricos, que constituyen los resultados del ensayo, los cuales deben ser tratados con métodos y herramientas bioinformáticas.

1.1.3 Análisis computacional de un microarreglo de ADN

A causa de que esta técnica es propensa al ruido causado por diferentes factores [16] los que provocarían la pérdida de validez en los resultados obtenidos junto a que se genera gran cantidad de información la cual es imposible analizar manualmente, es necesario el uso de herramientas que procesen estos datos y den como resultado información de valor. En este contexto cobran gran importancia las herramientas de selección de atributos relevantes en microarreglos de ADN, las cuales son las protagonistas de la fase final.

Luego de la **adquisición de la imagen** del microarreglo, es necesario procesarla digitalmente [17], lo cual tiene como objetivo final mejorar la imagen para luego ser capaz de cuantificar, lo más correctamente posible, el valor de fluorescencia de cada gen representado en el microarreglo en estudio. Luego de este paso, se obtiene una matriz en la cual sus valores muestran el grado de expresión de cada gen. La imagen de la figura 2 muestra esto último.

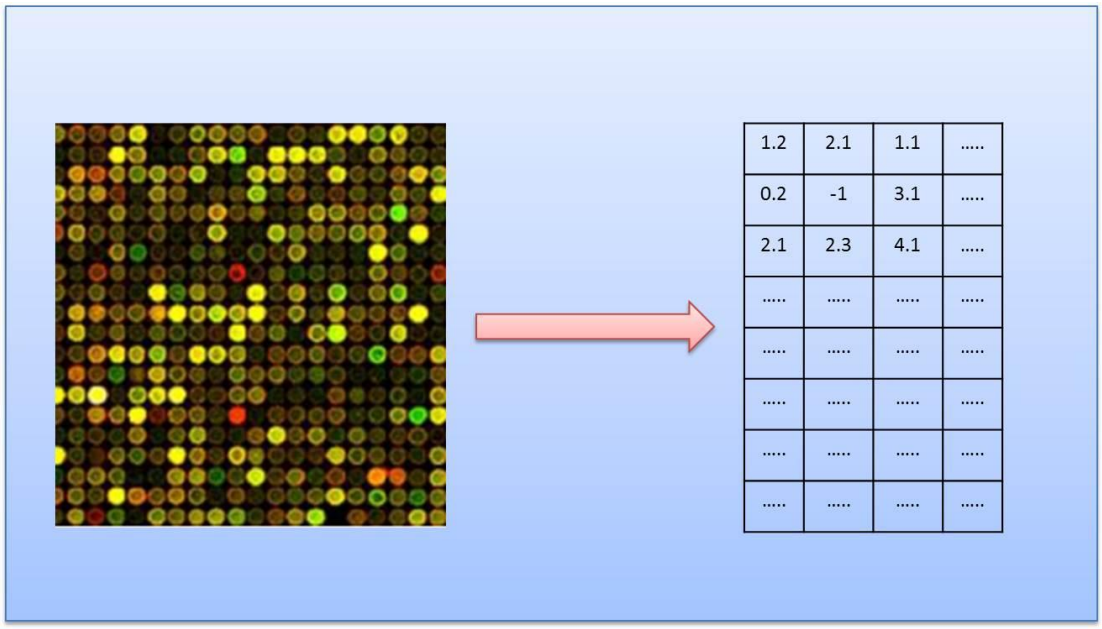


Figura 2. Conversión de imagen de un microarreglo a una matriz de valores reales

La matriz obtenida es analizada en busca de información útil, lo cual constituye la fase final de la técnica.

1.2 Métodos filtros de selección de atributos relevantes en microarreglos de ADN

1.2.1 Definición de método filtro

Un método filtro es un algoritmo para la selección de atributos relevantes en microarreglos de ADN caracterizados por un gran número de variables (genes) y pocas condiciones experimentales [18,19].

1.2.2 Características de los métodos filtros

Los métodos filtros se caracterizan por ser rápidos; por obtenerse calidad en los resultados; y por estar fuertemente basados en criterios de teoría de la información.

1.2.3 Métodos filtros existentes

El basamento teórico de los métodos filtros viene dado porque su funcionamiento está apoyado sobre fuertes criterios de teoría de la información tales como información común y pertinencia. El propósito de estos filtros es el de seleccionar el mejor subconjunto de variables que lleve al mejor modelo predictivo.

Actualmente existen 7 métodos filtros utilizados para selección de atributos relevante [20]:

- MASSIVE (Matrix of Average Sub-Subset Information for Variable Elimination).
- MRMR (Minimum Redundancy Maximum Relevance).
- Ranking.
- Markov blanket.
- FCBF (Fast Correlation Based Filter).
- CMIM (Conditional Mutual Information Maximization).
- Relevance.

1.3 Aplicaciones existentes relacionadas con la realización de experimentos

Las herramientas para la realización de experimentos tienen como objetivo brindar a los investigadores un medio a través del cual puedan verificar una o varias hipótesis relacionadas con un determinado fenómeno, para lo cual manipulan la(s) variable(s) que presumiblemente son su causa. El área de expresión génica ha demostrado ser invaluable en diagnóstico y tratamiento de enfermedades, por lo cual es de vital importancia la existencia de herramientas con las que se puedan realizar experimentos dentro de esta área.

1.3.1 Subio platform

Herramienta implementada con el lenguaje de programación java (multiplataforma), en la que se pueden analizar datos de cualquier tipo de microarreglos de alta calidad (*Affymetrix, Agilent, NimbleGen*). Provee herramientas para el análisis integral de microarreglos, las cuales se le integran en forma de *plugin* [21].

De esta herramienta se pueden utilizar un conjunto de *plugins* libres que vienen integrados a esta cuando se descarga. Si se quieren añadir otros *plugins* para hacer uso de sus funcionalidades, estos requieren su compra.

Esta herramienta fue rechazada como posible cumplimiento del objetivo propuesto porque de los *plugins* libres que tiene integrados por defecto, ninguno de ellos está destinado a la selección de atributos relevantes en microarreglos de ADN.

1.3.2 GeneChip® Command Console Software

Conjunto de herramientas intuitivas para el control y gestión del procesamiento de los microarreglos que fábrica esta empresa. Estas herramientas intervienen en todo proceso de la técnica, proceso que puede realizarse manual y/o automático [22].

Esta herramienta fue rechazada como posible cumplimiento del objetivo propuesto pues esta es aplicable solamente a la línea de microarreglos de la empresa, además de que aplicar esta tecnología es muy costosa por los elevados precios de los microarreglos antes mencionados.

1.3.3 Aplicación de consola MASSIVE

Aplicación de consola desarrollada con el lenguaje de programación C++. En esta se implementa el método filtro "*Massive*" para la selección de atributos relevantes en microarreglos de ADN [23].

Esta herramienta fue rechazada como posible cumplimiento del objetivo propuesto pues tiene como limitaciones que no cuenta con una interfaz para configurar su ejecución de forma fácil y que no permite guardar los resultados obtenidos para posterior consulta.

1.3.4 Weka

Entorno para Análisis del Conocimiento de la Universidad de Waikato (Weka -Waikato Environment for Knowledge Analysis) es una plataforma de software para aprendizaje automático y minería de datos escrito en Java y desarrollado en la Universidad de Waikato de Nueva Zelanda. Weka es un software libre distribuido bajo licencia GNU-GPL. La misma contiene una extensa colección de técnicas para pre-procesamiento de datos y modelado [24].

Esta herramienta fue rechazada como posible cumplimiento del objetivo propuesto porque en esta no se implementan métodos para la selección de atributos relevantes en microarreglos de ADN.

1.4. Lenguaje de programación

En esta sección se especifican las características más importantes del lenguaje de programación Visual C#, lenguaje seleccionado para la implementación de la Solución que se propone. Luego se fundamenta la elección de C# como lenguaje a utilizar para la implementación del software resultado de la presente investigación.

1.4.1. Lenguaje Visual C#

El lenguaje Visual C# está diseñado para compilar una variedad de aplicaciones que se ejecutan en la plataforma *.NET*. Este se caracteriza por ser simple, eficaz, y además por ofrecer seguridad de tipos y ser orientado a objetos. Con sus muchas innovaciones, permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

El lenguaje Visual C# ha sido diseñado para ser robusto, moderno y sencillo. El mismo promueve las buenas prácticas de programación (por ejemplo, el chequeo de tipos) y las reglas de globalización. Por otro lado, presenta un mecanismo robusto para el tratamiento de errores. Se destaca la existencia de tipos genéricos. Este lenguaje incorpora mecanismos para el desarrollo de aplicaciones que utilicen múltiples hilos.

1.4.2. Fundamentación de la elección

Debido a que el propósito de esta investigación es desarrollar una herramienta no comercial, robusta y confiable en la que, monitorizar el procesamiento de gran volumen de datos utilizando métodos filtros (tarea principal) que se puedan implementar independientes, en forma de *plugin*, de la herramienta base, además de la potencia y facilidad que ofrece el lenguaje C# para el trabajo en este tipo de arquitectura (uso de *reflection*).

1.5. Herramientas Utilizadas

Para el desarrollo de software es necesario contar con herramientas que faciliten el trabajo a realizar. Estas deben asegurar la automatización de tareas básicas que no tengan impacto en el desarrollo, pues la mayor parte del tiempo se debe dedicar a la implementación de las funcionalidades de interés. En fin, deben ser lo suficientemente inteligentes para que los desarrolladores se centren en las cuestiones importantes. A continuación se describen las herramientas utilizadas para dar solución a la problemática planteada.

1.5.1. Visual Studio 2010

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, y Visual Basic .NET, al igual que entornos de desarrollo web como ASP.NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles [25].

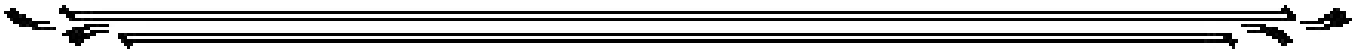
1.5.2. Visual Paradigm for UML 8.0

Visual Paradigm for UML (Unified Modeling Language - por su nombre en inglés) es una herramienta CASE (Computer Aided Software Engineering - por su nombre en inglés) aplicable en todo el ciclo de vida del desarrollo de software. Permite dibujar todos los tipos de diagramas de clases, generación de código desde diagramas y generación de documentación. Además proporciona tutoriales, demostraciones interactivas y proyectos UML. Es fácil de instalar y actualizar, además de ser compatible entre ediciones. Presenta licencia gratuita y comercial [26].

1.6. Plataforma y espacios de nombres

No a reinventar la rueda es uno de los paradigmas de mayor importancia dentro del desarrollo de software. A continuación se describe la plataforma sobre la cual se implementa la solución propuesta, además de los espacios de nombres más importantes utilizados con ese objetivo.

1.6.1. Plataforma .NET (versión 4.0)



.NET es una plataforma de Microsoft que hace énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones. Basado en ella, la empresa intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el sistema operativo hasta las herramientas de mercado. La versión 4.0 de esta plataforma cuenta con novedades con respecto a las anteriores, dentro de las más relevantes se encuentran: la inclusión de un nuevo modelo de programación paralela; mejoras en la seguridad y el rendimiento.

1.6.2. Espacio de nombres *System.Reflection*

Reflection es el nombre dado por Microsoft al conjunto de utilidades que permite leer la información y meta-información de las bibliotecas de enlace dinámico o más comúnmente DLL (Dynamic-Link Library - por su nombre en inglés) y ejecutables de .NET en tiempo de ejecución. Su utilidad fundamental está contenida en el espacio de nombres *System.Reflection* y contiene diversas funciones para cargar ensamblados, crear objetos, invocar métodos o leer información, todo ello en tiempo de ejecución.

1.6.3. Espacio de nombres *System.Threading*

El espacio de nombres *System*.*Threading* contiene clases e interfaces que permiten el trabajo concurrente en aplicaciones desarrolladas sobre la plataforma .Net. Además proporciona clases para la sincronización de actividades de subprocesos y el acceso a datos.

1.7. Metodología de desarrollo

El desarrollo de *software* no es una tarea de fácil realización. Prueba de ello es el caos que puede surgir durante su realización. De hecho, hubo una etapa que es conocida como “la crisis del *software*”, caracterizada principalmente por la desorganización en el desarrollo, en la que se evidenciaron los problemas derivados de no usar una guía que muestre el camino efectivo a seguir por los equipos de desarrollo durante esta actividad. Hoy en día existen varias guías que dirigen el desarrollo de *software*, las cuales se denominan metodologías de desarrollo de *software*. Una metodología define un conjunto de

actividades, acciones, tareas y productos de trabajo que se requieren para desarrollar *software* de alta calidad.

Estas metodologías se dividen en dos grupos: tradicionales y ágiles. Las primeras se caracterizan por seguir un proceso regido por normas y políticas basadas en estándares, y por mostrar cierta resistencia a los cambios. Estas se utilizan generalmente en proyectos de gran tamaño. Por otro lado, las segundas son menos estrictas, preparadas para los cambios y más concentradas en producir el mejor código posible. Su uso está mayormente dirigido a proyectos pequeños.
El tipo de metodología que mejor se adapta al desarrollo de la Solución que se presenta en este documento es la metodología ágil porque la Solución es pequeña, está sujeta a cambios en los requisitos y además se desea que el código quede lo más depurado posible y que sea mantenible. La metodología escogida es Programación extrema (XP – por sus siglas en inglés) por ser la más representativa de su categoría y estar bien documentada.

1.1.1. Metodología ágil XP

XP es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico ().

Características esenciales:

- Las Historias de Usuario**:** son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy

dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

- Proceso XP

  El ciclo de desarrollo consiste en seguir los siguientes pasos:

  1. El cliente define el valor de negocio a implementar.

  2. El programador estima el esfuerzo necesario para su implementación.

  3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.

  4. El programador construye ese valor de negocio.

  5. Volver al paso 1.

- Practicas utilizadas: XP define 12 prácticas de las cuales para desarrollar la Solución se utilizan las 6 siguientes:

  El juego de la planificación. Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

  Entregas pequeñas. Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio.

  Diseño simple. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

  Refactorización (*Refactoring*). Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

Integración continua. Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

Estándares de programación. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

Fases de XP

1. **Planeación:** La actividad de planeación comienza creando una serie de historias (también llamadas historias del usuario) que describen las características y funcionalidades requeridas en el *software* que se desarrollará. Una historia es una descripción de una característica o funcionalidad que el cliente quiere que posea el software que se va construir. Cada historia la escribe el cliente y le asigna una prioridad basándose en el valor de impacto que tiene esta en el software que se desarrolla.
2. **Diseño:** El diseño en XP sigue de manera rigurosa el principio MS (mantenerlo simple). Siempre se prefiere un diseño simple respecto a otro más complejo. Por otro lado, el diseño ofrece una guía de implementación para una historia como está escrita, ni más ni menos.
3. **Codificación:** Se recomienda que después de diseñar las historias y realizar el trabajo de diseño preliminar no se puede proceder a la codificación, sino que se debe desarrollar una serie de pruebas de unidad que ejerciten cada una de las historias que vayan a incluirse en el incremento de *software* actual.
4. **Pruebas de aceptación:** También llamadas pruebas del cliente, las especifica el cliente y se centran en las características generales y la funcionalidad del sistema, elementos visibles y revisables por el cliente.

En este capítulo se estudiaron los aspectos más importantes relacionados con la técnica de microarreglos de ADN, además de los sistemas existentes relacionados con la realización de experimentos que utilizan datos provenientes de aplicar esta técnica. También se describieron las herramientas, lenguaje de programación, plataforma y metodología de desarrollo de *software* que se utilizan para desarrollar la Solución que se propone en el siguiente capítulo.

CAPÍTULO 2. DESARROLLO DE LA SOLUCIÓN

A lo largo de este capítulo se abordan los elementos relacionados con el dominio de la investigación y del sistema. Se realiza el desarrollo del Modelo de Dominio, se plantean los requerimientos funcionales y no funcionales. Entre los objetivos a tratar se identifican: la concepción arquitectónica, estructura y elementos del sistema. Se hace énfasis en los principales aspectos a tener en cuenta para la correcta implementación del sistema.

2.1. Requerimientos

Para resolver el problema planteado se propone como solución un sistema basado en *plugins*. Las características básicas de dicho sistema estarán enfocadas en:

- ✓ Permitir la ejecución de un método filtro contenido en un *plugin.*
- ✓ Permitir importar un nuevo *plugin.*
- ✓ Permitir habilitar y deshabilitar *plugins.*
- ✓ Permitir la monitorización en tiempo real de los algoritmos implementados en los *plugins.*
- ✓ Guardar los resultados de los experimentos.
- ✓ Exportar los resultados a Weka.
- ✓ Monitorizar el uso de memoria y procesador del sistema operativo.
- ✓ Monitorizar el uso de memoria de la aplicación.

2.1.1. Requisitos funcionales

Los requisitos funcionales de un sistema surgen a partir de un análisis profundo de la situación problémica que se enfrenta. Para definir los requisitos del sistema que se propone se utilizó la técnica de entrevista con el objetivo de conocer las necesidades reales de los investigadores como usuarios finales del producto. Como resultado de este estudio, a continuación se listan los requerimientos funcionales definidos.

RF1. Gestionar configuración de plugin.

RF1.1. Habilitar plugin: posibilita que se pueda habilitar un *plugin* en la aplicación, anteriormente importado.

RF1.2. Deshabilitar plugin: posibilita que se pueda deshabilitar un *plugin* en la aplicación, anteriormente importado.

RF1.3. Importar plugin: posibilita que se pueda importar un *plugin* en la aplicación.

RF2. Gestionar ejecución de filtros.

RF2.1. Ejecutar filtro: posibilita que se pueda ejecutar un método filtro en la aplicación.

RF2.2. Pausar filtro: posibilita que se pueda pausar un método filtro en la aplicación que previamente estaba en ejecución.

RF2.3. Detener filtro: da la posibilidad de que se pueda detener y eliminar un método filtro en la aplicación que previamente estaba en ejecución, pausado o sin iniciar.

RF3. Guardar resultados: da la posibilidad de que se pueda al finalizar la ejecución de un método filtro los resultados arrojados por este puedan ser guardados para su posterior consulta.

RF4. Exportar resultados a Weka: da la posibilidad de que se puedan exportar los resultados arrojados por la ejecución de un método filtro a Weka para su análisis.

RF5. Monitorizar uso de memoria y procesador del sistema y la aplicación: da la posibilidad de monitorizar el uso de recursos que está teniendo la aplicación y el sistema operativo en general.

2.1.2. Requisitos no funcionales

Requerimientos de interfaz

Las ventanas de la aplicación mostrar los datos de forma estructurada y clara, además de permitir su correcta interpretación. La interfaz estará formada por listas, componentes de selección, etiquetas, botones y un árbol con estructura de carpetas. Ante la entrada de datos incorrectos se informará al usuario. Todos los textos y mensajes en pantalla aparecerán en idioma español.

Requerimientos de hardware

Estaciones de trabajo

Debido al alto procesamiento que se deben efectuar durante la ejecución de los filtros, es necesario que las estaciones de trabajo cuenten como mínimo con 256 Mb de memoria RAM y un microprocesador de 1.8 GHz.

Requerimientos de software

El sistema debe correr en sistemas operativos Windows utilizando la plataforma .Net en su versión 4.0 para asegurar su correcto funcionamiento.

2.2. Diseño

Para desarrollar exitosamente un producto de *software* es de gran importancia cumplir con cada uno de los pasos dentro del ciclo de desarrollo del proyecto. La fase diseño proporciona una visión más nítida del sistema que se quiere construir y tiene como objetivo principal, modelar el sistema de forma que soporte todos los requerimientos (funcionales y no funcionales).

2.2.1. Modelo de Dominio

El Modelo de Dominio captura los objetos más importantes que existen en el entorno donde estará enmarcado el sistema. Este es construido con las reglas de UML y es presentado como uno o más diagramas de clases. Se utiliza para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema.

2.2.2. Conceptos principales del dominio

Para brindar una mejor comprensión del Diagrama del Modelo de Dominio a continuación se describen los conceptos más importantes encontrados en el problema planteado.

Plugin: el concepto se refiere a una biblioteca de clases se puede añadir a una aplicación principal con el objetivo de ampliar las funcionalidades que esta última provee.

Filtro: el concepto se refiere al algoritmo encargado de analizar los datos de entrada y dar como salida datos filtrados.

2.2.3. Diagrama del Modelo de Dominio

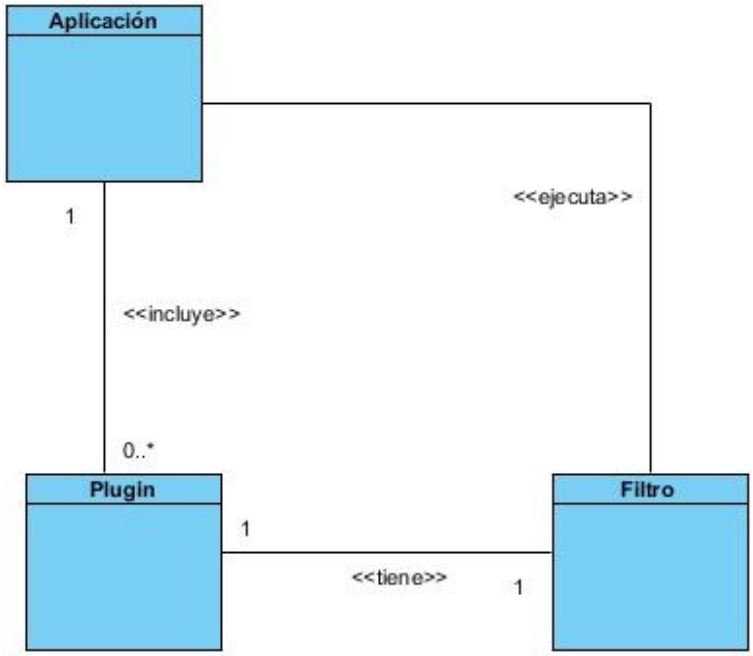


Figura 4. Diagrama del Modelo de Dominio

2.2.4. Fundamentación del uso de patrones de diseño

Los patrones de diseño son muy útiles para desarrollar *software* de forma rápida y eficiente, pues brindan una forma efectiva de enfrentar situaciones conocidas con anterioridad y a las que le han dado soluciones bastante buenas. Los patrones pueden considerarse como un conjunto de principios generales basados en la experiencia, que guían la creación de un *software*. Estos patrones no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables.

En el desarrollo del sistema se emplean los patrones generales de software para asignar responsabilidades (GRASP – por sus nombre en inglés) por su gran utilidad durante el diseño, y además, porque su uso se ha convertido en una condición indispensable si se quiere diseñar eficazmente *software* orientado a objetos [27, 28,29]. A continuación se describen los utilizados en la solución propuesta:

- ✓ **Experto:** es el principio básico de asignación de responsabilidades. Consisten en asignar una responsabilidad al experto en la información. En este se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto provee un bajo nivel de acoplamiento.
- ✓ **Creador:** ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. La creación de una instancia recaerá sobre la clase que posee toda la información necesaria para hacerlo.
- ✓ **Controlador:** asigna la responsabilidad del manejo de mensajes de los eventos del sistema a una clase global.
- ✓ **Bajo acoplamiento:** postula que una clase con bajo acoplamiento no depende de "muchas otras" clases. Las clases con alto acoplamiento recurren a muchas clases y no es conveniente porque son más difíciles de mantener, entender y reutilizar.
- ✓ **Proxy**. proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.
- ✓ **Alta cohesión:** postula que la información almacenada en una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase.

En la solución propuesta, estos patrones se utilizan mediante la asignación de responsabilidades a las clases que contienen la información necesaria para realizar las mismas, colaborando así con la organización del sistema. Además, es asignada la tarea de crear instancias de otras clases sólo a aquellas que contengan la información requerida para ello. Se usan clases controladoras para manejar eventos del sistema de forma limpia y eficiente. Se hace uso de una clase para mediar entre otras que necesitan compartir información.

2.3. Arquitectura de la Solución

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema. La misma debe describir diversos aspectos del software que son más comprensibles si se utilizan diferentes modelos o vistas. Es importante destacar que cada uno de ellos constituye una descripción parcial de una misma arquitectura y es deseable que exista cierto solapamiento entre ellos. Esto es así porque todas las vistas deben ser coherentes entre sí, evidente dado que describen la misma cosa. De manera general, la

arquitectura es el resultado de ensamblar los elementos arquitectónicos con el objetivo de satisfacer los requerimientos tanto funcionales como no funcionales de un sistema [30].

2.3.1. Arquitectura basada en plugins

La arquitectura basada en *plugins* da a un sistema la capacidad de añadirle más funcionalidades sin necesidad de modificarlo. Esta arquitectura (Ver Anexo 1) ha demostrado ser la mejor solución a la hora de implementar sistemas lo suficientemente genéricos como para adaptarse a las necesidades particulares de cada cliente. Se caracteriza principalmente por tener alta cohesión y bajo acoplamiento entre las partes. Los sistemas desarrollados sobre esta arquitectura son altamente modulares y extensibles, además de ser menos complejos, más baratos y fáciles de mantener.

En la actualidad existe la tendencia de que las aplicaciones se integren e intercambien información con otras, o sea que la idea es que ninguna aplicación es una isla; depende de cómo pueda ampliarse.

Beneficios de la arquitectura basada en plugins:

- Alta independencia entre módulos (también baja cohesión entre módulos). Puesto que cada módulo es, en principio, relativamente independiente de todos los demás es más fácil desarrollar pruebas sin que se vean afectado el resto de componentes de la aplicación.
- Facilidad para extender la aplicación sin necesidad de redistribuir un nuevo ejecutable.
- Si los plugins que se desarrollan son los suficientemente genéricos pueden reutilizarse en otras aplicaciones con lo que el periodo de desarrollo es significativamente menor (el necesario para adaptar el plugin al nuevo sistema).
- Facilidad para adaptarse a cambios en los requisitos añadiendo o modificando funcionalidad mediante la creación o modificación de un plugin adecuado.

Desventajas de la arquitectura basada en plugins:

- Transferencia de dependencias a tiempo de ejecución. Debido a que los plugins son independientes unos de otros pero pueden apoyarse unos en otros pueden surgir problemas derivados de la interacción de dichos plugins.

- Necesidad de un mayor control de errores. Puesto que los plugins son en principio desconocidos se hace especialmente necesario desarrollar un control adecuado sobre los parámetros de entrada así como los posibles errores durante la ejecución de un servicio que proporciona un plugin.
- Posibles compromisos de seguridad. En principio, y salvo que se proporcione un sistema de seguridad y permisos adecuados, cualquier plugin tiene acceso completo a la máquina con todos los permisos del usuario que ejecuta la aplicación.

2.4. Elementos que conforman la Solución

A continuación se describen los elementos más importantes que conforman la Solución propuesta. Su descripción incluye lo más relevante de cada uno.

2.4.1. Descripción de la solución

La solución está concebida para ser una aplicación de escritorio. La funcionalidad monitorizar ejecución tiene como objetivo conocer el estado de ejecución de un algoritmo para poder para poder realizar sobre este las acciones ejecutar, pausar y detener. Para apoyar la toma de decisiones sobre qué acciones realizar se encuentra la funcionalidad mostrar rendimiento, la cual muestra que cantidad de memoria y procesador se está utilizando en el sistema operativo, además de mostrar la cantidad de memoria que utiliza la aplicación en un momento dado. Al finalizar la ejecución de un algoritmo la aplicación es capaz de visualizar, guardar o exportar los resultados obtenidos de dicha ejecución.

El diagrama de la figura 5 muestra el conjunto de clases que conforman la aplicación y la forma en que estas interactúan para realizar las funcionalidades para las que fueron creadas. A continuación se describen cada una de estas clases y sus responsabilidades.

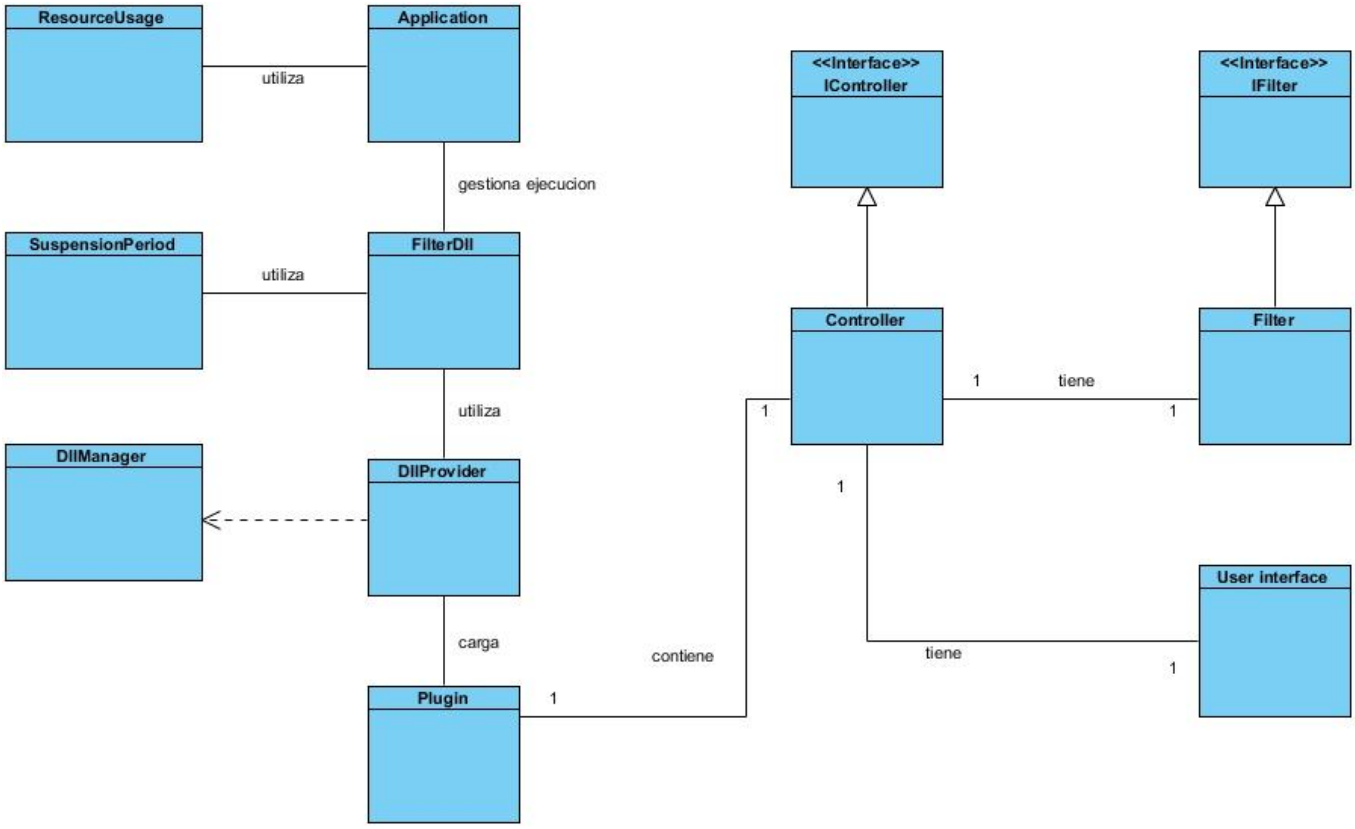


Figura 5. Diagrama de clases de la Solución

Cuando la aplicación se inicia esta carga todas las bibliotecas de clases (*plugins*), que esta reconozca como compatibles, contenidas dentro un directorio llamado **Available** que se encuentra debajo de la dirección **$Proyecto\Filters** siendo **$Proyecto** la dirección en la que se encuentra el directorio de la aplicación.

Criterio de compatibilidad de una biblioteca de clases:

Para que una biblioteca de clases sea reconocida y cargada por la aplicación esta debe contener básicamente una clase que herede de la clase interfaz IFilter y otra que lo haga de IController. Esta

operación se realiza comprobando si existe algún tipo dentro de la biblioteca que pueda ser asignado al tipo de la clase IFilter e IController respectivamente.

Para guardar la información concerniente a un método filtro cargado se utiliza la clase FilterDll. Esta clase constituye la representación de un *plugin* dentro de la aplicación y tiene como función almacenar la información relacionada con la ejecución de un método filtro dentro de un hilo. Ver figura 6.

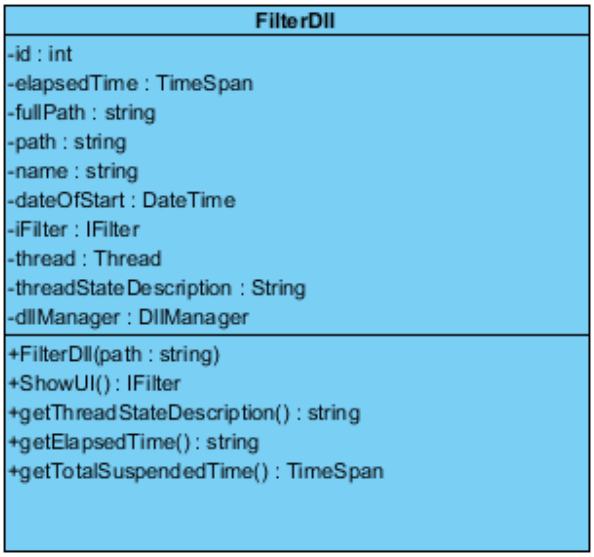


Figura 6. Clase FilterDll

Luego de que la aplicación ha cargado todas las bibliotecas reconocidas, entonces esta pone disponible desde la interfaz de usuario principal cada uno de los métodos filtros encontrados dentro de las bibliotecas reconocidas. En la figura 7 se muestra una biblioteca de clases cargada debido a que cumple con el criterio de compatibilidad antes expuesto.

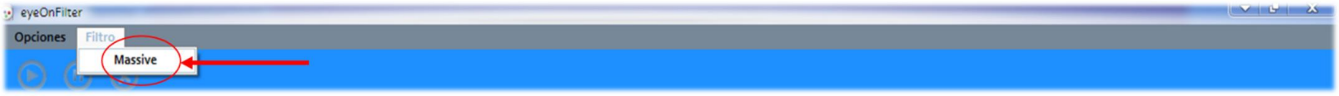


Figura 7. Carga de *plugins*

Cuando se selecciona un método para su ejecución este muestra la interfaz gráfica que define para la configuración de dicha ejecución. Ver figura 8.

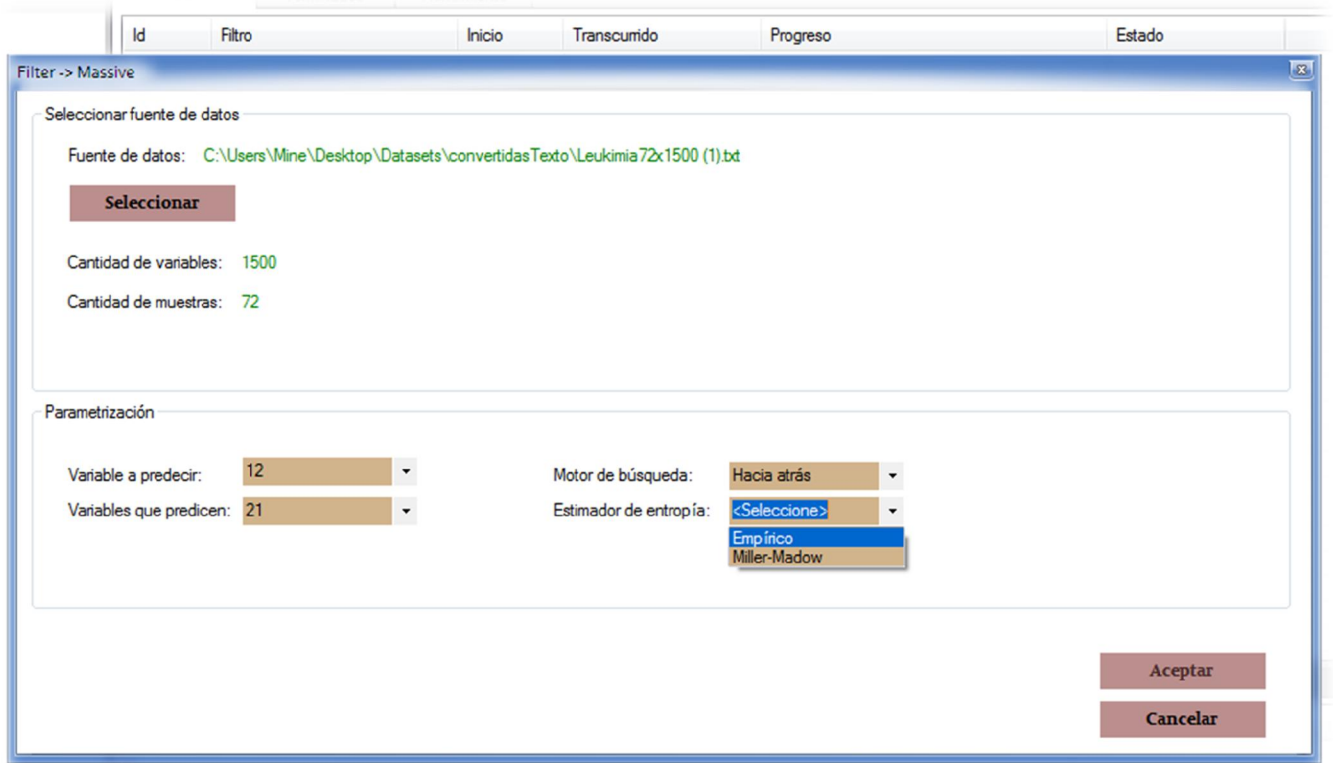


Figura 8. Configuración de un filtro

Luego de configurar el método filtro seleccionado este está listo para ser ejecutado. Este se muestra en una lista dedicada a visualizar la ejecución de los métodos configurados. Ver figura 9.

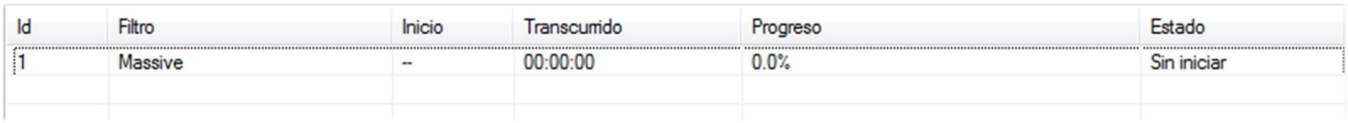


Figura 9. Filtro configurado y listo para ejecutarse

Una vez ubicada en esta lista, el método puede ejecutarse, pausarse y eliminarse. Estas operaciones se ejecutan haciendo llamadas a procedimientos del hilo contenido dentro del objeto de tipo FilterDll. Para

ejecutar el método filtro se utiliza el método ***Start()*** del hilo que lo contiene, pasando de esta forma al estado de ejecución. Si se requiere realizar una pausa sobre un filtro dado, entonces se utiliza el procedimiento ***Suspend()***. En caso de que un método esté pausado y se quiere reanudar su ejecución entonces se usa el procedimiento ***Resume()***. Por último, si se desea eliminar una ejecución de un filtro, se detiene si se estaba ejecutando y luego se borra. Ver figura 10.

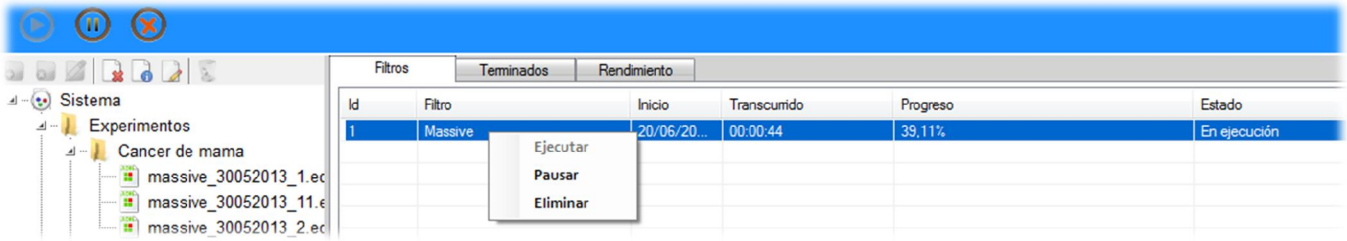


Figura 10. Filtro en ejecución

Conocer que porciento de ejecución lleva un determinado procedimiento en cierto instante de tiempo no es una tarea sencilla, pues no existe una medida exacta para determinar ese valor. Este problema puede ser formulado de la forma: ¿Qué tiempo demorará un procedimiento en terminar su ejecución? Para este enfoque existe una aproximación ampliamente aceptada que resulta en describir el comportamiento del tiempo de ejecución de un algoritmo como una función que depende del tamaño de las instancias del problema que resuelve. Caracterizar el desempeño de un algoritmo de esta manera es una abstracción que ignora detalles. Usar esta medida apropiadamente requiere ser consciente de los detalles ocultos por esta abstracción. Estos detalles están relacionados con que cada algoritmo se ejecuta sobre una plataforma, la cual generalmente engloba:

- La computadora en la cual se ejecuta el algoritmo, su velocidad de procesamiento, memoria caché, unidad de procesamiento con punto flotante, y otras características derivadas del *hardware*.
- El lenguaje de programación en el cual es escrito el algoritmo, junto con el compilador/intérprete y las configuraciones de optimización para el código generado.
- El sistema operativo.
- Otros procesos ejecutándose en segundo plano.

Un cambio en cualquiera de los parámetros que comprenden la plataforma cambiará el tiempo de ejecución de un algoritmo en un factor constante.

Es de notar que es necesario que cada *plugin* sea el que defina como calcular el porciento de ejecución del método filtro que este contiene. Para lograr esto la clase interfaz IFilter (ver figura 11) define el método **getPercentage** el cual recibe como parámetro el tiempo, en segundos, que lleva el filtro en ejecución y este devuelve el porciento que lleva el filtro de ejecutado. Hay que señalar que este valor no es exacto por lo se hace imprescindible una forma de manejar el caso de que el porciento esté próximo al valor máximo y al filtro le falte una parte considerable de ejecución. La solución más simple posible a este caso es aumentar el tiempo que supuestamente debería demorar la ejecución.

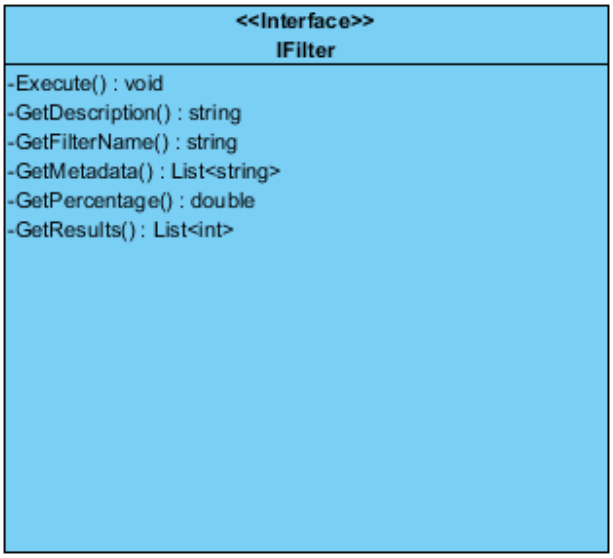


Figura 11. Clase interfaz IFilter

Una vez finalizada la ejecución, los resultados obtenidos de la misma pueden ser guardados, eliminados y hasta incluso se puede repetir la ejecución del mismo filtro. Ver figura 12.

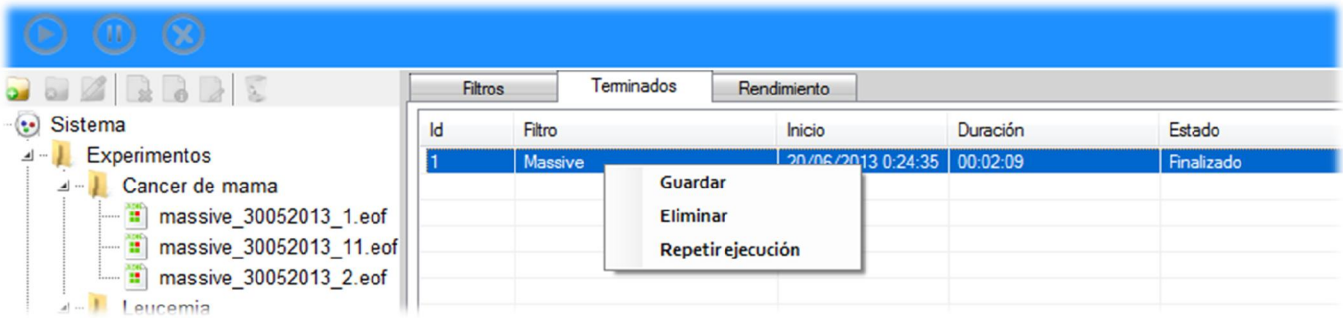


Figura 12. Opciones de un filtro cuando termina su ejecución

Al escoger la opción de guardar los resultados de una ejecución este debe ser ubicado dentro de alguna de los directorios que de experimentos que se tienen creado. Ver figura 13.

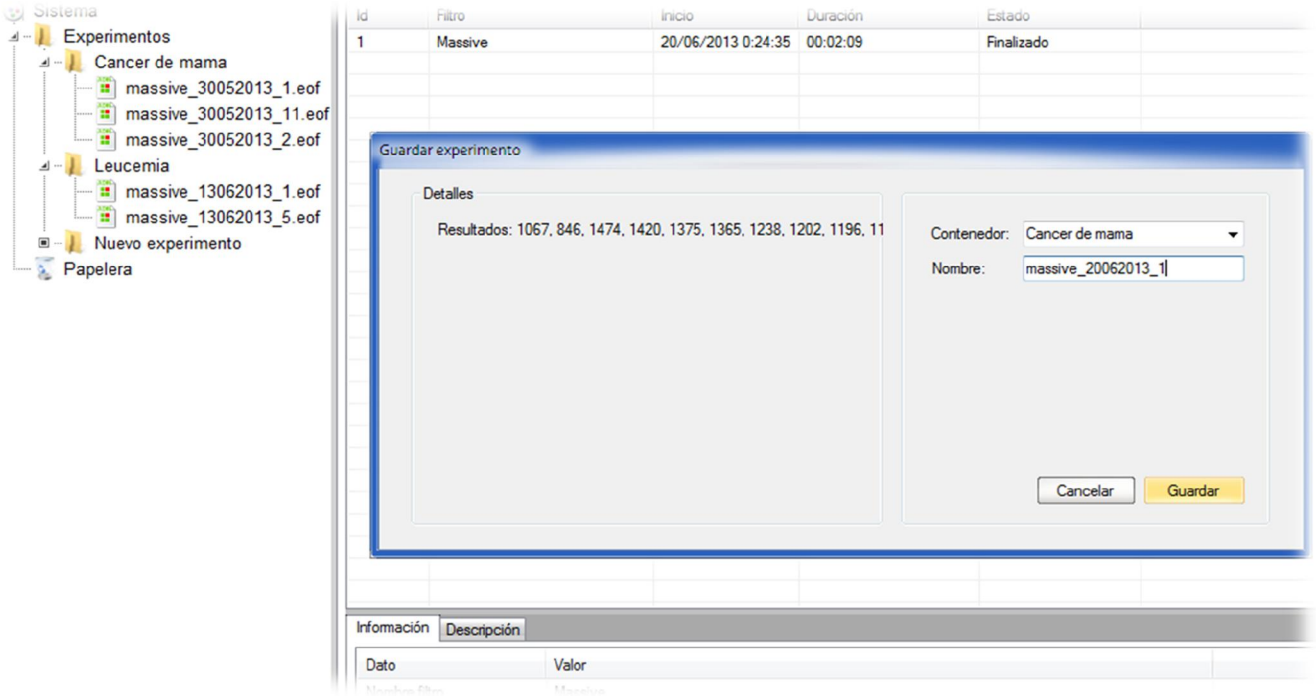


Figura 13. Guardar resultados de la ejecución de un filtro

Sobre el fichero guardado se pueden realizar varias acciones como visualizar, cambiar nombre, mover a otro experimento, abrir en su ubicación real, y por último la operación más importante que se puede realizar, la cual es **Exportar a fichero ARFF.** Esta opción permite exportar los resultados obtenidos en una ejecución de un método filtro a un fichero ARFF que puede ser analizado por la herramienta Weka, para la búsqueda de información valiosa relacionada con el experimento que el investigador esté llevando a cabo. Ver figura 14.

Figura 14. Exportar fichero a ARFF

2.4.2. Plugins

El sistema es capaz de importar *plugins*, cada uno de los cuales contiene un método de selección de atributos relevantes en microarreglos de ADN, además de la interfaz gráfica que permitirá su configuración. Los *plugins* son una parte muy importante dentro del sistema debido a que son los encargados de procesar los datos y dar un resultado como salida.

Ventajas de usar plugins:

- ✓ Permite que desarrolladores externos colaboren con la aplicación principal extendiendo sus funcionalidades.
- ✓ Reduce en tamaño de la aplicación.
- ✓ Permite al usuario utilizar solo las funcionalidades que necesita.
- ✓ Si algún plugin necesita ser actualizado no se afecta la aplicación principal.

2.4.3. Datos

Los datos con los que trabajará el sistema estarán contenidos en ficheros de texto plano, los cuales cargará en memoria y luego procesará para dar una salida. Los resultados podrán ser guardados para posterior análisis. El contenido de estos ficheros es muy sencillo debido a que consiste en una matriz rectangular o cuadrada en las que se distribuyen valores que indican el grado de expresión de un gen bajo una determinada condición experimental.

2.4.4. Hilos

Un hilo de ejecución es la unidad más pequeña de proceso que un sistema operativo puede generar. La implementación de hilos y procesos es distinta en cada sistema operativo, pero, en la mayoría de los casos, un hilo está contenido dentro de un proceso. Pueden existir varios hilos en un mismo proceso y pueden compartir recursos como la memoria, por otro lado distintos procesos no comparten estos recursos. En particular, los hilos de un proceso comparten las instrucciones del proceso (su código) y su contexto (los valores de sus referencias en un cierto momento). Para ofrecer una analogía, varios hilos en un proceso se parecen a varios cocineros leyendo el mismo libro de recetas y siguiendo las instrucciones, pero no necesariamente de la misma página. La tecnología que permite a los programadores el desarrollo de aplicaciones que realicen tareas concurrentes es conocida bajo el nombre de *Multithreading*.

En un solo procesador, el uso de múltiples hilos generalmente como una multiplicación de tiempo dividido en la que el procesador va cambiando entre distintas hilos. El cambio de contexto de este tipo sucede a tal velocidad que el usuario percibe que los hilos o las tareas están corriendo al mismo tiempo. En un multiprocesador o sistema de más de un núcleo, de hecho, los hilos o tareas van a correr al mismo tiempo, ya que cada procesador va a correr un hilo en particular.

Uno de los aspectos más importantes a la hora de trabajar con hilos en conocer el ciclo de vida que rige estas unidades de ejecución. En cualquier instante de tiempo un hilo se encuentra en uno de varios posibles estados. En la figura 15 se muestran los posibles estados por lo que puede transitar un hilo.

Un hilo inicia su ciclo de vida en el estado “Sin iniciar”, en el cual permanece hasta que el programa llame al método “Iniciar” del hilo, el cual lo ubica en el estado “Iniciado” e inmediatamente regresa el control al hilo que hizo la llamada. Luego el hilo que invocó “Iniciar” y cualquier otro u otros hilos en el programa se ejecutan concurrentemente.

Cuando un hilo iniciado recibe un procesador por primera vez y se convierte en el hilo en ejecución, este ejecuta su delegado ***ThreadStart***, el cual especifica las acciones que este hilo realizará durante su ciclo de vida. El delegado ***ThreadStart*** debe ser un método que devuelva ***void*** y que no reciba parámetros.

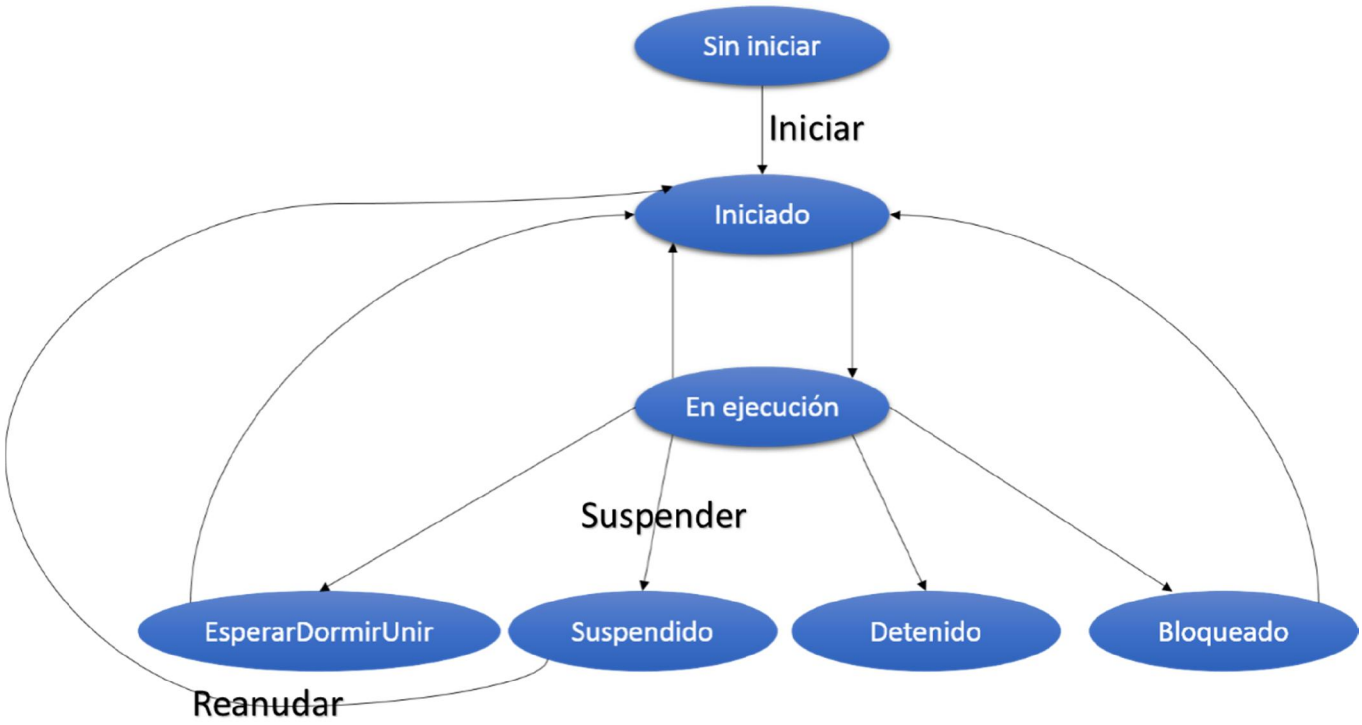


Figura 15. Ciclo de vida de un hilo

Comparación entre hilos y procesos

Los hilos son diferentes de los procesos de un sistema operativo en los siguientes puntos:

- Los procesos son típicamente independientes, mientras que los hilos existen como subcomponentes de un proceso.

- Los procesos llevan una cantidad importante de información de estado; múltiples hilos dentro de un proceso comparten estado, memoria y otros recursos.

- Los procesos tienen distintos espacios de direcciones, en cambio, los hilos comparten su espacio de dirección.

- Los procesos interactúan solo a través de mecanismos de comunicación entre procesos que provee el sistema.

- Un cambio de contexto entre hilos dentro del mismo proceso es típicamente más rápido que un cambio de contexto entre distintos procesos.

Uso de Hilos en la solución

Los hilos son utilizados para dar la posibilidad a un sistema de ir realizando ciertas tareas en segundo plano mientras el usuario puede ir realizando otras, de lo contrario la ejecución de las primeras bloquearían el sistema y no lo liberarían hasta terminar. Las ejecuciones que se llevan a cabo en la aplicación que aquí se presenta pueden demorar hasta días, en dependencia del tamaño del conjunto de datos de entrada. Es por esta razón que se hace uso de hilos para poder realizar varias ejecuciones, monitorizarlas y continuar trabajando en la aplicación. La aplicación coloca el código de ejecución de cada filtro dentro de un hilo y luego permite ejecutarlos concurrentemente [31]. Luego de iniciar la ejecución de un filtro dentro hilo entonces se tiene la posibilidad de detener o pausarla.

2.5 Integración con el Weka

Esta sección tiene como objetivo establecer como se realiza la exportación de un fichero en el formato ARFF (*Attribute-Relation File Format* – por su nombre en inglés) como medio de integración entre la Solución propuesta y el Weka.

ARRFF es un fichero en texto plano que describe una lista de instancias que comparten un grupo de atributos. Este tipo de fichero está formado por dos secciones. La primera sección se llama Cabecera y la segunda Datos. La sección cabecera contiene el nombre de la relación, una lista de atributos (las columnas en Datos) y sus tipos. La sección datos contiene una lista de instancias.

La solución propuesta se integra con el Weka a través de la generación de un fichero ARFF que se obtiene de transformar los resultados de la ejecución de un filtro (guardado). Esta transformación se lleva a cabo mediante la ubicación de cada variable dentro del fichero fuente sobre el que se ejecutó el método filtro y de ahí se extrae el nivel de expresión de esa variable (gen) ante las condiciones experimentales que son encontradas en el mismo fichero. Acompañando al fichero fuente tiene que encontrarse otro fichero en el que se encuentra la codificación utilizada para la primera columna del fichero fuente. Este fichero se utiliza para determinar a qué clase pertenece cada instancia reflejada en el fichero ARFF exportado.

En la figura 3 se muestra el fichero ARFF obtenido como resultado de exportar los resultados de una ejecución del filtro Massive. Para explicar de forma más clara el contenido del fichero solamente se tomó un fragmento del mismo. **@Relation** indica el nombre de la relación contenida en el fichero. **@Attribute** denota un atributo, del cual se especifica su tipo, dentro de las instancias que se encuentran debajo de **@Data** (indicando el comienzo de las instancias). Cada atributo debe ser especificado en una instancia en el mismo orden en que fue declarado. El atributo **gene-index** es el número de la columna en la que se encuentra el un gen determinado en el fichero fuente. El atributo **gene-expression** indica el grado de expresión gen dado. El atributo **predicted-gene** indica si una instancia corresponde al gen sobre el cual se buscaron las variables más predictivas. Por último, el atributo **class** se relaciona con la condición experimental o muestra sobre la cual un gen tiene cierto nivel de expresión.

```
@Relation test

@Attribute gene-index numeric
@Attribute gene-expression numeric
@Attribute predicted-gene {yes, no}
@Attribute class  {B-cell, T-cell, AML}

@Data
12,250,yes,B-cell
12,50,yes,B-cell
12,-60,yes,B-cell
12,250,yes,T-cell
12,250,yes,T-cell
17,150,no,B-cell
17,40,no,B-cell
17,-10,no,B-cell
17,232,no,T-cell
17,247,no,T-cell
```

Figura 3. Fragmento de un fichero ARFF exportado desde la Solución propuesta

2.6. Desarrollo de un nuevo plugin compatible con la Solución

En el presente epígrafe se describen los pasos y aspectos a tener en cuenta para desarrollar un *plugin* que sea compatible y se integre correctamente con la solución propuesta.

2.6.1. Aspectos a tener en cuenta

En la figura 4 se muestra el modelo de dominio de la Solución propuesta. En el mismo se pueden observar algunos aspectos a tener en cuenta para la correcta implementación de un nuevo *plugin* compatible con la solución propuesta.

La implementación de un *plugin* se divide en tres elementos principales:

1. **Filtro:** este elemento consiste en una clase que implementa la clase interfaz IFilter, la cual define una parte muy importante para la integración de todo filtro que se implemente y se pretenda integrar a la solución propuesta. Este contiene el método filtro sobre el que se gestiona la ejecución desde la aplicación principal.
2. **Interfaz de usuario:** este elemento es el más cercano al usuario, pues a través de este se le brinda la posibilidad de configurar los datos con los que trabajará el filtro cuando se ejecute.
3. **Controladora:** este elemento implementa la clase interfaz IController y se encarga de controlar la interacción entre el Filtro y la Interfaz de usuario, además de ser el punto de entrada del *plugin* implementado a la aplicación principal.

2.6.2. Biblioteca de clases common

La biblioteca de clases **common** encapsula dos elementos importantes. El primero son las clases interfaz necesarias para implementar un nuevo *plugin* (las cuales fueron descritas en el apartado anterior) y el segundo, los procedimientos que se identificaron como reiterativos por el uso necesario que hace un filtro para cargar los datos que necesita para su funcionamiento y guardar los resultados arrojados por el mismo para posterior consulta. Estos procedimientos son también encapsulados en la biblioteca de clases mencionada. Ver la figura 16.

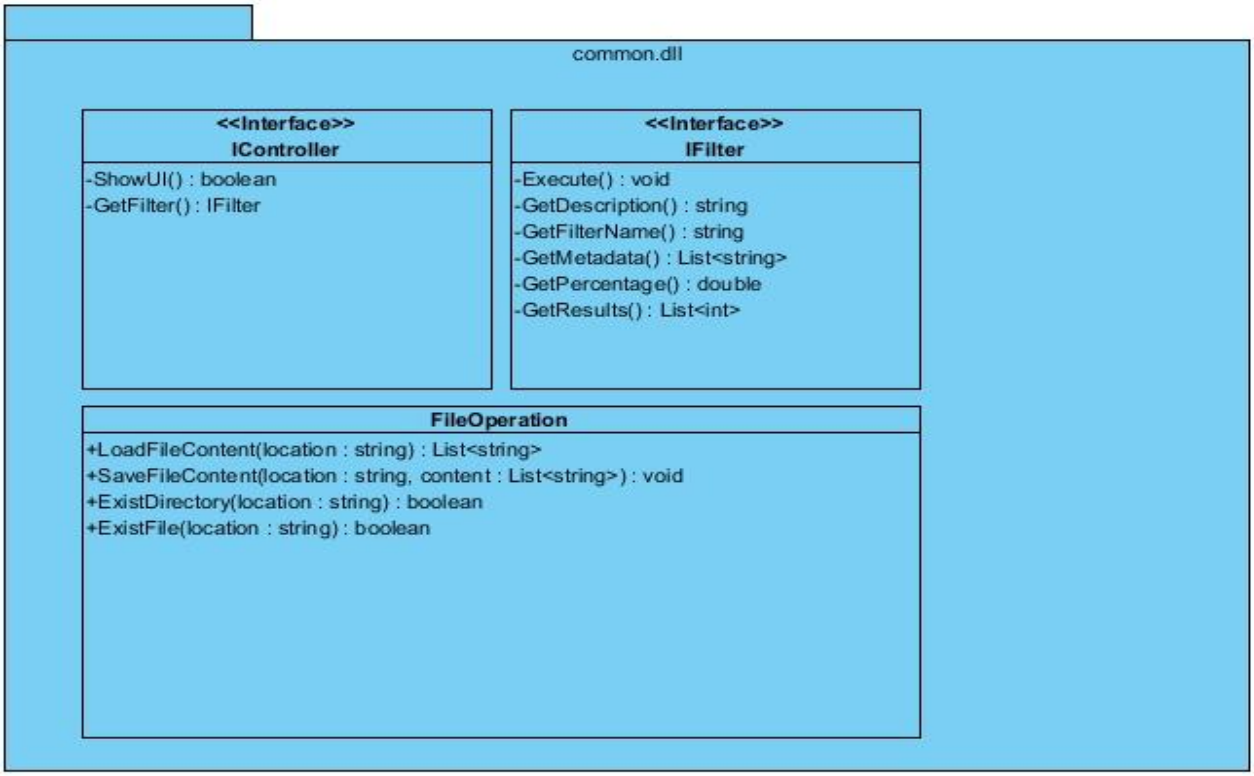


Figura 16. Elementos que conforman la biblioteca de clases *common*

Descripción de la clase interfaz IFilter

Como se mencionó anteriormente la clase interfaz **IFilter** establece las características comunes que tendrán todos los métodos filtros implementados con el objetivo de integrarlos a la solución propuesta. A continuación se describen los métodos definidos por esta clase interfaz.

Execute(): este método tiene la tarea de ejecutar todo el código relacionado con la ejecución del método filtro que se implemente. Al finalizar la ejecución del filtro, dentro de este mismo método deben recogerse los resultados y ser guardados en una variable que luego se devolverá a través del método **GetResults()**. Este método es muy importante pues donde se lleva a cabo todo el procesamiento de datos y es también el que se utiliza desde la aplicación principal para gestionar la ejecución del filtro que este contiene.

GetDescription(): método que tiene como objetivo devolver una descripción del filtro implementado para conocimiento del usuario que lo esté utilizando. Aquí, en forma de texto, se puede incluir información valiosa y útil que pueda ser utilizada para ampliar los conocimientos sobre un filtro dado.

GetFilterName(): devuelve el nombre del filtro implementado dentro del *plugin* sobre el que se realiza su llamada.

GetMetadata(): devuelve un conjunto de información sobre la ejecución de un filtro dado, la que se estima muy importante para el usuario. Esta información está estructurada en forma de diccionario, o sea que a cada fragmento de información se le asocia un identificador. Tener en cuenta que los datos que devuelva este método lo define el desarrollador del *plugin*.

GetPercentage(): devuelve el porciento de ejecución que lleva el método **Execute** en un instante dado. La implementación de este método es muy particular de cada filtro implementado, aunque se puede destacar que lo idóneo es utilizar una formula basada en teoría de complejidad de algoritmos que dependa únicamente de un instante de tiempo dado.

GetResults(): devuelve los resultados de la ejecución del método **Execute**.

Descripción de la clase interfaz IController

Como se hizo mención anteriormente la clase interfaz **IController** establece los puntos comunes que deben compartir todos los *plugins* implementados con el objetivo de ser reconocidos por la aplicación principal. A continuación se describen los métodos definidos por esta clase interfaz.

ShowUI(): la llamada de este método se realiza desde la aplicación principal. Este muestra la interfaz de usuario definida por el *plugin* que lo implementa. Luego del usuario completar los campos requeridos para la ejecución del método filtro y este cierre la interfaz, el método devuelve un valor indicando si ocurrió algún problema o todo transcurrió de forma correcta.

GetFilter(): la llamada de este método se realiza desde la aplicación principal luego de haberse mostrada la interfaz de usuario. Este método devuelve un objeto de tipo IFilter que no es más que un filtro configurado y listo para entrar en ejecución.

Descripción de la clase estática FileOperation

Esta clase estática contiene un conjunto de métodos estáticos que son necesarios para la carga de los datos que utilizarán los métodos filtros para su ejecución. A continuación se describen los mismos.

LoadFileContent(): devuelve el contenido de un fichero del cual su ubicación se pasa por parámetro.

SaveFileContent(): crea un fichero, dada su ubicación, almacenando la información contenida en una lista pasada por parámetro.

ExistFile(): método que devuelve si existe un fichero del cual se conoce su posible ubicación.

ExistDirectory(): método que devuelve si existe un directorio (carpeta) del cual se conoce su posible ubicación.

2.6.3. Pasos para la implementación de un plugin

Primer paso: Creación de un nuevo proyecto.

Para dar cumplimiento a este paso es necesario tener abierto el IDE Visual Studio, luego dar click al botón **NuevoProyecto**. Al realizar esta acción debe a aparecer una ventana en la que se debe escoger crear un proyecto de tipo Biblioteca de clases. Ver figura 17.

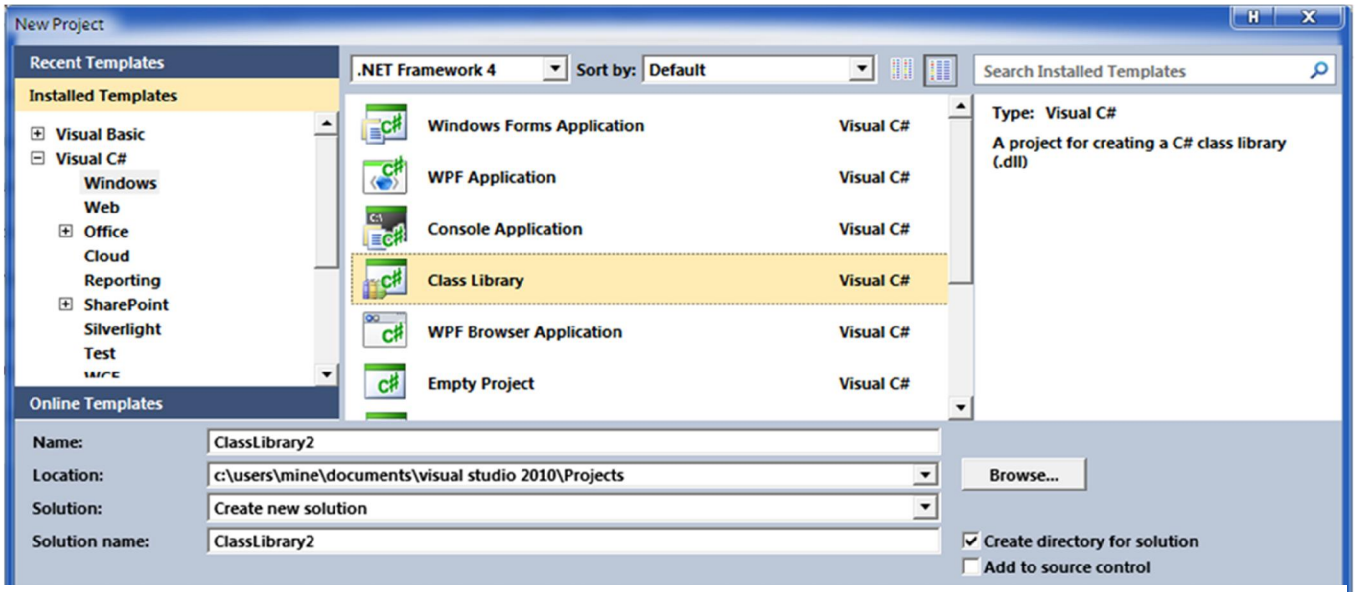


Figura 17. Crear un nuevo proyecto

Segundo Paso: Incluir la biblioteca de clases **common** (la cual fue descrita en el apartado anterior)

Para incluir la biblioteca de clases **common** en el nuevo *plugin* que se esté implementando se debe abrir el **Explorador de Solución**, luego seleccionar el proyecto y hacer derecho, del menú contextual que aparece se debe hacer click sobre la opción **Agregar referencia**. Después de esto tan solo se debe localizar la biblioteca y esta aparecerá como parte del proyecto. Ver figura 18.

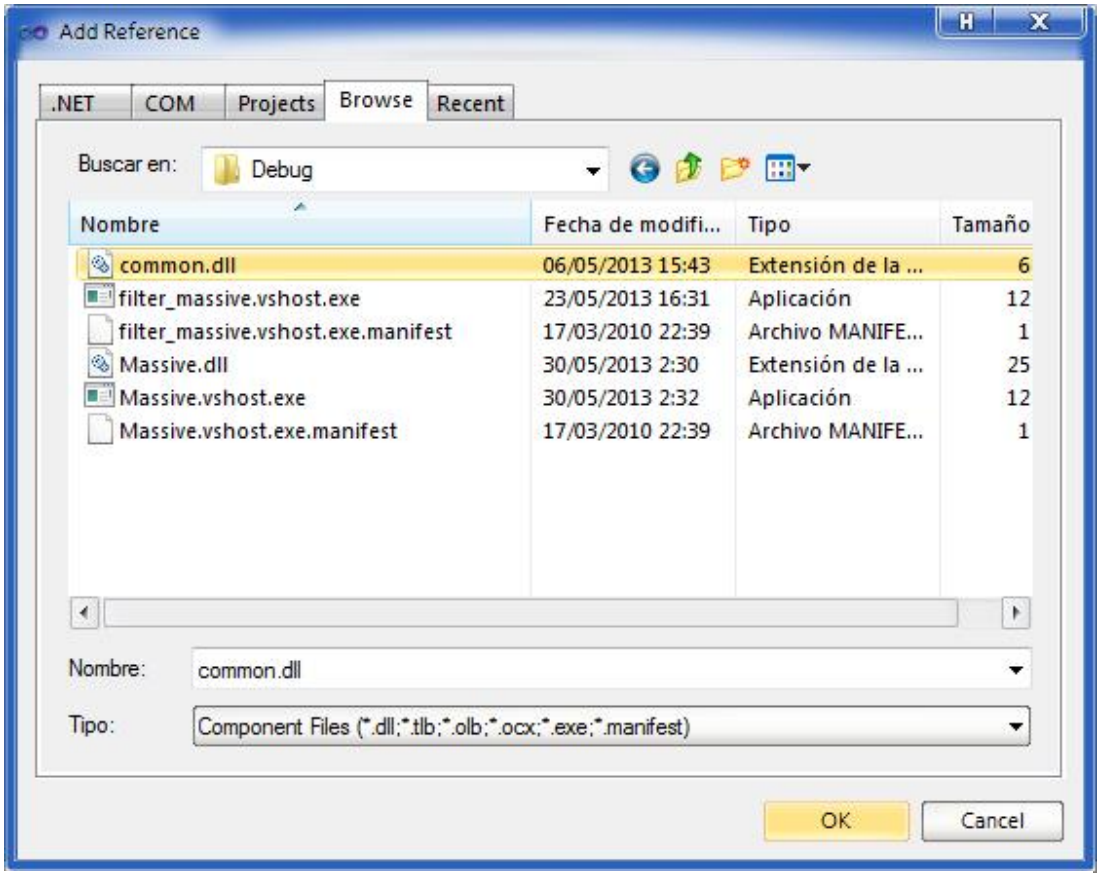


Figura 18. Añadir referencia a la biblioteca de clases *common*

Tercer Paso: Creación de la clase filtro

Al crear la clase filtro, la misma debe heredar de la clase interfaz IFilter e implementar todos sus métodos, los cuales fueron descritos en el apartado anterior.

Cuarto Paso: Creación de la interfaz de usuario

En este paso se deben crear uno o varios formularios, según las necesidades del filtro que se esté implementando. Luego de esta operación se deben agregar los componentes gráficos necesarios para capturar la entrada proveída por el usuario para poder instanciar el filtro implementado en el paso anterior. Hay que señalar que en este paso se tiene que validar la entrada de los datos, pues la aplicación principal no asume esta responsabilidad. Se sugiere el uso de componentes gráficos que sean de selección en lugar

de componentes de entrada por teclado, sin que esto último atente contra la flexibilidad del *plugin.* Ver figura 19.

```
public class Massive : IFilter
{
    attributes
    public Massive(string inputData, string estimator, string engine, int output_variable, int selection_size, int rows, int columns)

    public Massive()...

    private void LoadValues(string inputData, int m, int n)...

    Help methods

    #region IFilter methods
    public void Execute()...

    public List<int> getResults()
    {
        return results;
    }

    public String getDescription()...

    public Dictionary<String, String> getMetadata()...

    public String getFilterName()...

    public double getPercentage(long seconds)...
    #endregion


}
```

Figura 19. Herencia de la clase interfaz IFilter

Quinto Paso: Creación de la clase controladora

Esta clase es el punto de encuentro entre la clase filtro implementada y la interfaz de usuario creada para capturar la entrada de parámetros para instanciar el mismo. Esta clase debe heredar y en consecuencia implementar los métodos definidos por la clase interfaz IController, la cual fue descrita en el apartado anterior. Ver figura 20.

```
public class Controller:IController
{
    UserInterface _user_interface;
    Massive _massive_filter;

    public Massive MassiveFilter
    {
        get { return _massive_filter; }
        set { _massive_filter = value; }
    }

    public Controller()
    {
        _user_interface = new UserInterface();
    }

    public bool ShowUI()
    {

        if (_user_interface.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            _massive_filter = _user_interface.MassiveInstance;
            return true;
        }
        else
        {
            _massive_filter = null;
            return false;
        }
    }
    public IFilter GetFilter()
    {
        return _massive_filter;
    }

}
```

Figura 20. Herencia de la clase interfaz IController

Sexto Paso: Comprobación de la integración con la aplicación principal

En este punto se puede importar el *plugin* creado dentro de la aplicación, y debería importarse correctamente si se siguieron los pasos anteriores y se respetaron las reglas impuestas para la creación del *plugin.* Para una completa comprobación de integración se deben realizar todas las funcionalidades

presentes en la aplicación principal. Si luego de esta comprobación todo funciona de forma correcta, la implementación fue todo un éxito.

Nota: evidentemente antes de realizar esta acción se debe estar seguro que el *plugin* funciona como se espera, para lo cual se puede utilizar un pequeño truco consistente en cambiar el tipo de nuestro proyecto que es de tipo **Biblioteca de clases** a **Aplicación de Windows**, y de esta forma se lograría ejecutar nuestro *plugin* como si de una aplicación de escritorio se tratara. Para ello, se debe desde el **Explorador de Solución** hacer click derecho sobre el proyecto y seleccionar propiedades, una vez allí se debe seleccionar en la opción tipo de salida el tipo deseado, después se compila y se obtiene un ejecutable. Ver figura 21.

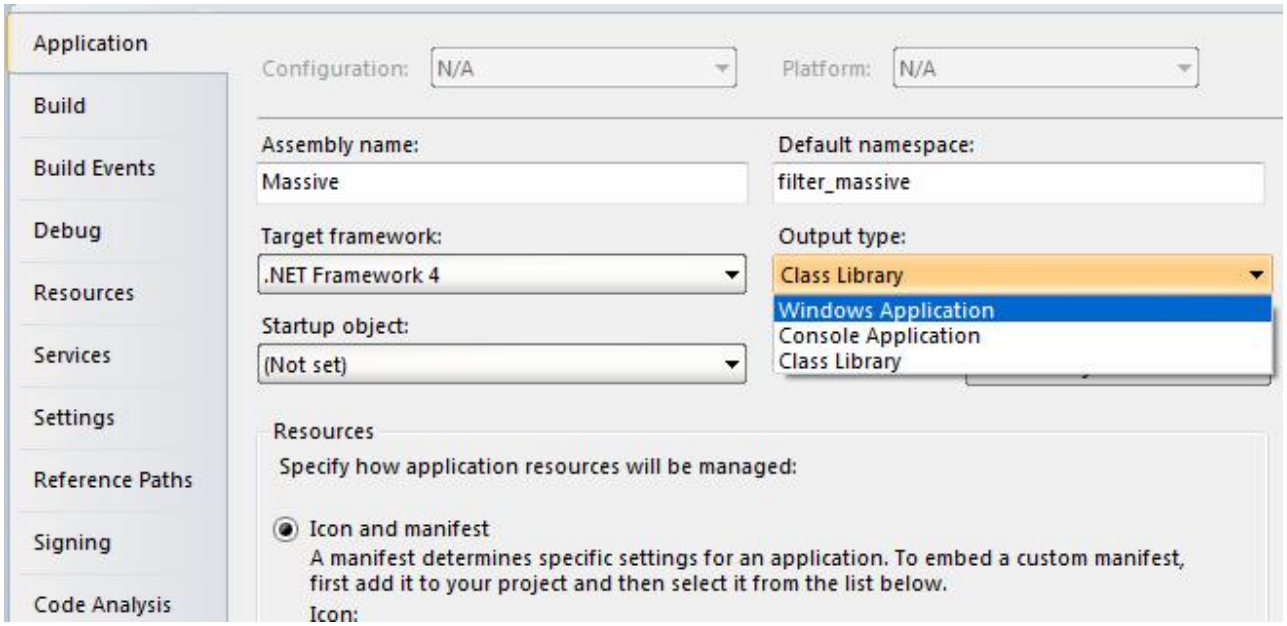


Figura 21. Cambio de tipo de proyecto

2.7. Estándar de codificación y tratamiento de excepciones

Las nuevas tendencias en el desarrollo de software requieren uniformidad en el código fuente de las aplicaciones desarrolladas además de un correcto tratamiento de excepciones dentro de las mismas para de esta forma aumentar su calidad, fiabilidad y robustez.

2.7.1. Estándares de codificación

Los estándares de codificación (llamados también convenciones de código o estilos de código) definen normas para escribir código fuente en ciertos lenguajes de programación.

Un buen estilo de código tiene como piedras angulares los siguientes elementos:

- Nombrar las variables apropiadamente.
- Estilo de indentación.
- Espaciado.

Ventajas de usar un buen estilo de código:

- Mayor legibilidad del código fuente.
- Mayor facilidad de mantenimiento.
- Mayor calidad en el software.

2.7.2. Tratamiento de excepciones

Durante la ejecución de un sistema, en este pueden darse condiciones que le conduzcan a funcionar de forma anómala. Estas situaciones inesperadas son denominadas excepciones. El correcto tratamiento de excepciones es una de las claves para lograr sistemas fiables y robustos, puesto que estas podrían desestabilizar su funcionamiento y con esto afectar las tareas que los mismos realizan, con el consiguiente impacto negativo que tengan estas últimas.

En la Solución propuesta se emplea la captura y tratamiento de excepciones extensivamente, en especial, en fragmentos de código donde es más probable la ocurrencia de excepciones, para de esta forma evitar la interrupción del sistema. En especial, se utilizan el tratamiento de excepciones en los procedimientos donde se realizan llamadas de control sobre hilos (ejecutar, pausar y detener), también se utilizan en las llamadas donde se realizan operaciones sobre ficheros (lectura, escritura, borrado), pues estas son propensas a provocar este tipo de situaciones.

CAPÍTULO 2. DESARROLLO DE LA SOLUCIÓN PROPUESTA

En este capítulo se describieron los aspectos más importantes relacionados con el desarrollo de la Solución propuesta. Se propusieron los pasos necesarios para la implementación de un nuevo *plugin* que se pueda integrar con la Solución. La metodología seleccionada se adaptó fácilmente al objetivo trazado. Todos los requerimientos fueron cumplidos, pues la aplicación obtenida brinda la gran ventaja de permitirle a un investigador realizar experimentos con datos provenientes de microarreglos de ADN, guardar organizadamente dichos datos y poder analizarlos en la herramienta Weka.

CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

El software se prueba para descubrir errores cometidos al realizar su diseño y construcción. Una prueba es un conjunto de actividades que se planean con anticipación y se realizan de forma sistemática. Con el objetivo de comprobar la validez de la solución implementada, en el presente capítulo se le realizan pruebas a la misma.

"El optimismo es el peligro ocupacional de la programación; la prueba, el tratamiento"

Kent Beck[1]

3.1. Validación solución vs MASSIVE

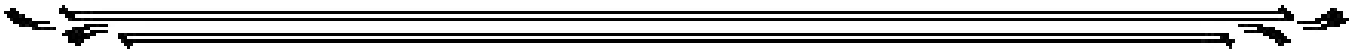
Se realizaron un conjunto de ejecuciones para comprobar que la solución implementada devuelve los mismos resultados que la implementación del autor del método. Para la realización de esta prueba se utilizó una computadora con las siguientes características: procesador Intel core2duo T5670 1.8GHz y 2GB de memoria RAM. Los resultados demuestran que la solución implementada devuelve los resultados mismos resultados que la implementación del autor.

En la siguiente tabla se recogen las ejecuciones realizadas con diferentes juegos de datos y diferentes configuraciones de selección de variables. La columna **fuente de datos** se refiere a los juegos de datos utilizados durante cada ejecución (Ver Anexo 2). La columna **tamaño** se refiere a las dimensiones de un conjunto dado (Filas x Columnas). La columna **selección** dice cuántas variables se seleccionaron. La columna **tiempo de ejecución** dice cuanto se demoró la solución propuesta (verde) y la implementación del autor (rojo) en devolver los resultados. La columna **coincidencia** dice si los resultados de amblas implementaciones coincidieron.

[1] Estadounidense, ingeniero de software, creador de la metodología ágil de desarrollo de software.

No	Fuente de datos	Tamaño	Selección	Coincidencia
1	1	300x15	10	✓
2	1	300x15	13	✓
3	2	1500x72	15	✓
4	2	1500x72	20	✓
5	3	5328x72	12	✓
6	3	5328x72	17	✓
7	4	1500x174	14	✓
8	4	1500x174	19	✓
9	5	2500x174	11	✓
10	5	2500x174	18	✓

Tabla 1. Muestra de ejecuciones realizadas



Conclusiones

A partir del estudio realizado y de los resultados obtenidos puede afirmarse que se han cumplido el objetivo general y las tareas de la investigación, por esto se arriba a las siguientes conclusiones:

- ✓ Se realizó una profunda fundamentación teórica sobre los métodos de selección de selección de atributos relevantes en microarreglos de ADN.
- ✓ Se identificaron los métodos filtros existentes de selección de atributos relevantes en microarreglos de ADN.
- ✓ Se implementó el método filtro MASSIVE, el cual se integra en forma de *plugin* a la solución propuesta.
- ✓ Se implementó una herramienta con interfaz gráfica para la parametrización, ejecución y monitorización de experimentos con microarreglos de ADN. La misma está basada en los principios de buenas prácticas en el diseño e implementación de aplicaciones concurrentes.
- ✓ Fue posible realizar la validación de los resultados de la investigación comprobando la eficacia de la solución propuesta.

Recomendaciones

Se recomienda para futuros trabajos:

- ✓ Implementar los otros métodos filtros identificados e integrarlos a la solución propuesta.

Referencias Bibliográficas

1. **Miralles, Ángela**. El Proyecto Genoma Humano: algunas reflexiones sobre sus relaciones con el Derecho. Tirant lo Blanch, 1997. 177.
2. **Goodman, Nathan**. Biological data becomes computer literate: new advances in bioinformatics. Current opinion in biotechnology, 2002, vol. 13, no 1, pág. 68-71.
3. **Wirta, Valtteri.** *Mining the transcriptome – methods and applications*. Royal Institute of Technology, Estocolmo, 2006. pág. 62.
4. **Catalán, Victoria y García-Foncillas, Jesús**. Tecnología microarray y cáncer. *Formación Médica Continuada*, 2008, vol. 3, no 1, pág. 18-30.
5. **Lastra, Guido; Manrique, Camila.** *Microarreglos: herramienta para el conocimiento de las enfermedades.* Asociación Colombiana de Reumatología, 2005, vol. 12, pág. 263-267.
6. Ídem 4.
7. **Li, Jinyan, Wong, Limsoon y Yang, Qiang.** Guest Editors´ Introduction: Data Mining in Bioinformatics. *IEEE Computer Society.* [En línea] [Citado el: 4 de Diciembre de 2012]. http://www.computer.org/csdl/mags/ex/2005/06/x6016.html.
8. **González Maestre, José.** *Análisis de datos de microarrays.* Ibime, España, 2010.
9. **Alba, E, García Nieto, J y Luque, G.** Algoritmos basados en cúmulos de partículas para el análisis de microarreglos de ADN. *Neo.* [En línea] http://neo.lcc.uma.es/staff/jmgn/doc/maeb07.pdf.
10. **National Center for Biotechnology Information**, Sitio oficial. [En línea] http://www.ncbi.nlm.nih.gov
11. Ídem 3.
12. **Zepeda Castilla, Ernesto José y Recinos Money, Edgar**. Clasificación molecular del cáncer de mama. Medigraphic, México DF, 2008, Vol. 76.

13. **Golub TR, Slonim DK y Tamayo P**. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science 1999.

14. **Perou CM, Sorlie T y Eisen MB**. Molecular portraits of human breast tumours. Nature 2000.

15. **Dhanasekaran, SM, Barrette, TR y Ghosh, D**. Delineation of prognostic biomarkers in prostate cancer. Nature 2001.

16. Ídem 3.

17. **Díaz Travieso, Servando y Suárez Suárez, Adair**. Software para el Procesamiento de Imágenes de Microarray. Universidad de las Ciencias Informáticas. La Habana, 2008. 75.

18. **Meyer, Patrick Emmanuel, Schretter, Colas y Bontempi, Gianluca**. Information-Theoretic Feature Selection in Microarray Data Using Variable Complementarity. IEEE Journal of Selected Topics in Signal Processing, vol. 2, no. 3, pág. 261-274.

19. **Peng, Hanchuan, Long, Fuhui y Chris Ding**. Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pág. 1226-1238.

20. Ídem 19.

21. **Subio platform**, Sitio oficial. [En línea]. http://www.subio.jp/products/platform

22. **Affymetrix**, Sitio oficial. [En línea]. http://www.affymetrix.com/estore/browse/products.jsp?productId=131429#1_1

23. Ídem 18.

24. **Weka**, Sitio oficial [En línea]. www.cs.waikato.ac.nz/ml/weka/

25. **Microsoft**, Sitio oficial [En línea]. http://www.microsoft.com/visualstudio

26. **Visual Paradigm**, Sitio Oficial. [En línea]. http://www.visual-paradigm.com
27. **Larman, Craig**. UML y patrones: Introducción al análisis y diseño orientado a objetos. Prentice Hall, México, 1999. 507.

28. **Larman, Craig**. UML y patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Prentice Hall, México, 1999. 520.

29. **Pilone, Dan y Miles, Russ**. Head First Software Development. O'Reilly, EEUU, 2008. 477.

30. **Keisler, Stephen**. Software Paradigms. Wiley, New Jersey, 2005.458.

31. **Deitel, Hervey**. C# How to program. Prentice Hall, Washington DC, 2002.1560.

Bibliografía

Affymetrix, Sitio oficial. [En línea]. http://www.affymetrix.com/browse/brand/affymetrixMicroarraySolutions/brandAffymetrixMicroarraySolutions-overview.jsp

Alba, E, García Nieto, J y Luque, G. Algoritmos basados en cúmulos de partículas para el análisis de microarreglos de ADN. *Neo.* [En línea] http://neo.lcc.uma.es/staff/jmgn/doc/maeb07.pdf.

Autores, Colectivo de. *Revista de Disformología y Epidemiología.* Madrid, 2010, vol. 5, no. 9.

Catalán, Victoria y García-Foncillas, Jesús. Tecnología microarray y cáncer. *Formación Médica Continuada*, 2008, vol. 3, no 1, pág. 18-30.

Deitel, Hervey. C# How to program. Prentice Hall, Washington DC, 2002.1560.

Dhanasekaran, SM, Barrette, TR y Ghosh, D. Delineation of prognostic biomarkers in prostate cancer. Nature, 2001.

Díaz Travieso, Servando y Suárez Suárez, Adair. Software para el Procesamiento de Imágenes de Microarray. Universidad de las Ciencias Informáticas. La Habana, 2008. 75.

Domenech Sánchez, Antonio y Vila, Jordi. Fundamento, tipos y aplicaciones de los arrays de ADN en la microbiología médica. Mallorca.

Garza Ramos, Ulises, Silva Sánchez, Jesús y Martínez Romero, Esperanza. Scielo. *Scielo.* [En línea] http://www.scielo.org.mx/pdf/spm/v51s3/a09v51s3.pdf

González Maestre, José. *Análisis de datos de microarrays.* Ibime, España, 2010.

Golub TR, Slonim DK y Tamayo P. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science, 1999.

Goodman, Nathan. Biological data becomes computer literate: new advances in bioinformatics. Current opinion in biotechnology, 2002, vol. 13, no 1, pág. 68-71.

Keisler, Stephen. Software Paradigms. Wiley, New Jersey, 2005.458.

Larman, Craig. UML y patrones: Introducción al análisis y diseño orientado a objetos. Prentice Hall, México, 1999. 507.

Larman, Craig. UML y patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Prentice Hall, México, 1999. 520.

Lastra, Guido, Manrique, Camila. *Microarreglos: herramienta para el conocimiento de las enfermedades.* Asociación Colombiana de Reumatología, 2005, vol. 12, pág. 263-267.

Li, Jinyan, Wong, Limsoon y Yang, Qiang. Guest Editors´ Introduction: Data Mining in Bioinformatics. *IEEE Computer Society.* [En línea] [Citado el: 4 de Diciembre de 2012]. http://www.computer.org/csdl/mags/ex/2005/06/x6016.html.

Pautas de uso de C# [En línea] [Citado el: 7 de abril de 2013.] **http://msdn.microsoft.com/en-US/library/ms229004%28v=vs.80%29.aspx**

Perou CM, Sorlie T y Eisen MB. Molecular portraits of human breast tumours. Nature 2000.

Peng, Hanchuan, Long, Fuhui y Chris Ding. Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pág. 1226-1238.

Pilone, Dan y Miles, Russ. Head First Software Development. O'Reilly, EEUU, 2008. 477.

Rodríguez Baena, Domingo Savio. Análisis de datos de Expresión Genética mediante técnicas de Biclustering. Sevilla, 2006. 101.

Rovira Forns, Joan. *La evaluación económica en Farmacogenómica Oncológica y Hematología.* Medical economics, Madrid , 2009.

Subio platform, Sitio oficial. [En línea]. http://www.subio.jp/

Meyer, Patrick Emmanuel, Schretter, Colas y Bontempi, Gianluca. Information-Theoretic Feature Selection in Microarray Data Using Variable Complementarity. IEEE Journal of Selected Topics in Signal Processing, vol. 2, no. 3, pág. 261-274.

Microsoft, Sitio oficial [En línea]. http://www.microsoft.com/visualstudio

MSDN. [En línea]. http://msdn.microsoft.com/es-es/library/system.threading%28v=vs.80%29.aspx

Miralles, Ángela. El Proyecto Genoma Humano: algunas reflexiones sobre sus relaciones con el Derecho. Tirant lo Blanch, 1997. 177.

National Center for Biotechnology Information, Sitio oficial. [En línea] http://www.ncbi.nlm.nih.gov

Nuñez Jover, Jorge y Capecchi, Vittorio. Innovaciones creativas y desarrollo humano. Ediciones Trilce, 2006.

Visual Paradigm, Sitio Oficial. [En línea]. http://www.visual-paradigm.com

Weka, Sitio oficial [En línea]. www.cs.waikato.ac.nz/ml/weka/

Wirta, Valtteri. *Mining the transcriptome – methods and applications*. Royal Institute of Technology, Estocolmo, 2006. pág. 62.

Xinmin, Li y Weikuan, Gu. *DNA microarrays: Their Use and Misuse.* Microcirculation, vol. 9, no. 1, pág. 13-22.

Zepeda Castilla, Ernesto José y Recinos Money, Edgar. Clasificación molecular del cáncer de mama. Medigraphic, México DF, 2008, Vol. 76.

Glosario de Términos

ADN: ácido desoxirribonucleico, es un ácido nucleico que contiene instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos conocidos y algunos virus, y es responsable de su transmisión hereditaria.

Informática: disciplina que estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales.

Bioinformática: es la aplicación de tecnología de computadores a la gestión y análisis de datos biológicos.

Genoma: es la totalidad de la información genética que posee un organismo o una especie en particular.

Genómica: es el conjunto de ciencias y técnicas dedicadas al estudio integral del funcionamiento, el contenido, la evolución y el origen de los genomas.

Bases de Datos: Se define como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular.

Genética: es el campo de la biología que busca comprender la herencia biológica que se transmite de generación en generación.

Paradigma: sistema de reglas y reglamentos que establecen límites o fronteras, ofrecen una guía para tener éxito al resolver los problemas que están dentro de esos límites, además ofrecen un modelo para resolver problemas.

Gen: es una secuencia ordenada de nucleótidos en la molécula de ADN que contiene la información necesaria para la síntesis de proteínas.

Weka: Herramienta de Minería de Datos.

Secuenciación de ADN: es un conjunto de métodos y técnicas bioquímicas cuya finalidad es la determinación del orden de los nucleótidos en un oligonucleótido de ADN.

GLOSARIO DE TÉRMINOS

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.

Modularidad: propiedad de un software que permite dividirlo en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

Extensibilidad: propiedad de un software que le permite añadir o quitar funcionalidades con el mínimo impacto sobre este.

Software: programa o conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

TIC: tecnologías de la información y comunicación, son tecnologías y herramientas que las personas utilizan para intercambiar, distribuir y recolectar información y para comunicarse con otras personas.

Plugin: es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de cierta interfaz.

Anexos

Anexo 1. Esquema de arquitectura basada en plugins

En la siguiente figura se muestra como la aplicación principal brinda servicios que los *plugins* pueden usar, además de un protocolo por el cual pueden intercambiar información. Los *plugins* dependen de los servicios proveidos por la aplicación, pues estos no trabajan por ellos mismos. Por el contrario, la aplicación trabaja independientemente de los *plugins,* haciendo posible a los usuarios añadir u actualizarlos dinámicamente sin necesidad de realizar cambios en esta.

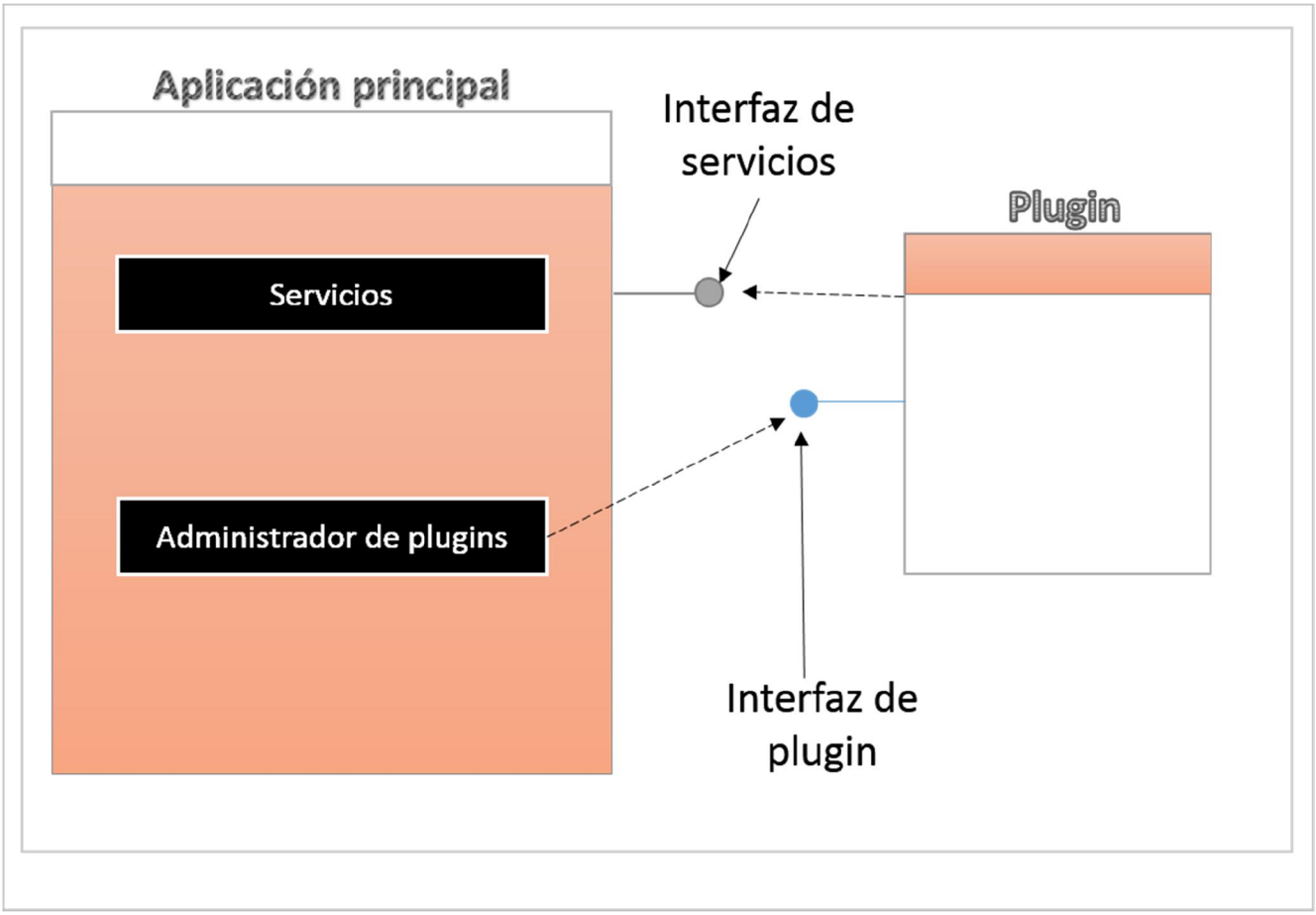


Figura 22. Esquema de la arquitectura basada en plugins

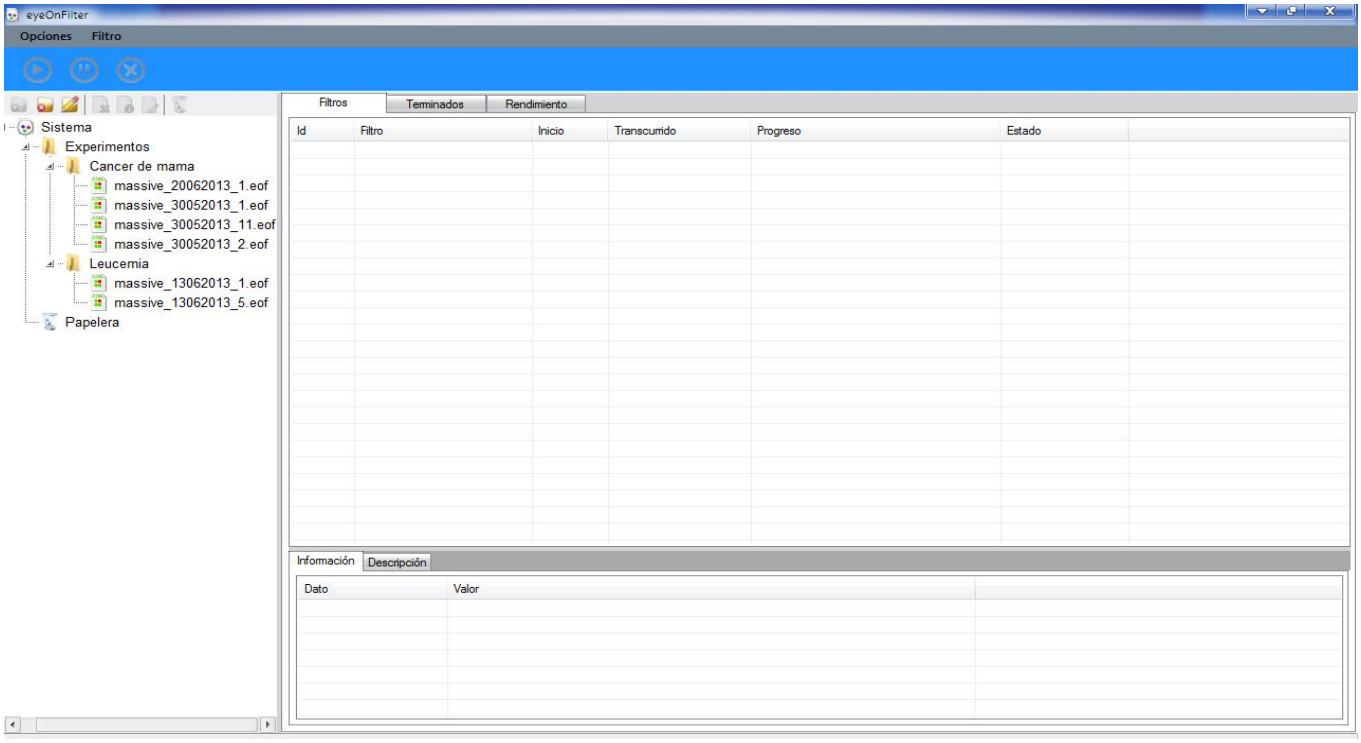
Anexo 2. Descripción de los juegos de datos

No	Nombre	Tamaño	Descripción
1	Aleatorio	300x15	Los datos son números generados aleatoriamente
2	Leukimia (1)	1500x72	Datos relacionados con la Leucemia
3	Leukimia (2)	5328x72	Datos relacionados con la Leucemia
4	Tumor (1)	1500x174	Datos relacionados con diferentes tipos de tumores
5	Tumor (2)	2500x174	Datos relacionados con diferentes tipos de tumores

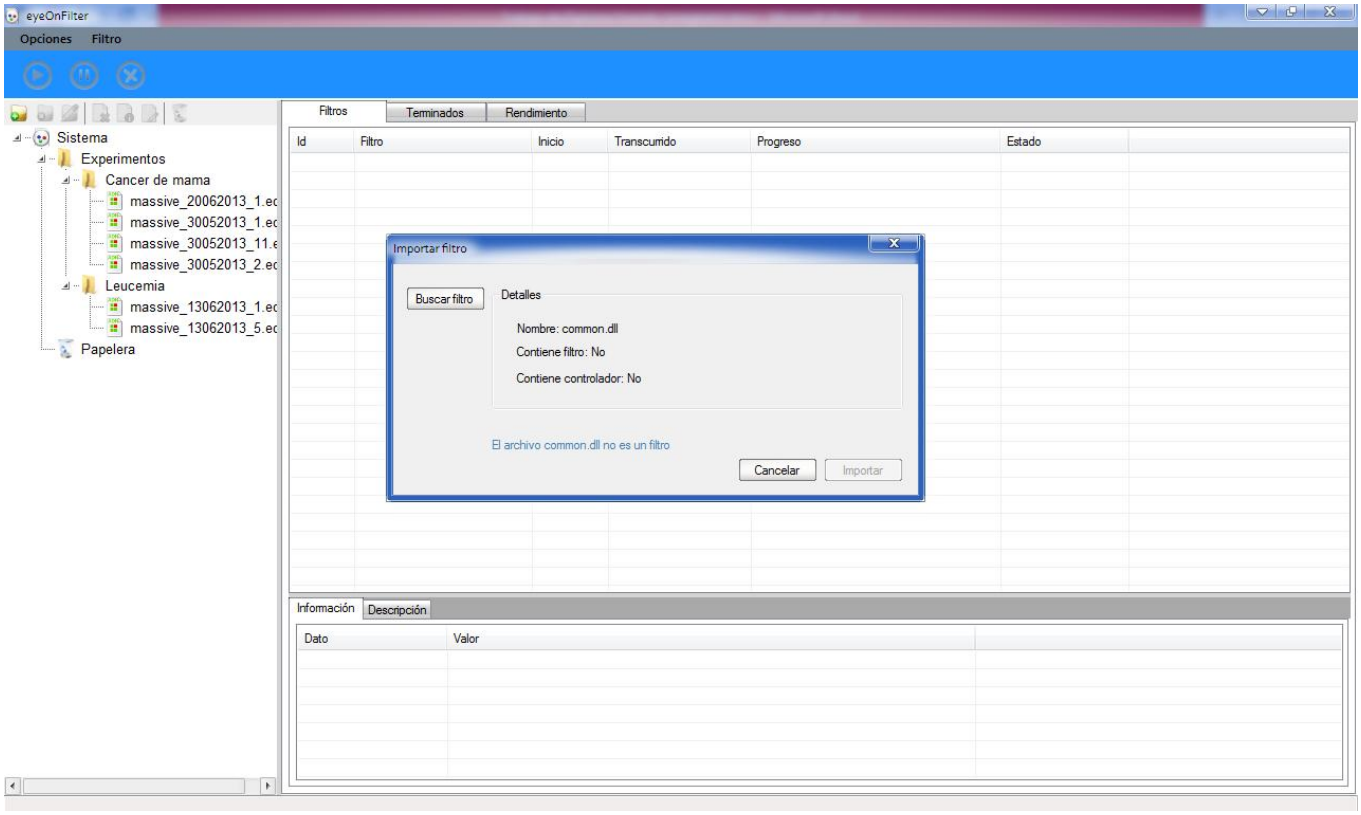
De 2 a 5 son datos de expresión de microarreglos. Son de dominio público, extraídos de la web de la Universidad de Plymouth[2].

[2] http://www.tech.plym.ac.uk/spmc/links/bioinformatics/microarray/microarray_cancers.html

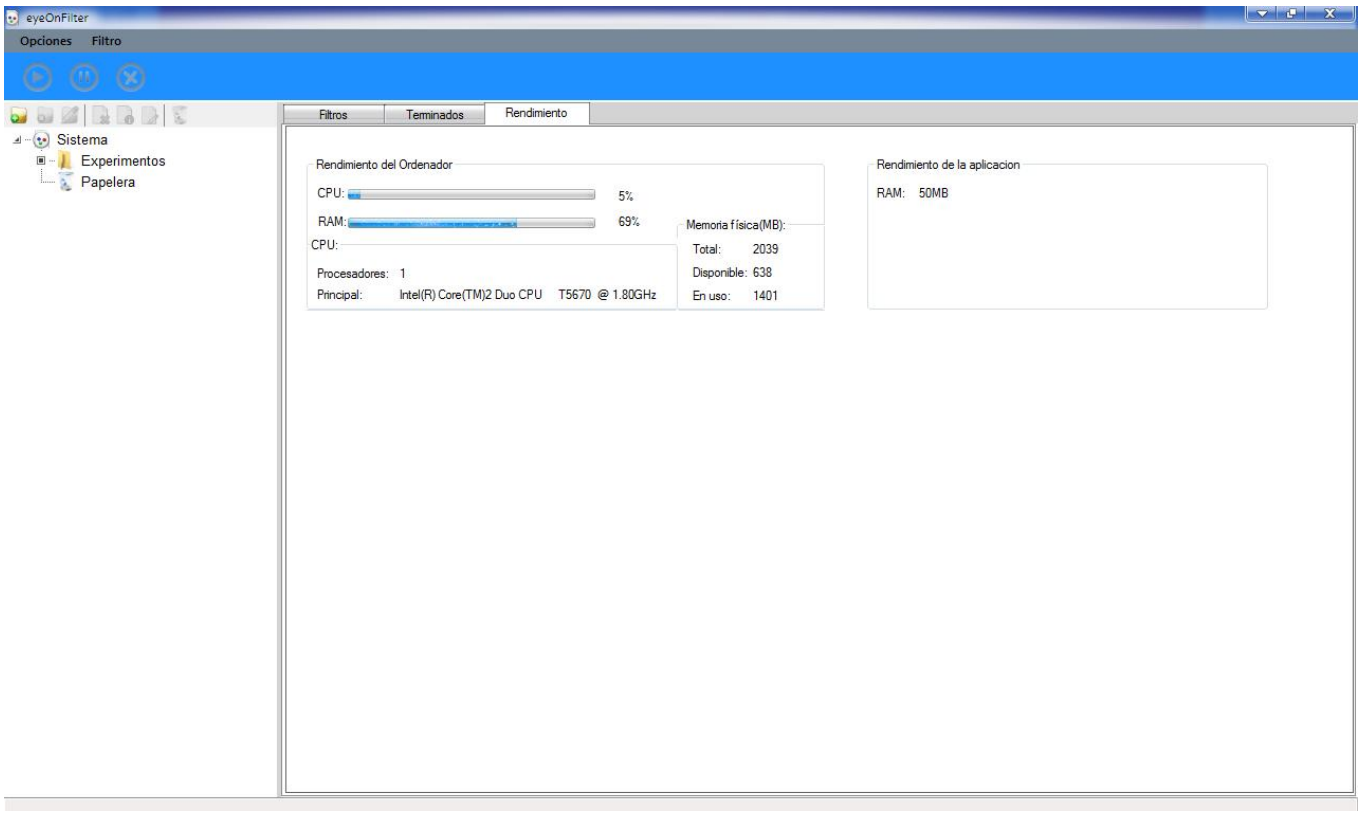
Anexo 3. Interfaz principal



Anexo 4. Interfaz Importar filtro



Anexo 5. Interfaz Monitorizar uso de recursos



Anexo 6. Interfaz principal con filtro en ejecución