Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño de una librería gráfica para la representación de configuraciones en sistemas P

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: María Samblás Martínez

Tutor: Jose María Sempere Luna

2013/2014

Agradecimientos

Quisiera agradecer a mi director de Trabajo Fin de Grado, Jose María Sempere Luna, su paciencia, su apoyo para la realización de este trabajo y su orientación para obtener un trabajo profesional.

A mis padres por todo el apoyo recibido en los buenos y malos momentos. Porque sin ellos, no sería nada de lo que soy hoy.

A mi pareja, por acompañarme estos últimos cinco duros años de mi vida.

Por último, a mi compañero Javier por toda su ayuda durante los estudios de grado y por su confianza depositada en mí.

Resumen

La Computación natural aproxima el esquema de célula eucariota a un modelo distribuido y paralelo capaz de procesar y generar información: Sistema P.

Los Sistemas P tienen dos características principales e importantes la concurrencia en las operaciones y el no-determinismo. Están formados por membranas, objetos y reglas de evolución. Las reglas pueden aplicarse tanto a objetos como a membranas en paralelo obteniendo una configuración instantánea.

La configuración instantánea obtenida tras aplicar las reglas de evolución, puede ser expresada en XML. La interpretación de un archivo en XML puede resultar dispendiosa y por ello transformaremos el XML en una representación gráfica, facilitando la interpretación de computaciones de Sistemas P.

PAnimator es una herramienta desarrollada en el lenguaje de programación orientado a objetos, Java, con el propósito de la animación de configuraciones de Sistemas P. Puede ensamblarse con P-Lingua, un lenguaje de programación de Computación con Membranas que se ha convertido en el estándar para definir Sistemas P.

Palabras clave: Computación natural, Sistemas P, Configuración instantánea, Animación de configuraciones.

Tabla de contenidos

Contenido

1. Introducción

1.1 Motivación

La mayor parte de los problemas resolubles algorítmicamente que se nos plantean se pueden resolver con un alto coste, ya sea de tiempo y/o espacio, es decir, son problemas con una resolución costosa.

Este hecho hace surgir la necesidad de buscar nuevos modelos que requieran de un menor coste de recursos y a su vez generen una solución.

Explicaremos que son los recursos y la forma de medirlos durante este trabajo.

Esta búsqueda ha dado como resultado el estudio y la introducción de nuevos modelos distintos a los convencionales (máquinas de Turing, funciones recursivas…) que proporcionan mejoras en las medidas de complejidad, espacio y tiempo, y pueden ser llevadas al terreno práctico, es decir, no son puramente teóricas.

Estos nuevos modelos basados en realidades físicas, especialmente biológicos, las búsquedas para obtenerlos y su estudio es lo que se denomina Computación Natural.

Existen diferentes tipos de Computación Natural, pero nosotros nos centraremos en Computación Celular con Membranas, apartado donde se lleva a cabo aproximaciones más detalladas del funcionamiento de las células. Este modelo fue introducido por Gh. Päun en 1998 llamado Sistemas P que da soporte a la computación con membranas. [1]

Päun define un Sistema P como un modelo de tipo distribuido y paralelo inspirado en el funcionamiento de una célula eucariota capaz de procesar y generar información.

En el apartado Sistemas P de esta memoria veremos con más profundidad y claridad este planteamiento, pero cabe decir para poner en situación al lector, que un Sistema P tiene una configuración inicial que tras una serie de operaciones, aplicaciones de reglas definidas en el mismo Sistema P, se convertirá en otra configuración.

Puesto al lector en contexto, explicaremos y detallaremos cual es la motivación de nuestro proyecto.

Toda la información donde se detalla tanto el estado inicial del Sistema P como su estado alcanzado se puede expresar en formato XML.

La estructura de un formato XML, que veremos más adelante, puede tener su intríngulis a la hora de interpretar las computaciones del Sistema P, por ello surge la necesidad de transformar el archivo XML a un formato gráfico para agilizar la interpretación de las computaciones.

1.1 Objetivos

PAnimator, como hemos llamado a nuestra librería gráfica para la representación de configuraciones en sistemas P, es una herramienta cuya finalidad es poder interpretar de manera ágil una computación de un Sistema P de transición y con Membranas Activas.

Se pretende:

Que PAnimator sea capaz de leer el archivo en formato XML y trate los datos.

Que una vez se ha leído el XML, se configure una representación gráfica acorde y lógica con los datos procesados.

Que las membranas que obtengan objetos se les asigne un color por defecto a cada objeto, rellenándose la membrana de un gradiente de color.

Que la cantidad de un color u otro en el gradiente venga marcada por la multiplicidad del objeto.

Que una vez obtenida la representación gráfica del Sistema P, el usuario pueda cambiar el color de los objetos si lo cree conveniente.

Por último, que se pueda guardar una imagen de la representación obtenida.

Con estos objetivos el usuario que desee conocer el resultado de una estructura inicial más unas reglas de evolución aplicadas a esta, no será necesario que conozca nada más que Computación de Sistemas P con membranas activas.

A través de este documento se pretende transmitir a la consciencia del lector que las nuevas ciencias avanzan a una velocidad inapreciable, siendo éstas capaces de resolver problemas hace poco impensados o computacionalmente imposibles. Asimismo que aprecie, teniendo los conocimientos previos claros, la naturaleza de esta herramienta como apoyo a la implementación práctica de la Computación de Sistemas P.

2. Sistemas P

2.1 Origen

En esta sección explicaremos los conceptos básicos de un Sistema P para familiarizar al lector con estas estructuras.

El área que estudia los Sistemas P se llama Computación Celular con Membranas.

Esta ciencia nace de la mano de Gheorghe Päun [1] como consecuencia de la necesidad del ser humano por encontrar soluciones a problemas resolubles algorítmicamente que con sistemas electrónicos como los de hoy en día somos incapaces de resolver a causa de limitaciones como la velocidad de cálculo y la cantidad de almacenamiento físico necesario.

La célula es la unidad fundamental de todo organismo vivo que posee una estructura muy compleja y organizada. Este tipo de estructura permite que en la célula se efectúen de manera simultánea una gran cantidad de reacciones químicas.

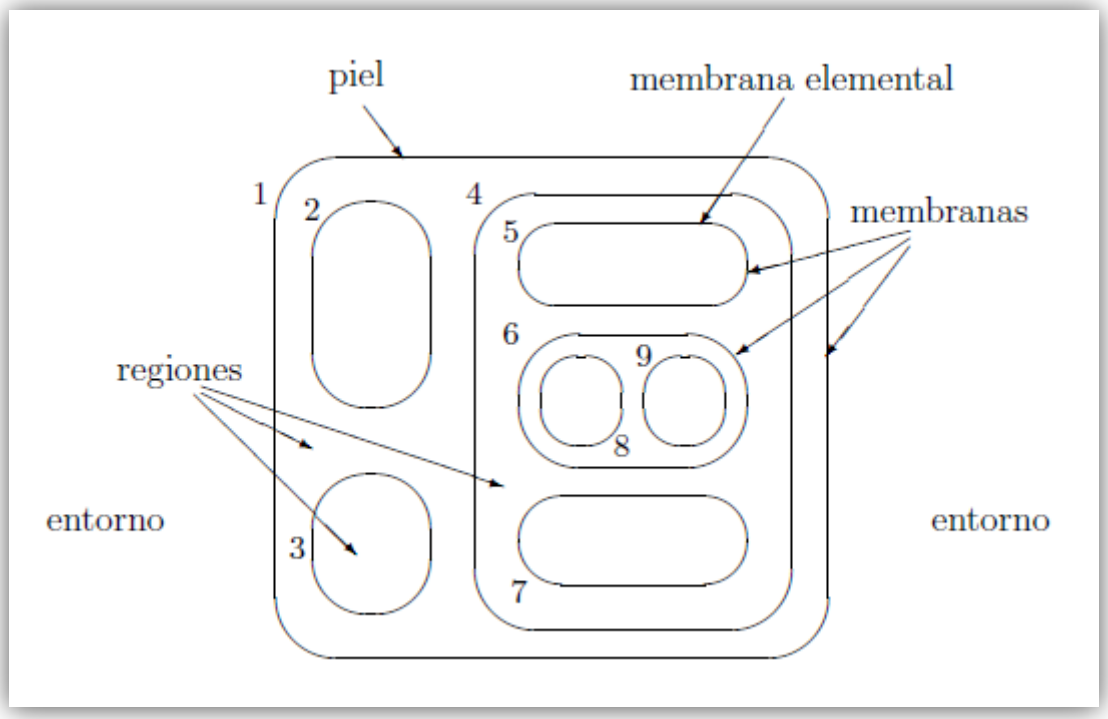

Figura 1. Estructura de membranas con zonas delimitadas (estructura interna de la célula)

Los Sistemas P consideran la célula como una máquina cuyos procesos químicos simultáneos se tratan como cálculos complejos. Es decir, los Sistemas P son los dispositivos computaciones de los modelos celulares que constituyen un marco teórico de computación inspirado en células vivas.

Los Sistemas P tienen dos características fundamentales: el paralelismo que permite realizar operaciones a la vez y su no-determinismo (característica que a día de hoy es imposible simular en un sistema electrónico.). Cabe decir que el paralelismo de un Sistema P se encuentra tanto a nivel de región, reglas aplicadas simultáneamente en una región, y a nivel de sistema donde todas las regiones evolucionan a la vez concurrentemente.

Formalmente hablando un Sistema P se representa gráficamente como muestra la figura siguiente:

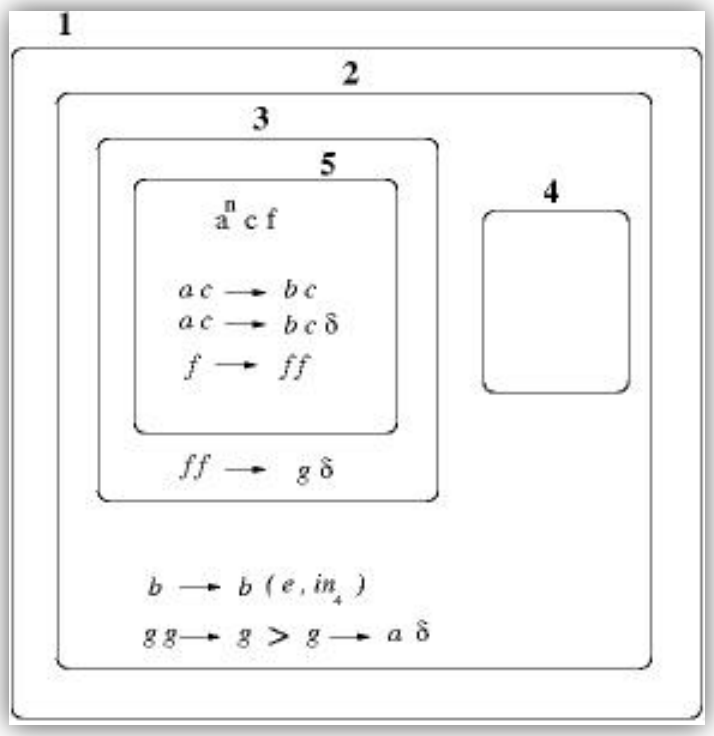

Figura 2. Estructura de un Sistema P

Los ingredientes que podemos observar y que forman un Sistema P son:

- Un alfabeto, conjunto de símbolos, cuyos elementos se denominan objetos: {a, b, c, e, f, g}.

- Una estructura de membranas, denominada también regiones, que tiene forma de árbol enraizado, etiquetadas con números: 1, 2, 3, 4 y 5.

- Unos multiconjuntos de objetos que están asociados a cada región: a^ncf de la membrana 5.

- Un conjunto de reglas de evolución asociado a cada membrana:
  Membrana 2 contiene las reglas $b \rightarrow b$ (e, in_4), r_1: $gg \rightarrow g$ y r_2: $g \rightarrow a\delta$
  (donde r_i marca la prioridad de reglas, en este caso r_1 se aplicará antes que r_2)
  Membrana 3 solo tiene una regla de evolución: $ff \rightarrow g\delta$.
  Por último la membrana 5 tiene tres reglas de evolución con igual orden de prioridad: $ac \rightarrow bc$, $ac \rightarrow bc\delta$ y $f \rightarrow ff$.

2.2 Sistemas P de transición

Ya explicado el origen y la importancia de los Sistemas P, nos adentraremos un poco más y analizaremos los Sistemas P básicos conocidos como Sistemas P de transición y los Sistemas P con membranas activas, base de este proyecto.

En este apartado nos centraremos en Sistemas P de transición, sus principales características y veremos un ejemplo para proyectar mejor la idea al lector.

Dependiendo de la evolución de los objetos, podemos clasificar los sistemas en tres tipos diferentes: [1]

- Sistema no-cooperativo donde los objetos evolucionan solos.
- Sistema cooperativo formado por reglas las cuales especifican la evolución de los objetos del sistema al mismo tiempo.
- Un caso intermedio en el que surgen ciertos objetos que no evolucionan solos, fruto de la aplicación de las reglas y que no son afectados por estas (catalizadores).

La principal característica de los Sistemas P de transición es la posibilidad de evolución de los objetos a través de ciertas reglas.

Un Sistema P de transición con grado n, siendo $n \geq 1$, tiene la siguiente sintaxis: [1]

$\Pi = (V, \mu, w_1, \ldots, w_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_0)$,

donde:

1. V es el conjunto de símbolos, alfabeto, que representarán a los objetos.

2. μ es la estructura de membranas de grado n. Las membranas y las regiones están etiquetadas con números. El número de membranas es n.

3. w_i, donde $1 \leq i \leq n$, son las cadenas pertenecientes a V^* que representan multiconjuntos sobre V asociados a las regiones.

4. R_i, donde $1 \leq i \leq n$, representa el conjunto finito de reglas de evolución sobre V asociadas a las regiones 1,2,...n de μ. ρ es la relación de prioridad entre las reglas de R_i si no existe prioridad, se representará con el símbolo $\varnothing$.
   Una regla de evolución está formada por el par (u, v), representado como $u \rightarrow v$ donde $u \in V^*$ y $v=v'$ o $v=v'\,\delta$ donde $v' \in V^*$. En general, una regla de evolución se escribe como:
   $(V \times \{here, out,\}) \cup (V \times \{in_j, \mid 1 \leq j \leq n\})$.
   δ es un símbolo especial que no pertenece a V. Representa la disolución de membranas

5. i_o, $1 \leq i_o \leq n$, representa la etiqueta de la membrana de la región de salida de Π.

Si una membrana esta vacía, no tiene objetos, puede representarse con el símbolo λ.

μ ,w_1, ..., w_n, representan la configuración inicial de Π.

A continuación veremos un ejemplo con la intención de reflejar la estructura explicada previamente:

Supongamos que tenemos un sistema de grado 4 especificado de la siguiente manera:[1]

$\Pi = (V, \mu, w_1, ..., w_4, (R_1, \rho_1), ..., (R_4, \rho_4), 4)$,

V={a.b.c.d},

$\mu = [_1\,[_2\,[_3\,]_3\,]_2\,[_4\,]_4\,]_1$,

w_1 = aac,

w_2 = a,

w_3 = cd,

$w_4 = \lambda$,

$R_1 = \{ r_1 : c \rightarrow (c, in_4), r_2 : c \rightarrow (b, in_4), r_3 : a \rightarrow (a, in_2)b, dd \rightarrow (a, in_4) \}$,

$\rho_1 = \{ r_1 > r_3, r_2 > r_3 \}$,

$R_2 = \{ a \rightarrow (a, in_3), ac \rightarrow \delta \}$,

$\rho_2 = \varnothing$,

$R_3 = \{ a \rightarrow \delta \}$,

$\rho_3 = \varnothing$,

$R_4 = \{ c \rightarrow (d, out), b \rightarrow b \}$,

$\rho_4 = \varnothing$.

En la figura siguiente veremos cómo evoluciona el sistema tras dos transiciones:

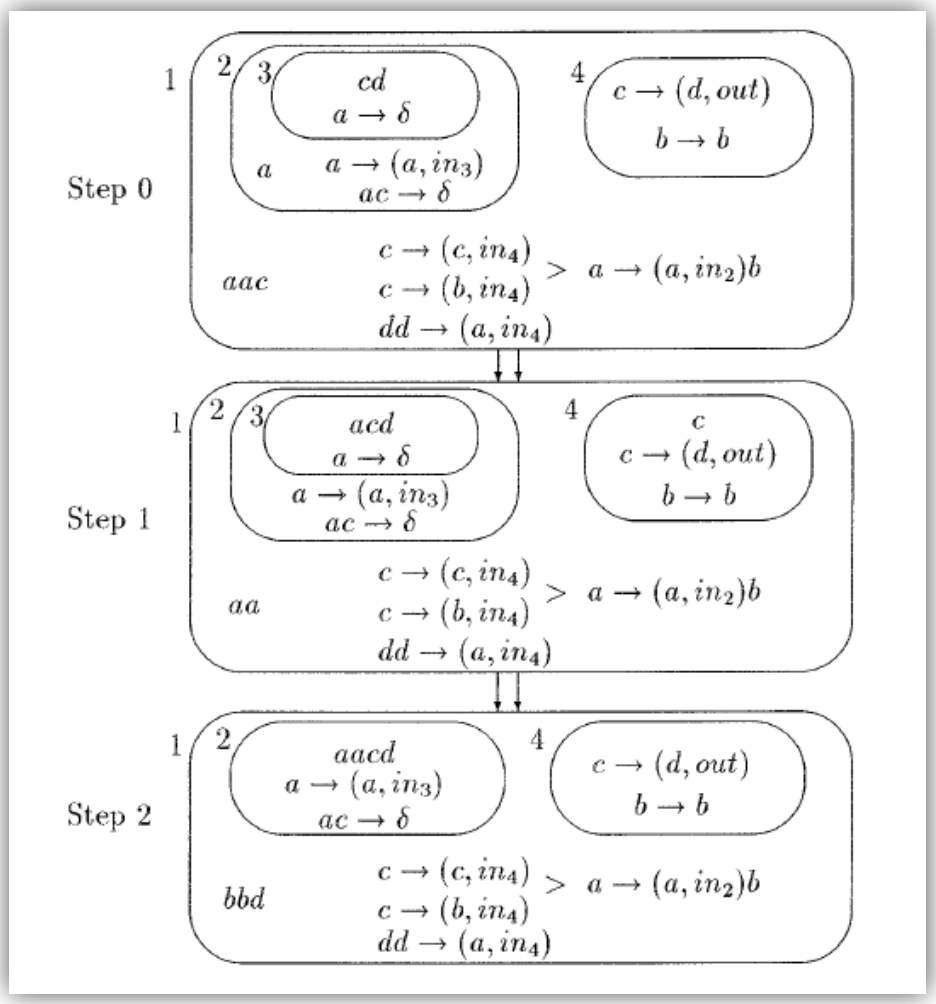

Figura 3. Ejemplo aplicación reglas de evolución [1]

En la representación etiquetada como “Step 0” se encuentra la configuración inicial del Sistema P de transición que hemos definido antes. La membrana de salida está etiquetada con el número 4, tal y como indica la variable i_0. También podemos observar la cadena de símbolos que representan los objetos dentro de las membranas y las reglas de evolución que se distinguen por la flecha de transición.

Analizaremos la membrana 1 para ver qué reglas de evolución podemos aplicar. Como vemos las reglas $c \to (c, in_4)$ y $c \to (b, in_4)$ tienen prioridad sobre $a \to (a, in_2)b$, así que escogeremos una de estas dos para aplicarla. Aplicaremos $c \to (c, in_4)$ (si decidiésemos aplicar $c \to (b, in_4)$, entraría un símbolo b en la membrana 4 que a su vez por la regla

$b \rightarrow b$ se transformaría en otro símbolo b y no pararía nunca). El símbolo c se enviará de la membrana 1 a la 4 y de la misma manera, con la regla $a \rightarrow (a, in_3)$ se enviara un símbolo a de la membrana 2 a la 3. De esta manera obtenemos la configuración etiquetada como "Step 1".

Para la configuración de "Step 2" se han aplicado las siguientes reglas de evolución: de nuevo $a \rightarrow (a, in_3)$ envía un simbolo a de la membrana 2 a la 3. Al mismo tiempo se ejecuta la regla $a \rightarrow \delta$ que disuelve la membrana 3 y la regla. En la membrana 1, como ya no tenemos símbolos c, se podrá usar la regla $a \rightarrow (a, in_2)$ que transformará la dos copias de a en dos de b y enviará a la membrana 2 una a, quedándose la cadena $aacb$ exactamente igual. Por ultimo aplicaremos la regla $c \rightarrow (d, out)$ que enviará el símbolo c de la membrana 4 a la membrana 1 pero en forma de símbolo d.

Se pretende aclarar que aunque se haya explicado la aplicación de las reglas de manera secuencial, para seguir un orden y estructura, estas transiciones se ejecutan de manera simultánea y paralela.

Entonces, en un instante de tiempo, se pueden crear o eliminar un número considerable de membranas. Si esto lo generalizamos, en un tiempo lineal polinómico, podemos usar o generar un espacio exponencial.

En la introducción ya presentamos el concepto de recursos sin llegar a definirlo, dejando esta tarea pendiente. Ahora que hemos introducido en el lector el conocimiento de la complejidad temporal, explicaremos que los principales recursos a medir son el espacio y el tiempo y que la cantidad necesaria para resolver un problema es lo que determina la complejidad del problema.

Por lo tanto, si hablamos en términos formales dentro de la Teoría de la Computación y complejidad, los Sistemas P son modelos de computación no deterministas y paralelos, capaces de resolver eficientemente problemas con una complejidad temporal no determinista polinómica, $f(n) \in O(n^p)$.[4]

Los problemas con estas características de complejidad son agrupados en la clase NP formada por problemas de decisión computables con complejidad temporal no determinista polinómica. Dentro de esta clase, podemos encontrar otras como la Clase P, caracterizados por modelo de computación determinista en tiempo polinómico, y la Clase NP-Completo (NP-duro) que contiene los problemas más difíciles cuya resolución necesita de un número polinómico de pasos de computación en un modelo no determinista.

La siguiente figura muestra las Clases de complejidad que hemos mencionado.

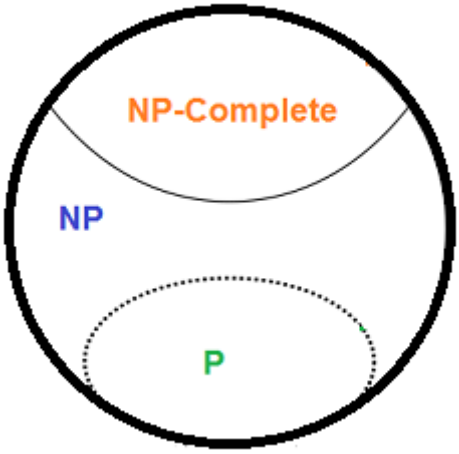

Figura 4: Clases de complejidad P, NP, NP-Completo

2.3 Sistemas P con membranas activas

En este apartado vamos a hablar de la base de nuestro proyecto e idea fundamental: Sistemas P con membranas activas.

Estos sistemas son una variante que permite la construcción de espacio exponencial en tiempo polinómico. Este hecho permite resolver, tal y como hemos hecho mención previamente, problemas difíciles proporcionando soluciones eficientes. En el marco de la computación, los problemas con estas características están agrupados y se llaman NP-completos y NP-duros.

Los sistemas P con membranas activas tienen más reglas de evolución que los anteriores. A continuación las detallaremos todas:

Dentro del conjunto de reglas de evolución podemos encontrar seis tipos:[2]

(a)-Conjunto de reglas de reescritura.

(b)-reglas de creación de objetos en las membranas

(c)-reglas para enviar objetos fuera de las membranas

(d)-reglas para disolver membranas

(e)-reglas para dividir membranas elementales

Además de estas reglas, dentro de los Sistemas P con membranas activas encontramos tres tipos más:

(g)-reglas de intercalado de membranas

(h)-reglas de separación de membranas

(i)-reglas de liberación de membranas

Definiremos formalmente un Sistema P con membranas activas como:

$\Pi = (V, H, \mu, w_1, \ldots, w_n, R)$, donde

1. V es el alfabeto de los objetos.
2. H es un conjunto finito de etiquetas para las membranas.
3. μ es la estructura de la membrana formada por n membranas. Las membranas y las regiones están etiquetadas con elementos de H.
4. w_i, donde $1 \leq i \leq n$, son las cadenas pertenecientes a V* que representan multiconjuntos sobre V asociados a las regiones.
5. R_i, donde $1 \leq i \leq n$, representa el conjunto finito de reglas de evolución como sigue:

(a) $[a \rightarrow v]_h$, para $h \in H, a \in V, v \in V^*$	reglas de evolución de objetos.
(b) $a[\,]_h \rightarrow [b]_h$ para $h \in H, a,b \in V$	reglas de comunicación.
(c) $[a\,]_h \rightarrow [\,]_h b$ para $h \in H, a,b \in V$	reglas de comunicación.
(d) $[\,a]_h \rightarrow b$ para $h \in H, a,b \in V$	reglas de disolución.
(e) $[a\,]_h \rightarrow [b]_h[c\,]_h$ para $h \in H, a,b,c \in V$	reglas de división de membranas elementales.
(g) $[\,]_h[\,]_h \rightarrow [\,]_h$ para $h \in H$	reglas de fusión de membranas elementales.
(h) $[V\,]_h \rightarrow [U]_h[V - U\,]_h$ para $h \in H, U \subset V$	reglas de separación de membranas elementales.
(i) $[\,[V\,]_h]_h \rightarrow [\,]_h V$ para $h \in H$	reglas de liberación de membranas

Algunas de las combinaciones de tipos de reglas en los Sistemas P con membranas activas pueden resolver problemas difíciles caracterizados por una complejidad espacial exponencial y una complejidad temporal lineal mediante el uso de separación de membranas.

Los problemas difíciles con esa complejidad tanto espacial como temporal se agrupan en una familia de problemas llamados NP-Completos, como hemos explicado previamente.

SAT fue el primer problema demostrado como NP-Completos conocido como Problema de Satisfacibilidad Booleana. Se trata de un problema donde se busca saber si una expresión booleana con variables y sin cuantificadores, tiene asociada una asignación de valores para las variables que hagan que la expresión sea verdadera.

Por ejemplo la expresión booleana $(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$ es cierta [10].

A continuación plantearemos la esquematización y definición formal de los datos para resolver un problema NP-Completo, SAT, con Sistemas P con membranas activas.

Para resolver SAT solo precisamos de las reglas de tipo (a), (b), (c) y o bien (e') [3] o el tipo (h').

Supongamos que tenemos las siguientes preposiciones:

$$\beta = C_1 \wedge \ldots \wedge C_m,$$

$$C_i = y_{i,1} \vee \ldots \vee y_{i,l_i},\ 1 \leq i \leq m,\ \text{donde}\ y_{i,k} \in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m,$$

$$1 \leq k \leq l_i$$

Pretendemos resolver una instancia del problema SAT β con Sistema P de membranas activas, con lo que codificaremos esta instancia con las reglas de evolución de la siguiente manera:

$$v_j = \{c_i \mid x_i \in \{y_{i,k} \mid 1 \leq k \leq l_i, \}, 1 \leq i \leq m\},\quad 1 \leq j \leq n$$

$$v'_j = \{c_i \mid \neg\, x_i \in \{y_{i,k} \mid 1 \leq k \leq l_i, \}, 1 \leq i \leq m\},\quad 1 \leq j \leq n$$

Así representaremos nuestras variables booleanas del SAT en forma de símbolos.

La configuración inicial del Sistema P quedará así:

$$\Pi = (V, H, \mu, w_o, w_1, w_S, R)$$

$$V = \{d_i, d'_i \mid 0 \leq i \leq n + m + 5\} \cup \{t_{i,j}, t'_{i,j}, f_{i,j}, f'_{i,j} \mid 1 \leq i \leq j \leq n\}$$

$$\{c_i \mid 1 \leq i \leq m\} \cup \{t, yes, no\},$$

$w_s = \lambda$,

$w_0 = d_0$,

$w_1 = d_0$,

$H = \{s, 0, 1,..., m+2\}$.

La estrategia a seguir para la resolución de SAT [2] es la siguiente: se generan las reglas necesarias de evolución (a), comunicación (b), (c) y de separación de membranas elementales (h).

La aplicación de las reglas provocará diferentes configuraciones del Sistema P.

Tras $n+m+5$ pasos, nos fijaremos en la membrana más externa, denominada como piel.

Si la preposición β tiene solución, es decir, si la expresión booleana que se quiere proyectar en β, tiene asociada un conjunto de valores que la hacen cierta, entonces en la membrana piel tendremos un objeto "*yes*". En caso contrario, el objeto de la membrana piel será "*no*".

De esta manera se resuelve SAT en un tiempo lineal con un modelo de computación no-determinista: Sistemas P con membranas activas.

3. PAnimator

Tras haber explicado la importancia de los Sistemas P con Membranas Activas, y la necesidad de agilizar la interpretación de las configuraciones a través de la animación de las configuraciones de Sistemas P, en este capítulo procederemos a explicar el diseño que hemos pensado para llevar a la herramienta que hace posible esto, PAnimator, así como su implementación.

1.1 Diseño (UML y Clases)

En este apartado analizaremos los componentes que se organizan para formar el software que estamos desarrollando como también las interfaces de usuario.

Como hemos aprendido durante nuestro proceso de formación y a lo que por ello nos conduce el instinto, hemos realizado una programación modular, hemos estructurado el código en diferentes clases.

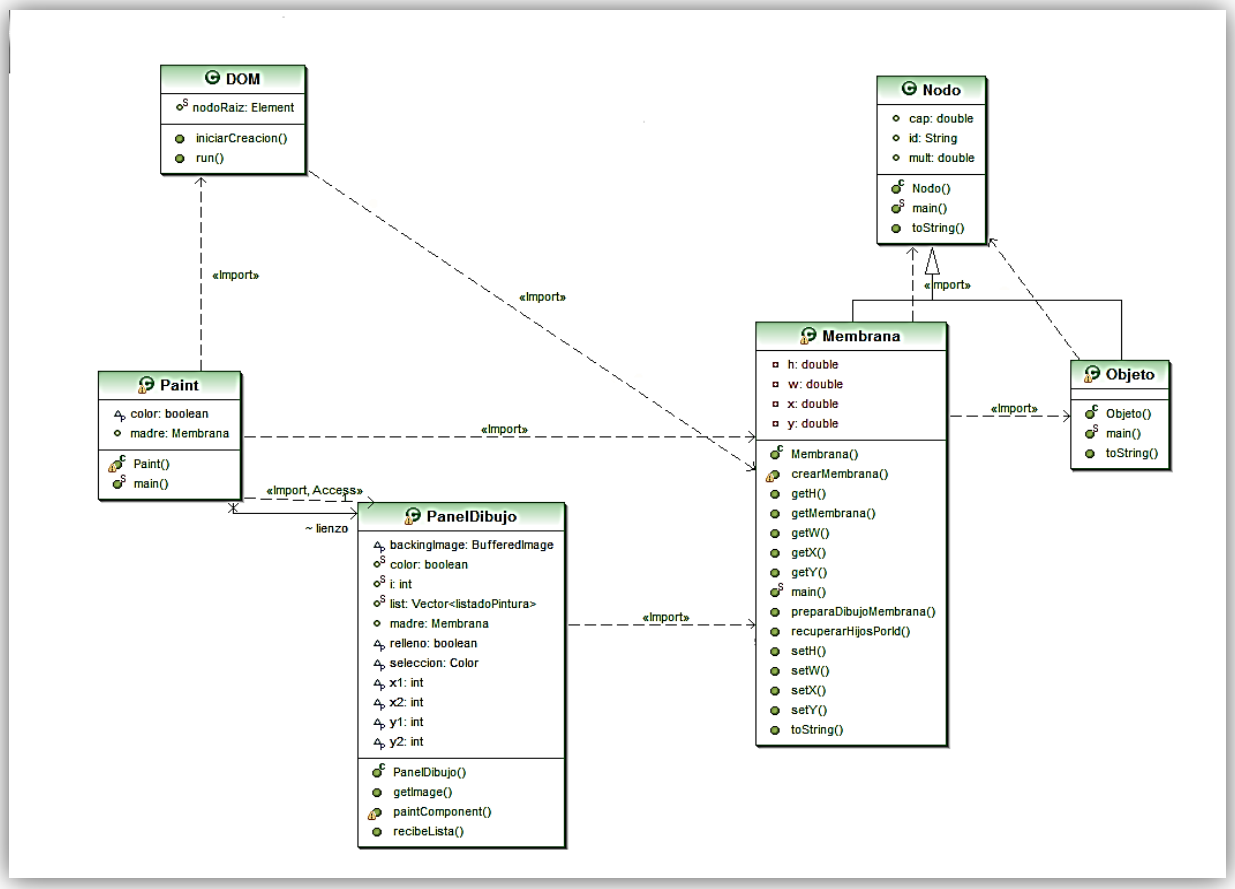

Figura 5. Esquema UML de PAnimator

En concreto, se ha decidido estructurar el proyecto en 4 clases bien diferenciadas:

. DOM

En esta clase se realizará la lectura del archivo en formato XML y tratará los datos extrayendo los nodos, etiquetas en XML. Trabajaremos en forma de árbol con los datos por su facilidad de acceso a ellos y manipulación.

Esta clase constará de dos métodos:

1. Run(): este método se encargará de obtener el archivo XML que indiquemos. Extraerá el nodo raíz, que será la primera etiqueta del XML y recuperará a sus hijos. Los hijos etiquetados como membranas, serán enviados al método que explicaremos a continuación.

2. IniciarCreacion(): aquí será donde crearemos un nuevo objeto membrana, e invocaremos al método crearMembrana que explicaremos más adelante. Si fuese necesario añadir parámetros al nuevo objeto membrana se podrá hacer en este método y luego crearemos la membrana, invocando el método correspondiente.

. NODO

Para ser más metódicos, crearemos una clase nodo. Esta clase tendrá objetos de tipo nodo que luego se especializarán en Membranas u Objetos dependiendo de las etiquetas de los datos que estemos procesando del XML.

En esta clase tendremos los atributos que son comunes tanto en membranas como en los objetos como por ejemplo multiplicidad y capacidad. Estos atributos serán accesibles desde las clases hijas de Nodo: Membrana y Objeto.

. MEMBRANA

La clase Membrana heredará de la clase Nodo.

Tras analizar el archivo XML, hemos concluido que sólo las membranas pueden tener nodos hijos, los objetos son hojas del árbol, no tienen hijos, con lo cual será en esta clase donde tratemos los nodos hijo de estas. Para ratificar esta información que aseguramos, veremos un ejemplo en el apartado 4 Estructura y procesado del XML.

Utilizaremos la estructura de datos tabla hash para almacenar los elementos leídos y procesados. Esta tabla tendrá como clave el nombre de la membrana y como valor el nombre del hijo. ¿Por qué una tabla hash? Hemos estudiado que las tablas hash son la estructura de datos cuyos métodos como buscar, añadir y modificar son de rápida

ejecución. Para poder dibujar las membranas y los objetos, intuimos que tendremos que realizar varias consultas, con lo cual se decide que utilizaremos esta estructura de datos.

Tendremos un constructor de membranas con sus atributos. Para poder ser dibujadas necesitaremos añadirle atributos como la posición en el eje x, la posición en el eje y, la altura y la anchura. Además tendrá su identificador, que será el nombre, para poder ser almacenada en la tabla hash.

Como necesitaremos acceder a estos atributos, crearemos los métodos set y get para poder consultarlos sin problemas.

También tendremos dos métodos importantes:

- crearMembrana(): ya hemos visto dónde se invocará, pero ahora contaremos cuál será su función. Será el método que procesará los datos llegados del XML.
  Recibirá un objeto nodo del que extraerá sus atributos: nombre, multiplicidad, capacidad, etc.
  Después se obtendrá sus elementos hijos donde se analizará si la etiqueta que los identifica es de tipo objeto o de tipo membrana.
  Si es del primer tipo, se invocará al constructor de objetos para su creación que analizaremos en la siguiente clase. Por el contrario, si es membrana, se volverá a invocar de manera recursiva a este mismo método.

- recuperarHijoPorId(): este método servirá para obtener la información necesaria, los elementos objeto y los elementos membrana que son descendentes de una membrana dada. Este método nos facilitará la representación gráfica de modo que podremos configurar los puntos y las dimensiones de los descendientes de una membrana sabiendo la cantidad de hijos que tiene y sus atributos.

. OBJETO

Como hemos mencionado antes, esta clase contendrá el constructor de los elementos de tipo objeto con sus atributos como la dimensión que tendrá el objeto a dibujar predefinida por ser una cadena de caracteres y su nombre identificativo.

Estas tres clases nos permitirán diferenciar los elementos de tipo membrana y objeto, y gracias a la Clase Nodo, podremos añadir atributos que sean comunes a estos elementos sin necesidad de modificar el resto de clases.

. PAINT

La clase Paint será la creadora de la interfaz de la herramienta. Esta clase deberá estar bien construida y será una de las más importantes puesto que la observará el usuario e interaccionará con ella.

Se utilizarán objetos de tipo JFrame, JButton y JToolBar pertenecientes de la API de Java que se explicarán en el apartado Implementación.

En el constructor de Paint será donde crearemos una membrana, invocaremos al método run() de la clase Dom. A su vez, también se creará un objeto de tipo

PanelPintura, veremos a continuación para qué, y conectaremos la membrana al Panel para poder llevar a cabo la representación gráfica.

. PANELDIBUJO

PanelDibujo generará una "hoja" en blanco que será donde se proyectará la representación gráfica.

En esta clase hay varios métodos importantes a destacar:

- formMouseCliked(): método para capturar los eventos ratón. Ya definido por la API. Lo usaremos para que cuando haga click el usuario en un objeto pueda pintarlo de nuevo con los colores que elija.
- paintComponent(): método definido por la API de Java. Será el método encargado de obtener los parámetros, calcularlos y construir la figura que representará cada membrana.

3.1 Implementación

3.1.1 Java

Una vez se ha tenido claro el diseño a seguir para que todo funcionase correctamente y estuviese estructurado, se ha procedido a implementar el código que realice todas las acciones necesarias para poder obtener el resultado esperado.

Se ha utilizado el lenguaje de programación Java. El motivo principal de esta decisión ha sido porque PAnimator puede ser sumado y ensamblado como herramienta al proyecto P-Lingua.[5]

P-Lingua es un lenguaje de especificación de sistemas P que se ha convertido en el lenguaje estándar para su programación.

P-Lingua fue desarrollado por el grupo de Computación Natural de la Universidad de Sevilla e implementado en el lenguaje de programación Java.

Asimismo Java es un lenguaje de propósito generar basado en clases y orientado a objetos diseñado específicamente para tener las mínimas dependencias de implementación posibles. Es decir, el objetivo de Java es permitir al desarrollador escribir el programa una vez pudiéndolo ejecutar en cualquier dispositivo o plataforma.

Se ha trabajado con la plataforma Eclipse Kepler debido a la familiaridad que ya se tenía por haber sido usada con anterioridad durante los cursos académicos.

3.1.2 Org.JDom2

Cuando observas un sistema P en imagen es sencillo observar que hay membranas que están dentro de otras. Esto se puede estructurar fácilmente en forma de árbol siendo el padre la membrana exterior y sus hijos los elementos que contiene dentro. De esta manera se obtiene una estructura ordenada y fácil de tratar.

Para la lectura y tratado del XML con operaciones de java se ha optado por la librería *org.jdom2*.

¿Por qué JDom y no SAX?

La librería JDom [10] de Java realiza una lectura del XML a la vez que lo transforma en un árbol. La estructura de árbol facilita la lectura y la obtención de los datos de las etiquetas del XML siendo rápido y preciso.

Por el contrario, SAX, que también sirve para el tratado de documentos XML, no dispone de la estructura en árbol y por ello es más difícil de manipular. Además realiza

una lectura secuencial del documento por lo que no permite la vuelta atrás, cosa que JDom sí.

Como hemos visto en el apartado de Diseño, gracias a esta estructura de árbol y a través de unos simples métodos podemos saber quién es el padre de una membrana, quienes son los hijos, sus atributos etc.

3.1.3 AWT.Graphics

AWT.Graphics [9]es una clase que se encuentra en la API Java 2D que amplia muchas de las capacidades graficas de la biblioteca AWT y que permite la creación de mejores interfaces de usuario y aplicaciones Java. Esta API es la que hemos utilizado para hacer posible la representación gráfica de nuestro sistema P.

En concreto, hemos hecho uso de *Graphics2D* que extiende de Graphics y que es mucho más completa.

Esta es la parte más interesante de la implementación del proyecto ya que ha sido completamente autodidacta y nueva para la autora de este. Además una pequeña dificultad añadida ha sido que desde que nos explicaron el sistema de coordenadas hasta llegar a Java2D, este se situaba en la esquina inferior izquierda, si no en el centro. Bien pues en el caso de Java, el centro de coordenadas del dispositivo se encuentra en la esquina izquierda superior, cosa que ha llevado a confusiones en alguna que otra ocasión.

A continuación destacaremos dos clases interesantes que hemos utilizado en nuestra implementación:

- Gradiente de Color (AWT.Graphics2D): Es un efecto producido en el interior de la figura cambiando el color progresivamente desde un color inicial hasta otro final. Para rellenar una figura con un gradiente de color hay que seguir los pasos:

    1. Se crea un objeto GradientPaint con los colores que vamos a degradas.

    2. Se llama al método setPaint() de la clase Graphics2D para asociar el gradiente.

    3. Se crea la figura que puede ser con un método de esta clase o de Graphics. Son compatibles.

    4. Se llama al método fill de la clase Graphics2D. Fill hace referencia a que el objeto se va a rellenar.

- Dialogo de Color (javax.swing.JColorChooser): Es un dialogo que muestra todos los colores y su gama para que puedas elegir el que quieras. Una vez está la interfaz generada, hay que invocarlo con una serie de parámetros.
  Esto es un gran avance dentro de la clase de interfaces.

En general, el desarrollo de la herramienta ha sido costoso. A pesar de haber estado familiarizados con el lenguaje de programación Java, no se conocía la API JAVA2D para las representaciones graficas ni tampoco se había llegado tan lejos con las interfaces con anterioridad. Además cabe añadir que no se había manipulado ningún documento XML con Java, aunque si con Python.

4. Estructura y procesado de XML

En este apartado vamos a explicar la estructura del Sistema P con todos sus ingredientes sintácticos modelados en el archivo XML.

Trataremos de proyectar al lector cómo sería el procesado que debería de hacer cualquier usuario que quisiera conocer los resultados obtenidos después de la aplicación de las reglas de evolución sobre la estructura de Membranas y objetos del Sistema P.

Se espera que el lector llegue a ser capaz de captar la esencia de esta herramienta.

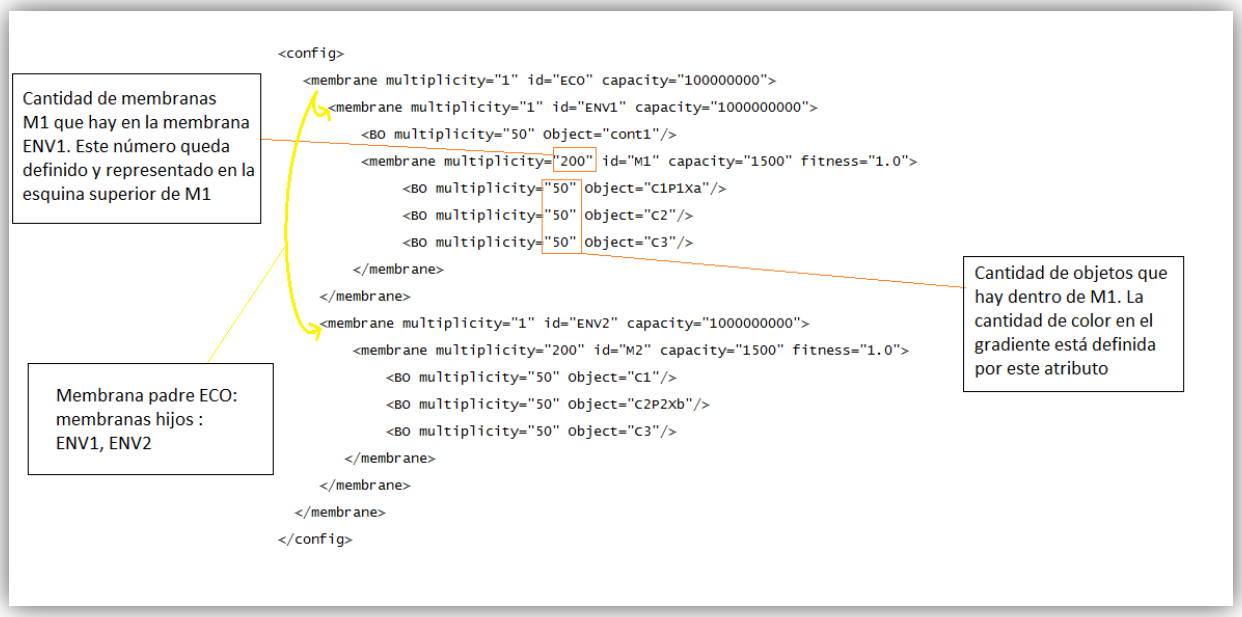

Figura 6. XML

El formato XML se caracteriza por la introducción de datos entre etiquetas. En nuestro ejemplo tenemos un XML cuyo cuerpo está formado por elementos con datos: “membrane” y “BO, Basic Object”. Estos elementos contienen a su vez atributos. En el caso de las membranas tenemos como atributos la “multiplicity” (cuantas membranas de este tipo tenemos), “id” (identificador o nombre de la membrana) y “capacity” (número de objetos que caben en la membrana).

En el caso de los Objetos Básicos (BO) tenemos los atributos “multiplicity” (cantidad de objetos de este tipo que tenemos) y “Object” (nombre del objeto basico).

A continuación veremos la representación gráfica que se obtendría de este archivo en XML.

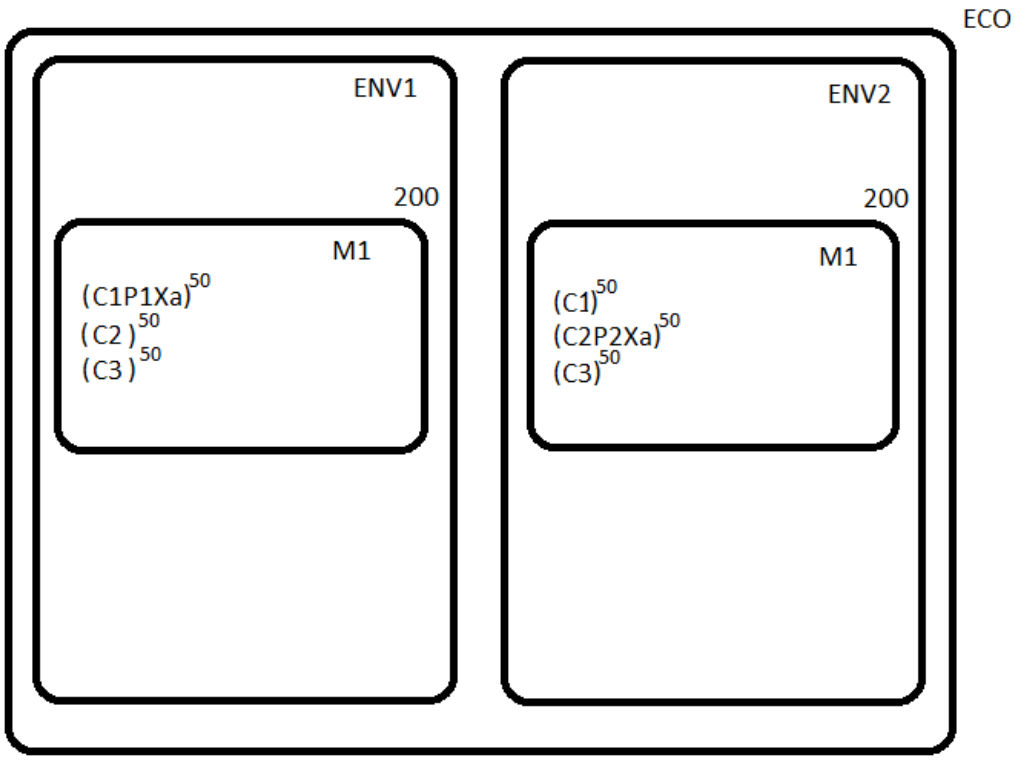

Figura 7. Representación gráfica del XML

Como podemos apreciar en la imagen, todos los atributos de los elementos van detallados con sus etiquetas correspondientes. En este caso se puede observar quien es hijo y quien hermano a través de la tabulación, pero el sistema lo detecta por el cierre de las etiquetas membrana y Objetos Básicos (BO).

5. Resultados

Vamos a considerar que el archivo XML a traducir en imagen es el siguiente:

```
<config>
  <membrane multiplicity="1" id="ECO" capacity="100000000">
    <membrane multiplicity="1" id="ENV1" capacity="1000000000">
      <BO multiplicity="50" Object="cont1"/>
      <membrane multiplicity="200" id="M1" capacity="1500" fitness="1.0">
        <BO multiplicity="50" Object="C1P1Xa"/>
        <BO multiplicity="50" Object="C2"/>
        <BO multiplicity="50" Object="C3"/>
      </membrane>
    </membrane>
    <membrane multiplicity="1" id="ENV2" capacity="1000000000">
      <membrane multiplicity="200" id="M2" capacity="1500" fitness="1.0">
        <BO multiplicity="50" Object="C1"/>
        <BO multiplicity="50" Object="C2P2Xb"/>
        <BO multiplicity="50" Object="C3"/>
      </membrane>
    </membrane>
  </membrane>
</config>
```

Figura 8. XML a procesar.

La representación de este Sistema P, ya explicada en el apartado Estructura y procesado de XML sería:

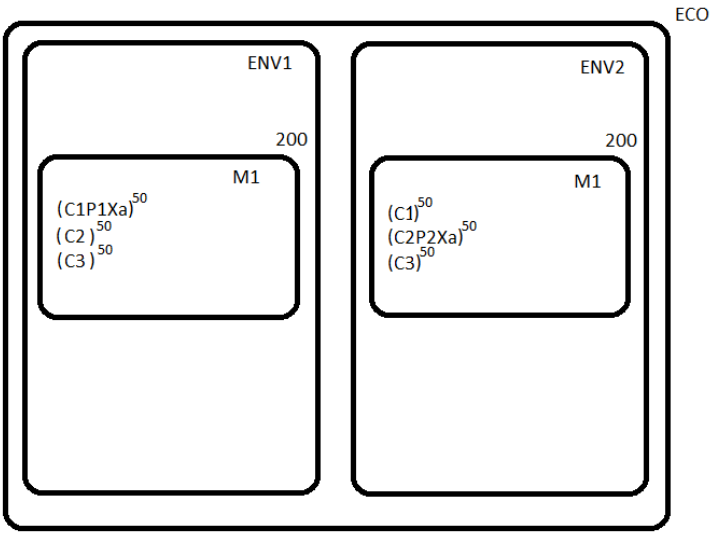

Figura 9. Representación gráfica de la figura 6.

Cuando la herramienta se inicia, realiza una lectura y tratado de los datos y etiquetas del XML generando los elementos Membrana y Objetos correspondientes. Una vez creada la estructura se procede a su representación gráfica obteniendo como resultado la siguiente imagen:

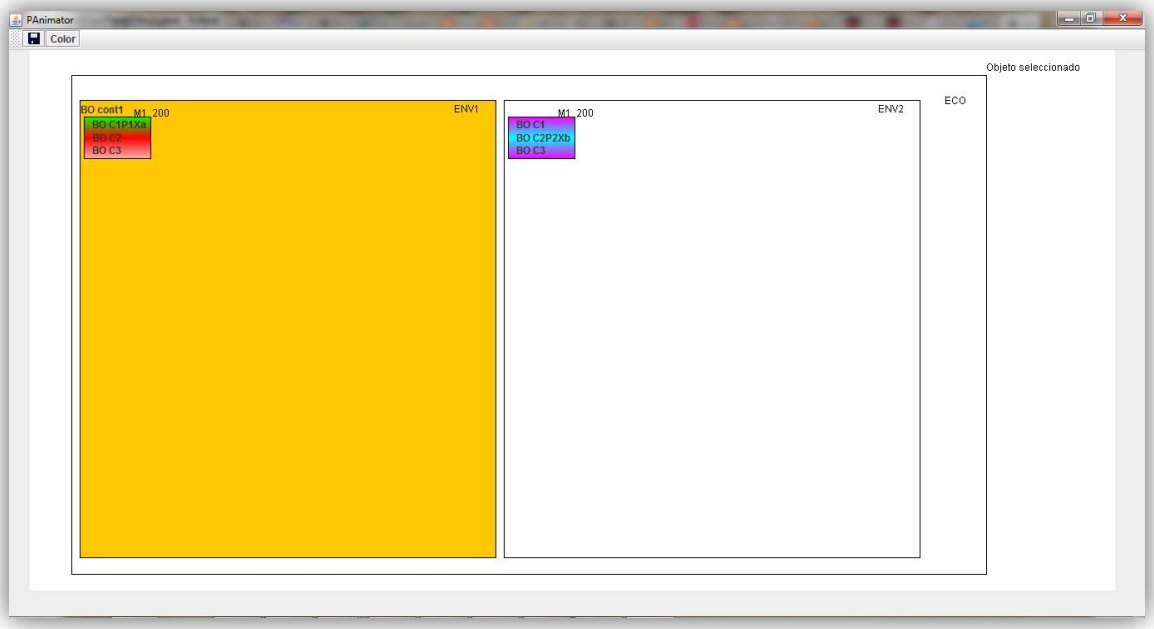

Figura 10

Supongamos que el usuario decide modificar los colores de las regiones ENV1 y M1 (dentro de ENV1). El siguiente paso, es seleccionar el objeto BO cont1 y seleccionar su color tal y como indican las imágenes:

Nota: En la esquina derecha superior se puede observar cómo indica el sistema el objeto que el usuario ha seleccionado.

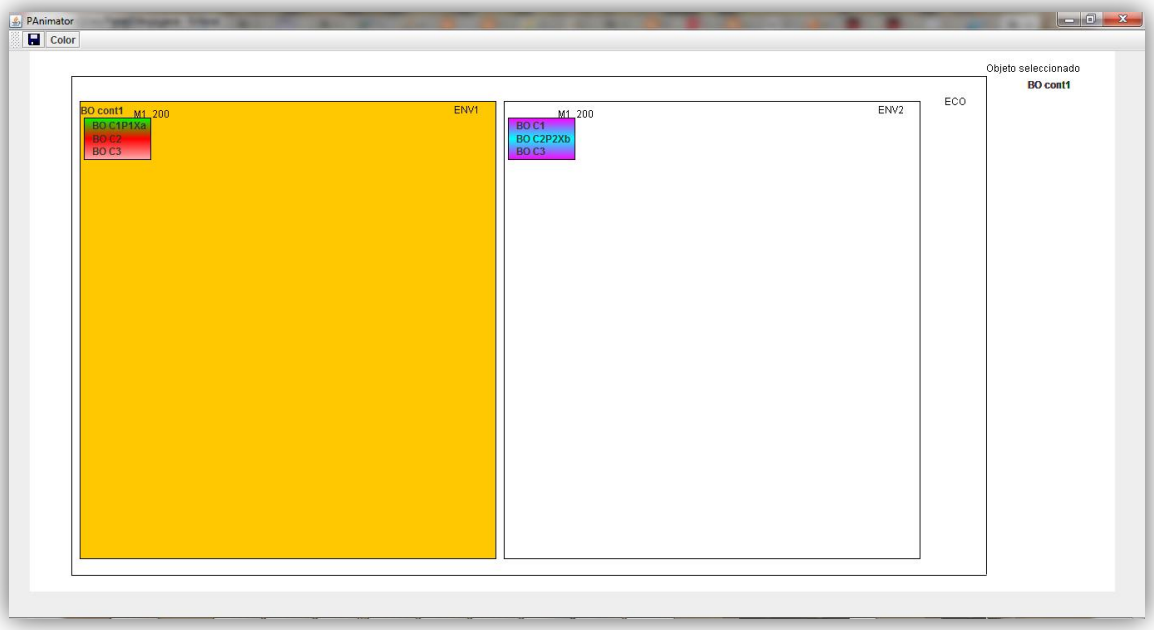

Figura 11

Ahora que el usuario ha seleccionado el objeto, tendría que pinchar en la pestaña de la barra superior: Color.

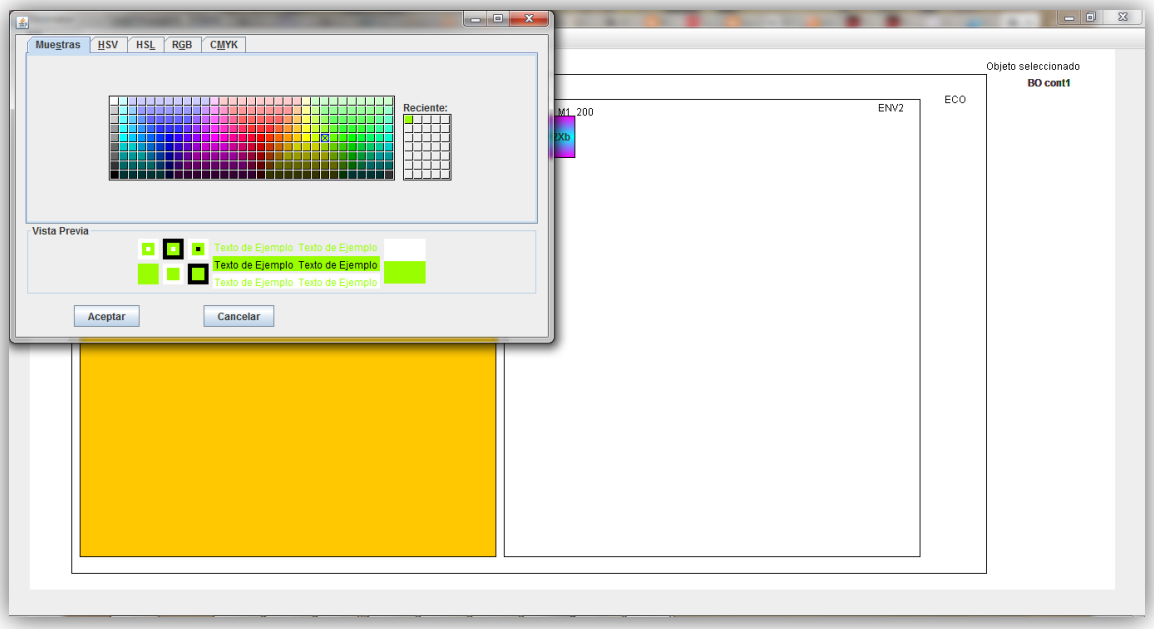

Figura 12

Podemos observar que el color escogido es el verde. En el momento en que se acepta esta acción se obtiene el resultado siguiente:

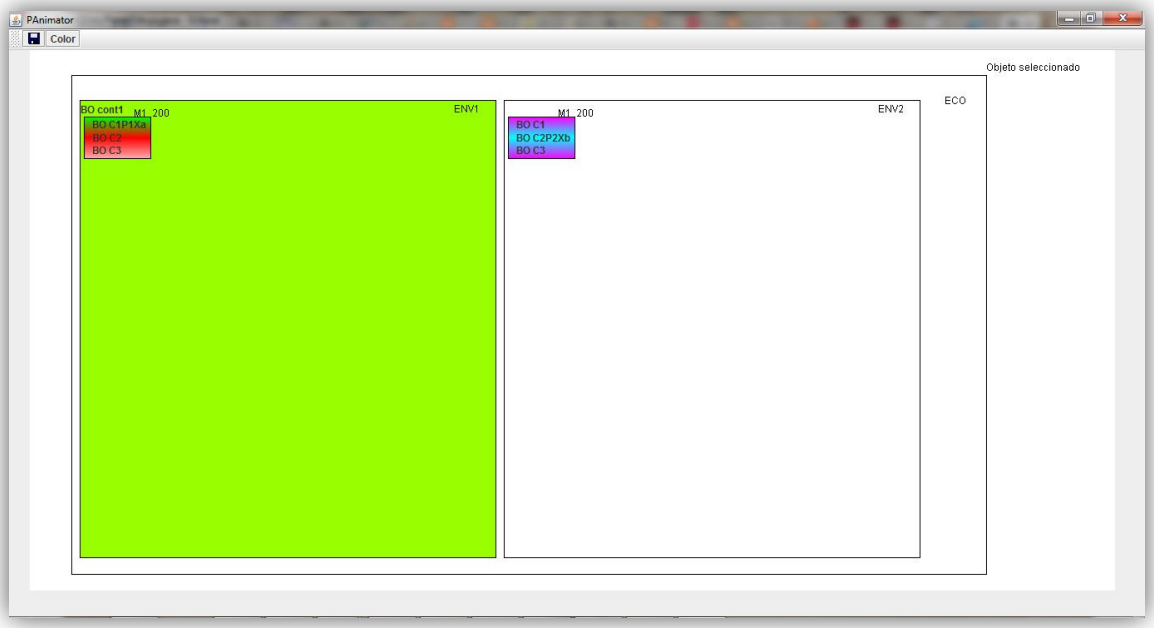

Figura 13

El usuario procederá a seleccionar los objetos contenidos en la membrana M1 situada dentro de ENV1 y su color:

Objeto C1P1Xa:

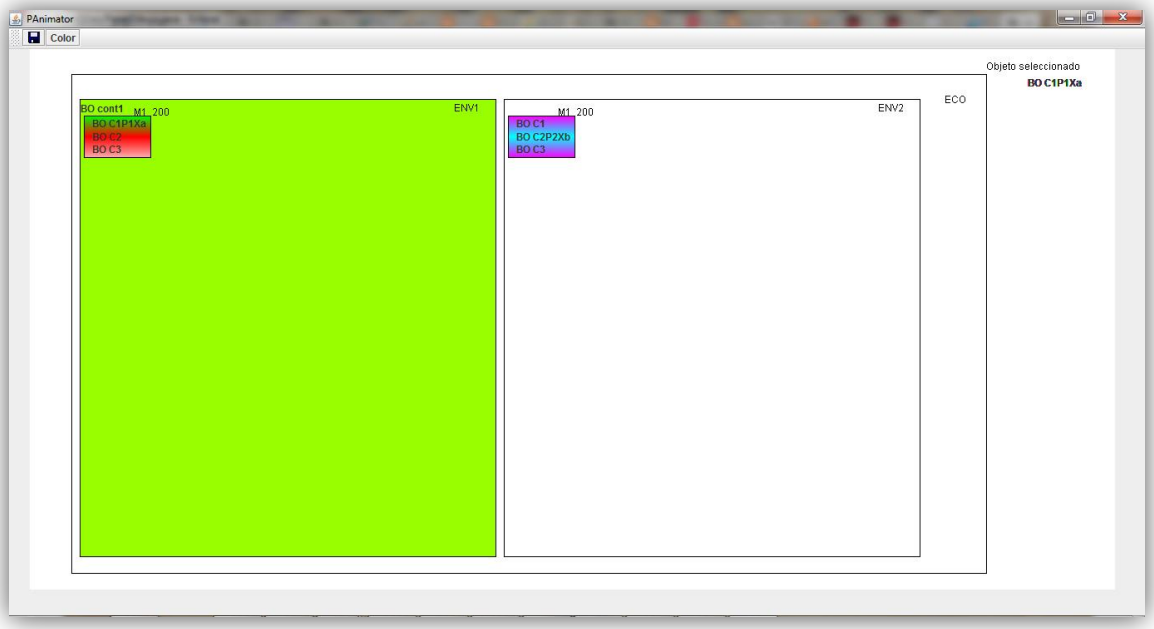

Figura 14

Color de C1P1Xa:

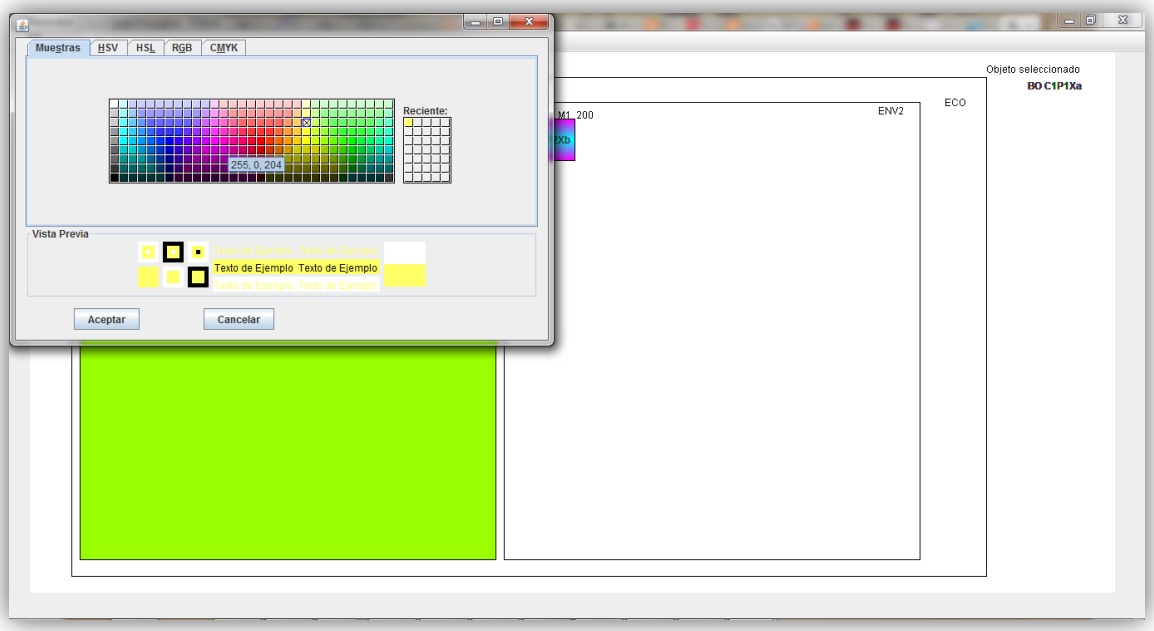

Figura 15

Objeto C2:

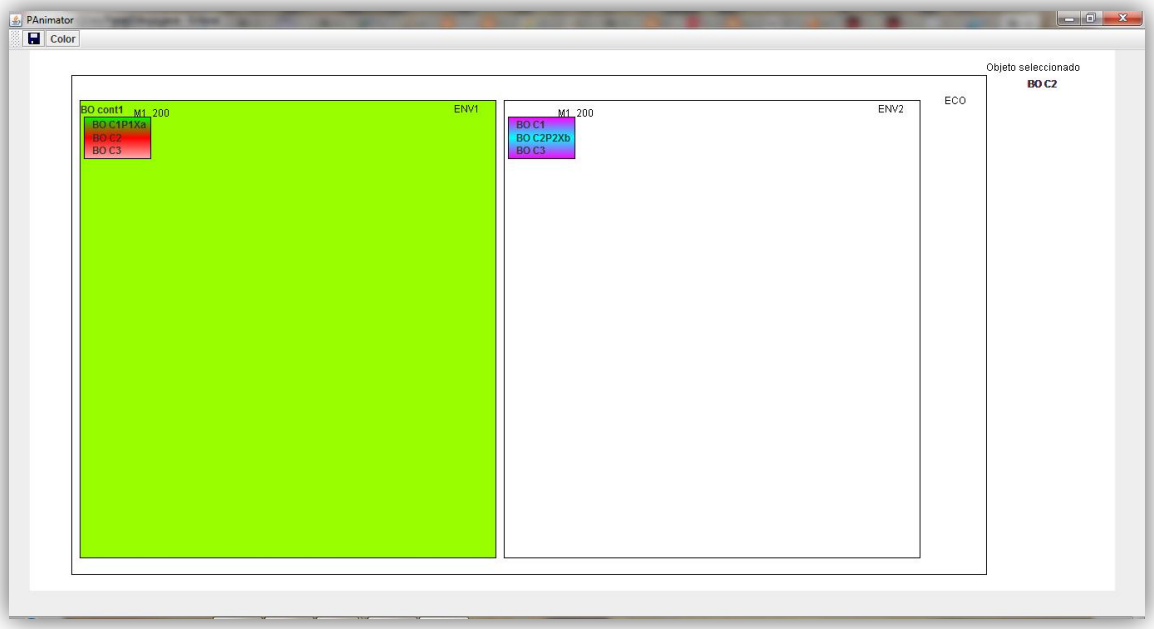

Figura 16

Color de C2:

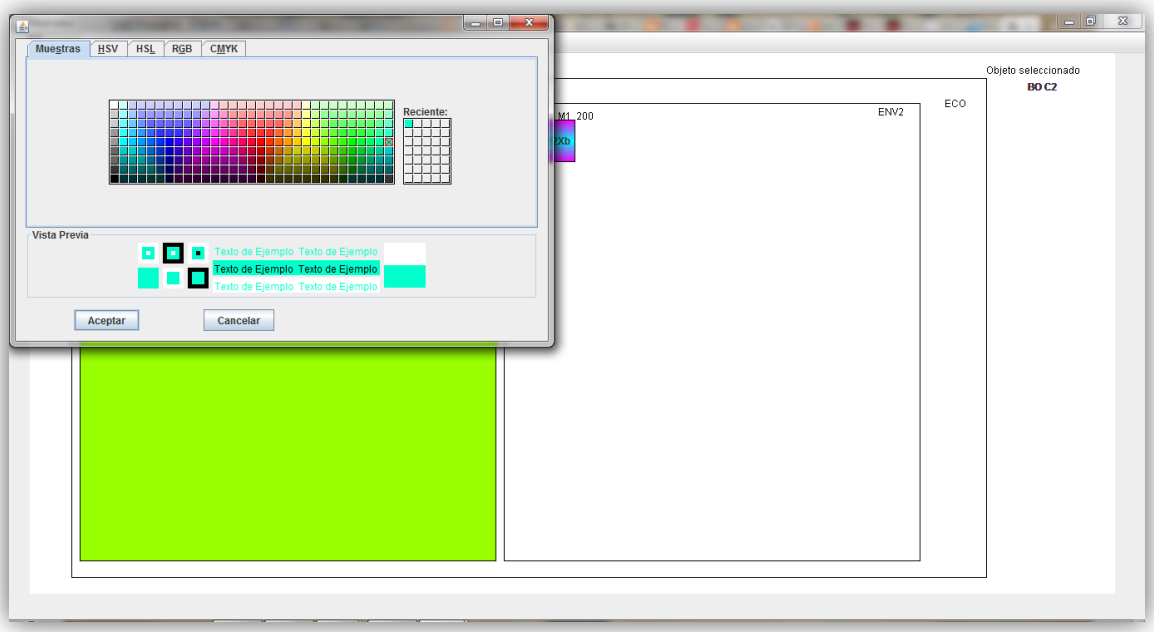

Figura 17

Objeto C3:

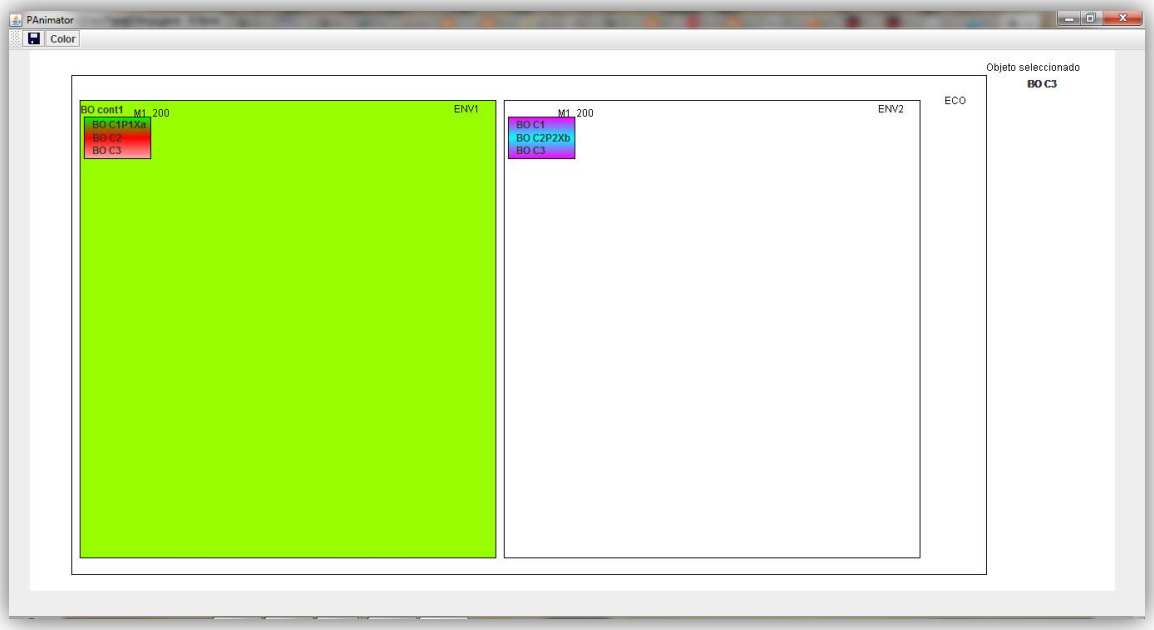

Figura 18

Color de C3:

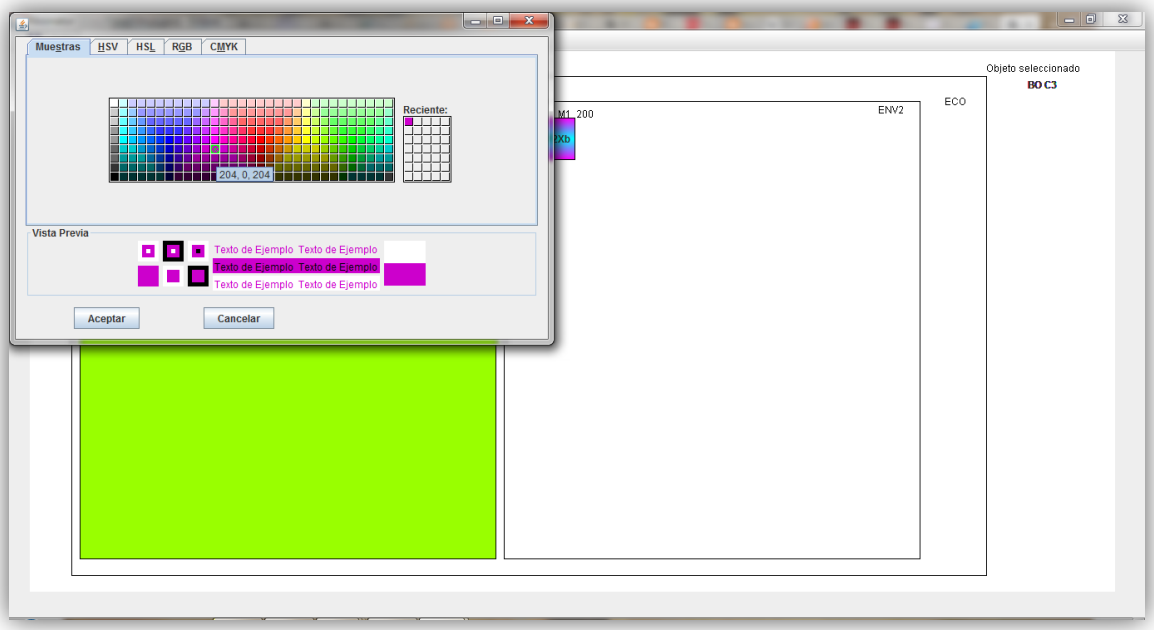

Figura 19

En el momento de aceptar el color que se le proporcionará al último objeto de la membrana M1, el sistema procederá a colorear la membrana en forma de gradiente con igual cantidad de colores, puesto que la cantidad de objetos contenidos en la membrana es la misma:

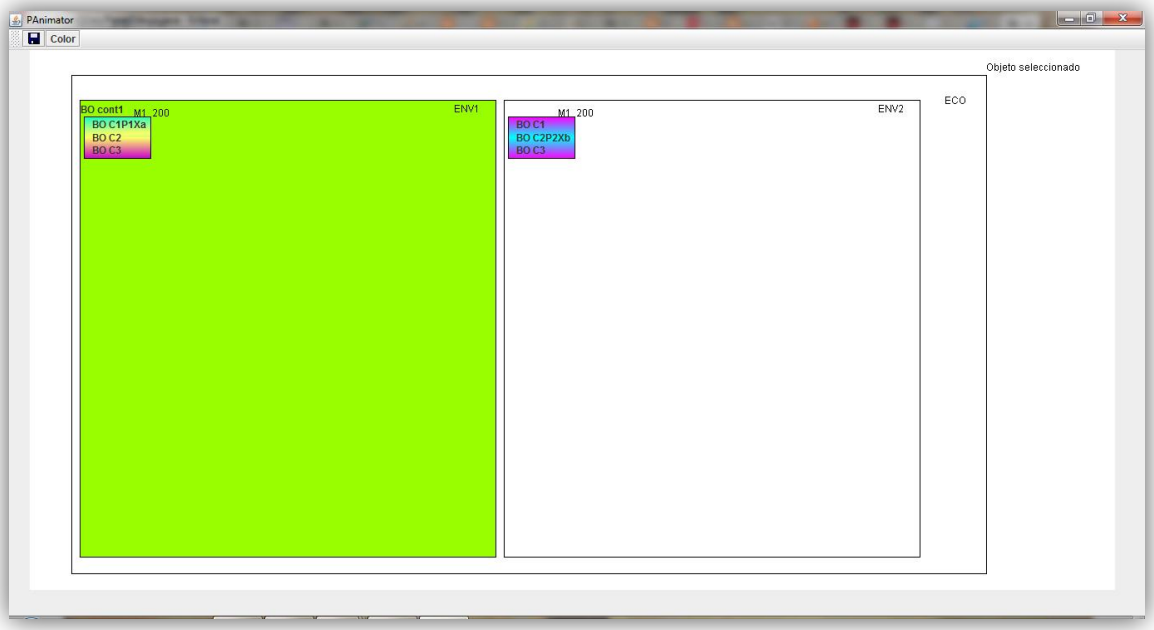

Figura 20

Llegados a este punto de la representación gráfica donde el usuario ha obtenido la configuración de colores deseada, se dirigirá a guardar la imagen obtenida.

Para ello hará click sobre el icono de guardado situado en la barra superior y aparecerá una nueva ventana donde podrá elegir el destino donde guardará la imagen y el nombre de ésta:

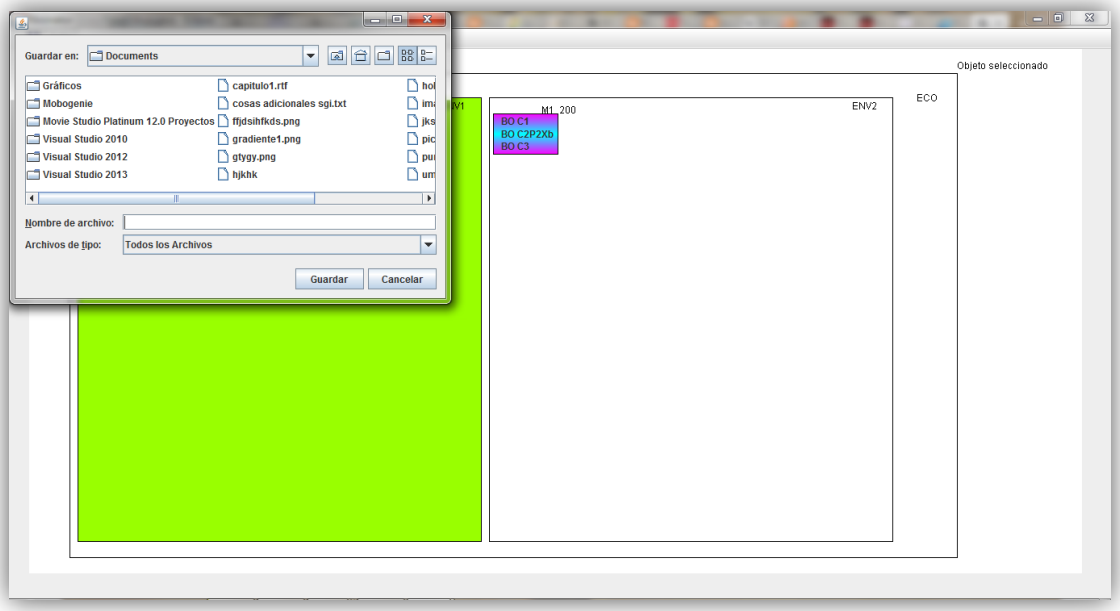

Figura 21

Se guardará la imagen en la carpeta Documentos con el nombre de Sistema1:

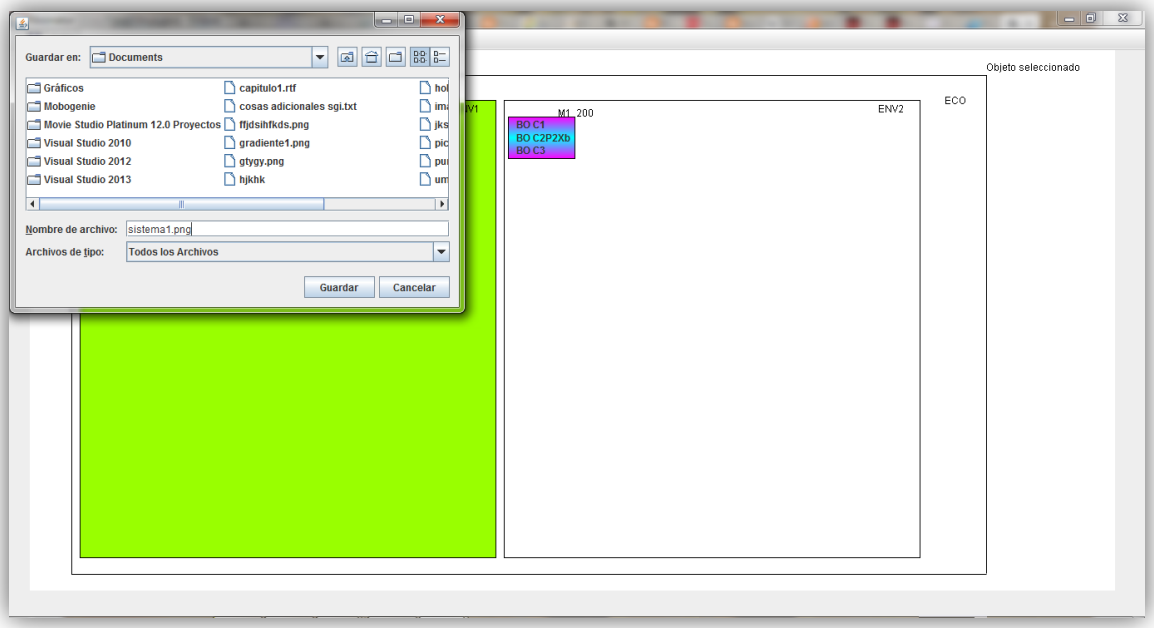

Figura 22

Cuando se pulse el botón Guardar, aparecerá un mensaje como el de la imagen que nos confirmará que ha sido guardada de manera correcta:

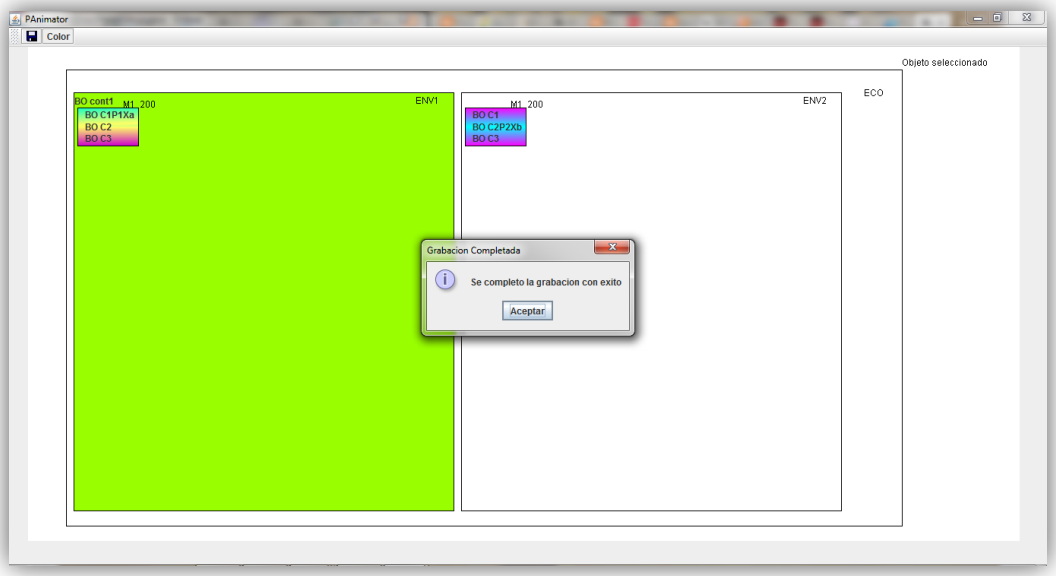

Figura 23

Si el usuario quisiese comprobar que la imagen se ha guardado, se dirigiría a su carpeta Documentos y abriría la imagen con el nombre Sistema1, tal y como la ha guardado.

Esta es la imagen obtenida vista desde el Visualizador de fotos del sistema operativo Windows 7:

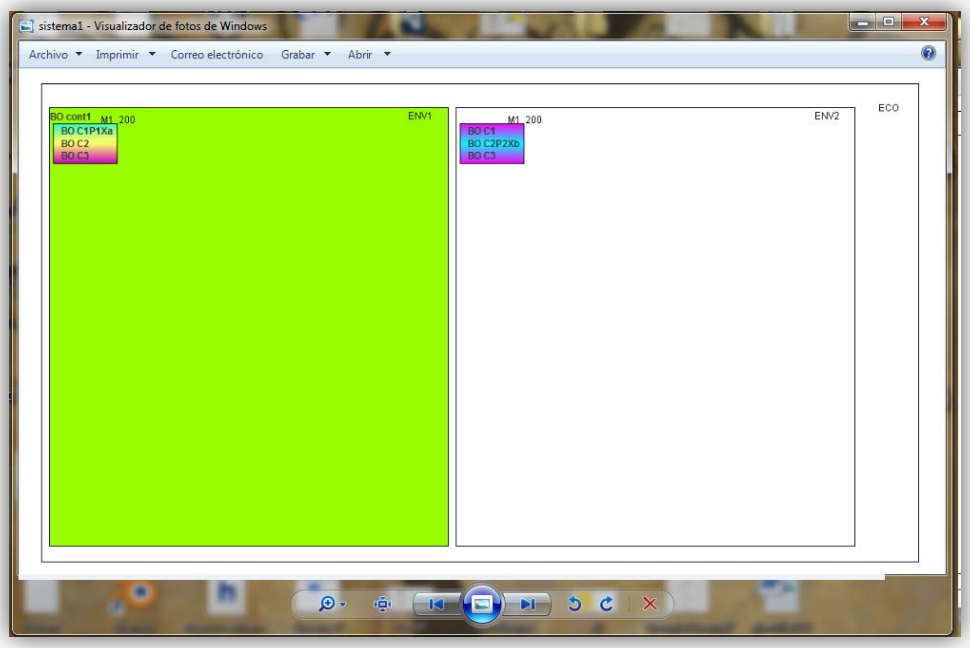

Figura 24

A través de este ejemplo hemos observado como el usuario ha escogido los colores de una matriz. También se puede escoger de manera diferente. A continuación veremos las formas de escoger los colores pudiendo configurar sus atributos como: iluminación, saturación y matiz, cantidad de Color en formato RGB (red, Green, blue), etc.

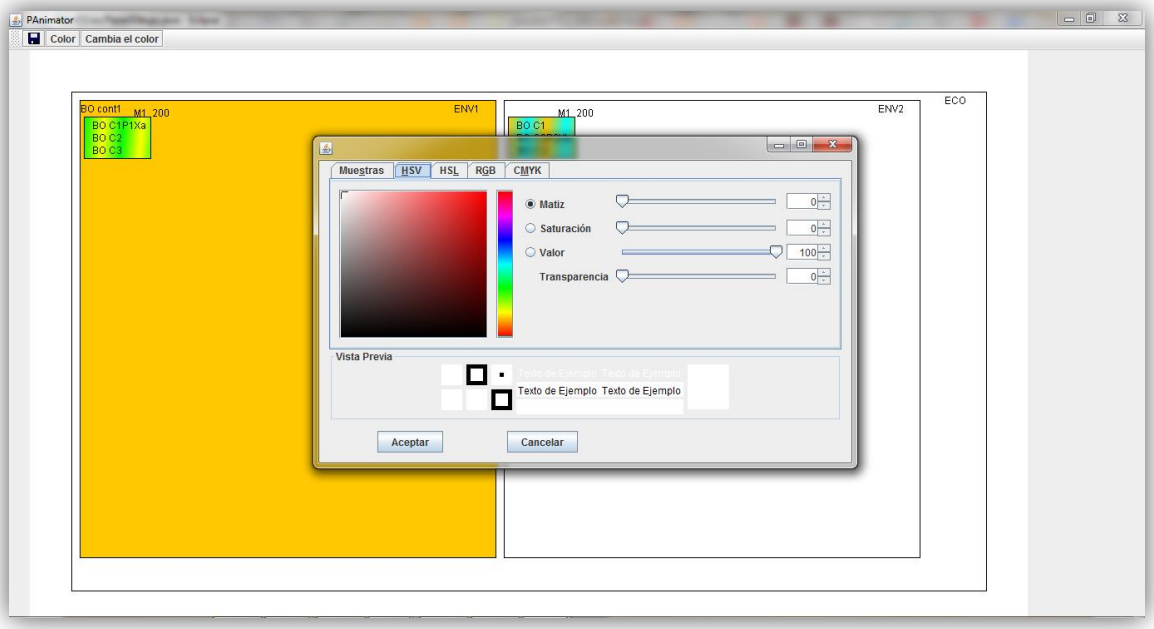

Figura 25

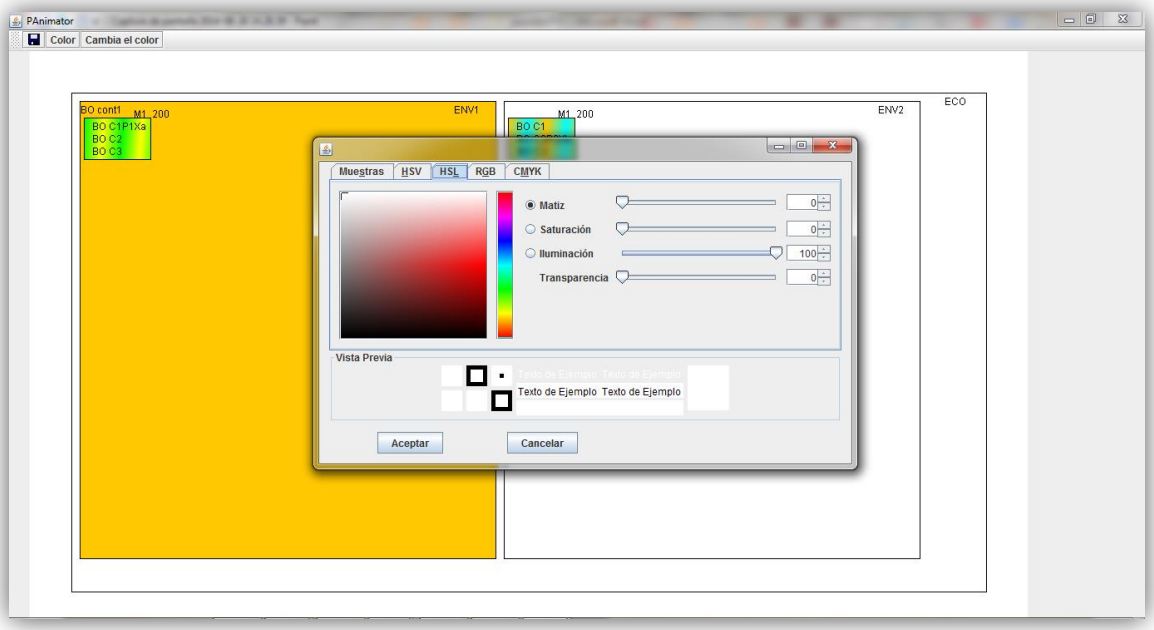

Figura 26

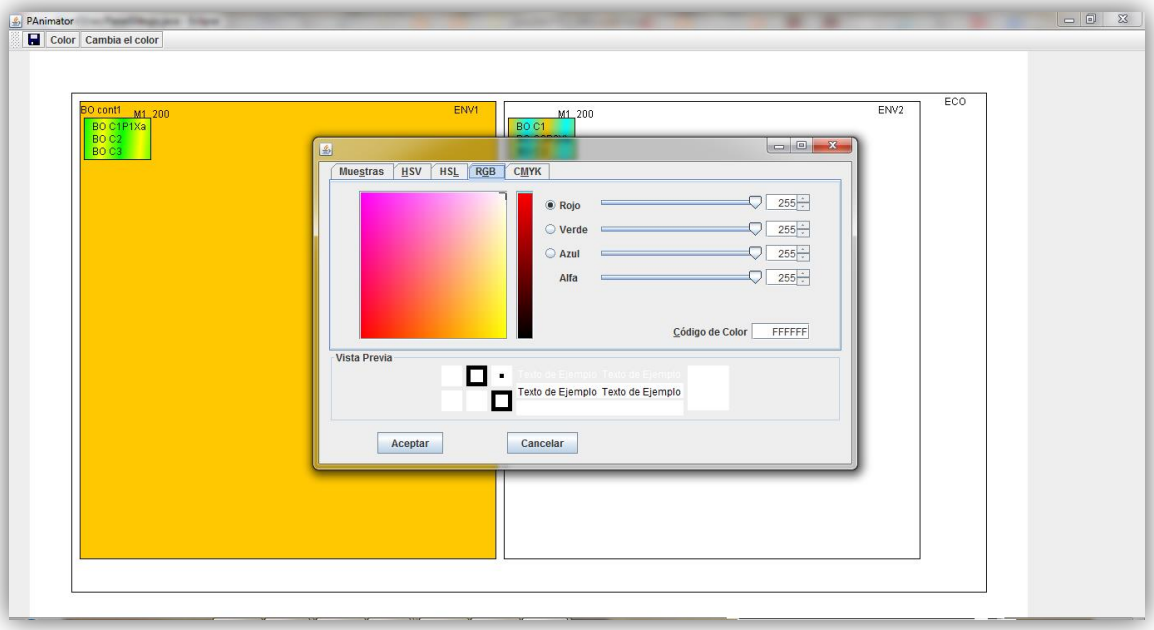

Figura 27

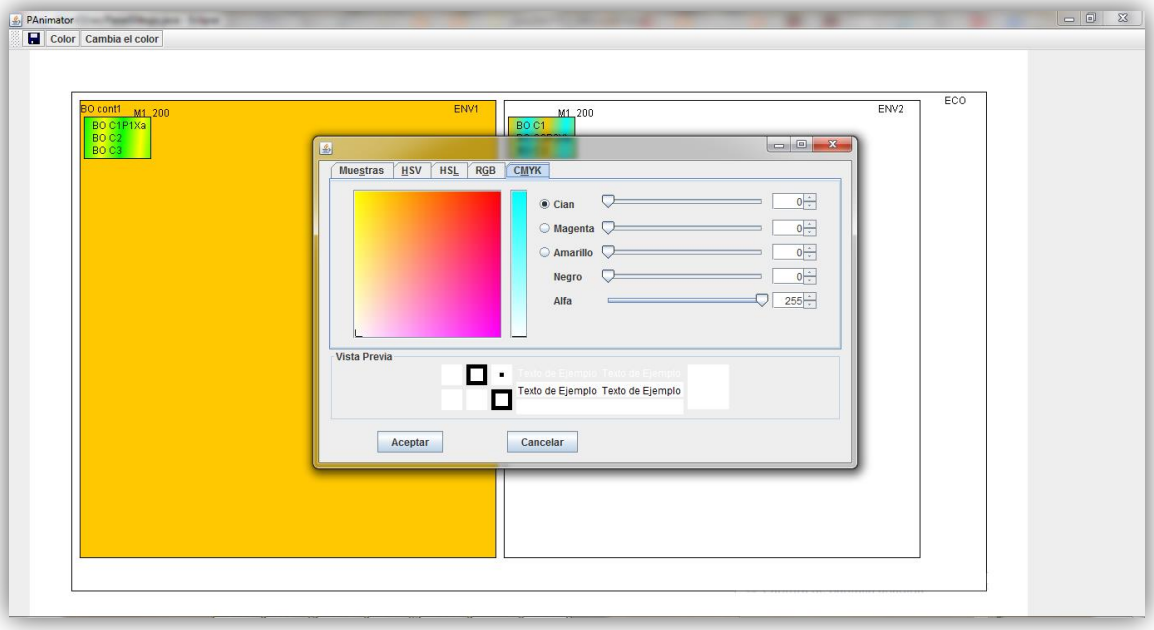

Figura 28

6. Conclusiones

Se han encontrado métodos y clases muy interesantes para nuestro proyecto, pero esto ha requerido el estudio de su funcionalidad y la modificación de estos fragmentos de código para que encajasen en nuestro diseño y cuyos resultados de ejecución fuesen los esperados.

Ha sido difícil parametrizar todos los atributos de los elementos membrana y objeto, sobre todo las posiciones en el espacio del dispositivo x e y.

Se ha experimentado el autoaprendizaje sobre las clases de las bibliotecas graficas de Java. Se tenían conocimientos previos sobre representaciones graficas pero con el lenguaje de programación C++ y la biblioteca OpenGL. Durante la investigación de las clases de Java y tras parte de la implementación de este proyecto se supo de la existencia de JOGL, que encapsula dentro de algunas clases de Java toda la potencia de la biblioteca de OpenGL. Se decidió obviar esta información por dos motivos. El primero es que ya se había empezado la etapa de implementación y la segunda es que JOGL no es orientada a objetos porque OpenGL no está remodelada y adaptada para Java, lo que nos podía llevar a confusiones y errores.

Otra de las dificultades encontradas es la disposición del origen de coordenadas del dispositivo que hemos comentado en la implementación. Dio a lugar a errores y confusiones que hicieron perder tiempo, pero con concentración y trabajo se solventó.

Se espera haber sido capaz de transmitir la motivación que me llevo a realizar este proyecto y la sabiduría adquirida durante la realización de este.

En general y concluyendo, es una satisfacción para la autora de este proyecto haber hecho funcionar de manera correcta y esperada el proyecto detallado. Se ha convertido en una fuente de conocimiento y experiencia donde se le ha permitido poder ser guía y programadora y aprender de errores.

En cuanto al proyecto desarrollado, cabe decir que existen varias extensiones y mucho trabajo que hacer sobre éste. Se ha desarrollado la herramienta con representación gráfica en dos dimensiones, pero también se puede hacer en tres dimensiones, por ejemplo. Igualmente se pueden añadir más herramientas al proyecto, por ejemplo, un editor de vídeo, que permitiese cargar las imágenes almacenadas y poder visualizar una secuencia de fotogramas.

Se puede añadir mucha funcionalidad a este proyecto si se desease.

7. Bibliografía

[1] G. Päun, Computing with Membranes, publicado en Journal of Computer and System Sciences 61, 108-143,(2000).

[2] A. Alhazov, T Ishdorj, Membrane Operations in P System with Active Membranes, Proceedings of the Second Brainstorming on Membrane Computing , 37 - 44, (2004).

[3] A. Alhazov, L. Pan, Gh. Päun, Trading Polarizations for Labels in P System with Active Membranes, Acta Informática 41 (2-3), 111-144(2004).

[4] Teoría de la Computación y Complejidad, asignatura Computación y Complejidad cursada en 3º de grado en la rama de Computación.(2012-2013)

[5]The P-Lingua website, www.p-lingua.org

[6] Tutorial API de Java: Swing, http://docs.oracle.com/javase/tutorial/uiswing/

[7] Tutorial API de Java: Java2d, AWT, http://docs.oracle.com/javase/tutorial/2d/

[8]API de Java de Oracle, http://docs.oracle.com/javase/6/docs/api/

[9]Gálvez Rojas, Sergio. Alcaide García, Manuel. Mora Mata, Miguel Ángel. Java a tope: Java2D (cómo tratar con Java figuras, imágenes y texto en dos dimensiones). Edición electrónica. ISBN 978-84-690-5677-6

[10] JDOM, API de java para acceso y manipulación de XML, http://www.jdom.org