3 HERRAMIENTAS SOFTWARE DE SIMULACIÓN EN COMPUTACIÓN CELULAR

Esta unidad tiene como objetivo presentar las principales herramientas software que se emplearán en las siguientes unidades. Se trata de herramientas que se encuentran a la vanguardia en el campo de la computación celular con membranas, proporcionando desde lenguajes que permitan escribir sistemas P de forma que puedan ser procesados por una máquina hasta entornos visuales para edición, depuración, simulación, análisis y verificación de modelos. Además, se propondrá una metodología para la resolución mecánica de problemas a través de la modelización y simulación.

1 INTRODUCCIÓN

Debido a la naturaleza bio-inspirada, masivamente paralela y no determinista de los sistemas P, su implementación real con la tecnología actual constituye un gran reto para la ciencia. Si bien es cierto que existen estudios preliminares analizando la problemática relativa a dicha implementación (e.g. [5]), aún queda un largo camino hasta alcanzar el objetivo final, respecto al que se muestra especialmente escéptico el colectivo de los biólogos, en lo que a dispositivos celulares se refiere.

Es por ello que la *simulación* de sistemas P utilizando dispositivos electrónicos convencionales se convierte en una necesidad de vital importancia para el progreso de las actividades científicas en Computación con Membranas.

Recordemos que todo modelo de computación consta de una especificación sintáctica y su dinámica se rige por medio de una semántica formal. En esta memoria, usaremos el término *simulador* para referirnos a una aplicación software/hardware que describe la especificación de un modelo formal de computación a través de un cierto lenguaje de programación, y que captura su semántica mediante la implementación de un algoritmo de simulación que debe reproducir la dinámica del modelo con fidelidad. Es decir, cada paso de computación del modelo formal es reproducido en el simulador a través un número finito de pasos, de tal manera que el simulador es capaz de determinar los elementos básicos del modelo que han intervenido de forma relevante en ese paso.

Los simuladores suelen seguir una política de *caja negra*; es decir, producen la misma salida que la máquina simulada, pero no informan al usuario sobre cómo se ha obtenido. En otras palabras, no muestran el funcionamiento del algoritmo de simulación.

En los últimos años, un gran número de aplicaciones de *software* y *hardware* para Computación con Membranas han sido presentadas. La mayoría de ellas son simuladores que replican una o varias de las posibles computaciones del sistema P simulado, siendo posible en algunos casos obtener todas las computaciones. Estos simuladores reciben como dato de entrada la configuración inicial de un sistema P junto con su conjunto de reglas, y producen un conjunto de computaciones como dato de salida, o cualquier otra información relevante para el usuario.

1. 1 Estructura general de un simulador para sistemas P

En esta sección se exponen los elementos comunes que, habitualmente, son incluidos en la mayoría de los simuladores de sistemas P (independientemente de su finalidad).

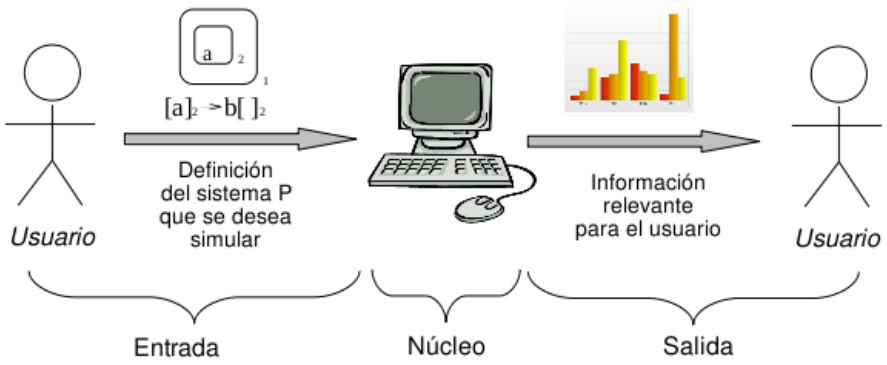

FIGURA 1: Elementos comunes en los simuladores de sistemas P

En la Figura 1 se presenta el esquema general de un simulador de sistemas P, donde se pueden observar los siguientes elementos comunes:

- Definición del sistema P que va a ser simulado.
- Núcleo de simulación.
- Presentación de resultados al usuario.

1. 2 Definición del sistema P

Antes de poder simular un sistema P es necesario establecer una especificación que permita definirlo completamente. Esto implica que al simulador se le ha de suministrar la siguiente información:

- El modelo de sistema P a simular.
- La estructura inicial de membranas.
- Los multiconjuntos iniciales.
- El conjunto de reglas.

Esta tarea de definición se complica cuando es necesario especificar familias de sistemas P (como es el caso cuando se diseñan soluciones a problemas de decisión) en donde el conjunto de reglas, el alfabeto, los multiconjuntos iniciales y, eventualmente, la estructura de membranas dependen de los valores asignados a unos parámetros iniciales.

Las principales soluciones para la definición de sistemas P empleadas por las aplicaciones existentes son las siguientes:

- Definición del sistema P dentro del código fuente.
- Definición del sistema P mediante interfaces de usuario.
- Definición del sistema P mediante ficheros externos.

Un fichero es un conjunto de bits que codifica algún tipo de información y se encuentra almacenado en un dispositivo periférico de almacenamiento como, por ejemplo, un disco duro, un pen-drive o un CD-ROM, DVD, etc. Dependiendo de la forma en que se codifique la información, los ficheros se pueden clasificar en ficheros de texto y ficheros binarios.

En los ficheros de texto la información se almacena mediante secuencias de caracteres codificados según un código determinado. Por ejemplo, ASCII es un código de caracteres de 7 bits + 1 bit de control basado en el alfabeto latino, el estandar ISO-8859-1 es una extensión que utiliza 8 bits para proporcionar caracteres adicionales usados en idiomas distintos al inglés, como es el español. En este tipo de ficheros, cada byte (secuencia de 8 bits) representa un carácter y, debido a esta codificación, suelen ser eficaces a la hora de guardar textos, sin olvidar el hecho de que existan muchos y variados programas informáticos para su edición.

Los ficheros binarios están compuestos por secuencias de bytes que pueden codificar cualquier tipo de información, pudiendo agruparse los bytes de diversas maneras para representar cualquier estructura de datos, siendo necesario conocer el formato del fichero para decodificar su información. Por ejemplo, un fichero binario podría contener una secuencia de números enteros, en donde cada número

se representa en binario con 4 bytes. La ventaja de los ficheros binarios con respecto a los ficheros de texto es que, en igualdad de condiciones, necesitan menos espacio para codificar la misma información. En cambio, tienen la desventaja de ser muy dependientes del formato empleado, necesitando programas informáticos específicos de cada formato para editarlos y manipularlos.

Es necesario establecer un formato de fichero para dar significado a las secuencias almacenadas con independencia de cómo se guarde la información. Es decir, hay que determinar la codificación de la información en el fichero.

Por ejemplo, unos formatos de fichero de texto muy utilizados son los que codifican la información según algún lenguaje basado en XML (eXtensible Markup Language). XML es un metalenguaje extensible de etiquetas que permite definir la gramática de lenguajes específicos. Por lo tanto XML no es un lenguaje en particular, sino más bien una manera de definir lenguajes para diferentes necesidades, pudiendo usar programas de edición de XML o edición de texto para editarlos. Un caso concreto de un lenguaje basado en XML es SBML (Systems Biology Markup Language) que es empleado para representar modelos de procesos biológicos.

Una buena solución podría ser aportar la definición de un sistema P a través de uno o varios ficheros (ya sean de texto o binarios) debido, principalmente, a los siguientes puntos.

- Es una solución de *poco acoplamiento*: la definición de un sistema P a través de ficheros puede ser utilizada por diferentes aplicaciones informáticas, independientemente de la plataforma utilizada o el lenguaje de programación en el que fueron desarrolladas. El único requisito es conocer el formato con el cual se codifica el sistema P.

- Es una solución *reutilizable*: la misma definición de un sistema P puede ser utilizada en diferentes entornos de software ya que al quedar guardada en un soporte físico, no es necesario volver a crearla en sucesivas simulaciones.

- Es una solución con *buena usabilidad*: en la mayoría de los casos, es posible implementar interfaces de usuario para editar, guardar y cargar los ficheros con los cuales se definen los sistemas P a simular. De esta manera, se implementa una solución mixta entre la utilización de ficheros y la aplicación de interfaces de usuario, posibilitando disfrutar de las ventajas de usabilidad de estos últimos.

  También es posible delegar la edición de ficheros a terceras aplicaciones bien conocidas por los usuarios, especialmente en el caso de los formatos basados en XML.

  En cualquier caso, la facilidad de aprendizaje del formato de codificación empleado será un factor determinante para conseguir una buena usabilidad.

La mayoría de las aplicaciones informáticas usadas en Computación con Membranas son simuladores que comparten una serie de elementos en común y, por

tanto, sus programadores han tenido que enfrentarse a problemas comunes de diseño, tales como

- Estrategias para definir los sistemas P a simular.
- Algoritmos de simulación que reproduzcan computaciones de sistemas P.
- Mecanismos para procesar las computaciones simuladas y presentar los datos relevantes al usuario.

Habitualmente, cada simulador aporta soluciones específicas a estos problemas, lo cual provoca una escasa reutilización de código y dificulta la adaptación de los usuarios a diferentes entornos de software.

En este capítulo se proponen soluciones generales a estos problemas de diseño, con la intención de facilitar el desarrollo de futuras aplicaciones y su utilización por parte de los usuarios.

En la primera sección se presenta un lenguaje de programación, que denominamos P-Lingua, como mecanismo estándar para la definición de sistemas P en ficheros de texto. De esta manera, se pueden reutilizar esos mismos ficheros en diferentes aplicaciones informáticas, mejorando el tiempo de adaptación del usuario a una nueva aplicación, así como el coste de desarrollo de nuevos simuladores.

2 EL LENGUAJE DE LA CÉLULA: P-LINGUA

la definición de sistemas P mediante ficheros de texto es una buena solución que proporciona

- bajo acoplamiento, ya que la especificación en ficheros de texto es independiente del programa que los gestiona;
- reutilización, debido a que una misma definición de un sistema P puede ser utilizada por diferentes aplicaciones; y
- usabilidad, gracias a la diversidad de programas de edición de texto existentes.

Los ficheros de texto que definen sistemas P deben seguir algún formato; la mayoría de ellos están diseñados para simuladores concretos, lo cual repercute negativamente en el tiempo y el esfuerzo que el programador requiere para desarrollar nuevos simuladores, así como en el tiempo y el esfuerzo que el usuario final necesita para asimilar los formatos de fichero de diferentes simuladores.

En esta memoria se presenta un lenguaje de programación, *P-Lingua*, como estándar para la codificación de sistemas P en ficheros de texto. De esta manera,

se pueden reutilizar los mismos ficheros de texto que definen sistemas P en diferentes aplicaciones informáticas, mejorando el tiempo de adaptación del usuario a una nueva aplicación y, mediante el desarrollo de bibliotecas de programación que procesen el formato estándar, se consigue mejorar el coste de desarrollo de nuevos simuladores.

La comunidad científica en Computación celular con Membranas está formada por un grupo heterogéneo de investigadores, desde matemáticos e informáticos hasta biólogos, ingenieros, físicos y ecólogos. Este grupo interdisciplinar comparte el lenguaje científico en el que se especifica los sistemas P y, por tanto, se hace necesario el desarrollo de un estándar orientado a esta comunidad que debería tener ciertas similitudes con el lenguaje utilizado tradicionalmente para especificar sistemas P. De esta manera, se reduciría la dificultad de aprendizaje y se mejoraría la usabilidad, permitiendo a los usuarios escribir en un lenguaje que les resulte familiar, dentro de las limitaciones inherentes a escribir en texto plano.

Por otra parte, la existencia de elementos comunes en soluciones propuestas a diferentes problemas numéricos **NP**-completos utilizando familias de sistemas P reconocedores con membranas activas ha permitido realizar una primera aproximación al desarrollo de un lenguaje de programación celular basado en subrutinas o módulos [6]. La especificación de sistemas P de manera modular presenta una serie de ventajas tales como la mejor comprensión de los programas escritos, la elegancia de código y la estructuración en módulos funcionales que se corresponden con secciones o conjuntos de reglas que se pueden utilizar repetidas veces en el la ejecución del programa.

El lenguaje de programación P-Lingua permite definir sistemas P pertenecientes a diferentes modelos o variantes de manera sencilla, pues su sintaxis está basada en la notación científica usada por los investigadores: por una parte, paramétrica, permitiendo definir familias de sistemas P mediante el uso de índices y parámetros; y por otra, modular, atendiendo a las ideas expuestas en [6].

Finalmente, los programas escritos en P-Lingua pueden servir de entrada para diversas aplicaciones informáticas o, mediante el uso de compiladores, pueden ser traducidos a otros formatos de especificación, aportando interoperabilidad entre simuladores, tal como se ilustra en la Figura 2.

2. 1 Sintaxis del lenguaje P-Lingua

La sintaxis de P-Lingua es suficientemente amplia como para definir una gran variedad de tipos/modelos/variantes de sistemas P. Sin embargo, al principio de cada fichero P-Lingua se debe especificar el modelo de sistemas P que se está utilizando, de tal manera que el compilador puede detectar errores semánticos (desde el punto de vista de la programación) cuando el sistema P escrito no satisface las restricciones del modelo especificado.

En la actualidad, en el marco de P-Lingua se pueden definir sistemas P pertenecientes a los siguientes modelos:

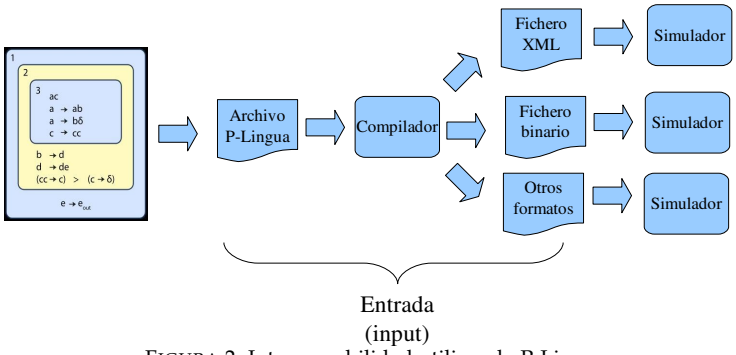

FIGURA 2: Interoperabilidad utilizando P-Lingua

- Sistemas P que trabajan a modo de célula (*cell-like*):
  - Sistemas P de transición.
  - Sistemas P symport/antiport.
  - Sistemas P con membranas activas y reglas de división.
  - Sistemas P con membranas activas y reglas de creación.
- Sistemas P que trabajan a modo tejido (*tissue-like*):
  - Sistemas P de tejido con reglas symport/antiport y reglas de división.
  - Sistemas P probabilísticos multientorno (PDP systems).
  - Simple kernel P systems.
- Sistemas P que trabajan a modo de neuronas (*spiking neural P systems*):
  - SN P systems con división de neuronas y budding.

Ampliar el lenguaje para soportar más modelos es uno de los retos permanentes en los que trabaja actualmente la comunidad de investigadores en Computación con Membranas.

A continuación se presenta la sintaxis de P-Lingua. Los detalles más específicos aparecen en un apéndice, para no sobrecargar demasiado la lectura de esta unidad.

2.0.1. Objetos

Los objetos del alfabeto de un sistema P se escriben usando letras o palabras (concretamente, usando *identificadores válidos*, ver el apéndice para más detalles), estando permitido incluir índices. Por ejemplo, $x_{i,2n+1}$ y Yes se escriben en P-Lingua como `x{i,2*n+1}` y `Yes` respectivamente.

La multiplicidad de un objeto se representa usando el operador `*`. Por ejemplo, si queremos representar $2n+1$ copias del objeto x_i escribiremos `x{i}*(2*n+1)`.

2.0.2. Especificación del modelo utilizado

Como el lenguaje P-Lingua soporta más de un modelo de sistemas P, es necesario especificar al principio del fichero qué modelo se está utilizando. Cada modelo incluye una serie de restricciones; por ejemplo, las reglas de creación de membranas no se permiten en sistemas P symport/antiport. En estos casos, el compilador de P-Lingua posee un analizador que detecta e identifica tales errores. El modelo utilizado se especifica usando la sentencia `@model<model_name>` al principio del fichero.

2.0.3. Definición de la estructura inicial de membranas

Para definir la estructura inicial de membranas de un sistema P, se escribe la siguiente sentencia:

```
@mu = expr;
```

donde `expr` es una secuencia de corchetes representando la estructura de membranas, incluyendo algunos identificadores para especificar la etiqueta y la carga eléctrica de cada membrana.

Ejemplos:

1. $[[\,]^{0}_{2}]^{0}_{1} \equiv$ `@mu = [[]'2]'1`

2. $[[\,]^{0}_{b}[\,]^{-}_{c}]^{+}_{a} \equiv$ `@mu = +[[]'b, -[]'c]'a`

2.0.4. Definición de las células iniciales de un sistema P de tejido

Cuando se define un sistema P de tejido (es decir, el fichero comienza por `@model<tissue_psystems>`), en lugar de definir la estructura inicial de membranas, basta con conocer el número inicial de células en el sistema. La siguiente sentencia cumple este propósito:

```
@mu = [[ ]'1 ...[ ]'q]'0;
```

Por ejemplo, para un sistema P de tejido con 2 células:

```
@mu = [[ ]'1 [ ]'2]'0;
```

2.0.5. Definición de multiconjuntos

La siguiente sentencia define el multiconjunto inicial asociado a la membrana etiquetada `label`.

```
@ms(label) = multiset_of_objects;
```

donde `label` es una etiqueta de membrana y `multiset_of_objects` es un multiconjunto de objetos separados por comas, pudiendo indicar la multiplicidad de los objetos con el operador $\star$. El carácter # se usa para representar el multiconjunto vacío. Con esta sentencia, todas las membranas etiquetadas con `label` reciben el multiconjunto inicial.
Por ejemplo: `@ms(2) = a1*10,b;`

En sistemas P de tejido, la etiqueta del entorno puede ser utilizada para definir el alfabeto del entorno, cuyos elementos son procesados como si tuvieran multiplicidad infinita (sin necesidad de añadir ningún símbolo adicional).
Por ejemplo: `@ms(0) = a,b,c;`

2.0.6. Definición de reglas

Como se ha mencionado previamente, cada modelo de sistemas P permite sólo unos tipos de reglas. A continuación se repasan los diferentes tipos de reglas de cada modelo mostrando ejemplos de cómo escribirlas de acuerdo a la sintaxis de P-Lingua.

`@model<membrane_division>`

1. Una regla de evolución del tipo $[a \rightarrow v]_h^\alpha$ (con $\alpha \in \{+, -, 0\}$) se puede representar como sigue: α`[a -->v]'h;`
2. Una regla de comunicación (*send-in*) del tipo $a[]_h^\alpha \rightarrow [b]_h^\beta$ se puede representar como sigue: aα`[]'h -->` β`[b]'h;`
3. Análogamente, para una regla de comunicación (*send-out*) del tipo $[a]_h^\alpha \rightarrow b[]_h^\beta$ podríamos escribir: α`[a]'h -->b`β`[]'h;`
4. Una regla de división del tipo $[a]_h^\alpha \rightarrow [b]_h^\beta[c]_h^\gamma$ se puede representar como sigue: α`[a]'h -->` β`[b]'h` γ`[c]'h;`
5. Una regla de disolución del tipo $[a]_h^\alpha \rightarrow b$ se puede representar como sigue: α`[a]'h -->b;`

donde a, b y c son objetos; v es un multiconjunto de objetos; h es una etiqueta; y α y β son identificadores de cargas eléctricas.

`@model<membrane_creation>`

1. Las reglas de los tipos 1, 2, 3 y 5 del apartado anterior se escriben con el mismo formato.
2. Una regla de creación de membranas del tipo $[a]_h^\alpha \rightarrow [[b]_{h_1}^\beta]_h^\alpha$ se puede representar como sigue: α`[a]'h -->`α`[`β`[b]'h`$_1$`]'h;`

donde a y b son objetos; h y h_1 son etiquetas de membranas; y α y β son identificadores de cargas eléctricas.

`@model<transition_psystem>`

1. Las reglas del tipo $[u \rightarrow v, w_{out}, w_{1(in_{h_1})} \ldots w_{n(in_{h_n})}]_h$ pueden ser escritas de la siguiente forma:
   `[u []'h1 ...[]'hn]'h -->w[v [w1]'h1 ...[wn]'hn]'h;`

donde $u, v, w, u_1, v_1, w_1, \ldots, w_n$ son multiconjuntos de objetos; y $h, h_1, \ldots, h_n$ son etiquetas de membranas.

De manera opcional, se permite incluir el objeto especial `@d` en una membrana de la parte derecha de cualquier regla con el fin de representar que la membrana que lo contiene será disuelta tras ejecutar la regla.

Los sistemas P de transición permiten prioridad entre sus reglas, el orden de prioridad se especifica en P-Lingua como una expresión numérica entre paréntesis a la izquierda de la regla, teniendo mayor prioridad aquellas que tienen menor número.

Algunos ejemplos:

- $[a^5\, c \rightarrow d_{out}\, e\, f_{in_1}\, g_{in_2}\, \lambda]_0 \equiv$
  `[a*5,c []'1 []'2]'0 -->d[@d,e,[f]'1 [g]'2]'0;`
- $[b^2 \rightarrow c\, d]_1 > [b \rightarrow x\, y]_1 \equiv$
  `(1) [b*2 -->c,d]'1;`
  `(2) [b -->x,y]'1;`

@model<probabilistic_psystem>

1. La sintaxis de las reglas con probabilidad asociada es análoga a los casos anteriores, pero añadiendo la probabilidad al final de la sentencia. Concretamente, una regla del tipo $u[v]_h^{\alpha} \xrightarrow{p} u_1[v_1]_h^{\beta}$ se podría escribir como sigue:

   `u`α`[v]'h -->u1`β`[v1]::p;`

donde u, v, u_1, v_1 son multiconjuntos de objetos; h es una etiqueta de membrana; α y β son identificadores de cargas eléctricas; y p es un número real comprendido entre 0 y 1 que contiene la probabilidad de la regla asociada.

@model<tissue_psystems>

1. El formato para definir reglas de comunicación del tipo $(h_1, u/v, h_2)$ se expresa como sigue:

   `[u]'h1 <-->[v]'h2`

2. Se pueden escribir reglas de división del tipo $[a]_h \rightarrow [b]_h[c]_h$ de las siguientes maneras:
   - `[a]'h -->[b]'h [c]'h`
   - `[a]'h -->[b][c]`

donde h_1, h_2 son etiquetas de células o la etiqueta del entorno, h es una etiqueta de célula; u, v son multiconjuntos de objetos; y a, b, c son objetos.

Algunos ejemplos:

- $(1, b_2c/b_3^2, 0) \equiv$ `[b{2},c]'1 <-->[b{3}*2]'0`
- $[A_1]_2 \to [B_1]_2[C_1]_2 \equiv$ `[A{1}]'2 -->[B{1}] [C{1}]`

2. 2 Definición en P-Lingua de un sistema P de transición

La Figura 3 ilustra un sistema P de transición que genera el conjunto $\{n^2 : n \geq 1\}$, en la Tabla 1 se muestra una posible computación.

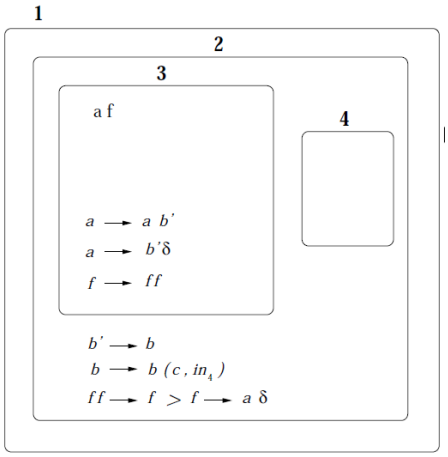

FIGURA 3: Un sistema P de transición que genera el conjunto $\{n^2 : n \geq 1\}$

2.0.7. Definición del sistema P en P-Lingua

En el código que se presenta a continuación se pueden observar las siguientes características sintácticas:

- Utilización de módulos y llamadas a módulos.
- Codificación de la estructura inicial de membranas, reglas y multiconjuntos iniciales de objetos.
- Utilización del símbolo especial `@d` para indicar que una membrana va a ser disuelta.

Paso	Membrana 1	Membrana 2	Membrana 3	Membrana 4
0			af	
1			$ab'f^2$	
2			$ab'^2f^{2^2}$	
3			$ab'^3f^{2^3}$	
⋮	⋮	⋮	⋮	⋮
m			$ab'^mf^{2^m}$	
$m+1$		$b'^{(m+1)}f^{2^{m+1}}$	*disuelta*	
$m+2$		$b^{m+1}f^{2^m}$	*disuelta*	
$(m+2)+1$		$b^{m+1}f^{2^{m-1}}$	*disuelta*	e^{m+1}
$(m+2)+2$		$b^{m+1}f^{2^{m-2}}$	*disuelta*	$e^{2(m+1)}$
$(m+2)+3$		$b^{m+1}f^{2^{m-3}}$	*disuelta*	$e^{3(m+1)}$
⋮	⋮	⋮	⋮	⋮
$(m+2)+m$		$b^{m+1}f^{2^{m-m}}$	*disuelta*	$e^{m(m+1)}$
$2m+3$	ab^{m+1}	*disuelta*	*disuelta*	$e^{(m+1)(m+1)}$

TABLA 1: Una posible computación del sistema P

- Utilización de prioridades entre las reglas.

Téngase en cuenta que en P-Lingua, el objeto b' se escribe `bp`, debido a que el carácter ' (comilla simple) no es un carácter permitido para representar identificadores válidos.

```
@model<transition>
def main()
{
  call n_cuadrados();
}
def n_cuadrados()
{
  @mu = [[[]'3 []'4]'2]'1;
  @ms(3) = a,f;
  [a --> a,bp]'3;
  [a --> bp,@d]'3;
  [f --> f*2]'3;
  [bp --> b]'2;
  [b []'4 --> b [c]'4]'2;
  (1) [f*2 --> f ]'2;
  (2) [f --> a,@d]'2;
}
```

3 MOTORES DE SIMULACIÓN DE SISTEMAS P

El núcleo de simulación es la parte de la aplicación que se encarga de simular una o varias computaciones del sistema P definido.

En este módulo, se parte del supuesto de que el sistema P definido no contiene errores y es coherente con un determinado modelo. Para la consecución de este objetivo se delega en el módulo previo.

El núcleo de simulación ejecuta un determinado algoritmo de simulación que obtiene una o varias computaciones del sistema P simulado. En esta sección se realiza un análisis y clasificación de las estrategias usadas frecuentemente a la hora de diseñar el núcleo de simulación.

Atendiendo al nivel de paralelismo real, el núcleo de simulación podría ser:

- *Secuencial*: se ejecuta un algoritmo de simulación que está diseñado para correr sobre una única CPU, habitualmente consiste en un bucle que selecciona y ejecuta reglas en cada iteración.

- *Multi-hilo*: en programación, un hilo corresponde a una secuencia de código que puede ser ejecutada en paralelo. Dependiendo de la arquitectura del sistema, se producirá un paralelismo real o un paralelismo virtual. Las CPUs que no poseen ningún nivel de paralelismo real, ejecutan los hilos de manera secuencial por intervalos de tiempo; en cambio, las CPUs que poseen un cierto nivel de paralelismo real en su arquitectura permiten la ejecución de hilos en diferentes procesadores o núcleos.

- *Paralelo*: existen determinadas plataformas totalmente paralelas, como es el caso de los *clusters* de ordenadores, en donde cada ordenador conectado corresponde a un nodo de ejecución. Por otra parte, existe hardware paralelo que se puede programar utilizando determinados lenguajes específicos, entre otros, las GPUs (Graphics Proccesor Units) o el hardware reconfigurable basado en FPGAs (Field Programmable Gate Arrays). En cualquier caso, la programación de hardware paralelo no es una tarea sencilla, ya que es necesario conocer bien las restricciones de la arquitectura con el fin de poder alcanzar, siempre que sea posible, un buen grado de paralelismo.

Según la diversidad de sistemas P que se pueden simular, el núcleo de simulación puede ser:

- Diseño para un sistema P concreto: se trata un simulador *ad hoc*. Esta solución suele ser empleada junto con la definición del sistema P en el código fuente y, además, en aquellos casos en los que únicamente interesa simular un sistema P en particular. Se trata de una solución poco flexible pero es la más sencilla y rápida de realizar.

- Diseño para un modelo de sistemas P: esta solución puede simular aquellos sistemas P que sigan la semántica de un modelo determinado. En este caso, el simulador recibe un sistema P definido por el módulo anterior y ejecuta la simulación de una o varias posibles computaciones siguiendo la semántica del modelo.

- Diseño para varios modelos de sistemas P. Es posible encontrar puntos en común entre diversos modelos; por ejemplo, el modelo de transición y el modelo symport/antiport sólo se diferencian en que el segundo es más restrictivo en las reglas permitidas, pues sólo se permiten reglas de comunicación con cooperación dependiente en cierto sentido. No obstante, desde el punto de vista de la simulación se podría utilizar el mismo núcleo de simulación.

En la biblioteca pLinguaCore implementa diferentes algoritmos de simulación secuenciales para los modelos soportados de sistemas P. Cada uno de estos algoritmos permite reproducir paso a paso una de las posibles computaciones del sistema P definido. En cada paso se almacena en memoria un objeto Java que codifica la configuración actual del sistema P. Según las necesidades del usuario, es posible volver a configuraciones previas para permitir una simulación interactiva.

Debido a las similitudes entre algunos modelos, es posible usar algunos algoritmos de simulación para diferentes modelos. Es importante hacer destacar que se supone que la definición del sistema P está libre de errores que puedan conducir a una computación incorrecta, debido a que el analizador de P-Lingua ha comprobado los posibles errores de programación.

4 MECOSIM, UN ENTORNO VISUAL PARA LA SIMULACIÓN

4. 1 Presentación de resultados al usuario

El núcleo de simulación reproduce, paso a paso, una de las posibles computaciones del sistema P definido, para lo cual se establecen estructuras de datos en la memoria de la máquina para almacenar las sucesivas configuraciones alcanzadas. Es posible almacenar todas las configuraciones por las que pasa el simulador, o bien se puede optar por almacenar exclusivamente la última configuración generada.

En cualquier caso, es necesario extraer información de estas configuraciones con el fin de mostrarlas al usuario. La información mostrada dependerá principalmente de la finalidad del simulador.

- En el caso de los simuladores que fueron diseñados con motivos pedagógicos, será interesante mostrar la máxima información posible: la estructura de membranas, los multiconjuntos de objetos, las reglas ejecutadas en el último paso de computación simulado, etc. Esta información se podrá presentar al usuario de varias maneras, siendo la utilización de interfaces gráficas una de las más apropiadas.

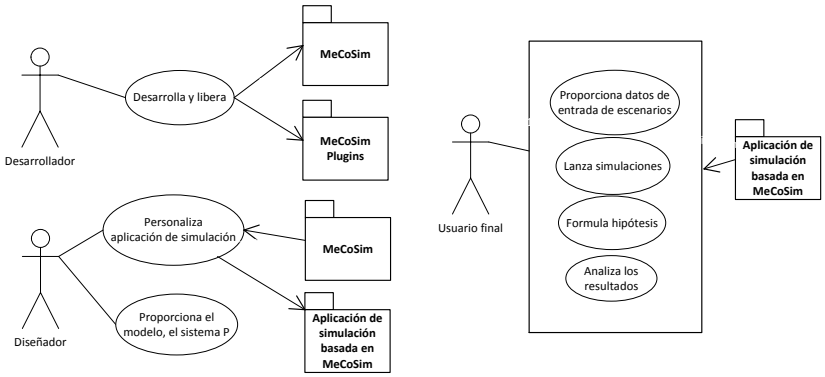

FIGURA 4: Roles y usos de MeCoSim

- En el caso de los simuladores cuyo objetivo es la simulación de procesos de la vida real a través de sistemas P, deja de tener sentido el recopilar los detalles pormenorizados de la computación simulada. De hecho, es muy probable que el usuario final no esté familiarizado con el paradigma de Computación con Membranas, sino que sea un experto en el proceso objeto de estudio a través de sistemas P. Para poder presentar los resultados de manera adecuada a este tipo de usuario, es necesario realizar un proceso de conversión entre la computación generada por el simulador y la información relevante que el usuario necesita.

La presentación de resultados al usuario depende mucho de la finalidad del simulador. En el caso de los simuladores con fines pedagógicos, son relevantes las propias configuraciones generadas por el simulador; en el caso de los simuladores que modelizan procesos de la vida real, la información relevante depende del proceso modelizado y deberá ser mostrada de manera comprensible para el tipo de usuario objetivo.

4. 2 MeCoSim

Como se ha comentado en el apartado de introducción y objetivos, la filosofía de MeCoSim lleva a una clara separación de los posibles roles implicados en el proceso de modelización y simulación de un determinado fenómeno: desarrollador de software, diseñador de sistemas P y usuario final de una aplicación de simulación. ¿Qué proporciona el software MeCoSim dentro de este esquema? Podemos verlo ilustrado mediante el diagrama de casos de uso de la Figura 4.

Como se ilustra en la figura, el desarrollador de software desarrolla y libera versiones de MeCoSim y/o sus plugins, de forma que tales versiones queden a

disposición de cualesquiera diseñadores de sistemas P y usuarios finales potenciales.

El diseñador de sistemas P parte de la existencia de MeCoSim y sus plug-ins. A partir de su funcionalidad general define una aplicación de simulación personalizada, adaptada al problema particular. Mediante esta aplicación, puede llevar a cabo la depuración del modelo y el análisis del sistema P correspondiente. Personalizar aplicaciones de simulación no requiere conocimientos de programación, sino de sistemas P y sus computaciones y simulaciones, de ahí que lo lleve a cabo el diseñador.

El usuario final, por su parte, emplea la aplicación de simulación a medida basada en MeCoSim, proporcionada por el diseñador (con el que interactúa, indicándole a éste las entradas y salidas que necesita), y estudia instancias de la solución proporcionada al problema en cuestión.

Partiendo de los roles y usos resumidos en el esquema anterior, podemos citar algunas de las funcionalidades principales de MeCoSim:

- Entorno general para la simulación de modelos basados en sistemas P. Este entorno se apoya en el framework de pLinguaCore, de modo que la aplicación de simulación por defecto, *General*, permite seleccionar un archivo de modelo escrito en P-Lingua, un *.pli*, y realizar depuración o simulación. Este entorno proporciona capacidades para la edición del archivo de modelo, así como para la visualización de la estructura de membranas o los multiconjuntos presentes en las distintas regiones, entre otras cosas.

- Mecanismo de definición de aplicaciones de simulación. Estas aplicaciones constan de:

  - Una estructura u organización jerárquica determinada por el usuario diseñador a través de un archivo de configuración en formato *.xls* (Excel, Calc).
  - Una definición de entradas y salidas. La principal representación de las entradas y salidas es en forma de tablas de entrada, aceptando la introducción de datos por parte del usuario; y de tablas de salida, mostrando información obtenida de las computaciones.
  - Salidas gráficas, proporcionando una definición de los campos sobre los que generar las gráficas; requiere la definición de una tabla de salida asociada a la gráfica, junto con la especificación de los roles que cumplen los distintos datos a la hora de componer las gráficas (categoría, subcategoría, serie, dato) y el establecimiento del tipo de gráfico (líneas, barras, columnas apiladas o tarta). Además, se pueden obtener salidas múltiples, compuestas por una lista de valores indicando categorías, de forma que al pulsar sobre cada valor para la categoría se muestre el gráfico adecuado, en lugar de mostrar toda la información junta, lo que en ocasiones puede resultar menos informativo. Estos son los casos de los gráficos de líneas y de barras múltiples.

FIGURA 5: Salidas - Datos de partida

Junto con la disposición/organización en pestañas, las tablas y los gráficos, componentes básicos de la aplicación personalizada en cuanto a su interfaz, se deben definir las dos cuestiones fundamentales que se ilustran en los siguientes items.

- Parámetros: datos del modelo que van a variar según la instancia concreta a tratar. Los usuarios van a proporcionar los datos mediante las tablas de entrada ya mencionadas, pero es necesario definir de algún modo cómo generar los parámetros del modelo a partir de dichas tablas. Así, un parámetro podría proceder directamente de una tabla, pero también puede venir derivado de una operación a partir de las tablas de entrada. Para ello se define todo un lenguaje de generación de parámetros.

- Salidas: antes de obtener las tablas y gráficos de salida, tendremos que indicar qué datos son los que se mostrarán. Estos datos partirán fundamentalmente de datos de la computación, de distintas configuraciones, poniendo el foco en los objetos que nos interesen dentro de las configuraciones. El conjunto de datos de partida es del tipo que aparece en la Figura 5.

  Durante la simulación, todas las configuraciones se almacenan según este esquema en una base de datos (generalmente de tipo *on-memory*, para minimizar el acceso a disco en la medida de lo posible). Así, la definición de las salidas consiste en la selección de información sobre lo almacenado, posiblemente incorporando parámetros de entrada o datos derivados teniendo en cuenta los mismos (por ejemplo, podríamos tener un dato de simulación correspondiente a una larva de mejillón cebra con una determinada multiplicidad, y tener un parámetro con el volumen de un determinado compartimento, y estar interesados en mostrar un dato de densidad de larvas haciendo uso de los dos datos anteriores).

  En consecuencia, se establece otro mecanismo para la selección, el filtrado y la agrupación y ordenación de los datos de salida a partir de la computación. Este mecanismo permite la anidación de salidas partiendo de otras salidas previamente definidas. Finalmente, el resultado de aplicar este mecanismo conllevará la generación y ejecución de una consulta **SQL** contra la base de datos.

5 UNA PROPUESTA METODOLÓGICA PARA LA MODELIZACIÓN Y SIMULACIÓN

A partir de la infraestructura proporcionada por P-Lingua y MeCoSim, se ha venido desarrollando una metodología de trabajo para la resolución de problemas tanto de la vida real como de literatura, interesantes bien desde un punto de vista teórico, computacional o de una índole más práctica, aplicado a algún problema de interés. En este sentido, se introdujo en [10] un primer esbozo de esta metodología para la simulación de modelos, extracción de propiedades y verificación de sistemas P, extendida posteriormente en [11]. A continuación se describen los aspectos principales de esta metodología, apoyada en las herramientas proporcionadas por P-Lingua y MeCoSim, que provee de un entorno de simulación que facilita la integración de las distintas funcionalidades involucradas en el proceso:

- Modelización: a partir del estudio del problema a tratar, se lleva a cabo el proceso de abstracción para capturar los datos relevantes para el fenómeno de interés, hasta diseñar un modelo basado en sistemas P que responda a las cuestiones planteadas en dicho estudio, incluyendo los procesos y datos necesarios. Algunos de los principales pasos involucrados en esta labor, en este caso en el ámbito de la variante conocida como Population Dynamics P systems (PDP systems), se encuentran en el protocolo establecido en [2].

  Una vez se dispone del modelo sobre un determinado fenómeno, este puede ser traducido al lenguaje P-Lingua [4] y almacenarse como archivo con extensión *.pli*. Introduciendo en el mismo archivo los valores concretos de los parámetros para un escenario, el modelo estaría listo para trabajar con pLinguaCore o MeCoSim[12].

- Simulación: el modelo anterior puede ser simulado mediante el framework de P-Lingua, o bien de forma gráfica desde MeCoSim, que delega en pLinguaCore el proceso de simulación. Los modelos de sistemas P soportados dentro del alcance de pLinguaCore incluyen diversas variantes de sistemas P de tipo Cell-Like, Tissue-Like, Spiking o Simple kernel, pudiendo encontrar un desglose completo en la página de P-Lingua[15]. Así, cuando se carga un modelo, según la variante de sistemas P a la que pertenezca el modelo tendremos una serie de posible simuladores asociados al mismo, de modo que podamos seleccionarlo mediante la interfaz de MeCoSim, como vemos en la Figura 6.

- Parametrización: el mecanismo de personalización de MeCoSim nos permite definir tablas de entrada para que el usuario pueda introducir datos específicos de cada escenario particular y se generen a partir de ellos los parámetros que complementen el modelo para su simulación, en lugar de estar éstos fijados en los archivos de modelo. De esta forma, podemos diseñar

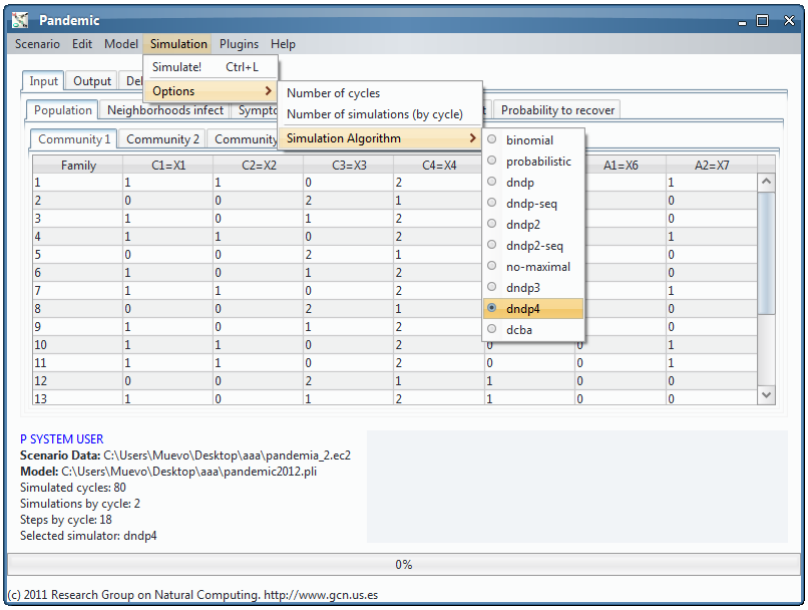

FIGURA 6: Simuladores asociados a un modelo

un único modelo para toda una familia de sistemas P incluyendo un importante número de parámetros, de forma que las distintas instancias del problema, los distintos escenarios, den lugar a un sistema P distinto al simular.

- Depuración: MeCoSim incluye un mecanismo, heredado de los simuladores de la familia Ecosim 1.0, generalizado y extendido, para llevar a cabo la depuración de los modelos de sistemas P. Dentro de las funcionalidades de depuración, tenemos la posibilidad de establecer el modelo ($SetModel$) si no se ha hecho previamente, inicializar el modelo ($InitModel$) para validar su corrección, mostrando en su caso los errores (pestaña $Errors$) y avisos (pestaña $Warnings$) detectados; si el modelo y sus parámetros instanciados a partir de las entradas son válidos se mostrarán los valores de los parámetros generados y las reglas detectadas en la pestaña $ParsingInfo$, entonces podremos proceder a hacer uso de las funcionalidades de simular un paso ($Step$) o un número determinado de pasos ($Runsteps$), mostrando todos los datos de las sucesivas configuraciones en la pestaña $SimulationInfo$. Podemos ver el aspecto de estas funcionalidades en la Figura 7.

  La depuración de los modelos nos permite ir detectando posibles errores en el modelo o los parámetros, según las restricciones sintácticas y semánticas del modelo y los valores de los parámetros según el escenario.

- Visualización y análisis de datos:

  - Definición de salidas personalizadas y post-procesadas: los simuladores

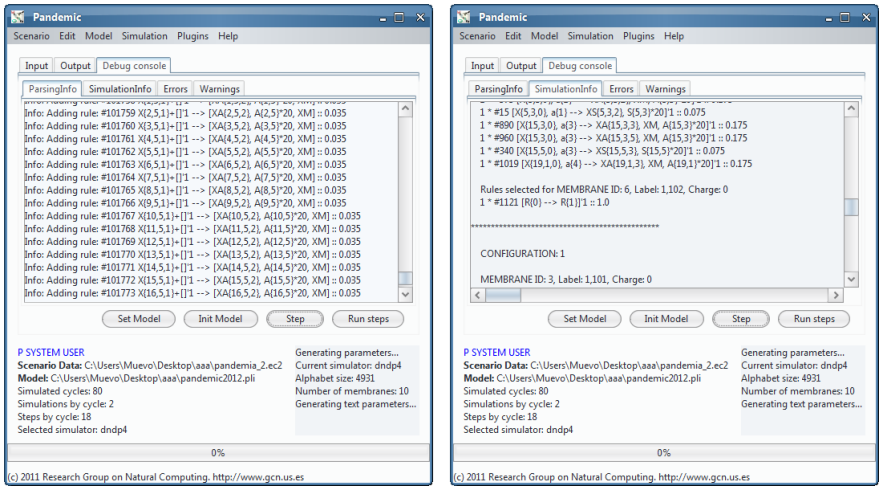

FIGURA 7: Depuración de un modelo

basados en P-Lingua muestran todas las configuraciones y aplicaciones de reglas durante la computación, tanto ejecutando externamente como dentro del modo de depuración de MeCoSim. Para complementar esta información "plana", como hemos visto, el mecanismo de definición de salidas de MeCoSim nos permite mostrar información específica en forma de tablas o gráficos. Esto nos da una mayor batería de funcionalidades de visualización de la parte de la información en la que estemos interesados. Además, nos permite mostrar salidas procesadas, aplicando algún post-procesamiento a los datos de las configuraciones, y mediante técnicas de filtrado y agrupación nos permite un mayor análisis sobre los datos obtenidos como resultado de las simulaciones. Estas herramientas están pensadas tanto para aportar la información al diseñador de sistemas P como para el propio usuario final de la aplicación basada en MeCoSim, interesado en el dominio de su problema y no propiamente en la forma de alcanzar los resultados a través de los sistemas P.

- Visualización avanzada de las estructuras y elementos propios de las configuraciones de los sistemas P: los plugins básicos de MeCoSim nos proporcionan ventanas de visualización orientadas al diseñador, para explorar el sistema P que está siendo simulado o depurado; entre estas ventanas, con estilo visual de árbol de directorios, se incluyen un visualizador del alfabeto (Figura 8, izquierda), otro de la estructura de membranas (Figura 8, derecha) y otro de los multiconjuntos incluidos en cada membrana (Figura 9).

- Visualización de grafos: además de las estructuras anteriores, se ha desarrollado un plugin adicional que permite definir en MeCoSim parámet-

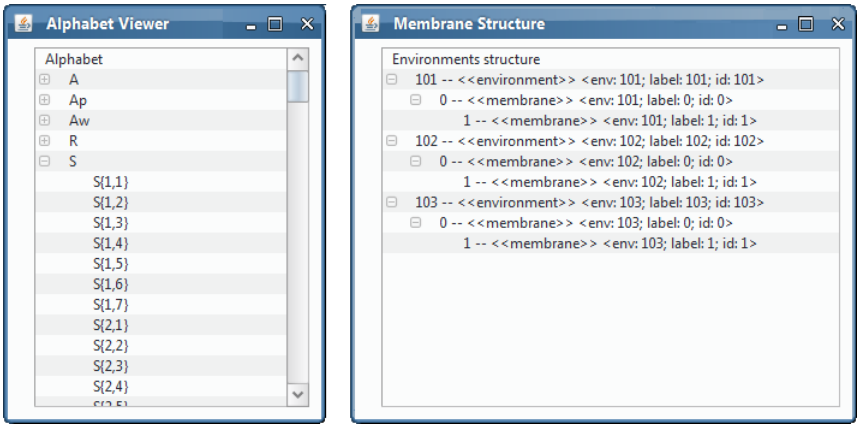

FIGURA 8: Visualizadores de alfabeto (izquierda) y estructura de membranas (derecha)

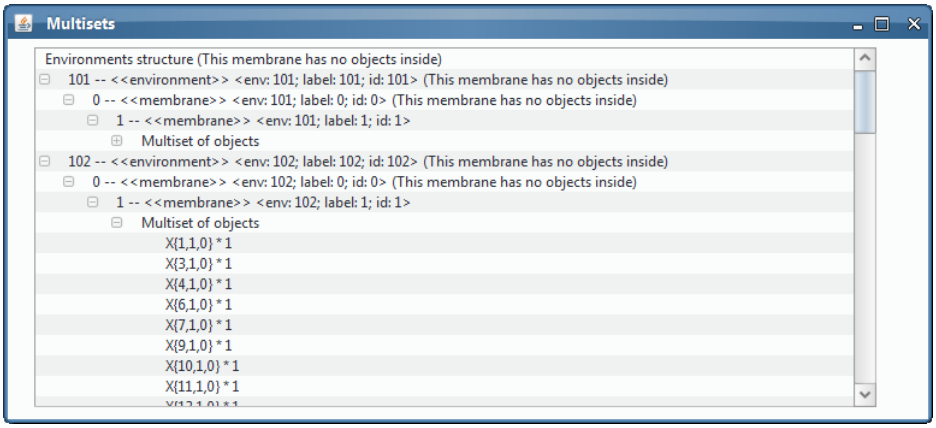

FIGURA 9: Visualizador de los multiconjuntos de cada membrana

ros (a partir de las entradas del usuario o de las tablas de salida de la computación) que actúen como vértices o aristas de grafos. Un ejemplo de ello serían los gráficos de la Figura 10.

Mediante el mecanismo de definición de parámetros de MeCoSim podemos establecer parámetros para el número de vértices (por defecto n), número de aristas (por defecto ne) y aristas específicas del grafo (por defecto $ek, 1$ para el primer nodo de la arista k, $ek, 2$ para el segundo). Con esto, el plugin para visualización de grafos podrá mostrar el grafo asociado a los parámetros generados. Es posible emplear otros nombres de parámetros, en cuyo caso a la hora de visualizar habrá que seleccionar el nombre del parámetro correspondiente a cada uno de estos roles.

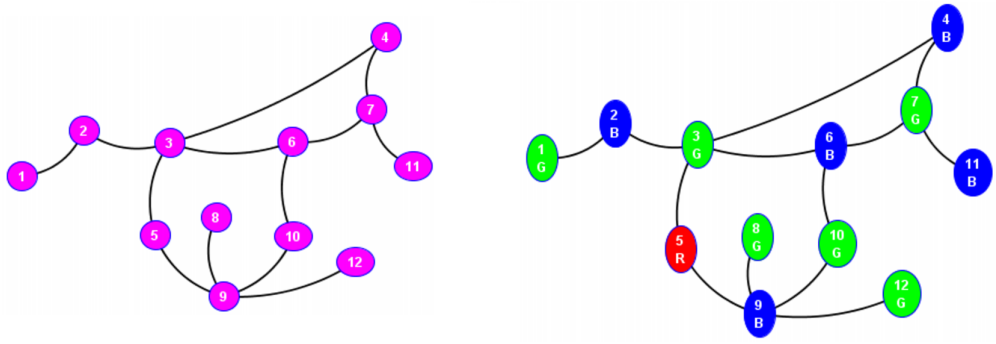

FIGURA 10: Plugin para la visualización de grafos

Con esta información podremos visualizar un grafo mediante parámetros, fundamentalmente generados a partir de la información de tablas de entrada a partir de un número de vértices más lista de aristas o únicamente a partir de la lista de aristas (numéricas o textuales). Podemos ver las opciones disponibles en la Figura 11.

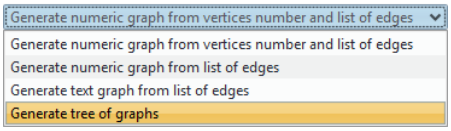

FIGURA 11: GraphsPlugin - Generation options

La última de las opciones, "Generate tree of graphs", permite la visualización de una interfaz conteniendo una estructura de árbol de dos niveles, de forma que cada hoja del árbol presente un grafo. Su origen fue la necesidad de visualizar para cada paso de computación y cada membrana, el grafo codificado por determinados objetos dentro de la membrana. Esto fue llevado a cabo mediante parámetros $versInfo k, l$ para cada nodo k de cada grafo, con l entre 1 y 3, indicando en 1 el valor para el primer nivel del árbol, en 2 el valor para el segundo nivel y en 3 el valor/la información del nodo en cuestión para el grafo determinado por 1 y 2. Así, para cada grafo se representará cada uno de los nodos según los datos en $versInfo$, incluyendo las aristas que puedan existir en el grafo original descrito por los parámetros descritos anteriormente. Podemos ver dos ejemplos de este tipo de visualizaciones de árboles de grafos en la Figura 12.

Además, de existir parámetros de tipo $versAttr$, éstos incluirán información adicional acerca de los nodos. Esta información adicional o meta-información sobre los nodos puede incluir una descripción textual para el nodo, que complemente al posible identificador numérico del nodo. Está prevista la inclusión de nuevos tratamientos sobre

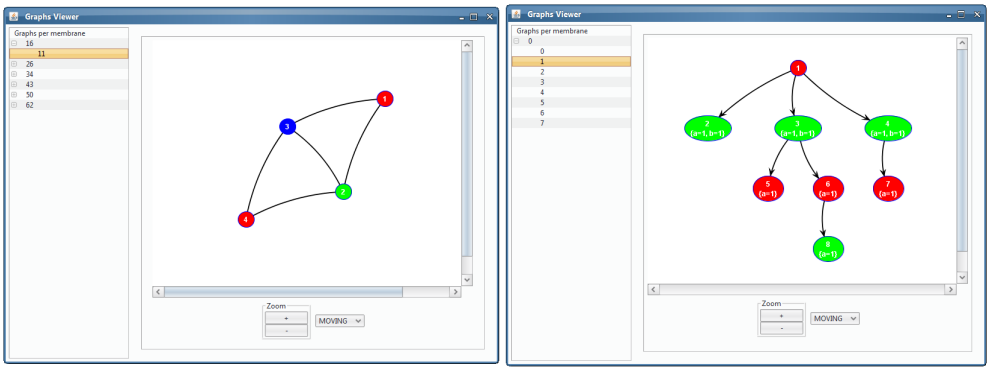

FIGURA 12: GraphsPlugin - Visualización de árboles de grafos

la meta-información. Lógicamente, este tipo de parámetros $versInfo$ y $versAttr$ no tienen por qué llamarse así, pueden definirse otros y establecer su rol a través de las opciones seleccionables, como se comentó anteriormente.

- Extracción de propiedades invariantes: la combinación de las capacidades de obtención de salidas personalizadas de MeCoSim, junto con el plugin desarrollado para la extracción de propiedades mediante la herramienta Daikon (ver [3, 14]), permiten extraer posibles propiedades invariantes en determinados elementos de los modelos diseñados. Este proceso está descrito en mayor detalle en [10].

- Verificación de propiedades: además de la posibilidad de extraer posibles propiedades invariantes de los modelos, a través de salidas a medida a partir de las computaciones/simulaciones, se ha llevado a cabo la integración con MeCoSim de herramientas para la verificación formal de propiedades sobre los modelos mediante técnicas de model checking. Para ello, se ha colaborado en el desarrollo de un plugin, $PromelaPlugin$, que a partir del modelo cargado en MeCoSim y el escenario concreto sobre el que simular, permite generar un archivo en formato Promela correspondiente al modelo. Este archivo puede ser modificado para incluir las propiedades a verificar, y mediante la acción correspondiente se puede lanzar la verificación a través de Spin Model checker [1, 7, 13]. Algunos detalles de este proceso son comentados en [11], teniendo en cuenta las bases para la verificación automatizada de sistemas P mediante Spin sentadas en [8, 9]. Podemos ver el aspecto de la interfaz gráfica del plugin para generación de código Promela y ejecución con Spin en la Figura 13

  Además del citado plugin, dado que el model checker Spin también presenta capacidades de simulación, se ha integrado en pLinguaCore un simulador de una nueva variante de sistemas P, simple kernel P systems, que lanza la simulación a través de Spin.

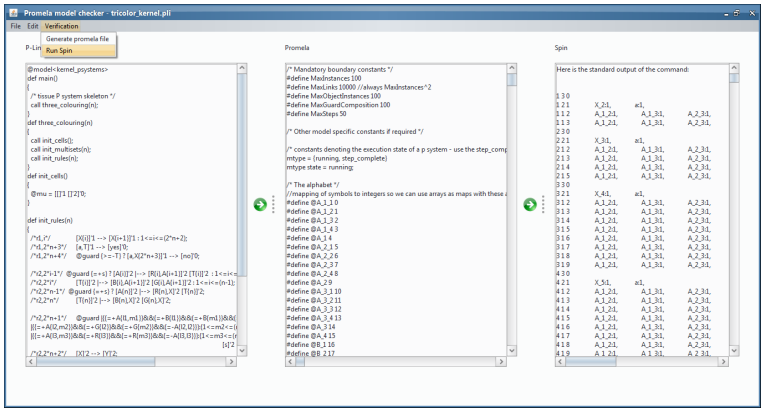

FIGURA 13: Plugin para generación de código Promela y verificación mediante Spin

6 P SYSTEMS BY EXAMPLE (PBE)

A pesar de la alta carga teórica que se ha podido ver en las unidades anteriores y las secciones anteriores de esta misma unidad, la pretensión de este curso no es otra que ser de índole práctica, proporcionando un acercamiento muy rápido a la modelización y simulación en el ámbito de la computación celular con membranas, a través de una sistemática sencilla ilustrada con ejemplos, de modo que todos presenten una estructura similar autocontenida, llevándonos desde el planteamiento del problema a su completa resolución a través de la metodología descrita.

Esta pretensión se ha traducido en lo que hemos denominado P systems By Example, o de forma abreviada PBE, que trata de conducir paso a paso desde el problema hasta la resolución y los resultados. La estructura de cada ejemplo es la siguiente:

- Nombre del ejemplo: un nombre descriptivo acerca del problema a abordar.
- Descripción del problema: descripción del escenario de interés, destacando los elementos del fenómeno estudiado que nos parecen relevantes.
- Descripción del modelo: descripción del sistema P que resuelve el problema.
- Instrucciones para la simulación: descripción de la aplicación de simulación diseñada o empleada, junto con las instrucciones necesarias para su simulación.
- Resultados esperados: detalle de los resultados esperados, aspectos de interés a visualizar y experimentar.

BIBLIOGRAFÍA

[1] M. Ben-Ari. *Principles of the Spin Model Checker*. Springer-Verlag, London, 2008.

[2] Colomer MÀ, Margalida A, Pérez-Jiménez MJ (2013) Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools. PLoS ONE 8(4): e60698. doi:10.1371/journal.pone.0060698

[3] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.*, 69(1-3):35–45, Dec. 2007.

[4] M. García-Quismondo, R. Gutiérrez-Escudero, M. A. M. del Amor, E. Orejuela-Pinedo, and I. Pérez-Hurtado. P-lingua 2.0: A software framework for cell-like p systems. *International Journal of Computers, Communications and Control*, IV:234–243, 09/2009 2009.

[5] R. Gershoni, E. Keinan, Gh. Păun, R. Puran, T. Ratner, S. Shoshani. Research topics arising from the (Planned) P systems implementation experiment in Technion. *Proceedings of the 6th Brainstorming Week on Membrane Computing*, 2008, 183–192.

[6] M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science*, **123** (2005), 93–110.

[7] G.J. Holzmann. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional 2003.

[8] F. Ipate, R. Lefticaru, and C. Tudose. Formal verification of P systems using Spin. *International Journal of Foundations of Computer Science*, 22(1):133–142, 2011.

[9] R. Lefticaru, C. Tudose, and F. Ipate. Towards automated verification of P systems using Spin. *International Journal of Natural Computing Research*, 2(3):1–12, 2011.

[10] Towards an integrated approach for model simulation, property extraction and verification of p systems. *Tenth Brainstorming Week on Membrane Computing*, I:291–318, 02/2012 2012.

[11] M. Gheorghe, F. Ipate, R. Lefticaru, M. J. Pérez-Jiménez, A. Turcanu, L. Valencia Cabrera, M. García-Quismondo, and L. Mierla. 3-col problem modelling using simple kernel p systems. *International Journal of Computer Mathematics*, 90(4):816–830, 2013.

[12] I. Pérez-Hurtado, L. Valencia-Cabrera, M. J. Pérez-Jiménez, M. A. Colomer, and A. Riscos-Núñez. Mecosim: A general purpose software tool for simulating biological phenomena by means of p systems. *IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010)*, I:637–643, 2010.

[13] Spin web site. `http://www.spinroot.com/`.

[14] M. P. A. Group. Daikon web page. `http://groups.csail.mit.edu/pag/daikon/`.

[15] P-lingua supported models variants. `http://www.p-lingua.org/wiki/index.php/Supported_models`.