



UNIVERSIDAD DE SEVILLA
Dpto. de Ciencias de la Computación
e Inteligencia Artificial

Un entorno para la experimentación virtual con modelos computacionales basados en sistemas P

Memoria presentada por
Luis Valencia Cabrera
para optar al grado de Doctor
por la Universidad de Sevilla

Luis Valencia Cabrera

V.º B.º Los Directores de la Tesis

Dr. D. Mario de Jesús Pérez Jiménez

Dr. D. Agustín Riscos Núñez

26 de octubre de 2014

*A los que ya no están,
cuyo vacío no se puede llenar*

*A mis padres
A mi hermana Ana, Rosalía, Niko y Tilman
A Marimeri, Antoñito, Pepi y Joaquín
A Manolito, Mati, Fidel y familias*

*A toda mi familia
A mis leales amigos
A Berlanga y Sevilla*

A los que vendrán...

Prólogo

Aprender de los mayores, reconocer la experiencia...

Aprender y beber de las evidencias del pasado, en la Ciencia y en la Vida, aprender de lo que otros nos transmitieron y nos dejaron, crecer sobre unos sólidos cimientos asentados con el paso del tiempo, colaborar con nuestro granito de arena, con nuestra energía, esfuerzo y capacidad, ayudar con nuestro aliento, testimonio y reflexión a quienes nos sucedan, transmitir lo aprendido de nuestros ancestros y nuestra visión...

Estudiar lo que otros pudieron aportar, alcanzar a comprender lo que nos hicieron llegar, observar el mundo con nuestros propios ojos e integrar nuestras ideas con lo que otros pudieron poner en este mundo, aunar reflexiones, conocimientos y esfuerzos con el pasado y el presente y sentar las bases para el futuro, tender puentes que unan los caminos de la Ciencia y del Universo, de la sociedad, del individuo, a través de la profundización en el conocimiento de cuanto nos rodea y ayudar a tratar de mejorarlo...

Aportar mecanismo vertebradores, integradores, que acerquen los campos, que acerquen posturas, que nos ayuden a comunicar distintos lenguajes, que pongan en común los conocimientos de los científicos de diversas áreas para que podamos entendernos, comunicarnos, enriquecernos mutuamente y juntos podamos alcanzar un mayor conocimiento de los fenómenos haciendo buena la máxima holística de que *el todo es mayor que la suma de las partes*.

El trabajo presentado en esta memoria ha pretendido participar en ese proceso integrador, proponiendo una suerte de pegamento en forma de metodología y herramientas que la soporten, para acercar caminos y tratar de reducir el salto entre la aproximación a un problema y su estudio, y la provisión de herramientas automatizadas que ayuden a profundizar en el conocimiento de los fenómenos a través de soluciones y modelos, abstracciones que sean capaces de capturar una parte relevante de la realidad y que lleven en su esencia en la mayor medida posible esa visión holística que adopta también la *Biología de Sistemas*, entendiendo el estudio de los fenómenos de la forma más global posible, no atendiendo a elementos individuales sino tratando de entender los sistemas y procesos en su totalidad a través de la interacción de múltiples elementos a distintos niveles. Todo ello encaminado a poder aumentar el conocimiento de los fenómenos y nos sirvan de ayuda a la mejor gestión de los sistemas subyacentes a través del uso de los modelos como herramientas prácticas.

Como comenzaba escribiendo, aprender de los antecesores que nos han llevado a lo que somos hoy, a lo que se conoce hoy, porque nada de lo que hacemos día a día merece un Premio Nobel o una Medalla Fields si no lo consideramos como un premio compartido, que parte del apoyo de tantos y tantos que nos llevaron al escalón necesario para que diéramos el siguiente paso, tanto y tantos conocidos y anónimos que a lo largo de la Historia fueron aportando una parte de lo que ha llegado hasta nosotros, de forma que hayamos podido partir de las herramientas teóricas y prácticas, conceptuales o tecnológicas, en tantas áreas de la Vida y el Conocimiento, tan potentes, elaboradas y desarrolladas a través de los siglos.

Sirva este prólogo como reflexión y agradecimiento general, antesala de los agradecimientos particulares a los antecesores y coetáneos más directos de los que he tenido la enorme fortuna de disfrutar y aprender.

Agradecimientos

Nada habría podido hacer sin el apoyo de cuantos me han proporcionado su experiencia y sabiduría cada día que he tenido la fortuna de compartir con ellos. En particular sin quien tenemos la inmensa suerte de tener entre nosotros como un referente, tanto a nivel profesional como moral, por los valores que de forma silenciosa transmite cada día con su ejemplo, con su trabajo, respeto, pasión, responsabilidad y a la vez empatía hacia cuantos acuden a él en busca de consejo o ayuda, a los que es incapaz de negar su abnegada colaboración, por más superado que pueda encontrarse a nivel físico y mental. Muchas gracias Mario, con ese ejemplo es muy difícil que nos veamos tentados a aferrarnos a cualquier excusa de la índole que sea para bajar los brazos, ni siquiera *la desazón por la vida*.

Mi más sincero agradecimiento también a Ignacio, a quien tanto debo desde que comenzara a principios de 2010 a trabajar con él en el desarrollo de MeCoSim a partir de sus desarrollos previos en el ámbito de los ecosistemas, sacando tiempo de donde no lo había, cuando aún me encontraba empleado a jornada completa para una importante Consultora Informática. Y mucho tengo que agradecerle también por ser quien me animó a contactar con Mario para ver si existían posibilidades de seguir mi vocación investigadora y pasar a trabajar para el Grupo de Investigación en Computación Natural.

Agredecer igualmente a Agustín, por su afable compañía, apoyo constante y sabia guía pese a su juventud. Y tanto a él como a Carmen por haber sostenido parte de mi mochila docente para permitirme terminar la labor desempeñada en la tesis presentada en esta memoria. No puedo olvidarme del apoyo humano y la ayuda tecnológica prestada por mi compañero y amigo Luis Felipe, ayuda también recibida siempre de Manu en el desarrollo de simuladores o de Miguel Ángel, en este último caso además siempre acompañadas de buenas dosis de humor, bien acompañado este último año por Francis, otro gran compañero. Una mención especial también a Álvaro, por su constante ayuda y soporte en tareas docentes, administrativas y de gestión. Gracias también a Fran, con quien he tenido una colaboración menos estrecha pero es un gran profesional que está realizando una gran labor que sigo con gran interés desde hace años.

Agredecer a todo el Grupo de Investigación en Computación Natural y el Departamento de Ciencias de la Computación e Inteligencia Artificial el apoyo, la compañía y el respeto demostrado desde mi incorporación en el primer caso en 2010 y el segundo en 2011. Me siento muy honrado de las oportunidades que se me han brindado desde el inicio y me llena de responsabilidad, manteniendo

el firme deseo y dedicación para tratar de corresponder a la confianza que han depositado en mí, esperando aportar el valor a nivel científico y docente que corresponde con el reto al que cada día debemos responder tanto a quienes depositaron tal confianza como a la Comunidad científica y el Alumnado que recibe finalmente el fruto de nuestro trabajo. Mi agradecimiento también a los miembros de distintos grupos de investigación con los que he podido colaborar a lo largo de estos años, fundamentalmente de las universidades de Sheffield, Bucarest, Pitesti, Wuhan y Chengdu, con un agradecimiento especial para Marian Gheorghe, que tanto se esforzó para que tuviera una estancia agradable en lo personal y provechosa desde el punto de vista científico durante mis tres meses en Sheffield, y a Gexiang Zhang, con quien tantos almuerzos, cafés y horas de trabajo conjunto he tenido el placer de compartir este último verano durante su estancia de un mes en Sevilla.

Tuve la suerte de aprender desde muy joven de mis mayores en las más diversas facetas de mi vida, compartiendo tantos momentos y conversaciones inolvidables con quienes tan importantes fueron para ayudar a conformar junto con mis padres y amigos la personalidad que he llegado a desarrollar. Muchas gracias allá donde estéis a Rafael, Lita, Candi, Primi, Fidel, Ana, Manuel, Carmela, Mariquita y Zaida, siento tanto no haberte podido conocer más primita... Y muchas gracia tía Antonia, nuestra tita... Me enseñaron tantas cosas tan importantes como la honradez, la nobleza, el tesón, la responsabilidad, el respeto, la familia, la generosidad, el cariño, la hermandad y también la alegría, la ilusión, el buen humor, levantarse cada día con una sonrisa... Y cómo no, la humildad (*jah sí, has sacado la nota más alta? ¡qué importaaaante!*, parece seguir recordándonos con retintín mi tía Antonia cada vez que a alguno se nos olvida que no hay nadie más importante que otro y que vayamos con humildad por la vida). Gracias a todos por el incalculable valor del tiempo que pasé con ellos, así como por el recuerdo permanente de que forman parte de mí, me siguen a donde voy y me llevan a ir en la buena dirección, tratando de que el camino que vaya trazando les haga sentirse orgullosos y tranquilos de que quienes les sucedemos dejamo el pabellón bien alto en la vida guiados por los principios que nos inculcaron. Del mismo modo que debemos recordar a los que nos trajeron aquí en la Vida, debemos honrar también en la Ciencia a quienes dejaron su sello y pusieron los cimientos de los que llegamos detrás...

Y no por recordar con cariño a los estuvieron vamos a quitar un ápice de agradecimiento y reconocimiento a los que siguen con nosotros, como es el caso de los padres, los que siempre están ahí cuando los necesitamos, que disfrutan con nuestros éxitos, nos ayudan a superar los fracasos y reciben al

hijo pródigo cuando ha estado ausente, los que siempre perdonan, los que nos enseñaron a vivir, y que tuvieron la madurez para no dejarse llevar por sus propias circunstancias en ocasiones difíciles y siempre dieron prioridad a sus hijos sobre sus intereses particulares. A quienes nos criaron no para que fuéramos como a ellos les gustaría haber sido, sino con la libertad necesaria para que fuéramos distintos de ellos y caminar con responsabilidad por la vida, porque no hay responsabilidad sin libertad y confianza, sólo cuando somos libres para elegir podemos ejercer nuestra responsabilidad y escoger el camino correcto. Podremos equivocarnos, pero no caer en la opción más fácil sino recorrer la senda que nos dicten nuestros principios, en todo caso ante la duda la más justa y difícil y la que nos lleve a ayudar a los más débiles, que nada puedan ofrecernos a cambio.

Terminar con este agradecimiento y reflexión para reforzar mi convicción en que deberemos estar siempre abiertos a aprender, en todos los aspectos y a todos los niveles, de nuestros mentores pero también de nuestros alumnos, de nuestros errores, de nuestros afines y de las bondades de quienes lo son menos. Espero seguir aprendiendo de todos, de los más cercanos y de cuantos puedan inspirarme en este mundo global, poniendo mi esfuerzo y capacidad a disposición de la investigación, la docencia y la divulgación, que pueda redundar finalmente en la medida de nuestras posibilidades en el beneficio de la sociedad, ayudando a regar una pequeña parcelita de ese terreno constituido por el conocimiento humano de cuanto nos rodea y de las herramientas que construimos para ayudarnos a profundizar en él.

Índice general

Introducción	1
I Preliminares	19
1. Computación Bioinspirada	21
1.1. Computación Natural	23
1.2. Computación Celular con Membranas	25
1.3. Variantes de Sistemas P	27
1.4. Resolución de problemas de decisión mediante sistemas celulares	38
1.5. Simuladores de sistemas P	53
2. Modelos en la vida real	61
2.1. Necesidad de diseñar modelos	62
2.2. Modelos formales	64
2.3. Modelización computacional	67
2.4. Diferentes aproximaciones de modelos formales	70
2.5. Modelos estocásticos versus modelos probabilísticos	76
3. Modelización computacional basada en Membrane Computing	81
3.1. Sistemas P multientorno	82
3.2. Tipos de sistemas P multientorno	85
3.3. Algoritmos de simulación para sistemas P multientorno	88
II Cuerpo	103
4. MeCoSim: Un entorno visual de simulación de propósito general para la computación celular con membranas	105

4.1. Introducción y objetivos	106
4.2. El software MeCoSim	114
4.3. Metodología	142
5. Simple Kernel P systems	151
5.1. Introducción	152
5.2. Simple Kernel P Systems	154
5.3. Herramientas software para modelización y simulación	159
5.4. Una solución para 3-COL basada en simple kernel P systems . .	167
6. Caso de estudio 1: Modelización de un ecosistema real	171
6.1. Modelización computacional de ecosistemas reales basada en sistemas PDP	172
6.2. El mejillón cebra en el embalse de Ribarroja	178
6.3. Planteamiento y marco de modelización	187
6.4. Diseño del modelo	194
7. Caso de estudio 2: Resolución de problemas NP-completos	235
7.1. El problema 3-COL en MeCoSim	237
7.2. Estudio y simulación del problema SAT	250
III Conclusiones y trabajo futuro	261
8. Conclusiones y trabajo futuro	263
8.1. Conclusiones	263
8.2. Trabajo futuro	267
A. La Historia Interminable del software	271
Bibliografía	277

Índice de figuras

1.1.	Una célula eucariota	25
1.2.	Una estructura de membranas	26
1.3.	Resolución de un problema de decisión por sistemas celulares . .	41
1.4.	Elementos comunes en los simuladores de sistemas P	54
4.1.	Roles y usos de MeCoSim	114
4.2.	Salidas - Datos de partida	116
4.3.	Base de datos interna de MeCoSim	118
4.4.	Generación de parámetros - Configuración	121
4.5.	Resumen de la gramática del lenguaje	126
4.6.	Registros de la simulación en la base de datos.	129
4.7.	Criterios asociados a un resultado.	131
4.8.	Criterios de selección asociados a un resultado.	131
4.9.	Criterios de filtrado.	132
4.10.	Criterios de agrupación.	133
4.11.	Gráfico de líneas.	134
4.12.	Gráfico de columnas.	134
4.13.	Gráfico de columnas apiladas.	135
4.14.	Lista de gráficos de líneas.	135
4.15.	Lista de gráficos de columnas.	136
4.16.	Definición de gráficos	136
4.17.	Daikon - Datos de salida - TableConfig	139
4.18.	Daikon - Extracción - DaikonConfig	139
4.19.	Simuladores asociados a un modelo	143
4.20.	Depuración de un modelo	144
4.21.	Visualizadores de alfabeto (izquierda) y estructura de membranas (derecha)	145
4.22.	Visualizador de los multiconjuntos de cada membrana	145
4.23.	Plugin para la visualización de grafos	146
4.24.	GraphsPlugin - Parámetros para la visualización	146

4.25. GraphsPlugin - Opciones de generación	147
4.26. GraphsPlugin - Visualización de árboles de grafos	147
4.27. Un salida a medida para extraer invariantes con Daikon	148
4.28. Selección de salida de la que extraer invariantes con Daikon	148
4.29. Extracción de invariantes con Daikon	149
4.30. Plugin para generación de código Promela y verificación mediante Spin	150
 5.1. Especificación, análisis y verificación de simple kernel P systems	160
6.1. Mejillón cebra	183
6.2. Cuenca del Ebro	186
6.3. División en zonas del embalse	186
6.4. División en zonas del embalse	190
6.5. Bloques del modelo	192
6.6. Aristas principales del grafo de entornos del sistema PDP	201
6.7. Traza de ejecución { rojo : a eliminar, verde : nuevos, naranja : modificados}	213
6.8. Configuración MeCoSim app - Estructura visual	220
6.9. MeCoSim app - Árbol de pestañas	221
6.10. Configuración MeCoSim app - Tablas	222
6.11. Configuración MeCoSim app - Definición de tabla	223
6.12. Configuración MeCoSim app - Parámetros	223
6.13. MeCoSim app - Temperaturas medias por semana	224
6.14. Configuración MeCoSim app - Parámetros avanzados	225
6.15. Configuración MeCoSim app - Resultado auxiliar	226
6.16. Configuración MeCoSim app - Resultado a mostrar	227
6.17. MeCoSim app - Depuración - Parsing	228
6.18. MeCoSim app - Depuración - Avisos	228
6.19. MeCoSim app - Simulación paso a paso	229
6.20. MeCoSim app - Ver alfabeto	229
6.21. MeCoSim app - Estructura de membranas	230
6.22. MeCoSim app - Multiconjuntos	230
6.23. MeCoSim app - Población inicial	231
6.24. MeCoSim app - Propiedades del área	232
6.25. MeCoSim app - Salida Adultos	233
6.26. MeCoSim app - Salida Gráfico Larvas	233
 7.1. Coloraciones - Definición y visualización de entrada	239
7.2. Coloraciones - Depuración	239

7.3.	Coloraciones - Configuración de la salida	240
7.4.	Coloraciones - Salida	241
7.5.	Coloraciones - Configuración de la extracción	242
7.6.	3-COL - MeCoSim app	245
7.7.	3-COL - Grafo inicial	245
7.8.	3-COL - Árbol de grafos	246
7.9.	3-COL - Un grafo solución personalizado	247
7.10.	SAT - MeCoSim app input	253
7.11.	SAT - Definición de parámetros	253
7.12.	SAT - Gráfico de salida	254
7.13.	SAT - Tabla de simulación	254
7.14.	MeCoSim - Ventana de SATPlugin	259
A.1.	Sitio web de MeCoSim - Instalación	275
A.2.	Sitio web de MeCoSim - Características	276
A.3.	Sitio web de MeCoSim - Documentación	276
A.4.	Sitio web de MeCoSim - Casos de estudio	277

Índice de tablas

Introducción

Desde el principio de los tiempos, el hombre se ha visto abocado a resolver problemas. Problemas muy concretos, como huir de un peligro inminente, buscar alimento en un momento determinado para proporcionarle la energía necesaria para subsistir o resguardarse del frío para mantenerse a una temperatura para la que su cuerpo esté preparado y pueda continuar ejerciendo sus funciones vitales hasta nuevo aviso.

Cuando las necesidades básicas inmediatas quedan cubiertas, el Hombre empieza a pensar, a dedicar algún tiempo a actividades lúdicas y a procesos que puedan hacerle mejorar su calidad de vida a un plazo digamos corto, no ya inmediato. De este modo, se puede comenzar a dotar de herramientas que puedan ayudarle a resolver problemas un poco menos concretos, situándole en una posición mejor de cara a los peligros, ayudándole a contar con una reserva de alimentos o disponiendo de elementos más estables que le ofrezcan cobijo y protección ante los elementos.

Salvarse del último ataque de un animal, de la última ventisca que asola su entorno, o conseguir una presa que evite el hambre durante un rato, puede cubrir una necesidad básica inmediata, pero será más importante para la subsistencia dar solución a problemas más generales como la búsqueda de alimento, el lugar más idóneo donde resguardarse o encontrar la mejor disposición para enfrentarse a un posible depredador. Para todo ello, es necesario plantear posibles estrategias que ayuden a resolver los problemas de manera general, no para cada caso particular, y para ello es conveniente dotarse de las mejores herramientas posibles.

Para solucionar toda esa clase de problemas, a lo largo de los siglos el Hombre ha ido acumulando experiencias que, a lo largo de la evolución, han ido situando a los supervivientes, justamente los mejor adaptados a las circunstancias de cada momento, cada vez en mejor situación a la hora de afrontar los retos que se plantean, tanto a través del conocimiento tácito, ese *know how* aprendido y transmitido de generación en generación, y de la memoria

genética de los individuos y las especies, como ya desde el comienzo de la Historia a través del testimonio escrito. Todo este proceso le ha conducido, sin duda, a una base de teorías y experiencias, de reflexiones y vivencias, que le ha ido proporcionando cada vez un mayor conocimiento de los fenómenos, de los entornos y circunstancias, de los problemas a los que se enfrenta, de sus herramientas y de sus posibles soluciones.

Dentro de la necesidad del ser humano de conocer más y mejor el mundo en el que vive, a todos los niveles, no todos los problemas son de la misma naturaleza ni todos ellos se pueden atacar de la misma forma. En todos los órdenes de la vida, el Hombre se ha ido encontrando problemas que pueden ser resueltos de *forma mecánica*; es decir, mediante un procedimiento sistemático que no requiera un conocimiento mayor del problema a resolver sino, más bien, unas habilidades para ejecutar unas tareas elementales. En ese sentido, cualquier entidad o máquina, viva o no, que sea capaz de llevar a cabo cada tarea o instrucción recogida en el procedimiento sistemático, será capaz de ejecutar la solución del problema resuelto de forma mecánica.

Posiblemente, muchas de las soluciones dadas por procedimientos mecánicos no habrán pasado a la Historia, quizás por ser anteriores a los registros conocidos. Un paso relevante fue el *ábaco* (primer instrumento conocido de cálculo en sus diversas variantes griega, babilonia o china, de origen indeterminado, posiblemente algunos milenios a.C., con algunas evidencias unos siglos a.C.) Por su parte, un lugar especial en la evolución de la resolución mecánica de problemas lo ocupa Abú Jáfar Mohammed ibn al-Khowarizmi, que estableció procedimientos mecánicos para el Álgebra en torno al año 825 d.C., de cuyo nombre se deriva el concepto de *algoritmo*, como un método especial para la resolución de cierto tipo de problemas en el sentido comentado, precisando que dicha resolución debía contar con un conjunto establecido de reglas ordenadas para llevar a cabo la actividad de forma precisa y no ambigua. El nombre de algoritmo proviene de Al-Khowarizmi, pero existen algoritmos conocidos anteriores, como el algoritmo de Euclides, procedimiento para el cálculo del máximo común divisor de dos números enteros, situado entre el 400 y el 300 a.C.

Si bien podemos observar lo antigua que es la necesidad humana de resolver problemas y los siglos que lleva resolviendo problemas a través de métodos mecánicos, un avance muy sustancial en este sentido se produjo con la aparición de los ordenadores electrónicos en la década de los cuarenta del pasado siglo. Cabe resaltar que las bases del ordenador moderno (una máquina de propósito general) ya fueron sentadas un siglo antes por parte de Charles Bab-

bage y Augusta Ada Lovelace, pero la tecnología del momento aún no estaba preparada para materializar el diseño realizado a nivel conceptual.

El salto cuantitativo en cuanto a la resolución de problemas concretos inabordables desde el punto de vista práctico que supuso la irrupción de los ordenadores electrónicos, tuvo un impacto enorme en el siglo XX y continúa hasta nuestros días. Se espera que esta tendencia continúe, si bien desde 1983 sabemos gracias a un resultado de R. Churchhouse que la velocidad de los procesadores electrónicos tiene un tope que jamás podrán superar. Esa limitación se traduce en la imposibilidad práctica de resolver determinados problemas que son muy relevantes en la vida real mediante ordenadores con soporte electrónico.

La necesidad de continuar resolviendo problemas que afectan al ser humano y la limitación citada acerca de la resolubilidad práctica de algunos mediante tecnología electrónica, conducen a la búsqueda de caminos alternativos en una huida hacia delante. Para ello, debemos idear mecanismos que no dependan de esa tecnología, y aquellas vías novedosas que se alejan de la principal línea anterior se suelen denominar no convencionales. En esa tesitura, una posibilidad por la que el Hombre ha optado en numerosas ocasiones es la de mirar a la Naturaleza como fuente de inspiración. La observación de que muchos de los fenómenos que tienen lugar en la Naturaleza pueden ser interpretados como procedimientos de cálculo, es la base de la *Computación Natural*, disciplina científica que recoge diversos paradigmas computacionales cuyo diseño se ha inspirado en procesos procedentes de la Naturaleza.

En los últimos años, desde su creación por parte de Gheorghe Păun en 1998, la Computación Celular con Membranas ha sido uno de los paradigmas de la Computación Natural que más ha atraído a la la comunidad científica, proporcionando un marco computacional inspirado en la estructura y el funcionamiento de las células vivas, que tiene presente la capacidad de las células para efectuar de manera simultánea un gran número de procesos a nivel bioquímico, a nivel de cada comportamiento de cada célula, de cada célula de un tejido, etc. En este sentido, ha habido diversas iniciativas desde su creación, tomando como referencia la célula y su estructura jerárquica interna, los tejidos formados por células o finalmente la comunicación que se produce entre neuronas, para producir modelos de computación muy diversos con un “tronco común”.

Esos nuevos modelos tratan de dar respuestas satisfactorias a la resolución de problemas que afectan al ser humano, concretamente la resolución de problemas abstractos mediante métodos mecánicos, pero tratando de salvar las

limitaciones inherentes a la tecnología electrónica. La computación celular con membranas proporciona modelos de computación *orientado a máquinas*, descritos a través de la sintaxis de los dispositivos o *máquinas* cuya ejecución (de acuerdo con una semántica muy singular) proporciona, teóricamente, todas las *funciones computables* del modelo. En definitiva, resolver un problema dentro de estos modelos de computación conllevará la descripción de las máquinas correspondientes.

Los dispositivos teóricos (las máquinas) de la computación celular con membranas se denominan, genéricamente, *sistemas P*. En función de la idea que los haya inspirado, tendremos sistemas P que trabajan a modo de células, sistemas P que trabajan a modo de tejidos o, incluso, sistemas P que trabajan a modo de neuronas. A su vez, existen diversas variantes de estos sistemas, la mayor parte de las cuales se han demostrado universales (en el sentido de tener la misma potencia computacional que las *máquinas de Turing*), por lo que, de acuerdo con la tesis de *Church-Turing* serán capaces de resolver, teóricamente, cualquier problema resoluble mecánicamente (en la acepción informal del término). Además, se ha demostrado que muchas de esas máquinas tienen la capacidad de resolver *de manera eficiente* (en el sentido de la semántica paralela, maximal y no determinista) problemas computacionalmente duros; en particular, problemas **NP**-completos. Obviamente, la resolución en tiempo polinomial de esos problemas en tiempo polinomial no implica una demostración negativa de la conjectura **P** ≠ **NP** ya que la solución polinomial se obtiene a cambio de usar una cantidad exponencial de espacio.

De esta forma se ha producido un avance interesante en la resolución de problemas complejos en un modelo de computación que reduce drásticamente el número de pasos necesarios para responder a instancias del problema. No obstante, surge la problemática de implementar esos nuevos modelos de computación bioinspirados en máquinas cuyo soporte no sea el electrónico, a fin de sacar rentabilidad a las soluciones teóricas propuestas. ¿Disponemos de la tecnología necesaria para materializar ese diseño en una máquina real? La respuesta es que, de momento, no es posible. Precisamente esto es lo que sucedió, por ejemplo, a mediados del siglo XIX con las ideas de Babagge cuya máquina analítica establecía elementos básicos muy similares a los que rigen el funcionamiento de los ordenadores actuales. Desgraciadamente el sueño de Babbage nunca pudo ser hecho realidad debido, principalmente, a que la tecnología de la época no estaba lo “suficientemente preparada” para la ocasión. La aparición de nuevos paradigmas teóricos de computación abre muchas expectativas, si bien debe venir acompañado de importantes avances tecnológicos

que permitan hacer realidad la construcción de máquinas inspiradas en esos paradigmas computacionalmente potentes.

Dicho lo cual y a pesar de que esos nuevos modelos están aún bastante lejos de poder ser implementados, obviamente en soporte no electrónico, la tarea de diseño de máquinas teóricas debe continuar, a la vez que habrá que esforzarse en construir simuladores que permitan su fácil manejo, aunque no capture de manera literal la semántica de esos modelos. Si bien esos simuladores no podrán explotar toda la eficiencia computacional de las soluciones diseñadas, sí servirán, en cambio, para explorar las capacidades de los modelos, sus limitaciones, su expresividad y potencia descriptiva, así como profundizar en la solución de los problemas empleando el tipo de procedimientos mecánicos asociados a estos novedosos modelos.

Desde la aparición de la Computación Celular con Membranas, se han proporcionado un buen número de soluciones a problemas tanto en el ámbito de lo que podríamos denominar *problemas de la vida real*, más cercanos a necesidades claramente identificables por las personas “de a pie”, como a problemas aparentemente un poco más lejanos que tienen una alta complejidad pero que, a su vez, pueden representar (modelizar) a problemas concretos interesantes. Ese es el caso, por ejemplo, del problema de la 3-coloración de un grafo, aparentemente “un problema matemático” teórico pero subyacente a muchos problemas de la vida real en cuya esencia se pueda encontrar una serie de elementos que deben satisfacer restricciones que involucran a varios de ellos como, por ejemplo la construcción de determinados tipos de circuitos o la elaboración de un calendario de exámenes, entre otros.

En el caso particular de fenómenos interesantes que aparecen en la vida real, es importante tratar de profundizar en el conocimiento en un sentido más amplio. Para ello, hay que capturar los detalles considerados como más relevantes acerca de los fenómenos que se estudian, a través de un proceso de abstracción en el que se determinen los elementos fundamentales a considerar (los fenómenos de la vida real en toda su extensión involucran, generalmente, demasiadas variables, parámetros y circunstancias, muchas de ellas desconocidas para nosotros) y se desechen otros que son considerados menos importantes para el fenómeno objeto de estudio. Como resultado de este proceso, analizando una serie de aspectos de un buen número de elementos y las posibles interrelaciones entre los mismos, se obtienen modelos formales que tratan de reproducir lo más fielmente posible la realidad del fenómeno que se estudia.

Las soluciones a este tipo de problemas reales y de alta complejidad proporcionadas en el marco de un modelo de computación, deben venir acompañadas

del desarrollo de simuladores a medida para resolver de forma práctica instancias concretas de los mismos que son las que suelen aparecer en la vida cotidiana. Estos simuladores *ad hoc* son asistentes del diseñador de soluciones para materializar respuestas concretas a problemas concretos que son modelizados por el problema abstracto. Es decir, parafraseando aquello de que “*pan para hoy, hambre para mañana*” es importante concentrar el esfuerzo en la resolución de problemas complejos abstractos de tal manera que las soluciones para instancias concretas tengan un amplio alcance.

En este contexto, un paso fundamental en la búsqueda de esa solución lo más general posible de los problemas en el ámbito de la simulación de sistemas celulares fue el marcado por P-Lingua. Esta plataforma software proporciona un lenguaje estándar para la especificación de estos dispositivos computacionales, junto con herramientas para el reconocimiento y depuración del lenguaje, así como simuladores capaces de reproducir la dinámica que tendría el sistema real si su materialización ya fuera una realidad. De este modo, en su búsqueda por solucionar problemas y tras la evidencia de las limitaciones encontradas en otros paradigmas, el hombre no sólo cuenta con una sólida teoría y una experiencia en el ámbito de la computación celular con membranas, sino que también dispone de unas herramientas para la simulación de los diseños realizados, a falta de una implementación real.

El diseñador de modelos formales en computación celular con membranas puede, así, dedicar sus esfuerzos al diseño de soluciones en el estudio de los fenómenos, sin preocuparse del desarrollo de simuladores para cada problema particular, dada la disponibilidad de simuladores de propósito general capaces de hacer ese trabajo.

Así pues, partamos de que ya se dispone de un lenguaje, unas herramientas y unos simuladores de los sistemas P para analizar su dinámica a partir de un escenario (configuración) inicial hasta una determinada situación de parada. A pesar de todo ello, siempre existirá una gran cantidad de problemas a los que nos podremos enfrentar y todo esfuerzo será pequeño con la inmensidad que nos queda por conocer, más y mejor, el mundo que nos rodea y resolver los problemas derivados de ello. Ahora bien, ¿qué limitaciones encontramos y dónde podría aportarse un valor a la hora de resolver problemas? Como se ha comentado, se han proporcionado herramientas para evitar el desarrollo de simuladores a medida, se ha resuelto ese problema general y, por tanto, ya podemos centrarnos en diseñar los sistemas P para resolver los problemas y éstos podrán ser ejecutados mediante los simuladores proporcionados, pero...

La provisión de un modelo computacional que resuelva un problema relacionado, por ejemplo, con un ecosistema real por parte del diseñador de sistemas P, ¿puede ayudar por sí solo a la mejor toma de decisiones por parte del ecólogo experto? ¿la existencia de un sistema P que represente, de forma sorprendentemente ajustada, una ruta señalizadora de proteínas relacionada con la *apoptosis* (muerte celular programada) puede dotar por sí sola de una herramienta interesante para el biólogo molecular?

La respuesta no puede ser un categórico no, pero si leemos el enunciado de la pregunta podremos estar de acuerdo en que antes de convertirse en herramienta útil este sistema debe ser bien comprendido por el ecólogo o el biólogo molecular, o bien, si estamos en el ámbito de los problemas **NP**-completos, por el lógico, el matemático o el experto en el dominio del problema del que se trate.

Con independencia del campo de aplicación o del dominio del problema relevante a resolver, si se va a trabajar con variantes de modelos de computación celular con membranas, el experto en el dominio del que se trate deberá tener ciertas nociones de los sistemas P. Para cada situación que se plantee el experto, en su profundización en el problema concreto en el que esté trabajando, deberá volver al modelo con el fin de codificar en el lenguaje **P-Lingua**, los datos correspondientes a la situación e interpretar la evolución de los elementos que participan en el modelo, en términos de la dinámica del sistema real que trata de analizar y cuyos problemas desea de resolver.

De esta manera, el diseñador de modelos en computación celular, ha conseguido resolver un problema con carácter general, dotar de un marco computacional que resuelve el problema abstracto e, incluso, codificarlo en **P-Lingua**, trabajar con el reconocedor del lenguaje, depurarlo y “reproducirlo” con el simulador de propósito general a disposición en el ámbito de los sistemas P. Este es el nivel al que trabaja el diseñador de sistemas P, con un profundo conocimiento de este tipo de variantes, y capaz de interactuar con los expertos del dominio del problema profundizando en ese conocimiento sobre los fenómenos, así como con los sistemas y simuladores.

Sin embargo, aparecen una serie de dificultades cuando hay que emplear este modelo con el fin de que sea una herramienta útil para la mejora en el conocimiento y la gestión práctica de los fenómenos por parte de los expertos en el dominio del problema. Sin olvidar que tales expertos deben trabajar con las mismas herramientas del diseñador, situarse al mismo nivel de abstracción y afrontar distintos problemas con esas herramientas.

El principal objetivo del trabajo presentado en esta tesis es dar un nuevo paso en la senda abierta por P-Lingua para la resolución de problemas mediante modelos no convencionales, más específicamente, mediante modelos de computación celular con membranas. Se trata, pues, de acercar tales herramientas a los expertos en los fenómenos, procesos o sistemas complejos objetos de estudio, de tal modo que se acorte el salto importante que va desde el modelo teórico o la solución a nivel de diseñador, hasta la disposición de una solución práctica a nivel del usuario final. En este punto nos podemos preguntar si existen modelos de computación celular con membranas que hayan sido utilizados en la resolución de problemas reales por los expertos en los dominios en cuestión, ajenos a las interioridades de los sistemas P. En efecto, eso se ha logrado para determinadas variantes pero, tradicionalmente, eso ha supuesto un coste muy significativo, en ocasiones teniendo que desarrollar simuladores software a medida para resolver un problema concreto y, en otras ocasiones, pese a disponer de simuladores y diseños generales basados en P-Lingua se ha tenido que desarrollar una aplicación software de alto nivel para situarlo en el dominio del usuario final.

El entorno de simulación presentado se denomina **MeCoSim** (**M**embrane **C**omputing **S**imulator) y pretende cubrir esa necesidad, resolver ese problema. Es decir, se trata de implementar un mecanismo general para poner a disposición de los usuarios aplicaciones adaptables a cada problema particular, abstrandéndoles de las interioridades de los sistemas P y los modelos basados en ellos, que utilizarán como herramienta, a modo y manera que P-Lingua proporciona a los diseñadores de sistemas P herramientas que los abstraigan del detalle de los reconocedores y simuladores. De esta forma, se pretende que esos problemas reales a los que se ha buscado una vía alternativa de solución, una especie de huida hacia delante para resolverlos, sobreponiéndonos a las limitaciones de paradigmas convencionales, pasen del ámbito de los diseñadores de sistemas P a terminar constituyéndose en herramientas útiles para la resolución práctica de instancias tratables de los problemas, precisamente para aquellos que estén especialmente interesados en aumentar su conocimiento acerca de los fenómenos, pudiendo así erigirse en potentes asistentes para la mejor toma de decisiones acerca de la gestión de los fenómenos de interés.

Para acometer el objetivo anterior y poner un pequeño granito de arena, una lágrima en el mar del conocimiento para sumar a los esfuerzos de tantos hermanos en la búsqueda de resolver los problemas que atañen a la Humanidad, se ha desarrollado el trabajo que, en la medida de lo posible, se trata de plasmar en los contenidos que se estructuran a continuación.

Contenido de la memoria

La presente memoria se ha estructurado en tres partes principales que en conjunto constituyen ocho capítulos cuyos contenidos pasan a describirse a continuación.

Parte I: Preliminares

La primera parte de la memoria se organiza a través de una serie de capítulos preliminares que tratan de sentar las bases sobre las que se sustenta el cuerpo de la tesis.

El **Capítulo 1** parte de la necesidad del Hombre de resolver problemas relevantes para su vida, y la existencia de determinadas soluciones a los mismos a través de procedimientos mecánicos. A continuación, se observa las limitaciones de los dispositivos de propósito general (ordenadores) que tienen soporte electrónico, a la hora de resolver determinadas instancias de problemas relevantes, así como la necesidad de explorar vías alternativas. Para ello, se recurre a métodos computacionales no convencionales inspirados en la Naturaleza viva, dentro de la disciplina de la Computación Natural. A continuación se profundiza un poco en el paradigma de la *Computación Celular con Membranas*, inspirada en la estructura y el funcionamiento de las células de los organismos vivos. Se proporcionan definiciones detalladas de varios dispositivos computacionales enmarcados en este paradigma, a saber, los sistemas P de transición, los sistemas P con membranas activas, así como los sistemas P que trabajan a modo de tejidos (incluyendo reglas de división celular o de separación celular). Además, se introduce el concepto de resolubilidad en tiempo polinomial de un problema de decisión en el ámbito de estos sistemas celulares. La última parte del capítulo está dedicada a explicar el rol que cumplen los simuladores de sistemas P, las características que debe presentar una buena plataforma de simulación y, finalmente, se realiza una revisión general de los simuladores paralelos desarrollados hasta el momento para sistemas P.

El **Capítulo 2** aborda la problemática que se encuentra el investigador a la hora de estudiar sistemas dinámicos complejos, con especial atención a aquellos que se dan en el mundo real y son de interés. Se profundiza en el proceso de *modelización* de la porción del mundo real que es objeto de estudio, desde la abstracción de los aspectos más relevantes de los fenómenos en cuestión, pasando por la representación de los elementos involucrados en los mismos, hasta el uso y finalidad de los modelos diseñados para aumentar el

conocimiento acerca de los fenómenos estudiados. Además, se analiza la posibilidad de emplear los modelos como herramientas útiles para ayudar a una toma de decisiones más informada por parte de los gestores de los fenómenos de interés. Seguidamente se define con precisión el concepto de *modelo formal* y las fases que caracterizan el proceso de modelización, se recuerda la noción de modelo de computación y se describen las propiedades deseables en todo buen modelo formal. A continuación, se analizan diversas aproximaciones de modelos formales, tanto computacionales como no computacionales, y el tema finaliza describiendo la aleatoriedad inherente a los procesos que se dan en la vida real, así como la idoneidad de capturar ciertos aspectos de la misma a través de modelos matemáticos, estocásticos o probabilísticos.

El **Capítulo 3** centra la atención en el ámbito de los modelos computacionales dentro del paradigma de la Computación Celular con Membranas, fundamentalmente en su empleo como marco de modelización para el estudio de fenómenos de interés de la vida real. Así, se da un marco formal uniforme, los *sistemas P multientorno*, para la representación de sistemas complejos de diversa índole, bajo los que se encuentran dos aproximaciones diferentes que encajan perfectamente dentro del marco: los sistemas P multicompartmentales, para fenómenos moleculares y celulares de naturaleza esencialmente estocástica, y los sistemas P de dinámica de poblaciones, una aproximación probabilística empleada, fundamentalmente, en procesos relacionados con la evolución en el tiempo de ecosistemas reales. El resto del capítulo presenta una serie de algoritmos que permiten capturar la semántica de los dos tipos de sistemas multientorno mencionados, planteando además distintas posibilidades a la hora de reproducir la dinámica de los sistemas, en función de las características de los fenómenos bajo estudio, como la existencia de cooperación entre individuos o la competencia por determinados recursos.

Parte II: Cuerpo

El cuerpo de esta memoria presenta el núcleo del trabajo realizado, describiendo en primer lugar las herramientas conceptuales, metodológicas y prácticas desarrolladas y, a continuación, la aplicación a diversos problemas de interés, tanto para la resolución de problemas de la vida real como problemas teóricos computacionalmente duros, como son los problemas **NP**-completos.

El **Capítulo 4** se centra en la descripción del entorno de simulación **MeCoSim**, software de propósito general para la validación experimental y experimentación virtual de los modelos analizados en los capítulos anteriores. Inicialmente

se introducen las herramientas, su origen y los objetivos principales que motivan su aparición. A continuación, se detalla la estructura interna del entorno y las principales funcionalidades proporcionadas tanto al diseñador de sistemas P, poniendo a su disposición un conjunto de facilidades para el diseño, depuración e introspección del modelo, como al usuario final, al que se le proporciona una aplicación visual que le permite trabajar a alto nivel centrando su atención en el estudio de los fenómenos de su interés, introduciendo sus escenarios y realizando las simulaciones correspondientes hasta obtener resultados, abstractayéndole de los detalles internos de los modelos. El capítulo finaliza con la presentación de una metodología para la solución de problemas en el ámbito de los sistemas P, desde el estudio inicial de los fenómenos hasta la utilización de herramientas adaptadas a las necesidades del usuario, para la experimentación virtual y la posible toma de decisiones anticipando las posibles consecuencias de los potenciales escenarios estudiados.

El **Capítulo 5** se enfoca en la novedosa aparición de los *simple kernel P systems*, una variante de sistemas celulares que pretende actuar como modelo integrador, aglutinador de diferentes elementos de diversas variantes de sistemas P que ya habían sido diseñadas y utilizadas con anterioridad. El objetivo de esa variante consiste en proporcionar un único formalismo capaz de simular la acción de la mayor cantidad de variantes de sistemas P anteriores. De esta manera, será posible realizar determinados tratamientos y estudios sobre los modelos, como la verificación de propiedades sobre los mismos, de manera uniforme, sin necesidad de desarrollar dichos mecanismos para cada variante de sistemas P. Además, los elementos introducidos en este modelo formal tratan de facilitar el diseño de soluciones basadas en sistemas P, en lugar de muchos otros formalismos destinados a disponer de un conjunto mínimo de elementos capaz de resolver el mayor espectro posible de problemas con dichos ingredientes.

Los **capítulos 6 y 7** presentan sendas aplicaciones en el ámbito de los problemas de la vida real y de los problemas NP-completos. El **Capítulo 6** comienza ofreciendo una panorámica y un poco de historia sobre la simulación de modelos de sistemas dinámicos que se dan en la vida real, e ilustra el uso del entorno y la metodología desarrollados en el Capítulo 4 para la modelización, análisis y simulación de fenómenos interesantes que aparecen en la vida real; por ejemplo, la gestión de un ecosistema acuático afectado por la presencia de una especie exótica invasora. Este caso de estudio supone un gran reto dado que se trata de un sistema complejo con un gran número de procesos y elementos involucrados interrelacionados entre sí y, a la vez, permite mostrar

las capacidades proporcionadas por las herramientas anteriores. Finalmente, en este capítulo se detalla la aplicación de P-Lingua y MeCoSim como entorno vertebrador de la metodología propuesta, articulando las diversas fases de la misma.

El **Capítulo 7** se centra en el uso de la plataforma proporcionada para trabajar en la resolución de problemas NP-completos bien conocidos. Para ello, se parte de la representación de dichos problemas y su modelización mediante sistemas P, incluyendo su traducción a P-Lingua, su reconocimiento sintáctico, depuración y simulación e introspección del sistema paso a paso. Así se consigue proveer una aplicación visual de alto nivel basada en MeCoSim para la introducción de diversas entradas adaptadas para cada problema, de tal manera que sea reconocible por el usuario y se pueda trabajar en el problema obviando las interioridades de su solución, con el fin de obtener resultados visuales adecuados para el dominio de su problema.

Parte III: Resultados

La tercera parte de la memoria pone el punto (y seguido) a la labor desempeñada. El **Capítulo 8** comienza con las conclusiones extraídas del trabajo realizado y de las aportaciones comentadas en esta sección. Posteriormente se proponen las principales líneas de investigación en las que ya se ha empezado a trabajar para continuar la senda abierta por esta tesis, así como las principales ideas de líneas de trabajo futuro que se considera podrían aportar grandes avances en la búsqueda de soluciones prácticas a problemas relevantes en diversos ámbitos.

Finalmente, se presentan algunos apéndices adicionales que tratan de cumplir varios objetivos complementarios al núcleo del trabajo presentado. Por un lado, intentar hacer justicia con la cantidad de paquetes software de los que se ha hecho uso durante el desarrollo del trabajo, ilustrando la funcionalidad que han aportado al entorno de simulación. Por otro lado, presentar el sitio web desarrollado para ilustrar las funcionalidades de MeCoSim a través de páginas, documentos y vídeos de uso, con el fin de servir de repositorio de aplicaciones, modelos y escenarios para las soluciones basadas en sistemas P. El objetivo es facilitar a la comunidad de *Membrane Computing* el trabajo práctico de modelización y simulación con sistemas P, así como la provisión a los usuarios de aplicaciones adaptadas a cada problema real para los usuarios finales, a modo y manera que las herramientas anteriores han sido de gran utilidad para el desarrollo del entorno de simulación MeCoSim.

Aportaciones

Entre las aportaciones realizadas a lo largo del desarrollo del presente trabajo, queremos destacar las que se relacionan a continuación.

- Diseño de una metodología para el empleo de soluciones basadas en sistemas P como herramientas capaces de asistir en la búsqueda de mayor conocimiento y la gestión de los problemas de interés, involucrando los pasos necesarios desde el estudio inicial del problema hasta la disposición de la aplicación software adaptada al mismo, así como los procesos de modelización, depuración, introspección, parametrización y personalización de aplicaciones, simulación, visualización, análisis de datos, extracción de propiedades invariantes y verificación de las mismas.
- Desarrollo de un entorno de simulación capaz de proporcionar los mecanismos adecuados para dar respuesta a los objetivos planteados por la metodología anterior, tanto a los diseñadores de soluciones basadas en sistemas P como a los usuarios finales de las aplicaciones. Estos mecanismos desarrollados han debido proporcionar el entorno de simulación general, y herramientas que sean a la vez lo suficientemente generales y flexibles para que puedan tratar cualquier solución basada en sistemas P, como personalizables para poder adaptarse a las necesidades de los usuarios de la mayor diversidad posible de problemas.
- Diseño e implementación de un mecanismo de extensibilidad para poder dotar al entorno general de nuevos mecanismos de extensibilidad, que amplíen la funcionalidad general, tanto actual como prevista del entorno, a través de una arquitectura de plugins en la que poder incorporar programas basados tanto en tecnologías similares a las empleadas por MeCoSim como completamente externos, conectables a MeCoSim a partir de una simple configuración. De tal modo que podamos llamar a dichos programas externos desde nuestro entorno general a través del sistema operativo. Los mecanismos desarrollados van en la línea de uno de los principales objetivos de MeCoSim: la vertebración, estandarización y articulación del proceso de modelización y simulación aunando para ello cuantos más herramientas y esfuerzos mejor, haciendo bueno el principio holístico de que *el todo es mayor que la suma de las partes*, premisa con la que ya se presentara MeCoSim en el congreso BIC-TA 2010 en la República Popular China.

- Desarrollo e integración de plugins para la solución de requisitos específicos en respuesta a diversos problemas, como la introducción de fórmulas proposicionales de forma amigable para el usuario (**SAT plugin**), la visualización de grafos para problemas relacionados con este tipo de elementos (**Graphs plugin**), la extracción de invariantes mediante la conexión con el software Daikon (**Daikon plugin**), la simulación externa a través del software Spin (**Promela plugin**), o el proyecto piloto para el desarrollo de funcionalidades adicionales para **MeCoSim** codificadas en diversos lenguajes de programación (con una conexión inicial al compilador GHC tras introducir código Haskell) (**Languages Integration plugin**).
- Diseño de un lenguaje para *simple kernel P systems* en **P-Lingua**, capturando los elementos esenciales del marco y extendiendo **P-Lingua** para incorporar los elementos necesarios.
- Desarrollo de herramientas para el reconocimiento léxico-sintáctico y desarrollo de un simulador en el framework de **P-Lingua**, con el fin de reconocer, depurar y simular soluciones basadas en *simple kernel P systems*.
- Aplicación de la metodología y las herramientas desarrolladas a un buen número de problemas de la vida real (aves carroñeras, rebeco, alianzas de plantas, tritones, anfibios, peces, mejillón cebra, pandemias, redes genéticas y producción animal porcina) y a un conjunto de problemas **NP**-completos (**SAT**, **3-COL**, **Subset Sum**, **Partition**, **Vertex Cover** y **Broadcasting**).
- Desarrollo de un sitio web para la descarga e instalación de **MeCoSim** y difusión de la información documental y audiovisual para el empleo de las funcionalidades básicas proporcionadas, así como para la centralización de repositorios de aplicaciones, modelos y escenarios correspondientes a soluciones basadas en sistemas P.

Publicaciones

Algunos de los principales resultados más relevantes del trabajo presentado en esta memoria han sido publicados en las revistas, capítulos de libros y actas de congresos que se enumeran a continuación.

Revistas indexadas

1. M.A. Colomer, A. Margalida, L. Valencia-Cabrera, A. Palau. Application of a computational model for complex fluvial ecosystems: the population dynamics of zebra mussel *Dreissena polymorpha* as a case study. *Eco-logical Complexity*, **20** (2014), 116-126. Factor de impacto: **2.000**
2. M. Gheorghe, F. Istrate, R. Lefticaru, M.J. Pérez-Jiménez, A. Turcanu, L. Valencia-Cabrera, M. García-Quismondo, L. Mierla. 3-COL problem modelling using simple Kernel P systems. *International Journal of Computer Mathematics*, **90**, 4 (2013), 816-830. Factor de impacto: **0.721**
3. D. Orellana-Martín, C. Graciani, L.F. Macías-Ramos, M.A. Martínez-del-Amor, A. Riscos-Núñez, Á. Romero-Jiménez, L. Valencia-Cabrera. Sevilla Carpets Revisited: Enriching the Membrane Computing Toolbox. *Fundamenta Informaticae*, **134** (2014), 153-166. Factor de impacto: **0.479**
4. I. Pérez-Hurtado, L. Valencia-Cabrera, J.M. Chacón, A. Riscos-Núñez, M.J. Pérez-Jiménez. A P-Lingua based simulator for tissue P systems with cell separation. *Romanian Journal of Information Science and Technology*, **17**, 1 (2014), 89-102. Factor de impacto: **0.453**
5. L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez, Y. Su, H. Yu, L. Pan. Modeling Logic Gene Networks by Means of Probabilistic Dynamic P Systems. *International Journal of Unconventional Computing*, **9**, 5-6 (2013), 445-464. Factor de impacto: **0.433**
6. M. García-Quismondo, L.F. Macías-Ramos, M.A. Martínez-del-Amor, I. Pérez-Hurtado, L. Valencia-Cabrera. Open Problems on Simulation of Membrane Computing Models. *International Journal of Foundations of Computer Science*, **24**, 5 (2013), 619–622. Factor de impacto: **0.326**

Revistas no indexadas

1. M.A. Colomer, S. Lavín, I. Marco, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, E. Serrano, L. Valencia-Cabrera. Modeling population growth of Pyrenean Chamois (*Rupicapra p. pyrenaica*) by using P systems. *Lecture Notes in Computer Science*, **6501** (2011), 144-159.
2. L.F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M.J. Pérez-Jiménez, A. Riscos-Núñez. A P-Lingua based simulator for Spiking Neural P systems. *Lecture Notes in Computer Science*, **7184** (2012), 257-281.
3. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, Á. Romero-Jiménez, C. Graciani, A. Riscos-Núñez, M.A. Colomer, M.J. Pérez-Jiménez. DCBA: Simulating population dynamics P systems with proportional objects distribution. *Lecture Notes in Computer Science*, **7762** (2013), 257-276.
4. L.F. Macías-Ramos, M.J. Pérez-Jiménez, A. Riscos-Núñez, M. Rius. The efficiency of tissue P systems with cell separation relies on the environment. *Lecture Notes in Computer Science*, **7762** (2013), 243-256.
5. F. Ipate, R. Lefticaru, L. Mierla, L. Valencia-Cabrera, H. Hang, G. Zhang, C. Dragomir, M.J. Pérez-Jiménez, M. Gheorghe. Kernel P systems: Applications and Implementations. *Advances in Intelligent Systems and Computing*, **212** (2013), 1081-1089.
6. M.J. Pérez-Jiménez, A. Riscos-Núñez, M. Rius, L. Valencia-Cabrera. The relevance of the environment on the efficiency of tissue P systems. *Lecture Notes in Computer Science*, **8340** (2014), 308-321.
7. L.F. Macías-Ramos, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, A. Riscos-Núñez, L. Valencia-Cabrera. The Role of the Direction in Tissue P Systems with Cell Separation. *Journal of Automata, Languages and Combinatorics*, **19**, 1-4 (2014), 185-199.

Capítulos de libros

1. M.A. Colomer, M. García-Quismondo, L.F. Macías-Ramos, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, L. Valencia-Cabrera. Membrane System-Based Models for Specifying Dynamical Population Systems. In P. Frisco, M. Gheorghe, M. J. Pérez-Jiménez (eds.) *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer, Series: Emergence, Complexity and Computation, Vol. 7, 2014, Chapter 4, pp. 97-132.

Actas de congresos

1. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez. MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems. In K. Li, Z. Tang, R. Li, A.K. Nagar, R. Thamburaj (eds.) *Proceedings 2010 IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010)*, IEEE Press, Volume 1, September 23-26, 2010, Changsha, China, pp. 637-643.
2. M.A. Colomer, C. Fondevilla, L. Valencia-Cabrera (2011). A New P System to Model the Subalpine and Alpine Plant Communities. *Proceedings of the Ninth Brainstorming Week on Membrane Computing*, Seville, Spain, January 31- February 4, 2011, Report RGNC 01/2010, Fénix Editora, 2011, pp. 91-112.
3. R. Lefticaru, F. Ipate, L. Valencia-Cabrera, A. Turcanu, C. Tudose, M. Gheorghe, M.J. Pérez-Jiménez, I.M. Niculescu, C. Dragomir. Towards an integrated approach for model simulation, property extraction and verification P systems. *Proceedings of the Tenth Brainstorming Week on Membrane Computing*, Volume I, Seville, Spain, January 30- February 3, 2012, Report RGNC 01/2012, Fénix Editora, 2012, pp. 291-318.
4. M. Gheorghe, F. Ipate, C. Dragomir, L. Mierla, L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez. Kernel P systems - Version I. *Proceedings of the Eleventh Brainstorming Week on Membrane Computing*, Seville, Spain, February 4-8, 2013, Report RGNC 01/2013, Fénix Editora, 2013, pp. 97-124.

5. L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez, Y. Su, H. Yu, L. Pan. Analising gene networks with PDP systems. *Arabidopsis thaliana*, a case study. *Proceedings of the Eleventh Brainstorming Week on Membrane Computing*, Seville, Spain, February 4-8, 2013, Report RGNC 01/2013, Fénix Editora, 2013, pp. 257-272.

Parte I

Preliminares

1

Computación Bioinspirada

La necesidad de resolver problemas concretos de la vida real ha obligado al hombre a buscar procedimientos sistemáticos que permitan obtener soluciones de los mismos a través de *métodos* que denominaremos *mecánicos*. Dichos procedimientos consisten, básicamente, en la realización de una serie de tareas “elementales” debidamente secuenciadas. De esta forma, toda entidad capaz de ejecutar de manera autónoma esas tareas, puede implementar/llevar a cabo ese tipo de procedimientos y, por tanto, será capaz de proporcionar soluciones de dichos problemas aunque no disponga, propiamente, de conocimiento alguno acerca de los mismos.

La irrupción de los ordenadores electrónicos (de propósito general) a mediados de la década de los cuarenta del siglo pasado, significó un avance cuantitativo en la resolución de problemas concretos, en tanto que se pudo dar respuesta a muchos problemas que, hasta ese momento, habían sido inabordables desde el punto de vista práctico. No obstante, en 1983, R. Churchhouse estableció la existencia de una limitación física para los procesadores electrónicos convencionales. Más concretamente, demostró la existencia de una cota para la velocidad y el tamaño que dichos procesadores podían alcanzar. Lo más grave radica en que esa cota es claramente insuficiente para poder implementar en ordenadores electrónicos soluciones mecánicas de problemas muy relevantes de la vida real. Dicho con otras palabras, existen muchos problemas importantes que jamás podrán ser resueltos usando dispositivos electrónicos.

Es entonces cuando el hombre se plantea la posibilidad de diseñar y construir nuevas máquinas cuyo soporte físico no sea, precisamente, el medio electrónico. ¿Porqué este medio es el *más adecuado* para poder implementar computaciones? En ese escenario aparece la Naturaleza viva como una fuente alternativa de inspiración para el diseño de nuevos paradigmas computacionales que algún día pudieran ser implementados en dispositivos no electrónicos.

La *Computación Natural* es una disciplina científica cuyo objetivo fundamental es el análisis, diseño y simulación de procesos dinámicos que tienen lugar en la Naturaleza viva desde hace billones de años y que son susceptibles de ser interpretados como procedimientos de cálculo. Se trata pues de describir modelos y estudiar técnicas computacionales inspiradas por la Naturaleza, tratando de comprender el mundo que nos rodea en términos de procesamiento de la información.

El objetivo fundamental de este capítulo es la presentación del marco formal en el que se desarrolla el presente trabajo, a saber, la *Computación Celular con Membranas (Membrane Computing)* y está estructurado como sigue.

En primer lugar se describen brevemente algunos paradigmas y modelos de Computación Natural. En la Sección 1.2 se introducen las primeras ideas y conceptos de la Computación Celular con Membranas. La Sección 1.3 está dedicada a la presentación de algunas variantes que van a tener un papel importante en esta memoria, relacionados con sistemas P que trabajan a modo de células o a modo de tejidos. La Sección 1.4 trata de justificar que el paradigma computacional considerado es adecuado para proporcionar soluciones eficientes, en cierto sentido, de problemas computacionalmente duros. Para ello, se introducirán los ingredientes necesarios que permiten desarrollar una Teoría de la Complejidad Computacional en *Membrane Computing*, y se presentarán soluciones de los problemas SAT y 3-COL en las variantes antes mencionadas. Esas soluciones nos permitirán estudiar las diferentes funcionalidades que aporta la plataforma **MeCoSim**, en conjunción con **P-Lingua**, para la depuración y el análisis de propiedades de esos diseños. La última sección está dedicada al análisis de diversos aspectos relacionados con aplicaciones software que permiten simular el comportamiento de los dispositivos computacionales celulares. Específicamente, se presentarán algunos elementos básicos del proyecto software **P-Lingua**, la estructura de una buena plataforma de simulación de sistemas P y, finalmente, un visión general de los simuladores paralelos desarrollados hasta el momento.

1.1. Computación Natural

Existen problemas abstractos que modelizan/representan problemas concretos relevantes de la vida real tales que cualquier solución mecánica (algorítmica) conocida en un modelo convencional de computación (funciones recursivas, funciones λ -calculables, máquinas de Turing, máquinas URM, modelo **GOTO**, etc.) requiere un alto coste para su ejecución, ya sea en tiempo y/o en espacio, siendo habitual que el intento de disminuir una de las dos medidas provoca un crecimiento exponencial en la otra.

Las limitaciones deducidas del resultado obtenido por R. Churchhouse oscurecen bastante el panorama a la hora de dar respuesta a dichos problemas si nos limitamos a emplear los dispositivos electrónicos disponibles en la actualidad. Por todo ello, surge la necesidad de buscar nuevos paradigmas computacionales que sean capaces de dar respuestas aceptables (en tiempo razonable) a esos problemas importantes. Esto se podría conseguir reduciendo ambos parámetros (tiempo/espacio) o, al menos, incluyendo procedimientos en los que un coste elevado en una de las medidas sea *asimilado*, en cierto sentido, por el propio modelo en beneficio de una reducción considerable sobre la otra, o bien permitiendo nuevos procesos que pudieran ser implementados de manera más eficiente usando máquinas cuyo soporte físico no sea el medio electrónico.

En este contexto, la búsqueda de nuevos modelos alternativos de computación está encaminada a la mejora cuantitativa de los resultados que proporciona la *teoría de la complejidad computacional*, en tanto que permita solucionar problemas concretos relevantes que son intratables usando dispositivos electrónicos como medio auxiliar.

La Computación Natural surge como una de las posibles alternativas a la computación que podríamos denominar convencional o clásica, en la búsqueda de nuevos paradigmas que puedan proporcionar vías novedosas para superar las limitaciones que poseen los modelos convencionales. La Computación Natural engloba un conjunto de modelos que tienen como característica común la simulación del modo en que la naturaleza actúa/opera sobre la materia (algunos investigadores extienden este concepto hasta abarcar modelos tales como la *computación cuántica*, que no se ajusta fielmente a la interpretación dada anteriormente). Esos modelos estudian la forma en que las diversas leyes de la naturaleza producen modificaciones en determinados sistemas complejos que van desde hábitats hasta conjuntos de moléculas, pasando por organismos vivos, y que pueden ser interpretadas como procesos de cálculo u operaciones sobre los elementos básicos que lo integran.

Entre las ramas de la Computación Natural, destacamos los *algoritmos genéticos* (o más en general, la *computación evolutiva*), las *redes neuronales artificiales*, la *computación molecular* y la *computación celular con membranas*.

Los algoritmos genéticos fueron introducidos por J. Holland [74] en 1975 y están inspirados en el proceso genético de los seres vivos a través del cual evolucionan. El ingrediente fundamental es el principio de selección natural que permite encontrar una *buena* solución a partir de una gran cantidad de posibles soluciones candidatas.

Las redes neuronales artificiales fueron introducidas por W.S. McCulloch y W. Pitts [94] en 1943 y están inspiradas en las interconexiones y en el funcionamiento de las neuronas en el cerebro. Originariamente, las redes neuronales artificiales surgieron como una simulación del sistema nervioso, formado por un conjunto de unidades conectadas entre sí, simulando a las *neuronas* de los sistemas nerviosos biológicos.

La computación molecular tiene como objetivo el uso de moléculas orgánicas (ADN, ARN, proteínas, etc.) como hardware biológico que permite realizar computaciones. Esta rama de la Computación Natural nace, propiamente, a finales de 1994 con los trabajos de L. Adleman [15], en particular con el primer experimento realizado en un laboratorio que permitió resolver una instancia concreta de un problema NP-completo a través de la manipulación de moléculas de ADN.

La computación celular con membranas fue introducida por Gh. Păun [116] en 1998 y está inspirada en la estructura y el funcionamiento de las células de los organismos vivos así como de sus interacciones en tejidos o en estructuras biológicas de orden superior, principalmente en lo que respecta a su capacidad para procesar y generar información. En octubre de 2003, el Institute for Scientific Information (ISI, USA) designó a la *computación celular con membranas* como *Fast Emerging Research Front* en el área de *Computer Science*.

Los algoritmos genéticos y las redes neuronales artificiales han sido implementadas en ordenadores electrónicos convencionales con el objetivo de mejorar la eficiencia de programas que son ejecutados sobre dichas máquinas. Por su parte, como se ha comentado anteriormente, la computación molecular basada en ADN ha sido implementada en medios bioquímicos (el experimento de L. Adleman permitió resolver una instancia concreta del problema del camino hamiltoniano, en su versión dirigida y con dos nodos distinguidos). En cambio, la computación celular con membranas aún no ha sido implementada ni en medios electrónicos ni bioquímicos, ni en ningún otro medio.

1.2. Computación Celular con Membranas

La célula es la unidad fundamental de todo organismo vivo. Posee una estructura compleja y, a la vez, muy organizada que permite la ejecución simultánea de un gran número de reacciones químicas. Existen dos tipos de células: las *procarióticas* (propias de ciertos organismos unicelulares, como las bacterias y las cianofíceas) que carecen de membrana nuclear y, por tanto, no hay separación entre núcleo y citoplasma, y las células *eucarióticas* (propias de los animales y plantas) cuyo núcleo está separado del citoplasma por una doble membrana. En ambos tipos de células se realizan de manera similar una serie de procesos que son fundamentales para la vida: (a) replicación del ADN; (b) producción de energía; (c) síntesis de proteínas; y (d) realización de procesos metabólicos.

En un primer análisis se pueden distinguir en una célula tres partes claramente diferenciadas: una especie de película muy fina (*membrana plasmática*) que delimita a la célula de su entorno; un corpúsculo central (*núcleo*), que contiene y almacena la información genética en moléculas de ADN y de ARN; y el *citoplasma*, que es la parte comprendida entre el núcleo y la membrana plasmática.

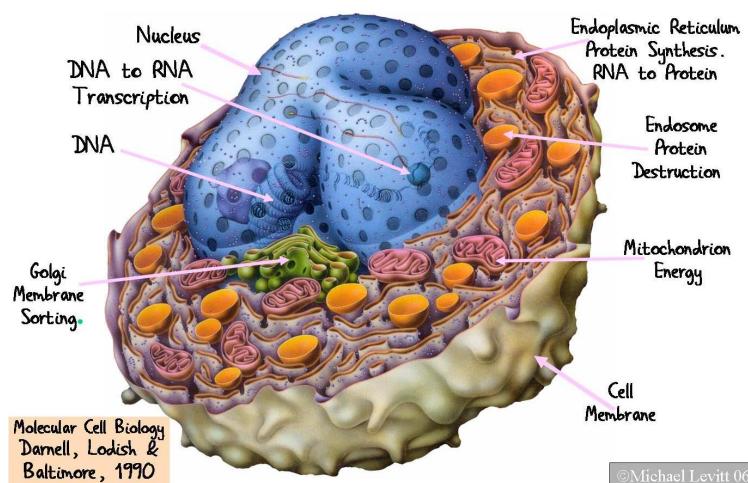


Figura 1.1: Una célula eucariota

El comportamiento de una célula puede ser considerado, en esencia, como el de una máquina, en tanto que realiza una serie de procesos dinámicos que,

en cierto sentido pueden ser interpretados como procedimientos de cálculo.

La Computación Celular con Membranas es un paradigma de computación inspirado en la estructura y el funcionamiento de las células de los organismos vivos. Los dispositivos computacionales de los modelos celulares se denominan genéricamente *sistemas P*. Los ingredientes sintácticos de los modelos básicos son los siguientes: (a) una *estructura de membranas* formalizada a través de un árbol enraizado, cuya raíz se denomina *piel* y cuyas hojas se denominan *membranas elementales*; (b) una serie de multiconjuntos finitos de objetos (abstracciones de las sustancias químicas) colocados en las distintas regiones o compartimentos que delimitan los nodos de la estructura; y (c) un conjunto finito de reglas de reescritura (abstracciones de las reacciones químicas) que permiten la evolución de los objetos en los compartimentos a lo largo del tiempo.

En un sistema P básico, la piel (la raíz del árbol que define la estructura de membranas) proporciona una entidad propia al sistema en tanto que posibilita distinguir el interior del exterior. Éste se denominará genéricamente *entorno* del sistema.

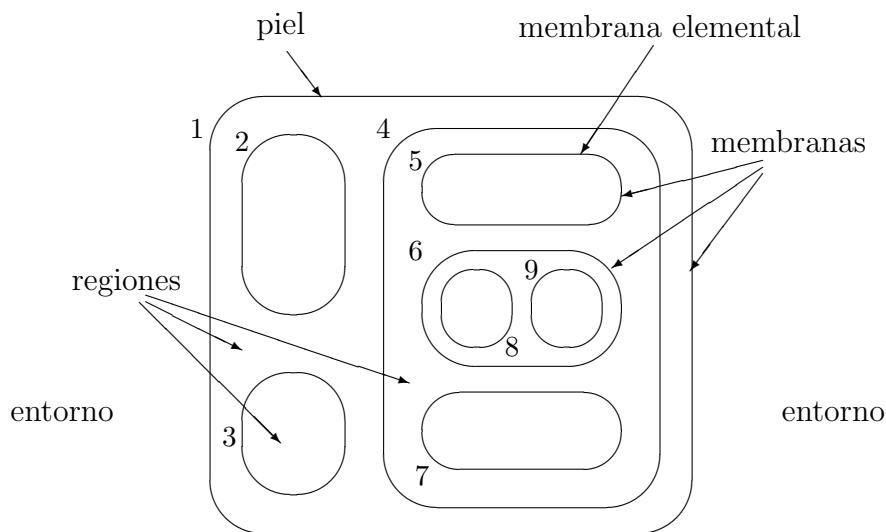


Figura 1.2: Una estructura de membranas

Los ingredientes semánticos de los modelos de computación celular con membranas se basan en el *paralelismo maximal* y el *no determinismo*. En la

dinámica de los sistemas P se aplica un doble paralelismo: el primero de ellos se encuentra a nivel de regiones o compartimentos ya que las reglas son aplicadas de forma simultánea en una región, y el otro se encuentra a nivel del sistema en general ya que todas las regiones evolucionan a la vez de forma concurrente en cada uno de los compartimentos que conforman el sistema. La condición de maximalidad significa que a la hora de realizar un paso de computación, se seleccionará un multiconjunto de reglas a aplicar al sistema de tal manera que tras su aplicación no quede ningún objeto por evolucionar que podría haber evolucionado (cualquier regla que se le añada al multiconjunto maximal de reglas, daría un multiconjunto de reglas que no sería globalmente aplicable). En lo que respecta al no determinismo, a la hora de evolucionar el sistema pueden existir distintos objetos y diferentes reglas que pueden ser usadas. Pues bien, los objetos a evolucionar y las reglas por las cuales evolucionan dichos objetos se elegirían de forma no determinista.

1.3. Variantes de Sistemas P

En esta sección, se van a presentar algunas variantes de modelos de computación celular con membranas que serán usados a lo largo de este trabajo. Específicamente, se introducirán dos sistemas P que trabajan a modo de céluas (sistemas P de transición y sistemas P con membranas activas) y dos variantes que trabajan a modo de tejidos (sistemas P de tejidos con división celular y sistemas P de tejidos con separación celular).

1.3.1. Sistemas P de transición

Vamos a comenzar presentando los *sistemas P de transición* que nos permitirán ilustrar algunos conceptos aludidos en la sección anterior. Estos sistemas fueron, precisamente, los que introdujo Gh. Păun en el primer artículo relativo a la computación celular con membranas [116].

En primer lugar, introduciremos algunos preliminares básicos. Un *alfabeto*, Γ , es un conjunto no vacío y sus elementos se denominan *símbolos*. Una sucesión finita de símbolos de Γ (es decir, una aplicación de un número natural en Γ) es una *cadena* o *palabra* de Γ . El conjunto de todas las cadenas sobre el alfabeto Γ se denota por Γ^* . Un *lenguaje* sobre Γ es un subconjunto de Γ^* ; es decir, un *lenguaje* sobre Γ es un conjunto de palabras sobre Γ .

Un *multiconjunto* m sobre un alfabeto Γ es un par (Γ, f) donde f es una función de Γ en el conjunto \mathbb{N} de los números naturales. Si $m = (\Gamma, f)$ es un multiconjunto, entonces se define el *soporte* de m como sigue: $supp(m) = \{x \in \Gamma \mid f(x) > 0\}$. Si $x \in supp(m)$, entonces diremos que x pertenece al multiconjunto m y escribiremos $x \in m$. Se dice que un multiconjunto es finito si su soporte es un conjunto finito. Si $m = (\Gamma, f)$ es un multiconjunto finito sobre Γ y $supp(m) = \{a_1, \dots, a_k\}$, entonces se notará $m = a_1^{f(a_1)} \dots a_k^{f(a_k)}$ (el orden aquí es irrelevante), y diremos que $f(a_1) + \dots + f(a_k)$ es el cardinal de m , denotado por $|m|$. El multiconjunto vacío (todos los elementos tienen multiplicidad 0) se denota por \emptyset o λ . Notaremos por $M_f(\Gamma)$ el conjunto de todos los multiconjuntos finitos sobre Γ . Es interesante observar que, de acuerdo con lo anterior, todo multiconjunto finito sobre Γ puede ser representado mediante una cadena sobre Γ ; no obstante, varias cadenas diferentes pueden representar el mismo multiconjunto.

Si $m_1 = (\Gamma, f_1)$, $m_2 = (\Gamma, f_2)$ son multiconjuntos sobre Γ , entonces definimos la *unión* de m_1 y m_2 como $m_1 + m_2 = (\Gamma, g)$, donde $g = f_1 + f_2$; es decir, $g(x) = f_1(x) + f_2(x)$, para cada $x \in \Gamma$. Diremos que m_1 está *contenido* en m_2 , y escribiremos $m_1 \subseteq m_2$, si se verifica la siguiente condición: $m_1(x) \leq m_2(x)$, para cada $x \in \Gamma$. A partir de aquí se extiende de manera natural las relaciones binarias del conjunto $Rel = \{\leq, \geq, =, \neq, >, <\}$ a relaciones binarias entre multiconjuntos de un alfabeto (por ejemplo, $m_1 \leq m_2$ si y sólo si $m_1 \subseteq m_2$).

Una vez introducidos estos conceptos preliminares, vamos a definir los sistemas P de transición, siguiendo las pautas marcadas por Gh. Păun en el trabajo que daría origen a la disciplina *Membrane Computing* (ver [116] para más detalles).

Definición 1.1. *Un sistema P de transición de grado $q \geq 1$ es una tupla*

$$\Pi = (\Gamma, \mu_\Pi, \mathcal{M}_1, \dots, \mathcal{M}_q, (R_1, \rho_1), \dots, (R_q, \rho_q), i_{out})$$

en donde:

- Γ es un alfabeto finito denominado *alfabeto de trabajo*.
- μ_Π es una estructura de membranas (un árbol enraizado) que consta de q membranas etiquetadas de forma única desde 1 hasta q .
- Para cada i , $1 \leq i \leq q$, \mathcal{M}_i es un multiconjunto finito de objetos (símbolos del alfabeto) sobre Γ asociado a la membrana i , representando el contenido inicial de esa membrana.

- Para cada i , $1 \leq i \leq q$, R_i es un conjunto finito de reglas sobre Γ asociado a la membrana i . Una regla sobre Γ es un par (u, v) , que se suele denotar por $u \rightarrow v$, en donde u es un multiconjunto finito sobre Γ y $v = v'$ o $v = v'\delta$, siendo v' un multiconjunto finito sobre $\Gamma \times (\{\text{here}, \text{out}\} \cup \{in_j : 1 \leq j \leq q\})$ y $\delta \notin \Gamma$ es un símbolo especial.
- Para cada i , $1 \leq i \leq q$, ρ_i es un orden parcial estricto sobre R_i .
- i_{out} , $1 \leq i_{out} \leq q$, representa la membrana de salida del sistema.

Una *configuración* o *descripción instantánea* de un sistema P de transición en un instante t consta de una estructura de membranas y una familia de multiconjuntos de objetos asociados a cada membrana existente en ese instante. En la descripción de Π , la configuración $(\mu_\Pi, \mathcal{M}_1, \dots, \mathcal{M}_q)$ se denomina *configuración inicial*.

Diremos que una regla $r \equiv u \rightarrow v$ asociada a una membrana i es *aplicable* a una configuración \mathcal{C} de Π en un instante t si se satisfacen las condiciones siguientes: (a) la membrana i aparece en la estructura de membranas asociada a dicha configuración; (b) el multiconjunto u está contenido en el multiconjunto de objetos asociado a la membrana i en \mathcal{C} ; (c) Si $(v_1, in_j) \in v$, entonces la membrana j debe ser hija de la membrana i ; (d) si $v = v'\delta$, entonces la membrana i no puede ser la raíz del árbol ni la membrana de salida i_{out} ; y (e) no existe ninguna regla r' de R_i que sea aplicable a esa configuración y tenga *mayor* prioridad; es decir, tal que $(r, r') \in \rho_i$.

Para definir cómo se pasa de una configuración a otra en un paso de transición hay que precisar cómo se aplican las reglas a las configuraciones. Sea $r \equiv u \rightarrow v$ una regla asociada a una membrana i que es aplicable a una configuración \mathcal{C} . La ejecución de r se realiza como sigue: (a) el multiconjunto u se elimina de la membrana i ; (b) para cada $(a, out) \in v$, un objeto $a \in \Gamma$ se incluye en la membrana padre de i (si i es la raíz o membrana piel, entonces el objeto a se envía al entorno); (c) para cada $(a, here) \in v$, un objeto $a \in \Gamma$ se añade a la membrana i ; (d) para cada $(a, in_j) \in v$, un objeto $a \in \Gamma$ se introduce en la membrana hija etiquetada por j ; y (e) si $\delta \in v$, entonces la membrana i se disuelve y su contenido pasará a la membrana que sea el primer antecesor no disuelto en ese paso.

Las reglas del sistema serán aplicadas de manera *no determinista, paralela y maximal*; es decir, las reglas que se van a ejecutar y los objetos involucrados en dichas reglas, serán elegidos de tal manera que en cada paso de computación

y tras la aplicación de esas reglas, no pueden quedar objetos por evolucionar que *puedan* hacerlo mediante la aplicación de alguna regla.

Por otra parte, es interesante hacer notar que las *relaciones de prioridad* dadas sobre las reglas, se interpretan en sentido fuerte, como sigue: si la regla r_1 tiene mayor prioridad que la regla r_2 (es decir, si $(r_2, r_1) \in \rho_i$, para un cierto i) y se puede aplicar r_1 a una cierta configuración, entonces no se podrá aplicar r_2 a esa configuración, aunque fuera aplicable.

Diremos que una configuración \mathcal{C} es de *parada* si no existe ninguna regla del sistema que sea aplicable a dicha configuración. Dadas dos configuraciones \mathcal{C} y \mathcal{C}' del sistema Π , diremos que \mathcal{C}' se obtiene de \mathcal{C} tras la ejecución de un *paso de computación* o *transición*, y notaremos $\mathcal{C} \Rightarrow_{\Pi} \mathcal{C}'$, si la configuración \mathcal{C}' se obtiene a partir de la configuración \mathcal{C} al aplicar un multiconjunto maximal de reglas. Por tanto, el modelo de computación resultante es no determinista, en tanto que para una configuración \mathcal{C} pueden existir dos configuraciones distintas $\mathcal{C}', \mathcal{C}''$ tales que $\mathcal{C} \Rightarrow_{\Pi} \mathcal{C}'$ y $\mathcal{C} \Rightarrow_{\Pi} \mathcal{C}''$; es decir, la configuración \mathcal{C} poseería más de una configuración siguiente.

Una *computación* en un sistema P es una sucesión (finita o infinita) de configuraciones tal que: (a) el primer término de la sucesión es una configuración inicial del sistema; (b) para cada $n \geq 2$, el n -ésimo término de la sucesión se obtiene a partir del anterior mediante la ejecución de un paso de computación; y (c) si la sucesión es finita (en cuyo caso se denominará *computación de parada*), entonces el último término de la sucesión es una configuración de parada. Sólo las computaciones de parada proporcionan un resultado, el cual estará codificado en la membrana de salida (por ejemplo, a través del multiconjunto de objetos de la membrana de salida de la correspondiente configuración de parada).

Notación: Si $\mathcal{C} = \{\mathcal{C}_t\}_{t \leq r}$ ($r \in \mathbb{N}$) es una computación de parada de Π , entonces diremos que la longitud de \mathcal{C} es r y la notaremos por $|\mathcal{C}|$.

En una versión posterior (*sistemas P con salida externa* [118]) se contempló la posibilidad de que la salida o respuesta del sistema a través de una computación estuviera codificada en el entorno; es decir, se permitía que la región de salida fuera el entorno del sistema (en este caso, se escribirá $i_{out} = 0$).

Así pues, una computación en un sistema P parte de una configuración inicial y mediante sucesivas transiciones, o bien se llega a una situación/configuración de parada (en tanto que el sistema no puede seguir evolucionando), o bien, el sistema nunca se detiene (en tanto que sigue evolucionando de manera indefinida).

Admitiremos que toda computación ejecuta un proceso sincronizado; es decir, se supone que hay una especie de *reloj universal* que marca las actuaciones de todos los elementos que integran el sistema.

Como se ha comentado anteriormente, hay que destacar la existencia de dos niveles de paralelismo en la ejecución: por una parte, las reglas asociadas a una membrana son aplicadas de manera simultánea dentro de esa membrana y, por otra, estas operaciones son realizadas en el mismo instante en todas las membranas que vertebran el sistema. Es decir, en un paso de reloj se ejecutarán todas las reglas que pueden ser aplicadas (de manera maximal) y en todas las membranas.

Los dispositivos que proporcionan la computación celular con membranas constituyen un modelo de computación que conjuga la elegancia y sencillez del modelo, con su proximidad a la realidad de la célula, desde un punto de vista biológico, abstrayendo de la estructura y funcionamiento de las células una serie de ideas computacionalmente relevantes.

Son modelos matemáticos con propiedades atractivas desde el punto de vista computacional: algunos son *completos* en el sentido de tener la misma potencia computacional que las máquinas de Turing, y otros son *eficientes* en tanto que son capaces de proporcionar soluciones “eficientes” (en cierto sentido, polinomiales) de problemas computacionalmente duros, en particular de problemas **NP**-completos.

Aunque los sistemas que se definen en la computación celular con membranas tienen una inspiración biológica, se debe resaltar el hecho de que constituyen igualmente un buen modelo teórico de computación distribuida, en donde distintas unidades de cálculo trabajan independientemente pero estructuradas en una cierta jerarquía vertical. Por ejemplo, la jerarquización usada en las conexiones establecidas en redes de ordenadores como Internet puede ser representada como una estructura de membranas, en la que los nodos de la red se interpretan como las membranas que forman el sistema, y el flujo de información entre nodos conectados se interpreta como las componentes químicas que dichas membranas generan e intercambian.

Asimismo, muchos sistemas dinámicos complejos como, por ejemplo, aquellos que surgen en el estudio de la dinámica de poblaciones, pueden ser también interpretados como “sistemas celulares” en los que los individuos de las distintas especies interaccionan entre sí dentro de hábitats que permiten el traspaso de los mismos.

1.3.2. Sistemas P con membranas activas

A continuación, en el marco de la computación celular con membranas se introduce una variante que va a posibilitar la construcción de una cantidad exponencial de espacio (expresado en términos del número de objetos y el número de membranas) en un tiempo polinomial.

Esta variante determinará un modelo computacional eficiente, en el sentido de ser capaz de proporcionar soluciones de problemas **NP**-completos [160] cuya ejecución requiere un número polinomial de pasos de computación, si bien se necesita el uso de una cantidad exponencial de espacio.

El ingrediente distintivo y diferenciador de esta variante radica en una regla específica de *división celular* que está inspirada en el proceso de *mitosis*. Recordemos que la mitosis es un proceso de división celular que da como resultado la producción de dos células hijas a partir de una sola célula original.

Este proceso ha sido formalizado en el marco de los sistemas celulares con membranas a través de una regla de división dando lugar a una variante denominada *sistema P con membranas activas*.

Definición 1.2. *Un sistema P con membranas activas de grado q ≥ 1 es una tupla $\Pi = (\Gamma, H, \mu_\Pi, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$, en donde:*

- Γ es un alfabeto finito, denominado alfabeto de trabajo.
- H es un conjunto finito, denominado alfabeto de etiquetas, tal que $\Gamma \cap H = \emptyset$.
- μ_Π es un árbol enraizado cuyos nodos (membranas) están etiquetados con elementos de $H \times \{0, +, -\}$. Si una membrana está etiquetada por $(h, \alpha) \in H \times \{0, +, -\}$, entonces notaremos $[]_h^\alpha$ y diremos que esa membrana tiene etiqueta h y carga eléctrica α . Inicialmente todas las membranas tienen asociadas carga neutra (0) y, además, membranas distintas tienen asociadas etiquetas distintas.
- $\mathcal{M}_1, \dots, \mathcal{M}_q$ son multiconjuntos finitos sobre el alfabeto Γ .
- \mathcal{R} es un conjunto finito de reglas de los tipos siguientes:
 - (a) $[a \rightarrow v]_h^\alpha$, donde $a \in \Gamma$, $v \in M_f(\Gamma)$, $\alpha \in \{0, +, -\}$, $h \in H$ (regla de evolución).

- (b) $[a]_h^\alpha \rightarrow b []_h^\beta$, donde $a, b \in \Gamma$, $\alpha, \beta \in \{0, +, -\}$, $h \in H$ (regla de comunicación hacia fuera).
- (c) $a []_h^\alpha \rightarrow [b]_h^\beta$, donde $a, b \in \Gamma$, $\alpha, \beta \in \{0, +, -\}$, $h \in H$, h no es la etiqueta de la raíz del árbol (regla de comunicación hacia dentro).
- (d) $[a]_h^\alpha \rightarrow b$, donde $a, b \in \Gamma$, $\alpha \in \{0, +, -\}$, $h \in H \setminus \{i_{out}\}$, h no es la etiqueta de la raíz del árbol (regla de disolución).
- (e) $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$, donde $a, b, c \in \Gamma$, $\alpha, \beta, \gamma \in \{0, +, -\}$, $h \in H \setminus \{i_{out}\}$, h no es la etiqueta de la raíz del árbol (regla de división).
- $i_{out} \in H \cup \{\text{env}\}$, donde $\text{env} \notin H \cup \Gamma$ es un símbolo que etiqueta el entorno del sistema.

Un sistema P con membranas activas $\Pi = (\Gamma, H, \mu_\Pi, \mathcal{M}_1, \dots, \mathcal{M}_p, \mathcal{R}, i_{out})$ de grado $q \geq 1$ puede ser considerado como un conjunto de q membranas dispuestas según una estructura de árbol enraizado, de tal manera que cada nodo (membrana) tiene asociado una etiqueta (un elemento de H) y una carga eléctrica (positiva (+), negativa (-), o neutra (0)).

Obsérvese que, a diferencia de los sistemas P de transición, las reglas de un sistema P con membranas activas están asociadas a etiquetas (elementos de H) y no a membranas.

Además, las reglas del tipo (e) permiten la existencia de distintas membranas en el sistema con la misma etiqueta a lo largo de la ejecución.

Seguidamente se especifica cómo se aplican las reglas en un sistema P con membranas activas.

- (a) *Reglas de evolución:* la regla $[a \rightarrow v]_h^\alpha$, será aplicable a toda membrana h que tenga carga eléctrica α y contenga el objeto a . Su aplicación no altera ni la etiqueta ni la carga de la membrana en la que se aplica. Su ejecución produce la sustitución de un objeto a por un multiconjunto v de objetos.
- (b) *Reglas de comunicación hacia fuera:* la regla $[a]_h^\alpha \rightarrow b []_h^\beta$ será aplicable a toda membrana h que tenga carga eléctrica α y contenga el objeto a . Su aplicación no altera la etiqueta de la membrana pero, en cambio, la carga pasa a ser β (que puede coincidir con α). Además, su ejecución produce la eliminación de un objeto a de la membrana h y la aparición de un objeto b en la membrana padre de h (si h es la membrana piel, el objeto b pasaría al entorno del sistema).

- (c) *Reglas de comunicación hacia dentro:* la regla $a []_h^\alpha \rightarrow [b]_h^\beta$ será aplicable a toda membrana h , distinta de la membrana piel, que tenga carga eléctrica α y tal que el objeto a aparece en la membrana padre de h . Su aplicación no altera la etiqueta de la membrana h pero, en cambio, la carga pasa a ser β (que puede ser α). Además, su ejecución produce la eliminación de un objeto a de la membrana padre de h y la aparición de un objeto b en la membrana h .
- (d) *Reglas de disolución:* la regla $[a]_h^\alpha \rightarrow b$, será aplicable a toda membrana h , distinta de la piel y de la membrana de salida (en su caso), que tenga carga eléctrica α y contenga al objeto a . Su aplicación provoca la eliminación de la membrana h del sistema y su contenido pasará a la membrana que sea el primer antecesor de h no disuelto.
- (e) *Reglas de división:* la regla $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_l^\gamma$ será aplicable a toda membrana h que sea elemental y distinta de la piel y de la membrana de salida (en su caso), que tenga carga eléctrica α y contenga el objeto a . Su aplicación produce la creación de dos membranas con la misma etiqueta, la primera de ellas con carga β y la segunda con carga γ . El objeto a que dispara la regla desaparece y, en su lugar, aparece b en la primera de las membranas creadas y c en la segunda de esas membranas. Los restantes objetos que aparecen en la membrana donde se ha aplicado la regla, son replicados en las dos membranas creadas.

Las reglas del sistema se aplicarán de tal forma que:

1. En cada paso de computación, a una membrana sólo se le puede aplicar, *a lo sumo*, una regla de entre las que no son del tipo (a).
2. Una regla de evolución se puede ejecutar sobre una membrana simultáneamente y de manera maximal, con reglas del tipo (b), (c), (d) o (e).

Los conceptos de configuración, paso de computación (o de transición) de una configuración a otra y computación, se definen en el marco de los sistemas P con membranas activas de forma similar al de los sistemas P de transición. Todas las computaciones comienzan con una configuración inicial y evolucionan de acuerdo con lo indicado anteriormente. Únicamente las *computaciones de parada* proporcionan un resultado que estará codificado por los objetos presentes en una cierta región de salida en la *configuración de parada*. Esa región puede ser una membrana ordinaria o bien el entorno.

1.3.3. Sistemas P que trabajan a modo de tejidos

Los sistemas P estudiados hasta ahora poseen una estructura de membranas jerarquizadas a través de un árbol enraizado, por ello se dice que este tipo de sistemas trabaja *a modo de células* (*cell-like*).

Además de tales sistemas, surgen otros que tratan de extender la idea anterior para poder describir el modo en que operan los tejidos como conjunto de células independientes que actúan de manera cooperativa a través de una serie de interacciones entre ellas: son los sistemas P de tejidos (*tissue-like*).

En este nuevo contexto computacional de sistemas P existirán una serie de unidades básicas o fundamentales que denominaremos *células* y, como en el caso *cell-like*, un elemento singular externo al propio sistema que denominaremos *entorno*.

Las células del sistema estarán identificadas a través de una etiqueta y se pueden comunicar entre sí o con el entorno, a través de ciertas reglas de comunicación del tipo *symport/antiport*.

Además, en estos sistemas pueden existir dos tipos diferentes de reglas que permiten la creación de dos células nuevas a partir de una célula inicial. Se tratan de reglas que implementan, por una parte, la división celular (reglas de división) y, por otra, la fisión celular (reglas de separación).

Definición 1.3. *Un sistema de tejidos de grado $q \geq 1$ es una tupla*

$$\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$$

tal que:

1. Γ es un alfabeto finito.
2. $\mathcal{E} \subseteq \Gamma$.
3. $\mathcal{M}_1, \dots, \mathcal{M}_q$ son multiconjuntos sobre Γ .
4. \mathcal{R} es un conjunto finito de reglas del siguiente tipo:
 - (a) Reglas de Comunicación: $(i, u/v, j)$, para $i, j \in \{0, 1, 2, \dots, q\}$, $i \neq j$, $u, v \in M_f(\Gamma)$, $|u + v| \neq 0$;
5. $i_{out} \in \{0, 1, 2, \dots, q\}$.

Definición 1.4. Un sistema P de tejidos con división celular de grado $q \geq 1$ es una tupla $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$, tal que:

1. $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$ es un sistema P de tejidos.
2. \mathcal{R} es un conjunto finito de reglas del siguiente tipo:
 - (a) Reglas de Comunicación: $(i, u/v, j)$, para $i, j \in \{0, 1, 2, \dots, q\}$, $i \neq j$, $u, v \in M_f(\Gamma)$, $|u + v| \neq 0$;
 - (b) Reglas de División: $[a]_i \rightarrow [b]_i [c]_i$, donde $i \in \{1, 2, \dots, q\}$, $i \neq i_{out}$ y $a, b, c \in \Gamma$.

Definición 1.5. Un sistema de tejidos con separación celular de grado $q \geq 1$ es una tupla $\Pi = (\Gamma, \Gamma_1, \Gamma_2, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$, tal que:

1. $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$ es un sistema P de tejidos.
2. $\{\Gamma_1, \Gamma_2\}$ es una partición de Γ .
3. \mathcal{R} es un conjunto finito de reglas del siguiente tipo:
 - (a) Reglas de Comunicación: $(i, u/v, j)$, para $i, j \in \{0, 1, 2, \dots, q\}$, $i \neq j$, $u, v \in M_f(\Gamma)$, $|u + v| \neq 0$;
 - (b) Reglas de Separación: $[a]_i \rightarrow [\Gamma_1]_i [\Gamma_2]_i$, donde $i \in \{1, 2, \dots, q\}$, $i \neq i_{out}$ y $a \in \Gamma$.

Un sistema de tejidos $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$ de grado $q \geq 1$, puede ser considerado como un conjunto de q células, etiquetadas de 1 a q , con un entorno (etiquetado por 0) y de tal manera que:

- (a) $\mathcal{M}_1, \dots, \mathcal{M}_q$ son multiconjuntos finitos sobre Γ que representan los objetos (elementos del alfabeto de trabajo Γ) inicialmente colocados en las q células del sistema.
- (b) \mathcal{E} es el conjunto de objetos inicialmente localizados en el entorno del sistema, admitiremos que todos ellos aparecen en un *número arbitrario de copias*.
- (c) $i_{out} \in \{0, 1, 2, \dots, q\}$ representa una *región* distinguida que nos va a permitir codificar la salida del sistema. Usaremos el término *región i* ($0 \leq i \leq q$) para referirnos a la célula i , en el caso $1 \leq i \leq q$, o al entorno, en el caso $i = 0$. Téngase presente que en el caso en que la región de salida sea una célula del sistema, entonces dicha célula no puede ser “dividida” ni “separada”.

Una regla de comunicación $(i, u/v, j)$, para $i, j \in \{0, 1, 2, \dots, q\}, i \neq j$, $u, v \in M_f(\Gamma)$, es aplicable a todo par de regiones distintas i, j tales que la región i contiene el multiconjunto de objetos u y la región j contiene el multiconjunto de objetos v . La aplicación de dicha regla provoca que el multiconjunto de objetos u pase de la región i a la región j y, simultáneamente, el multiconjunto de objetos v pase de la región j a la región i .

Una regla de división $[a]_i \rightarrow [b]_i[c]_i$ es aplicable a una célula i tal que contiene el objeto a y no sea, en su caso, la célula de salida. La aplicación de dicha regla produce la creación de dos nuevas células con la misma etiqueta i , de tal manera que en la primera célula creada, el objeto a es reemplazado por b , y en la segunda, el objeto a se sustituye por c . Los restantes objetos existentes en la célula i , se **replican** y se copian en las dos células nuevas.

Una regla de separación $[a]_i \rightarrow [\Gamma_1]_i[\Gamma_2]_i$ es aplicable a una célula i tal que contiene el objeto a y no sea, en su caso, la célula de salida. La aplicación de dicha regla produce la creación de dos nuevas células con la misma etiqueta i , de manera que el objeto a se consume en esa célula y los restantes objetos de la célula se **distribuyen** entre las nuevas células: los objetos pertenecientes a Γ_1 van a la primera célula creada y los objetos pertenecientes a Γ_2 a la segunda.

Las reglas de un sistema P de tejidos se ejecutan, de manera paralela maximal no determinista, como es habitual en sistemas celulares con membranas. No obstante, en la aplicación de las reglas existe una restricción fundamental: si en un cierto paso, se aplica una regla de división (o, en su caso, una regla de separación) a una célula, entonces en ese paso sólo se aplicará esa regla a esa célula; es decir, se puede pensar que cuando se está aplicando una regla de división o de separación a una célula, entonces los “canales de comunicación” de la misma con otras células del sistema y con el entorno quedan cerrados.

Una *configuración* o *descripción instantánea* de un sistema P de tejidos en un instante está descrita por los multiconjuntos de objetos sobre el alfabeto Γ asociados a todas las células presentes en el sistema, así como el multiconjunto de objetos sobre $\Gamma \setminus \mathcal{E}$ asociados al entorno en ese momento. Obsérvese que inicialmente los elementos de \mathcal{E} aparecen en el entorno con multiplicidad infinita. Por tanto, en la configuración del sistema en un instante determinado, no produce efecto alguno en la descripción del entorno el hecho de que un elemento de \mathcal{E} sea introducido en el sistema a través de una célula, o bien sea enviado al entorno procedente de una célula. La *configuración inicial* del sistema $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$ es $(\mathcal{M}_1, \dots, \mathcal{M}_q; \emptyset)$. De manera similar a como se ha hecho en las secciones anteriores, se introduce el concepto de computación, utilizándose la misma notación que la indicada en la sección 1.2.

1.4. Resolución de problemas de decisión mediante sistemas celulares

En esta sección vamos a usar el término *sistema celular* para referirnos indistintamente a un sistema P de transición, un sistema P con membranas activas o a un sistema P de tejidos con división celular.

El objetivo de esta sección es introducir el concepto de resolubilidad eficiente de un problema de decisión a través de una familia de sistemas celulares.

Informalmente, un *problema abstracto* se dice que es *de decisión* si sólo admite como respuesta *Sí* o *No*. Formalmente, un problema de decisión, X , es un par ordenado (I_X, θ_X) , en donde I_X es un lenguaje sobre un alfabeto finito (cuyos elementos se denominan *instancias* del problema) y θ_X es una función booleana total sobre I_X . Así pues, para cada instancia $a \in I_X$ se tiene que o bien $\theta_X(a) = 1$, en cuyo caso diremos que la respuesta del problema es afirmativa para esa instancia, o bien $\theta_X(a) = 0$ en cuyo caso diremos que la respuesta del problema es negativa para esa instancia.

Todo problema de decisión X tiene asociado, de manera natural, un lenguaje L_X formado por todas aquellas instancias $a \in I_X$ tales que $\theta_X(a) = 1$; es decir, tales que la respuesta del problema es afirmativa. Y recíprocamente, dado un lenguaje L sobre un alfabeto finito Γ , existe un problema de decisión X_L tal que el lenguaje asociado al problema X_L es, precisamente, el lenguaje de partida L .

En un modelo de computación, los dispositivos reconocedores de lenguajes admiten como entrada una cadena del alfabeto sobre el que está definido el lenguaje. Entonces, el dispositivo siempre para y devuelve *Sí* en el caso en que la cadena de entrada pertenezca al lenguaje y devuelve *No* en caso contrario. En tal situación, diremos que el dispositivo computacional *reconoce* el lenguaje.

Pues bien, de acuerdo con lo indicado anteriormente, para resolver un problema de decisión en un modelo de computación basta considerar dispositivos del modelo que sean reconocedores de lenguajes. Entonces, se dirá que un problema es resoluble en un tal modelo de computación si es capaz de reconocer el lenguaje asociado al problema de decisión.

Los *sistemas celulares reconocedores* serán utilizados como marco formal para atacar la resolución de problemas de decisión computacionalmente duros de manera eficiente.

Definición 1.6. Diremos que un sistema celular con membranas Π es un sistema reconocedor si satisface las condiciones siguientes:

- El alfabeto de trabajo Γ del sistema contiene dos objetos distinguidos **yes** y **no**. Además, en sistemas P de tejidos, al menos una copia de cada uno de esos objetos ha de estar presente en alguno de los multiconjuntos iniciales del sistema.
- Existe un alfabeto (de entrada) Σ estrictamente contenido en Γ . En sistemas P de tejidos se verifica, además, que $\Sigma \cap \mathcal{E} = \emptyset$, siendo \mathcal{E} el alfabeto del entorno.
- Los multiconjuntos iniciales del sistema $\mathcal{M}_1, \dots, \mathcal{M}_q$ son multiconjuntos sobre el alfabeto $\Gamma \setminus \Sigma$.
- Existe una membrana de entrada (en el caso *cell-like*) o una célula de entrada (en el caso *tissue-like*) que notaremos por i_{in} .
- La región de salida será el entorno.
- Todas las computaciones paran.
- Si \mathcal{C} es una computación de Π , entonces al menos uno de los objetos **yes** o **no**, pero no ambos, debe haber sido enviado a la región de salida y, además, sólo en el último paso de la computación.

Para cada multiconjunto m sobre Σ , la *configuración del sistema Π con entrada m* se obtiene de la configuración inicial del sistema añadiendo al multiconjunto $\mathcal{M}_{i_{in}}$ el multiconjunto m ; es decir, el multiconjunto asociado a la región de entrada i_{in} es $\mathcal{M}_{i_{in}} + m$.

Si Π es un sistema celular reconocedor y \mathcal{C} es una computación de Π , entonces diremos que el resultado es **yes** (respectivamente, **no**) si este objeto aparece en el entorno asociado a la correspondiente configuración de parada y, en cambio, ni el objeto **yes** ni el objeto **no** aparecen en el entorno asociado a ninguna configuración anterior. Diremos que una configuración \mathcal{C} es de *aceptación* (respectivamente, de *rechazo*) si la respuesta es **yes**, afirmativa (respectivamente, **no**, negativa).

Notaremos por **AM** (respectivamente, **TDC**) la clase de todos los sistemas P reconocedores con membranas activas (respectivamente, sistemas P de tejidos reconocedores con división celular).

1.4.1. Clases de complejidad polinomial en sistemas celulares

Los sistemas celulares proporcionan dispositivos no deterministas y, en consecuencia, a la hora de estudiar la resolubilidad de problemas de decisión en este nuevo marco, será aconsejable establecer “distancias” con el concepto clásico de resolubilidad por máquinas de Turing no deterministas.

Recuérdese que a la hora de definir dicho concepto, la clave radica en qué se entiende formalmente por *aceptar una instancia* del problema de decisión. En el caso de máquinas de Turing no deterministas, una instancia de un problema de decisión es aceptada si y sólo si existe, al menos, una computación del sistema que es de aceptación para dicha instancia entrada. Obsérvese que según esta definición, una instancia de un problema de decisión puede ser aceptada incluso si existe alguna computación del sistema que tenga como entrada esa instancia y que, en cambio, no sea de parada. En cierto sentido, se puede decir que esta definición de aceptación por máquinas de Turing no deterministas, no captura el “verdadero concepto de algoritmo”, en tanto en cuanto no podríamos decidir “mecánicamente” la no aceptación de una instancia, en algunos casos.

En el caso de los sistemas celulares, vamos a tratar de capturar el “verdadero concepto de algoritmo” incluso trabajando con sistemas no deterministas. Para ello, exigiremos que una instancia de un problema será aceptada si y sólo si toda computación del sistema con entrada dicha instancia sea de aceptación.

Otra aclaración interesante a realizar antes de pasar a la definición de resolubilidad eficiente de problemas de decisión a través de sistemas celulares es la siguiente: un problema de decisión puede ser resuelto por una única máquina de Turing (determinista o no) ya que en la descripción de dichas máquinas existe un elemento (la cinta) con capacidad infinita. En cambio, si se tiene presente que en la descripción de los sistemas celulares todos los ingredientes son finitos, para resolver un problema de decisión que, en general, contendrá una cantidad infinita numerable de instancias, será necesario una sucesión infinita de sistemas celulares.

Definición 1.7. *Un problema de decisión $X = (I_X, \theta_X)$ es resoluble en tiempo polinomial por una familia $\Pi = \{\Pi(n) \mid n \in \mathbb{N}\}$ de sistemas celulares reconocedores si se verifican las siguientes propiedades:*

- *La familia Π es polinomialmente uniforme por máquinas de Turing; es decir, existe una máquina de Turing determinista que trabaja en tiempo polinomial y tal que construye el sistema $\Pi(n)$ a partir de $n \in \mathbb{N}$.*

- Existe un par de funciones (cod, s) sobre I_X que son computables en tiempo polinomial, de tal manera que satisfacen lo siguiente:
 - Para cada instancia $w \in I_X$, $s(w)$ es un número natural y $cod(w)$ es un multiconjunto de entrada del sistema $\Pi(s(w))$.
 - Para cada $n \in \mathbb{N}$, el conjunto $s^{-1}(n)$ es finito.
 - La familia Π es polinomialmente acotada con respecto a (X, cod, s) ; es decir, existe una función polinómica $p(n)$, tal que para cada instancia $w \in I_X$, toda computación de $\Pi(s(w))$ con entrada $cod(w)$ es de parada y realiza, a lo sumo, $p(|w|)$ pasos.
 - La familia Π es adecuada con respecto a (X, cod, s) ; es decir, para cada instancia $w \in I_X$, si existe una computación de aceptación de $\Pi(s(w))$ con entrada $cod(w)$, entonces $\theta_X(w) = 1$.
 - La familia Π es completa con respecto a (X, cod, s) ; es decir, para cada $w \in I_X$, si $\theta_X(w) = 1$, entonces toda computación de $\Pi(s(w))$ con entrada $cod(w)$ es de aceptación.

De la adecuación y completitud se deduce que todo sistema de la familia es *confluente*, en el siguiente sentido: cada computación de dicho sistema con el mismo multiconjunto de entrada, debe dar siempre la misma respuesta.

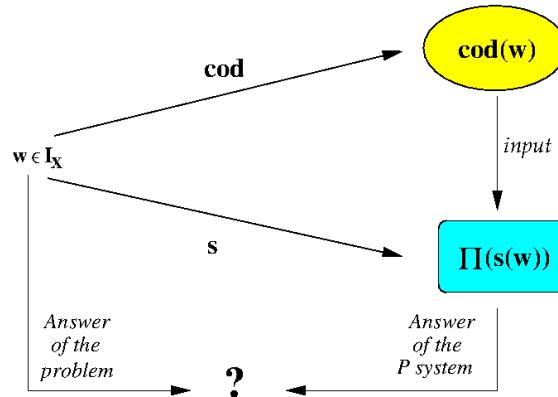


Figura 1.3: Resolución de un problema de decisión por sistemas celulares

Si \mathbf{R} una clase de sistemas celulares reconocedores, notaremos $\mathbf{PMC_R}$ al conjunto de problemas de decisión que pueden ser resueltos en tiempo polinomial por familias de sistemas de \mathbf{R} .

1.4.2. Una solución de SAT mediante sistemas P con membranas activas

A continuación vamos a presentar una solución eficiente del problema **SAT** de la satisfactibilidad de la lógica proposicional, a través de una familia de sistemas P reconocedores con membranas activas. El problema **SAT** se formula como sigue: “*dada una fórmula proposicional en forma normal conjuntiva, determinar si es satisfacible*”; es decir, determinar si existe, al menos, una valoración de verdad que hace verdadera la fórmula.

Pues bien, comencemos observando que la aplicación f de $\mathbb{N} \times \mathbb{N}$ en \mathbb{N} definida por $f(m, n) = \frac{(m+n) \cdot (m+n+1)}{2} + m$ es una función computable y biyectiva entre los conjuntos citados. Usualmente, notaremos $f(m, n) = \langle m, n \rangle$.

Para cada par de números naturales $m, n \in \mathbb{N}$ se considera el sistema P reconocedor con membranas activas $\Pi(\langle m, n \rangle) = (\Gamma, \Sigma, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in})$ de grado 2, definido como sigue:

- El alfabeto de entrada es $\Sigma = \{x_{i,j}, \bar{x}_{i,j} : 1 \leq i \leq m, 1 \leq j \leq n\}$.

- El alfabeto de trabajo es

$$\begin{aligned} \Gamma = \Sigma \cup \{c_k : 1 \leq k \leq m+2\} \cup \{d_k : 1 \leq k \leq 3n+2m+3\} \cup \\ \cup \{r_{i,k} : 0 \leq i \leq m, 1 \leq k \leq 2n\} \cup \{e, t\} \cup \{\text{Yes}, \text{No}\} \end{aligned}$$

- El conjunto de etiquetas es $\{1, 2\}$.

- La estructura inicial de membranas es $\mu = [\]_2]_1$.

- Los multiconjuntos iniciales son $\mathcal{M}_1 = \emptyset$ y $\mathcal{M}_2 = \{d_1\}$.

- La membrana de entrada es la etiquetada por 2.

- El conjunto \mathcal{R} consta de las siguientes reglas:

- (a) $\{[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n\}$.
- (b) $\{[x_{i,1} \rightarrow r_{i,1}]_2^+, [\bar{x}_{i,1} \rightarrow r_{i,1}]_2^- : 1 \leq i \leq m\};$
 $\{[x_{i,1} \rightarrow \lambda]_2^-, [\bar{x}_{i,1} \rightarrow \lambda]_2^+ : 1 \leq i \leq m\}.$
- (c) $\{[x_{i,j} \rightarrow x_{i,j-1}]_2^+, [x_{i,j} \rightarrow x_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\};$
 $\{[\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^+, [\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}.$
- (d) $\{[d_k]_2^+ \rightarrow [\]_2^0 d_k, [d_k]_2^- \rightarrow [\]_2^0 d_k : 1 \leq k \leq n\};$
 $\{d_k[\]_2^0 \rightarrow [d_{k+1}]_2^0 : 1 \leq k \leq n-1\}.$

- (e) $\{[r_{i,k} \rightarrow r_{i,k+1}]_2^0 : 1 \leq i \leq m, 1 \leq k \leq 2n - 1\}$.
- (f) $\{[d_k \rightarrow d_{k+1}]_1^0 : n \leq k \leq 3n - 3\};$
 $[d_{3n-2} \rightarrow d_{3n-1}e]_1^0$.
- (g) $e[]_2^0 \rightarrow [c_1]_2^+$; $[d_{3n-1} \rightarrow d_{3n}]_1^0$.
- (h) $\{[d_k \rightarrow d_{k+1}]_1^0 : 3n \leq k \leq 3n + 2m + 2\}$.
- (i) $[r_{1,2n}]_2^+ \rightarrow []_2^- r_{1,2n}$.
- (j) $\{[r_{i,2n} \rightarrow r_{i-1,2n}]_2^- : 1 \leq i \leq m\}$.
- (k) $r_{1,2n}[]_2^- \rightarrow [r_{0,2n}]_2^+$.
- (l) $\{[c_k \rightarrow c_{k+1}]_2^- : 1 \leq k \leq m\}$.
- (m) $[c_{m+1}]_2^+ \rightarrow []_2^+ c_{m+1}$.
- (n) $[c_{m+1} \rightarrow c_{m+2}t]_1^0$.
- (o) $[t]_1^0 \rightarrow []_1^+ t$.
- (p) $[c_{m+2}]_1^+ \rightarrow []_1^- Yes$.
- (q) $[d_{3n+2m+3}]_1^0 \rightarrow []_1^+ No$.

- $i_{in} = 2$; es decir, la membrana de entrada es la etiquetada por 2.

Sea $\varphi = C_1 \wedge \dots \wedge C_m$ una fórmula proposicional en forma normal conjuntiva tal que $Var(\varphi) = \{x_1, \dots, x_n\}$ y $C_i = y_{i,1} \vee \dots \vee y_{i,k_i}$, $1 \leq i \leq m$, donde $y_{i,j} \in \{x_j, \neg x_j : 1 \leq j \leq n\}$ son los literales de φ . Consideramos las funciones

$$\begin{aligned} cod(\varphi) &= \bigcup_{i=1}^m \{x_{i,j} : x_j \in C_i\} \cup \{\bar{x}_{i,j} : \neg x_j \in C_i\} \\ s(\varphi) &= \langle m, n \rangle = \frac{(m+n) \cdot (m+n+1)}{2} + m \end{aligned}$$

Obsérvese que la fórmula φ será procesada por el sistema $\Pi(s(\varphi))$ con multiconjunto de entrada $cod(\varphi)$, cuya ejecución trata de implementar un algoritmo de fuerza bruta estructurado en las siguientes fases:

- *Fase de generación y primer chequeo:* De manera simultánea, por una parte, se van generando todas las posibles valoraciones o asignaciones de verdad asociadas al conjunto de variables $\{x_1, \dots, x_n\}$ de la fórmula proposicional de entrada y, por otra, se van comprobando qué cláusulas van siendo satisfechas por esas valoraciones.

- *Fase de sincronización:* Debido a la simultaneidad en la generación y primer chequeo, es necesario determinar cuándo finaliza ambos procesos, a fin de comenzar el chequeo final para determinar si existe alguna valoración que hace verdadera todas las cláusulas.
- *Fase de segundo chequeo:* En cada membrana se comprueba cuántas cláusulas son satisfechas por la asignación de verdad codificada por ella.
- *Fase de salida:* El sistema proporcionará la salida, enviando al entorno el objeto correspondiente de acuerdo con el chequeo efectuado en la fase anterior.

La familia **Π** es polinomialmente uniforme ya que las reglas están descritas recursivamente y la cantidad de recursos usados para su construcción es:

- El número total de objetos del alfabeto de trabajo es $4mn + 3m + 5n + 9 \in \Theta(m \cdot n)$.
- El número de membranas iniciales es $2 \in \Theta(1)$.
- El máximo cardinal de los multiconjuntos iniciales es $1 \in \Theta(1)$.
- El número total de reglas es $6mn + 3m + 6n + 10 \in \Theta(m \cdot n)$.
- La máxima longitud de una regla es $9 \in \Theta(1)$.

La familia **Π** es polinomialmente acotada con respecto a (cod, s) ya que el tiempo de ejecución de $\Pi(s(\varphi))$ con entrada $cod(\varphi)$ es:

- $3n - 1$ pasos en la fase de generación.
- $2n$ pasos en la fase de sincronización.
- $2m$ pasos en la fase de chequeo.
- 4 pasos en la fase de salida.

Por tanto, el tiempo total de ejecución del sistema $\Pi(s(\varphi))$ con entrada $cod(\varphi)$ es $5n + 2m + 3 \in \Theta(n + m)$.

Para analizar los detalles de la demostración de la adecuación y completitud de la familia **Π** respecto del problema SAT, ver [124].

1.4.3. Una solución de 3-COL mediante sistemas P de tejidos con reglas de división

A continuación vamos a presentar una solución eficiente del problema 3-COL a través de una familia de sistemas P de tejidos reconocedores con división celular.

El problema 3-COL se formula como sigue: “*dado un grafo no dirigido, determinar si existe una coloración válida del grafo con tres colores*”. Recordemos que una coloración de un grafo no dirigido $\mathcal{G} = (V, E)$ con tres colores es una aplicación $f : V \rightarrow \{1, 2, 3\}$ de tal manera que para cada nodo x , $f(x)$ es el color asignado por la coloración f al nodo x . Se dice que una coloración f de \mathcal{G} es válida si satisface la siguiente condición: para cada arista $\{u, v\} \in E$ del grafo se tiene que $f(u) \neq f(v)$; es decir, una coloración válida asigna colores distintos a nodos del grafo que sean adyacentes.

Por cuestiones de simplicidad de notación en la solución que se va a presentar, usaremos los elementos del conjunto $\{R, G, B\}$ para designar los tres colores $\{1, 2, 3\}$ (R de “red”, G de “green” y B de “blue”).

Para cada $m, n \in \mathbb{N}$, consideramos el sistema P de tejidos reconocedor con división celular $\Pi(\langle n, m \rangle) = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in})$ definido como sigue:

- El alfabeto de trabajo Γ es el conjunto

$$\begin{aligned} & \{A_i, R_i, G_i, B_i, T_i, \overline{R}_i, \overline{G}_i, \overline{B}_i : 1 \leq i \leq n\} \cup \\ & \{a_i : 1 \leq i \leq 2n + m + \lceil \log m \rceil + 11\} \cup \{c_i : 1 \leq i \leq 2n + 1\} \cup \\ & \{d_i : 1 \leq i \leq \lceil \log m \rceil + 1\} \cup \{z_i : 2 \leq i \leq m + \lceil \log m \rceil + 6\} \cup \\ & \{A_{ij}, P_{ij}, \overline{P}_{ij}, R_{ij}, G_{ij}, B_{ij} : 1 \leq i < j \leq n\} \cup \{b, D, \overline{D}, e, T, S, N, \text{b, yes, no}\} \end{aligned}$$

- El alfabeto del entorno es $\mathcal{E} = \Gamma \setminus \{\text{yes, no}\} \setminus \Sigma$
- El alfabeto de entrada Σ es el conjunto $\{A_{ij} : 1 \leq i < j \leq n\}$
- Los multiconjuntos iniciales contenidos en las células son: $\mathcal{M}_1 = \{a_1, b, c_1, \text{yes, no}\}$ y $\mathcal{M}_2(n) = \{D, A_1, \dots, A_n\}$
- El conjunto \mathcal{R} consta de las siguientes reglas:
 - (a) $\{[A_i]_2 \rightarrow [R_i]_2[T_i]_2, [T_i]_2 \rightarrow [G_i]_2[B_i]_2 : 1 \leq i \leq n\}$
 - (b) $\{(1, a_i/a_{i+1}, 0) : 1 \leq i \leq 2n + m + \lceil \log m \rceil + 10\}$
 - (c) $\{(1, c_i/c_{i+1}^2, 0) : 1 \leq i \leq 2n\}$

- (d) $(1, c_{2n+1}/D, 2)$
- (e) $(2, c_{2n+1}/d_1 \bar{D}, 0)$
- (f) $\{(2, d_i/d_{i+1}^2, 0) : 1 \leq i \leq \lceil \log m \rceil\}$
- (g) $(2, \bar{D}/e z_2, 0)$
- (h) $\{(2, z_i/z_{i+1}, 0) : 1 \leq i \leq m + \lceil \log m \rceil + 5\}$
- (i) $\{(2, d_{\lceil \log m \rceil + 1} A_{ij}/P_{ij}, 0) : 1 \leq i < j \leq n\}$
- (j) $\{(2, P_{ij}/R_{ij} \bar{P}_{ij}, 0) : 1 \leq i < j \leq n\}$
- (k) $\{(2, \bar{P}_{ij}/B_{ij} G_{ij}, 0) : 1 \leq i < j \leq n\}$
- (l) $\{(2, R_i R_{ij}/R_i \bar{R}_j, 0) : 1 \leq i < j \leq n\}$
- (m) $\{(2, B_i B_{ij}/B_i \bar{B}_j, 0) : 1 \leq i < j \leq n\}$
- (n) $\{(2, G_i G_{ij}/G_i \bar{G}_j, 0) : 1 \leq i < j \leq n\}$
- (o) $\{(2, \bar{R}_j R_j/\flat, 0) : 1 \leq j \leq n\}$
- (p) $\{(2, \bar{B}_j B_j/\flat, 0) : 1 \leq j \leq n\}$
- (q) $\{(2, \bar{G}_j G_j/\flat, 0) : 1 \leq j \leq n\}$
- (r) $(2, e \flat/\lambda, 0)$
- (s) $(2, e z_{m+\lceil \log m \rceil + 6}/T, 0)$
- (t) $(2, T/\lambda, 1)$
- (u) $(1, b T/S, 0)$
- (v) $(1, S \text{ yes}/\lambda, 0)$
- (x) $(1, b a_{2n+m+\lceil \log m \rceil + 11}/N, 0)$
- (y) $(1, N \text{ no}/\lambda, 0)$

- $i_{in} = 2$; es decir, la célula de entrada es la etiquetada por 2.

Sea $\mathcal{G} = (V, E)$ un grafo no dirigido con n vértices ($V = \{1, \dots, n\}$) y m aristas. Consideramos las funciones $cod(\mathcal{G}) = \{A_{ij} : \{i, j\} \in E \wedge 1 \leq i < j \leq n\}$ y $s(\mathcal{G}) = \langle m, n \rangle$. La ejecución del sistema $\Pi(s(\mathcal{G}))$ con entrada $cod(\mathcal{G})$ se estructura en cuatro fases:

- *Fase de generación de coloraciones*: se generan todas las posibles coloraciones del grafo mediante la aplicación sucesiva de reglas de división en la célula 2. En esta fase se puede observar con bastante claridad el no determinismo de los sistemas P de la familia y, a la vez, la confluencia de los mismos.

- *Fase de pre-chequeo:* esta fase permite introducir objetos R_{ij}, G_{ij}, B_{ij} , para cada arista $\{i, j\}$ del grafo, en todas las células etiquetadas por 2, a fin de analizar la validez de la coloración que cada una de ellas codifica.
- *Fase de chequeo:* esta fase chequea en cada célula 2 si la coloración que codifica es válida, para ello hará uso de los objetos R_{ij}, G_{ij}, B_{ij} .
- *Fase de salida:* el sistema proporciona la salida correspondiente en función de lo analizado anteriormente.

La familia **Π** es polinomialmente uniforme ya que las reglas están descritas recursivamente y la cantidad de recursos usados para su construcción es:

- El número total de objetos del alfabeto de trabajo es $3n^2 + 9n + 2m + 3\lceil \log m \rceil + 28 \in \Theta(n^2 + m)$.
- El número de células iniciales es $2 \in \Theta(1)$.
- El número inicial de objetos es $n + m + 6 \in \Theta(n + m)$.
- El número total de reglas es $3n^2 + 6n + 2m + 3\lceil \log m \rceil + 25 \in \Theta(n^2 + m)$.
- La máxima longitud de una regla es $4 \in \Theta(1)$.

La familia **Π** es polinomialmente acotada con respecto a (cod, s) ya que el tiempo de ejecución de $\Pi(s(u))$ con entrada $cod(u)$ es:

- $2n$ pasos en la fase de generación.
- 3 pasos en la fase de pre-chequeo.
- $m + \lceil \log m \rceil + 4$ pasos en la fase de chequeo.
- A lo sumo 3 pasos (respuesta negativa) en la fase de salida.

Por tanto, el tiempo total de ejecución del sistema $\Pi(s(u))$ con entrada $cod(u)$ es $2n + m + \lceil \log m \rceil + 10 \in \Theta(n + m)$ en el caso de una respuesta negativa.

Para analizar los detalles de la demostración de la adecuación y completitud de la familia **Π** respecto del problema 3-COL, ver [48].

1.4.4. Una solución de 3-COL mediante sistemas P de tejidos con reglas de separación

Seguidamente vamos a presentar una solución eficiente del problema SAT a través de una familia de sistemas P de tejidos reconocedores con reglas de separación.

Para cada par de números naturales $m, n \in \mathbb{N}$ se considera el sistema P de tejidos reconocedor, de grado 3, con reglas de separación celular y que usa reglas de comunicación de longitud, a lo sumo 3,

$$\Pi(\langle m, n \rangle) = (\Gamma, \Gamma_1, \Gamma_2, \Sigma, \mathcal{E}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{R}, i_{in}, i_{out})$$

definido como sigue:

- El alfabeto de entrada es:

$$\Sigma = \{x_{i,j}, \bar{x}_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m\}$$

- El alfabeto de trabajo es: $\Gamma = \Sigma \cup \Gamma_1 \cup \Gamma_2$, en donde:

$$\begin{aligned} \Gamma_1 = & \{A_i, B_i : 1 \leq i \leq n+1\} \cup \{a_i, b_i, T_i, F_i, y_i, v_i, w_i : 1 \leq i \leq n\} \cup \\ & \{c_i, t_i, f_i, s_i, z_i : 1 \leq i \leq n-1\} \cup \{E_j : 1 \leq j \leq m+1\} \cup \\ & \{\alpha_i : 0 \leq i \leq 3n+2m+1\} \cup \{\beta_i : 0 \leq i \leq 3n+2m+2\} \cup \\ & \{q_{i,j}, r_{i,j}, u_{i,j} : 1 \leq i, j \leq n-1\} \cup \\ & \{x_{i,j}, \bar{x}_{i,j}, e_{i,j}, \bar{e}_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m\} \cup \\ & \{d_{i,j,k}, \bar{d}_{i,j,k} : 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq k \leq n\} \cup \{q_0, S, \text{yes}, \text{no}\} \end{aligned}$$

$$\Gamma_2 = \{A'_i, B'_i : 1 \leq i \leq n+1\} \cup \{a'_i, b'_i, T'_i, F'_i : 1 \leq i \leq n\}$$

- El alfabeto del entorno es:

$$\begin{aligned} \mathcal{E} = & \{S\} \cup \{A_i, B_i, A'_i, B'_i : 2 \leq i \leq n+1\} \cup \{E_j : 1 \leq j \leq m+1\} \cup \\ & \{a_i, a'_i, b_i, b'_i, v_i : 2 \leq i \leq n\} \cup \{T'_i, c_i, t_i, f_i, s_i, z_i : 1 \leq i \leq n-1\} \cup \\ & \{T_i, F_i, F'_i, y_i, w_i : 1 \leq i \leq n\} \cup \{\alpha_i : 1 \leq i \leq 3n+2m+1\} \cup \\ & \{\beta_i : 1 \leq i \leq 3n+2m+2\} \cup \{q_{i,j}, r_{i,j}, u_{i,j} : 1 \leq i, j \leq n-1\} \cup \\ & \{e_{i,j}, \bar{e}_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m\} \cup \\ & \{d_{i,j,k}, \bar{d}_{i,j,k} : 1 \leq i, k \leq n, 1 \leq j \leq m\} \end{aligned}$$

- Los multiconjuntos iniciales son:

$$\begin{aligned}\mathcal{M}_1 &= A_1 B_1 \\ \mathcal{M}_2 &= a_1 a'_1 b_1 b'_1 v_1 q_{1,1} \alpha_0 \text{ yes no} \\ \mathcal{M}_3 &= \beta_0\end{aligned}$$

- El conjunto \mathcal{R} de reglas está formado por las que se relacionan a continuación:

- (1) $(1, A_i / a_i a'_i, 2)$, for $1 \leq i \leq n$, and $(1, A_{n+1} / E_1, 2)$.
- (2) $(1, A'_i / a_i a'_i, 2)$, for $1 \leq i \leq n$, and $(1, A'_{n+1} / E_1, 2)$.
- (3) $(1, B_i / b_i b'_i, 2)$, for $1 \leq i \leq n$.
- (4) $(1, B'_i / b_i b'_i, 2)$, for $1 \leq i \leq n$.
- (5) $(1, T_i / t_i, 2)$, for $1 \leq i \leq n - 1$.
- (6) $(1, T'_i / t_i, 2)$, for $1 \leq i \leq n - 1$.
- (7) $(1, F_i / f_i, 2)$, for $1 \leq i \leq n - 1$.
- (8) $(1, F'_i / f_i, 2)$, for $1 \leq i \leq n - 1$.
- (9) $(1, t_i / T_i T'_i, 0)$, for $1 \leq i \leq n - 1$.
- (10) $(1, f_i / F_i F'_i, 0)$, for $1 \leq i \leq n - 1$.
- (11) $(1, b_i / B_{i+1} S, 0)$, for $1 \leq i \leq n$.
- (12) $(1, b'_i / B'_{i+1}, 0)$, for $1 \leq i \leq n$.
- (13) $(1, a_i / T_i A_{i+1}, 0)$, for $1 \leq i \leq n$.
- (14) $(1, a'_i / F'_i A'_{i+1}, 0)$, for $1 \leq i \leq n$.
- (15) $(2, A_i / c_i, 0)$, for $1 \leq i \leq n - 1$, and $(2, A_i / \lambda, 0)$, for $n \leq i \leq n + 1$.
- (16) $(2, A'_i / c_i, 0)$, for $1 \leq i \leq n - 1$, and $(2, A'_i / \lambda, 0)$, for $n \leq i \leq n + 1$.
- (17) $(2, B_i / c_i, 0)$, for $1 \leq i \leq n - 1$, and $(2, B_n / \lambda, 0)$.
- (18) $(2, B'_i / c_i, 0)$, for $1 \leq i \leq n - 1$, and $(2, B'_n / \lambda, 0)$.
- (19) $(2, c_i / b_{i+1} b'_{i+1}, 0)$, for $1 \leq i \leq n - 1$.

- (20) $(2, v_i / y_i^2, 0)$, for $1 \leq i \leq n$.
- (21) $(2, y_i / z_i w_i, 0)$, for $1 \leq i \leq n-1$, and $(2, y_n / w_n, 0)$.
- (22) $(2, z_i / v_{i+1}, 0)$, for $1 \leq i \leq n-1$.
- (23) $(2, w_i / a_{i+1} a'_{i+1}, 0)$, for $1 \leq i \leq n-1$, and $(2, w_n / E_1, 0)$.
- (24) $(2, q_{1,1} / r_{1,1}, 0)$.
- (25) $(2, q_{i,j} / r_{i,j}^2, 0)$, for $1 \leq i \leq n-1$, $2 \leq j \leq n-1$.
- (26) $(2, r_{i,j} / s_i u_{i,j}, 0)$, for $1 \leq i, j \leq n-1$.
- (27) $(2, s_i / t_i f_i, 0)$, for $1 \leq i \leq n-1$.
- (28) $(2, u_{1,j} / q_{1,j+1} q_{2,j+1}, 0)$, for $1 \leq j \leq n-2$.
- (29) $(2, u_{i,j} / q_{i+1,j+1}, 0)$, for $2 \leq i, j \leq n-2$.
- (30) $(2, u_{i,n-1} / \lambda, 0)$, for $1 \leq i \leq n-1$.
- (31) $(2, T_i / \lambda, 0)$, for $1 \leq i \leq n-1$.
- (32) $(2, T'_i / \lambda, 0)$, for $1 \leq i \leq n-1$.
- (33) $(2, F_i / \lambda, 0)$, for $1 \leq i \leq n-1$.
- (34) $(2, F'_i / \lambda, 0)$, for $1 \leq i \leq n-1$.
- (35) $[S]_1 \longrightarrow [\Gamma_1]_1 [\Gamma_2]_1$
- (36) $(2, \alpha_i / \alpha_{i+1}, 0)$, for $0 \leq i \leq 3n+2m$.
- (37) $(3, \beta_i / \beta_{i+1}, 0)$, for $0 \leq i \leq 3n+2m+1$.
- (38) $(3, x_{i,j} / d_{i,j,1}^2, 0)$, $(3, \bar{x}_{i,j} / \bar{d}_{i,j,1}^2, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m$
- (39) $(3, d_{i,j,k} / d_{i,j,k+1}^2, 0)$, $(3, \bar{d}_{i,j,k} / \bar{d}_{i,j,k+1}^2, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq n-1$.
- (40) $(3, d_{i,j,n} / e_{i,j}, 0)$, $(3, \bar{d}_{i,j,n} / \bar{e}_{i,j}, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m$.
- (41) $(1, T_i E_j / e_{i,j}, 3)$, $(1, F_i E_j / \bar{e}_{i,j}, 3)$, $(1, T'_i E_j / e_{i,j}, 3)$, $(1, F'_i E_j / \bar{e}_{i,j}, 3)$, for $1 \leq i \leq n$, $1 \leq j \leq m$.

(42) $(1, e_{i,j} / T_i E_{j+1}, 0), (1, \bar{e}_{i,j} / F_i E_{j+1}, 0)$, for $1 \leq i \leq n, 1 \leq j \leq m - 1$.

(43) $(1, e_{i,m} / E_{m+1}, 0), (1, \bar{e}_{i,m} / E_{m+1}, 0)$, for $1 \leq i \leq n$.

(44) $(3, T_i / \lambda, 0), (3, F_i / \lambda, 0), (3, T'_i / \lambda, 0), (3, F'_i / \lambda, 0)$, for $1 \leq i \leq n$.

(45) $(3, E_j / \lambda, 0)$, for $1 \leq j \leq m$.

(46) $(1, E_{m+1} / \text{yes } \alpha_{3n+1+2m}, 2)$.

(47) $(1, \text{yes } / \beta_{3n+1+2m+1}, 3)$.

(48) $(2, \alpha_{3n+1+2m} / \beta_{3n+1+2m+1}, 3)$.

(49) $(2, \text{no } \beta_{3n+1+2m+1} / \lambda, 0)$.

(50) $(3, \text{yes } / \lambda, 0)$.

- La célula de entrada es $i_{in} = 3$.
- La región de salida es el entorno, $i_{out} = 0$.

Sea $\varphi = C_1 \wedge \dots \wedge C_m$ una fórmula proposicional en forma normal conjuntiva tal que $Var(\varphi) = \{x_1, \dots, x_n\}$ y $C_i = y_{i,1} \vee \dots \vee y_{i,k_i}$, $1 \leq i \leq m$, donde $y_{i,i'} \in \{x_j, \neg x_j : 1 \leq j \leq n\}$ son los literales de φ . Consideramos las funciones

$$cod(\varphi) = \bigcup_{i=1}^m \{x_{i,j} : x_j \in C_i\} \cup \{\bar{x}_{i,j} : \neg x_j \in C_i\}$$

$$s(\varphi) = \langle m, n \rangle = \frac{(m+n) \cdot (m+n+1)}{2} + m$$

Obsérvese que la fórmula φ será procesada por el sistema $\Pi(s(\varphi))$ con multiconjunto de entrada $cod(\varphi)$, cuya ejecución trata de implementar un algoritmo de fuerza bruta estructurado en las siguientes fases:

- *Fase de generación:* Se generan todas las posibles asignaciones de verdad asociadas al conjunto de variables de la fórmula proposicional de entrada. Para ello, se usarán de forma adecuada las reglas de separación del sistema de tal manera que cada célula codificará una y sólo una asignación de verdad.
- *Fase de chequeo:* En cada célula se comprueba si la asignación de verdad codificada por ella, dar valor verdadero a la fórmula.

- *Fase de salida:* El sistema envía al entorno la salida correspondiente de acuerdo con el chequeo efectuado en la fase anterior.

La familia **Π** es polinomialmente uniforme ya que las reglas están descritas recursivamente y la cantidad de recursos usados para su construcción es:

- El número total de objetos del alfabeto de trabajo es $2mn^2 + 5mn + 3n^2 + 5m + 27n + 12 \in \Theta(mn^2)$.
- El número de células iniciales es 3.
- El número inicial de objetos en las células iniciales es $12 \in \Theta(1)$.
- El número total de reglas es $mn^2 + 3mn + 3n^2 + 5m + 30n + 12 \in \Theta(mn^2)$.
- La máxima longitud de una regla de comunicación es 3..

La familia **Π** es polinomialmente acotada con respecto a (cod, s) ya que el tiempo de ejecución de $\Pi(s(\varphi))$ con entrada $cod(\varphi)$ es:

- $3n + 1$ pasos en la fase de generación.
- $2m$ pasos en la fase de chequeo.
- 3 pasos en la fase de salida.

Por tanto, el tiempo total de ejecución del sistema $\Pi(s(\varphi))$ con entrada $cod(\varphi)$ es $3n + 2m + 4 \in \Theta(n + m)$.

Para analizar los detalles de la demostración de la adecuación y completitud de la familia **Π** respecto del problema SAT, ver [125].

1.5. Simuladores de sistemas P

Recordemos que un modelo de computación consta, básicamente, de una especificación sintáctica que permite su identificación y una semántica formal que proporciona la dinámica, el comportamiento del mismo a lo largo del tiempo.

Una *implementación* del modelo consiste en diseñar una máquina real que tiene la capacidad de reproducir fiel y exactamente las computaciones que se producen en el mismo; en particular, cada paso de computación realizado en el modelo requiere una unidad de tiempo de ejecución en la máquina implementada. Como tal máquina real, será un dispositivo finito y, por tanto, tendrá unas limitaciones en lo que respecta a la cantidad de recursos. Quiere ello decir que todo lo calculable, desde el punto de vista teórico, en un modelo de computación no puede ser reproducido en una máquina que lo implemente, simplemente aquello para lo que la máquina disponga de recursos computacionales suficientes.

A diferencia de otros paradigmas computacionales, es bien conocido que las distintas variantes de la computación celular con membranas no han sido implementadas en ningún medio (bioquímico, electrónico, etc.). Más aún, debido a la naturaleza bio-inspirada, masivamente paralela y no determinista, la implementación de alguna de las variantes de sistemas celulares constituye un gran reto para la ciencia y la tecnología actual, hasta el punto de existir relaciones interesantes entre la implementación eficiente de ciertos sistemas celulares y la resolución del famoso problema **P** versus **NP**, desde un punto de vista práctico.

Si bien es cierto que existen estudios preliminares analizando la problemática relativa a dicha implementación (e.g. [60]), aún queda un largo camino hasta alcanzar el objetivo final, respecto al que se muestra especialmente escéptico el colectivo de los biólogos moleculares y celulares.

Ahora bien, ciertas variantes de sistemas celulares han sido utilizadas para la resolución “eficiente” de problemas abstractos con un alto nivel de complejidad computacional, o bien como marco para la modelización matemática de fenómenos relevantes y muy variados de la vida real.

En particular, en este último caso, a la hora de extraer información acerca de los procesos objetos de estudio es necesario manejar esas variantes (en primer lugar, para la validación experimental del modelo y, en segundo lugar, para la obtención de hipótesis plausibles relacionadas con escenarios de interés

para los expertos). Por ello es imprescindible diseñar programas informáticos que puedan ser ejecutados sobre ordenadores convencionales y que permitan reproducir, al menos parcialmente, el comportamiento formal de esos sistemas celulares.

Nosotros usaremos el término *simulador* de un modelo formal de computación para referirnos a una aplicación de ordenador (software y/o hardware) que describa la especificación sintáctica del mismo a través de un lenguaje de programación y materialice la semántica mediante un algoritmo de simulación susceptible de ser implementado en una máquina real (generalmente electrónica). Obviamente, ese algoritmo debe capturar la dinámica del modelo a nivel “global” en el sentido de proporcionar no sólo las salidas que correspondan a unos datos de entradas del modelo sino que, además, cada paso de computación del modelo debe ser “reproducido” en el simulador mediante un número finito de pasos y de tal manera que la ejecución del simulador nos permita determinar los elementos básicos del modelo que han intervenido de manera relevante en ese paso.

En los últimos años se han presentado un gran número de aplicaciones de *software* y *hardware* para *Membrane Computing*, lo cual ha propiciado un gran avance en el desarrollo de simuladores [46]. La mayoría de ellos muestran una arquitectura software típica, incluyendo tres módulos principales: *entrada* (la definición del modelo de sistema celular), *núcleo* (el motor de simulación), y *salida* (la presentación de resultados).

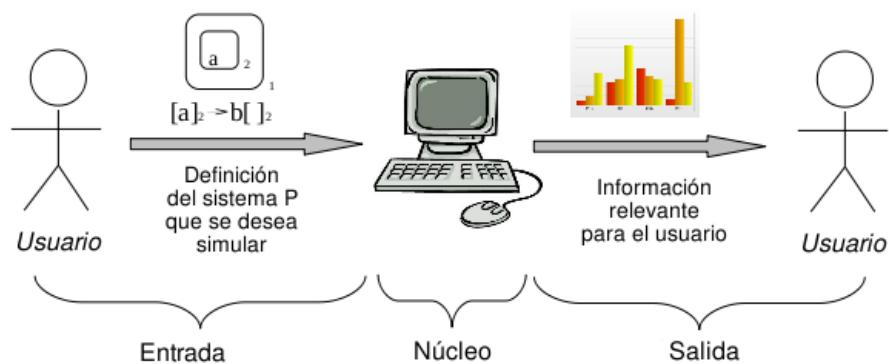


Figura 1.4: Elementos comunes en los simuladores de sistemas P

Para simular un sistema celular es necesario establecer una especificación que permita definirlo completamente, para lo cual se le ha de suministrar: (a) la variante de sistema celular a simular; (b) la estructura inicial del sistema (de membranas o de células); (c) los multiconjuntos iniciales asociados a cada componente básica del sistema; y (d) el conjunto de reglas del sistema. Esta tarea se complica notoriamente cuando es necesario especificar familias de sistemas celulares, como es el caso de soluciones de problemas de decisión computacionalmente duros, en donde el conjunto de reglas, el alfabeto, los multiconjuntos iniciales y, eventualmente, la estructura del sistema dependen de los valores asignados a unos parámetros iniciales.

En lo que respecta al núcleo de simulación, se trata de la parte de la aplicación que se encarga de simular las computaciones del sistema celular definido. En este módulo, se parte del supuesto de que el sistema introducido es consistente con un determinado modelo, para lo cual se delega en el módulo previo. El núcleo procede a la ejecución de un determinado algoritmo de simulación para proporcionar una o varias computaciones del sistema celular simulado, reproduciendo, paso a paso, una de las posibles computaciones. Para ello, se establecen estructuras de datos en la memoria de la máquina para almacenar las sucesivas configuraciones alcanzadas, de tal manera que sea 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, información que dependerá principalmente de la finalidad del simulador. En el caso de aquellos que fueron diseñados con fines pedagógicos, son relevantes las propias configuraciones generadas por el simulador, mientras que 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 que, en general, será simplemente un experto en el proceso objeto de estudio. En el Capítulo 4 se profundizará en estas consideraciones.

1.5.1. El proyecto software P-Lingua

Cada modelo de computación celular con membranas tiene unas ciertas peculiaridades de tipo semántico en lo que respecta a la forma en que las reglas son aplicadas. Por tanto, sería conveniente que los simuladores tuvieran la capacidad de soportar diferentes escenarios correspondientes a distintos modelos de sistemas *P* y, por ello, los simuladores deben recibir como *entrada* una definición precisa del sistema celular que va a ser simulado.

P-Lingua [131, 120] es, en sí mismo, un lenguaje de definición para describir sistemas P y permite estandarizar la definición de la entrada del simulador, a fin de que los simuladores procesen siempre el mismo formato. El proyecto P-Lingua provee herramientas de software libre bajo licencia GNU GPL [2], para la compilación, simulación y depuración, las cuales están integradas en la biblioteca Java llamada *pLinguaCore*. La biblioteca también incluye varios simuladores secuenciales para reproducir las computaciones de las variantes de sistemas P soportadas.

Existe la posibilidad de exportar ficheros de definición de P-Lingua a otros formatos de fichero (libres de errores) con el fin de proporcionar interoperabilidad entre los distintos entornos software de simulación. Conviene destacar algunas de las ventajas de P-Lingua y pLinguaCore. En primer lugar, P-Lingua es un formato estándar para definir sistemas P que se puede traducir a otros formatos sintácticamente libre de errores. En segundo lugar, el desarrollo de un simulador en pLinguaCore es un proceso relativamente sencillo y, a la vez, permite implementar nuevos algoritmos de simulación. Finalmente, el mismo fichero de entrada P-Lingua puede ser usado tanto para los simuladores desarrollados como por los simuladores de pLinguaCore.

1.5.2. Plataformas de simulación de sistemas P

Una buena plataforma de computación para simular sistemas P debería proveer un buen balance de rendimiento, flexibilidad, escalabilidad, paralelismo y coste [69]. Las tres primeras propiedades son consideradas los atributos de calidad más importantes de las plataformas para aplicaciones de Computación con Membranas [105]:

- *Rendimiento*: se refiere a la velocidad a la cual la plataforma es capaz de simular sistemas P. Una medida adecuada para ello puede ser el número de aplicaciones de reglas que se realizan por unidad de tiempo.
- *Flexibilidad*: significa la habilidad de la plataforma para soportar la ejecución de un gran rango de sistemas P. Un alto grado de flexibilidad implica soportar una gran diversidad de sistemas P (de acuerdo a las variantes designadas a ser simuladas).
- *Escalabilidad*: se refiere a la habilidad de la plataforma para simular sistemas P con aumento de tamaño sin repercutir en la habilidad de realizar sus funciones o reducir el rendimiento. El tamaño de los sistemas

P se puede determinar a este respecto mediante el número de membranas, el número de reglas definidas y los multiconjuntos iniciales.

Es difícil asegurar que cualquier plataforma de computación incluya los tres atributos. Un factor que promueve un atributo puede demorar otro. De hecho existen dos conexiones principales entre ellos [105]: *flexibilidad vs rendimiento* y *flexibilidad vs escalabilidad*.

1.5.3. Simulación paralela de sistemas P

En esta sección se aporta una revisión general de los simuladores paralelos desarrollados hasta el momento para sistemas P.

- *Simuladores basados en cluster* (un conjunto de computadores interconectados mediante una red local [130]):
 - Simulador de *Ciobanu y Guo* [35]: simula un conjunto restrictivo de sistemas P de transición usando un cluster basado en Linux, usando C++ y la biblioteca MPI. Los autores indican que la mayor limitación estaba en la comunicación y cooperación entre membranas.
 - Simulador de *Fernández et al.* [151]: aporta ciertas mejoras al simulador de *Ciobanu y Guo* con el fin de resolver parcialmente el problema del proceso de comunicación.
 - Simulador de *Syropoulos et al.* [150]: este simulador trabaja en Java, y hace uso de la biblioteca *RMI* (Remote Method Invocation) para distribuir el trabajo. Al igual que en los simuladores anteriores, se distribuye cada membrana por cada procesador. Sin embargo, el objetivo era mostrar el uso de los sistemas P como un fundamento de la computación distribuida.
 - Simulador de *Diez Dolinski et al.* [51]: se trata de una alternativa novedosa, que provee una solución altamente escalable al problema del crecimiento exponencial del espacio creado por los sistemas P. Se utilizan algoritmos *MapReduce* sobre los entornos distribuidos, y se amplía P-Lingua (P-Lingua distribuido) para definir las correspondientes entradas.
- *Simuladores basados en FPGA* (chips reconfigurables que el diseñador puede programar, permitiendo usar ciertos recursos como procesadores y memoria “a la carta” [93]):

- Simulador de *Petreska y Teuscher* [126]: el primer simulador basado en FPGAs que da un total soporte para un subconjunto particular de sistemas P de transición, también permitiendo reglas de división y disolución. Está implementado en *VHDL* (VHSIC hardware description language), y tiene algunas limitaciones, como por ejemplo [105]: no implementa paralelismo dentro de las membranas, es inflexible, y tiene una limitada escalabilidad.
 - Simulador de *Fernández et al.* en FPGA [56]: es una alternativa que presenta un diseño de un circuito que puede ser implementado en FPGAs para la selección de las reglas activas de un sistema P de transición.
 - Simulador de *Nguyen et al.* [106, 105, 107]: se trata de una evolución en la simulación de sistemas P en FPGAs. Su sistema hardware, denominado Reconfig-P, junto a P-BUILDER, provee una aproximación elegante al balanceo de rendimiento, flexibilidad y escalabilidad. El sistema construido por encima de la FPGA es capaz de adaptar el diseño del circuito a las características del sistema P. Por ejemplo, el no determinismo se soporta gracias a un algoritmo diseñado para tal fin, denominado DND [107].
- *Simuladores basados en microcontroladores* (circuito integrado que contiene procesador, memoria y unidades de entrada/salida):
- Simulador de *Gutiérrez et al.* [70, 69]: el objetivo de este simulador es balancear la flexibilidad y el rendimiento a un bajo coste. Se utiliza el *PIC* (Peripheral Interface Controller) como solución, y para tal fin, se han diseñado distintos algoritmos y arquitecturas para mejorar la comunicación entre las unidades que manejan las membranas [151, 23, 69].
- *Simuladores basados en GPUs* (procesadores de las tarjetas gráficas, que pueden ser usados como multiprocesadores especiales altamente paralelos [111], y programados con modelos de programación como CUDA [85, 5] y OpenCL [100, 6]):
- *PMCGPU*: consiste en una serie de subproyectos que tienen como objetivo simular diferentes variantes de sistemas P:
 - *PCUDA*: es el primer simulador para sistemas P basado en CUDA, conformando una plataforma flexible para simular sistemas P, pero comprometiendo rendimiento y escalabilidad.

- *PCUDASAT*: es una rama del proyecto PCUDA centrada en la simulación eficiente de una familia de sistemas P con membranas activas que resuelve instancias del problema SAT. Se trata de una plataforma inflexible, pero que incrementa el rendimiento y la escalabilidad.
- *TSPCUDASAT*: es una variante de PCUDASAT, donde se trata a una familia de sistemas P a modo de tejidos con división celular que resuelven SAT.
- *ABCD-GPU*: es un proyecto reciente en la simulación de sistemas PDP, que serán descritos en la Sección 3.2.2. Incluye simuladores basados en C++, y para plataformas de procesadores multinúcleo (con OpenMP [7]) y GPU (con CUDA).
- *Simulación de sistemas P neuronales con impulsos*: los primeros simuladores fueron iniciados por Cabarle *et al.* [24, 25]. El algoritmo implementado usa una representación de matrices del modelo, introducida por Zeng *et al.* [161]. Por tanto, la implementación es relativamente sencilla para la GPU. Además, se aprovecha la simulación del no determinismo como un nivel de paralelismo, de tal manera que cada camino de computación se lleva a cabo por un multiprocesador de la GPU.
- *Simulación de sistemas P de evolución-comunicación con energía y sin reglas antiport*: el trabajo inicial fue llevado por Juayong *et al.* [82]. Como para los sistemas P neuronales de impulsos, el simulador hace uso de una representación de matrices, donde se representa la semántica del modelo.
- *Simulación de sistemas P numéricos y enzimáticos*: este trabajo es llevado a cabo por García-Quismondo *et al.* [59]. Los modelos simulados son usados para modelizar controladores de robots, por lo que los resultados son significativos para el campo de la Inteligencia Artificial.
- *Simuladores de soluciones de procesamiento de imágenes con sistemas P a modo de tejidos*: Díaz-Pernil *et al.* [119] presentan un nuevo trabajo en la simulación de modelos de sistemas P a modo de tejidos, pero específicos de algoritmos de procesamiento de imágenes (por ejemplo, suavizado de imágenes). Estas aplicaciones son soluciones específicas para imágenes, donde la simulación de los sistemas P se lleva a cabo implícitamente en el código fuente.

2

Modelos en la vida real

En este capítulo se aborda la problemática general que existe a la hora de estudiar sistemas dinámicos complejos (fenómenos, entidades, procesos u organismos) que aparecen en el mundo real y que pueden resultar de interés por motivos diversos. El método que se va a usar se denomina *modelización* y consiste en representar el sistema objeto de estudio en un determinado marco (gráfico, informal, matemático, etc.) con el fin de obtener más información y, a ser posible, aumentar el conocimiento que se tiene acerca del mismo, posibilitando la formulación de hipótesis predictivas en diferentes escenarios. Para ello, se diseñará un *modelo* que tratará de capturar los elementos más significativos del sistema y, posteriormente, se manejará el modelo usando diferentes técnicas a fin de saber, en primer lugar, la fiabilidad del mismo y, en su caso, poder inferir información nueva que pudiera ser de interés para el mejor conocimiento del propio sistema.

Este capítulo está estructurado como sigue. La primera sección está dedicada a justificar la necesidad de usar modelos que representen sistemas dinámicos complejos a fin de extraer algún tipo de información que pueda ser de utilidad en un determinado contexto.

En la segunda sección se describen brevemente algunas aproximaciones para el diseño de modelos matemáticos (es decir, en el marco de alguna teoría formal), desde la pionera formulación continua basada en sistemas de ecua-

ciones diferenciales ordinarias o parciales (ODEs/PDEs), a las formulaciones discretas basadas en distintos paradigmas de computación (sistemas basados en agentes, redes de Petri y álgebra de procesos, entre otras).

La sección 2.3 está dedicada a la presentación de dos tipos de algoritmos que describen sucintamente las semánticas asociadas a ciertos modelos computacionales y que capturan la aleatoriedad inherente a los procesos que se producen en la naturaleza. En primer lugar, una semántica estocástica basada en el algoritmo clásico de Gillespie y, en segundo lugar, una semántica de tipo probabilístico. Ambas semánticas tienen como característica común el hecho de asociar constantes o funciones numéricas a las distintas expresiones (reglas o ecuaciones) que aparecen en el modelo, de tal manera que los valores numéricos asociados a cada expresión en un instante determinado juegan un papel relevante a la hora de determinar la siguiente configuración del sistema.

2.1. Necesidad de diseñar modelos

En cierto sentido se puede afirmar que el intento de miniaturizar ciertas partes del mundo ha sido un sueño constante de la humanidad. Los esfuerzos encaminados a capturar exactamente esa parte concreta del mundo que es realmente significativa, en ciertos contextos, con el fin de destilar su esencia, ha conducido a avances importantes en la ciencia.

Informalmente, un *modelo* de un cierto sistema complejo es una representación concreta, abstracta, conceptual, gráfica o formal, que permite estudiar, analizar, describir, explicar y razonar acerca del mismo. Así por ejemplo, un modelo de un objeto físico trata de imitar la apariencia o fisonomía de ese objeto como, por ejemplo, sería la maqueta de un barco o un retrato de una cierta persona. Lo que importa de un modelo es, básicamente, la información que es capaz de proporcionar acerca del sistema que representa. Con un modelo se obtiene una imagen simplificada de la realidad que, por tanto, nunca podrá contener todos y cada uno de los detalles del sistema real que describe y representa. Es muy importante saber distinguir entre componentes y factores que son esenciales, de otros que no lo son tanto y que, en consecuencia, no deberían ser incorporados al modelo. En resumen, los modelos son abstracciones o representaciones simplificadas de ciertos aspectos o sistemas del mundo real que tienen su interés y utilidad en determinados contextos.

Las leyes de Newton pueden ser consideradas como modelos en tanto que proporcionan información cualitativa acerca de la dinámica aproximada de

ciertos sistemas. Como suele suceder con todos los modelos, dichas leyes contienen simplificaciones; en efecto, una velocidad constante presupone en muchos casos que no existe fricción, la caída de una manzana de un árbol admite que no hay resistencia del aire y que la masa es constante, etc. A pesar de estas simplificaciones, no hay duda que las leyes de Newton son muy útiles en numerosos cálculos mecánicos ya que las simplificaciones sólo introducen errores menores que, además, pueden ser evaluados, por ejemplo, en estos casos por medio de otras leyes físicas sobre fricción o resistencia del aire. En función de la precisión que se quiera obtener en nuestros cálculos, habría que analizar si las simplificaciones efectuadas son adecuadas o necesitan ser revisadas con otras consideraciones adicionales.

Existe un problema de “equilibrio” a la hora de diseñar un modelo ya que, por una parte, éste debe incorporar las características del sistema que sean esenciales o relevantes en el contexto del problema científico que trata de ser resuelto; es decir, deben ser lo suficientemente simples con el fin de entenderlos y manejarlos de la mejor y más eficiente manera posible. Pero, por otra parte, deben ser lo suficientemente ricos a fin de proporcionar comportamientos que puedan ser sorprendentes, interesantes, útiles y significativos. Así por ejemplo, los mapas geográficos son modelos de ciertas realidades y el diseño de los mismos estará en función de su utilidad para propósitos que pueden ser muy diversos como, por ejemplo, mapas para aeronaves, coches, trenes, barcos, lugares arqueológicos, genoma de organismos vivos, etc.

El uso de modelos es intrínseco a cualquier actividad científica y es básico a la hora de abordar ciertos problemas de la vida real en donde, además, como es bien conocido existen muchas interacciones entre otros factores que operan de manera simultánea. La construcción de modelos nos puede ayudar a describir esas interacciones, a relacionar las experiencias u observaciones previas con otras actuales y a analizar qué podría suceder en el futuro bajo determinadas circunstancias o escenarios iniciales.

Como ya hemos comentado, los científicos usan regularmente abstracciones o simplificaciones de la realidad tales como diagramas, grafos, leyes, relaciones, etc., con el fin de tratar de entender mejor ciertos aspectos concretos que estudian, analizan y examinan. El desarrollo de modelos suele requerir una aproximación multidisciplinar ya que en el propio problema que se estudia pueden incidir relaciones muy variadas entre diferentes disciplinas. Los médicos, químicos, biólogos, ecólogos y economistas, entre otros, han estado siempre familiarizados con unos mecanismos de modelización, generalmente informales, estrechamente relacionados con los experimentos que realizaban

en el laboratorio o sobre el terreno. En este contexto, las Matemáticas y la Informática han sido utilizadas por dichos colectivos simplemente como herramientas auxiliares para el mejor desarrollo cuantitativo de esos experimentos. Sin embargo, durante las últimas décadas se ha producido un extraordinario avance en las técnicas usadas en la experimentación así como en la extracción y procesamiento de la información, que ha propiciado la recopilación de una masiva cantidad de datos sobre los sistemas dinámicos analizados; por ejemplo, en la búsqueda de las moléculas involucradas en ciertas funciones y en el análisis de su estructura como fue el caso de la insulina, en la secuenciación del genoma de distintas especies, incluida la humana, y la búsqueda de genes, en el análisis de factores en ecosistemas que inciden en la evolución de especies en peligro de extinción o de especies exóticas invasoras, etc. Así se llegó a la conclusión que el avance en las técnicas de laboratorio o de campo, unido a los distintos avances tecnológicos, únicamente habían proporcionado algo así como la *partitura* que describía la complejidad de los sistemas dinámicos que se analizaban, sin darnos información directa acerca de cómo interpretar la música o de cómo componer nuestras propias melodías. Entonces surge el problema de cómo manejar esa ingente cantidad de datos a fin de extraer, por una parte, información cualitativa que puede ser deducida a partir de los datos y, por otra, realizar predicciones acerca de los comportamientos posibles de los sistemas a lo largo del tiempo ante diferentes escenarios. Los modelos proporcionan, en general, mucha más información acerca del problema objeto de estudio que un simple análisis estadístico de observaciones ya que tienen la ventaja de poder incorporar gran parte del conocimiento teórico sobre el tema.

Los logros alcanzados durante finales del siglo pasado, tanto en Biología celular y molecular como en Ecología y dinámica de poblaciones, en general, en Economía así como, por supuesto, en Ciencias de la Computación tanto en su vertiente teórica como práctica, han propiciado la convergencia de dichas disciplinas a través del uso de modelos formales, con el objetivo de conseguir progresos científicos significativos en la ciencia.

2.2. Modelos formales

Un *modelo formal* es una abstracción de un aspecto concreto del mundo real dentro de un marco matemático que trata de resaltar algunos aspectos relevantes del sistema objeto de estudio a través del uso de teorías formales. Así pues, un modelo formal es una especie de traducción simplificada de una parte de la realidad a un nuevo sistema expresado en términos específicamente matemáticos; es decir, en el contexto de una teoría formal.

El proceso de modelización formal consta de un conjunto de etapas o fases semiformales que nos guían en la tarea de diseño, en la descripción en un lenguaje formal, así como en su implementación para la evaluación y el análisis. Dicho proceso se puede estructurar como sigue:

■ *Identificación y delimitación del fenómeno objeto de estudio.*

El desarrollo de modelos formales es un proceso arduo en donde, con frecuencia, habrá que reconsiderar los supuestos, las simplificaciones, etc. realizadas en su diseño inicial. El primer paso consiste en establecer la parte específica del sistema que se quiere modelizar detallando las componentes más relevantes, los objetivos concretos que se pretenden alcanzar, las cuestiones particulares que se trata de responder, así como los procesos de validación y análisis que se van a considerar.

■ *Definición y formulación formal.*

Seguidamente hay que especificar o traducir en el contexto de una teoría formal, la descripción informal de las componentes del sistema que se estudia, los objetivos y las cuestiones a tratar.

■ *Implementación o simulación computacional.*

A continuación hay que detallar la forma de implementar o, en su caso, aproximar numéricamente o simular el modelo usando ordenadores electrónicos, a fin de poder manejarlo y extraer información fiable. En este punto es fundamental la corrección de los algoritmos usados para capturar la semántica del modelo considerado debido, básicamente, a la gran cantidad de parámetros, variables y estructuras involucradas en los elementos fundamentales (por ejemplo, funciones celulares, redes de genes, rutas señalizadoras de proteínas, interacciones entre individuos de poblaciones, etc.) de los sistemas dinámicos que se estudian. Esta tarea entra dentro de la Ingeniería del software.

■ *Validación y calibración.*

En el estudio y análisis de sistemas complejos es posible que determinados parámetros o bien no se hayan podido calcular experimentalmente o bien se han obtenido dentro de un rango de posibles valores. Por ello, antes de realizar simulaciones habrá que proceder a *calibrar* nuestro modelo; es decir, habrá que usar un método de estimación de parámetros y realizar un análisis de sensibilidad para determinar qué parámetros son realmente cruciales en el modelo. El proceso de calibración debe ser seguido de

un proceso de validación contrastando los resultados de la simulación del modelo con datos fiables obtenidos experimentalmente. Esta etapa proporciona, obviamente, un proceso iterativo de *refinamiento*. Conviene advertir que el método de validación depende, en cierto sentido, de los objetivos del modelo y, por ello, habría que traducir, en cierta manera, esos objetivos en criterios de validación. La validación debe proporcionar el grado de fiabilidad o de incertidumbre del modelo.

■ *Análisis y chequeo.*

Una vez que se ha encontrado un conjunto de parámetros fiables del modelo y éste ha sido validado, podemos centrarnos en algunas cuestiones que se formularon como objetivos cuando se inició el proceso de modelización. Existen diferentes métodos de análisis según los modelos formales diseñados que van desde una simple generación de simulaciones en un ordenador electrónico hasta complicados análisis estadísticos, pasando por el uso de técnicas de *model checking*.

En un modelo formal de un sistema complejo se pueden distinguir una serie de componentes o elementos básicos: *variables de estado*, *funciones de control* o *variables externas*, *expresiones matemáticas*, *parámetros* y *constantes universales*, si las hubiere.

1. Las variables de estado permiten describir completamente el sistema objeto de estudio en un instante determinado. La elección de estas variables es esencial en el diseño del modelo.
2. Las funciones de control o variables son expresiones “externas” que influyen sobre el estado del sistema. El modelo tratará de analizar cómo los cambios en los valores de esas variables o funciones pueden afectar al estado del sistema.
3. Las expresiones matemáticas (ecuaciones, reglas de reescritura, etc.) describen las relaciones entre las funciones de control y las variables de estado. Se usan para representar procesos físicos, químicos, biológicos y de comportamiento de poblaciones, en general.
4. Los parámetros son coeficientes que intervienen en la representación matemática de los procesos a fin de especificar algunos hechos relevantes de los elementos que intervienen en el modelo.

5. Las constantes universales, en su caso, intervienen en el sistema analizado y no dependen, propiamente, del mismo.

Parece claro que para evitar que un modelo sea innecesariamente complejo habría que restringir “tanto como se pueda” el número de componentes básicas.

Es bien conocido que en muchos sistemas complejos reales no es posible realizar experimentos con todo el sistema completo a fin de obtener información. No obstante, recurriendo a modelos de dichos sistemas será posible realizar modificaciones sobre las funciones de control y variables a fin de estudiar los cambios producidos en los mismos a través de una evaluación de los cambios en las variables de estado.

2.3. Modelización computacional

Recordemos que dar un modelo de computación consiste, básicamente, en proporcionar una definición formal del concepto de procedimiento mecánico. En este marco, la resolución de un problema abstracto tiene la ventaja de poder ser implementada (o, en su caso, simulada) en una máquina y, por tanto, de ser ejecutada para valores que especifican problemas concretos.

Está ampliamente aceptado ([138]) que un buen modelo formal debe satisfacer, al menos, las cuatro siguientes propiedades: *relevancia*, *comprendibilidad*, *extensibilidad* y *tratabilidad matemático-computacional*.

- Un modelo debe ser *relevante*, en el sentido de capturar las propiedades básicas o esenciales del fenómeno investigado de una manera unificada, tanto en su estructura como en su comportamiento a lo largo del tiempo.
- Un modelo debe facilitar una mejor *comprendibilidad* del sistema que se estudia; es decir, los formalismos abstractos usados en el modelo deberían corresponderse bien con los conceptos informales e ideas del sistema, de tal manera que pueda ser fácilmente interpretado por los expertos en el sistema objeto de estudio.
- Un modelo debe ser fácilmente *extensible* y *escalable* a otros niveles de organización y fácilmente modificable con el fin de incluir nuevo conocimiento o eliminar hipótesis falsas. En general, nuestro conocimiento del sistema es dinámico y casi constantemente se está generando nueva información sobre el mismo; por ello, el modelo debería ser fácilmente extensible, en el sentido de poder incorporar esa nueva información.

- Un modelo debe ser *tratable computacionalmente*, en tanto en cuanto debe permitir su implementación en dispositivos electrónicos a fin de poder ejecutar simulaciones que permitan manejar el modelo y estudiar la dinámica del sistema en diferentes escenarios, a través de la manipulación de las condiciones experimentales, sin necesidad de realizar experimentos complejos y costosos en el laboratorio o sobre el terreno, posibilitando, de esta manera, su análisis matemático y computacional.

El desarrollo de modelos en un marco computacional ofrece la ventaja de poder manejarlos usando máquinas o simuladores del propio marco. La habilidad de los modelos computacionales para manejar la complejidad hacen de ellos potentes instrumentos para explorar la naturaleza. Estos modelos son esenciales para entender más y mejor sistemas complejos en donde los cálculos directos podrían tardar años o, incluso, siglos. Son aplicables tanto a nivel micro (por ejemplo, a nivel de reacciones químicas moleculares, rutas señalizadoras de proteínas, regulación de la expresión de genes, etc.) como a nivel macro (por ejemplo, a nivel de ecosistemas, galaxias, etc.) y están posibilitando la aparición de importantes y útiles herramientas científicas.

De la misma forma que la interacción entre teoría y experimentación es la fuerza que dinamiza la ciencia, se puede afirmar que, actualmente, la modelización computacional y la simulación están en el propio corazón del método científico que podríamos calificar como moderno. En este caso, como en tantos otros, la teoría y los experimentos prácticos van de la mano.

Uno de los objetivos fundamentales de cualquier modelo formal es su capacidad de *predicción*; es decir, la posibilidad de realizar conjeturas o hipótesis plausibles acerca del posible comportamiento del sistema modelizado que es objeto de investigación, ante diferentes escenarios que puedan ser considerados de interés por los expertos.

Existen diversas formas de enfocar el proceso de modelización formal de sistemas complejos. A continuación, enumeramos algunas aproximaciones.

1. En relación con el escalado del espacio, se puede distinguir entre enfoque *macroscópico*, *microscópico* y *mesoscópico*. En el primero de ellos, el sistema se observa como un todo, sus componentes se representan con poco detalle y no se proporcionan mecanismos de interacción entre ellas. En el enfoque microscópico, cada parte del sistema se representa con bastante detalle; por ejemplo, en el caso de fenómenos moleculares, se tiene en cuenta cada molécula y se especifica su posición y momento. Esta aproximación proporciona mucha más información que la anterior pero es

intratable desde el punto de vista computacional en la mayoría de casos. Por último, el enfoque mesoscópico se centra en el número de componentes individuales que integran el sistema, sólo tiene en cuenta las partes del sistema que se consideran más relevantes, despreciando parámetros como la posición y el momento, en el caso de las moléculas. Esta aproximación es más tratable que la microscópica y, a la vez, es capaz de manejar y conservar información más relevante que la macroscópica.

2. De acuerdo con el tipo de análisis realizado, un modelo formal puede ser *cuantitativo* o *cualitativo*. El primero proporciona información y datos cuantitativos acerca del sistema que se estudia, mientras que el segundo proporciona información cualitativa acerca del sistema y de su dinámica. A su vez, según el tipo de datos cuantitativos generados y el carácter de la especificación del sistema, un modelo puede ser *discreto* o *continuo*. En el primero de ellos, las componentes del sistema modelizado son representadas mediante individuos o entidades discretas y los datos generados son, también, discretos. En un modelo *continuo*, las componentes del sistema son representadas a través de variables continuas y los datos generados son continuos.
3. Respecto de su dinámica o evolución, los modelos se clasifican en *determinista* y *no determinista*. En el primero de ellos, existe una única posible evolución del modelo a partir de unos parámetros o situación inicial, mientras que en el segundo existen muchas posibles evoluciones del modelo a partir de unos parámetros iniciales. En esa dinámica, el paso de una configuración a una configuración siguiente se realiza, por ejemplo, con una cierta aleatoriedad; por tanto, a la hora de obtener resultados mínimamente fiables a partir de un determinado escenario, es necesario realizar un número elevado de simulaciones y trabajar con medias, evaluando la desviación típica y el intervalo de confianza.
4. Finalmente, en relación con el origen de la información utilizada para su diseño, los modelos formales pueden ser *empíricos* o *heurísticos*. El primero de ellos es construido a partir de observaciones directas o resultados experimentales, mientras que los modelos heurísticos son diseñados a través de los mecanismos conocidos del sistema que se estudia.

Es bien conocido que uno de los objetivos fundamentales de la física moderna es el desarrollo de una teoría unificada de campos que permitiera explicar todas las fuerzas naturales a través de un cierto conjunto de ecuaciones. Ese

objetivo vendría a ser algo así como una *teoría de todo*. Por el mismo motivo, sería maravilloso poder disponer de un *modelo de todo* que permitiera unificar todos los modelos de todos los sistemas del mundo real. En ese sueño, dispondríamos de una especie de copia del mundo, a la vez humano y físico, que sería capaz de responder a cualquier pregunta y predecir el futuro, una especie de bola mágica de cristal que materializaría, a su vez, el sueño de los enciclopedistas del siglo XVIII al contener todo el conocimiento habido e, implícitamente, por haber. Los modelos computacionales son diseñados de acuerdo con una serie de datos conocidos u obtenidos experimentalmente y, presumiblemente, de unas leyes conocidas, pero no son exactamente la propia realidad que trata de representar. Por ello, será muy importante poder evaluar el nivel con el que se ha capturado la realidad.

Los modelos computacionales y sus eventuales predicciones pueden provocar ciertos recelos en muchos entornos de expertos. Desconfianza como la que encontró Galileo tras realizar las primeras observaciones astronómicas con un telescopio y publicar sus descubrimientos en los que se afirmaba la existencia de montañas lunares, las lunas de Júpiter, los anillos de Saturno, etc. Esas afirmaciones violaban las creencias aristotélicas acerca del cosmos, llegando los críticos a afirmar que esos descubrimientos eran, propiamente, artefactos derivados del modelo usado por Galileo, ilusiones ópticas creadas por el propio telescopio a imagen y semejanza de un caleidoscopio.

En ese contexto se podría afirmar que un modelo de computación viene a ser como un telescopio virtual que, en lugar de analizar la inmensidad del universo, se centra en alguna parte específica del mismo con el objetivo de conocerla hasta el más mínimo detalle, a fin de comprenderla íntegramente e, incluso, predecir su comportamiento.

2.4. Diferentes aproximaciones de modelos formales

En esta sección se va a hacer un pequeño recorrido sobre los distintos marcos de modelización formal de sistemas dinámicos complejos que aparecen en el mundo real.

Los sistemas de ecuaciones diferenciales ordinarias o en derivadas parciales (ODEs/PDEs) constituyeron el primer marco para esa modelización, lo cual era hasta cierto punto natural debido a que las ecuaciones diferenciales habían

sido utilizadas con éxito para el estudio y análisis de procesos dinámicos en general, complejos, a través de una aproximación macroscópica, continua y determinista. No obstante, este enfoque llegaba a ser cuestionable en ciertos casos; por ejemplo, en sistemas celulares con un bajo número de moléculas, en reacciones lentas o en estructuras no homogéneas [12, 65]. Además, esta aproximación no puede utilizar un número de variables excesivamente grande debido a la dificultad en esos casos para implementar los métodos de aproximación numérica que son indispensables para poder manejar el modelo diseñado mediante un sistema de ecuaciones diferenciales ordinarias. Todo ello restringe bastante el marco de actuación a la hora de modelizar, por ejemplo, ciertas rutas señalizadoras de proteínas, mecanismos moleculares de bacterias o dinámica de poblaciones, en las que intervienen un elevado número de elementos.

Por otra parte, la complejidad de los procesos biológicos y la dinámica de poblaciones hacen necesario el uso de ordenadores electrónicos para desvelar los mecanismos y funcionalidades subyacentes a dichos procesos y poblaciones. En el caso de modelos continuos basados en ODEs/PDEs es necesario realizar una aproximación numérica que permita manejar el modelo a fin de extraer información acerca del sistema que se estudia, mientras que en la mayoría de los modelos computacionales es posible su implementación *ad hoc* a través de las propias máquinas virtuales de dichos modelos. En las dos últimas décadas, se han propuesto nuevos formalismos bien fundamentados para el diseño de modelos computacionales que pueden servir de novedosos marcos para la modelización de sistemas dinámicos complejos. En esta sección se presentan brevemente las aproximaciones computacionales más extendidas: los sistemas basados en agentes [73], las redes de Petri [66], y el álgebra de procesos (π -cálculo [138]). No obstante, en estas aproximaciones no se tienen en cuenta estructuras jerárquicas y compartimentalizadas que aparecen de manera natural en organismos vivos como, por ejemplo, las derivadas de las membranas biológicas en el funcionamiento de las células.

2.4.1. Modelos basados en ODEs/PDEs

Como ya se ha comentado, los sistemas de ecuaciones diferenciales ordinarias o en derivadas parciales (ODEs/PDEs) han sido utilizados, tradicionalmente, para el estudio de procesos dinámicos y, por esta razón, constituyeron el primer marco de modelización formal para el análisis de procesos biológicos. En particular, hoy día, las ODEs/PDEs representan, sin lugar a dudas, la aproximación más usada para modelizar mecanismos moleculares, funciones

celulares, dinámica de poblaciones, en general, y ecosistemas reales, en particular. Ahora bien, conviene hacer hincapié en la existencia de una serie de restricciones importantes cuando se usan las ODEs/PDEs como marco de modelización; así por ejemplo, en el caso de sistemas celulares los modelos basados en ODEs/PDEs parten de dos supuestos o hipótesis básicas:

- *Las células tienen volúmenes homogéneos y las concentraciones no cambian con respecto al espacio;* es decir, se presupone que las células son volúmenes homogéneos y bien mezclados (el número de moléculas está uniformemente distribuido en el volumen). Desde luego, este supuesto depende de las unidades de tiempo y espacio con las que se trabaje. En el caso de las bacterias se discute la validez de esta hipótesis ya que se ha probado que la difusión es suficiente para que una proteína o molécula recorra en segundos todo el volumen de la bacteria. La hipótesis de homogeneidad espacial no se suele cumplir, por lo general, en células eucariotas, ni en muchos fenómenos biológicos; por ejemplo, en aquellos procesos en los que el tiempo para que ciertas moléculas se desplacen a través del sistema sea grande en comparación con el tiempo (promedio) que tardan en producirse las reacciones químicas.
- *Las concentraciones químicas varían continuamente a lo largo del tiempo y de manera determinista.* Este supuesto es válido si el número de moléculas en el volumen que reacciona es suficientemente grande (al menos, miles) y las reacciones son rápidas. Sin embargo, en sistemas con pocas moléculas (centenares de moléculas es un número considerado pequeño) las interacciones moleculares tienen lugar de forma discreta y están separadas por intervalos de tiempo no constantes.

Esas condiciones no se satisfacen, por ejemplo, en sistemas celulares con un número pequeño de moléculas o sujetos a reacciones lentas, o en sistemas con estructuras organizadas en diferentes compartimentos. No obstante, conviene resaltar el hecho de que las ODEs/PDEs han sido usadas con gran éxito como marco de modelización formal en diferentes escenarios tanto en la disciplina de la Biología de Sistemas como en el análisis de la evolución de poblaciones de individuos en Ecología y en Economía.

Más concretamente, los sistemas de ecuaciones diferenciales ordinarias han sido ampliamente usados para modelizar la cinética de reacciones químicas *macroscópicas* cuyo objetivo es el análisis de la evolución media de la concentración de las sustancias químicas a través de todo el sistema. En esta

aproximación, el cambio de concentraciones a lo largo del tiempo es descrita para cada especie química, admitiendo implícitamente que la fluctuación en torno al valor medio de la concentración es pequeño en relación con la concentración. Este supuesto de homogeneidad puede ser razonable en determinadas circunstancias pero, como ya se ha comentado, deja de serlo en muchos casos; por ejemplo, cuando el número de moléculas es bajo o la distribución de ciertas moléculas relevantes en la célula no es uniforme. Así pues, aunque el modelo basado en ODEs pueden producir resultados interesantes y útiles bajo ciertas restricciones, también pueden proporcionar una visión distorsionada de lo que está ocurriendo en una célula [19].

Ahora bien, teniendo presente la complejidad de, por ejemplo, algunas rutas señalizadoras de proteínas, con frecuencia es necesario utilizar un sistema de ecuaciones diferenciales con un número muy elevado de variables para modelizar esos procesos e, incluso, la interdependencia entre muchas ecuaciones diferenciales puede hacer muy sensible al modelo en relación con las condiciones iniciales. En este marco es difícil modelizar parámetros tales como el tiempo de retraso y el efecto espacial [129], y pequeños cambios en la topología de la red, pueden exigir cambios sustanciales en muchas de las ecuaciones diferenciales básicas [20]. En estos casos se está analizando la posibilidad de sustituir la aproximación basada en ODEs/PDEs por otra que utilice sistemas de ecuaciones en derivadas parciales que sea útil para estudiar la dinámica de los sistemas espacialmente extendidos sobre escalas que son grandes comparadas con las longitudes de escalas moleculares.

2.4.2. Aproximación basada en agentes

La modelización basada en agentes trata cada componente individual del sistema como una entidad simple (un agente) que tiene asociado un cierto conjunto de reglas que especifica su comportamiento en el sistema. Los agentes tienen la capacidad de interaccionar con el entorno, así como con agentes vecinos, de acuerdo con determinados protocolos.

Los agentes pueden representar cualquier componente de un sistema; así, por ejemplo, para una ruta señalizadora de proteínas, es posible representar mediante un agente desde una simple molécula (receptor, ligando, etc.) hasta una cadena de interacciones moleculares, pasando por los individuos de una cierta población. De esta manera se obtiene un marco de modelización extensible y modular.

Un modelo bioquímico basado en agentes no tiene las restricciones a las que está sujeta la aproximación basada en ODEs/PDEs ya que cualquier número y distribución de moléculas puede ser modelizado, las cuestiones espaciales y los tiempos de retraso en los procesos celulares pueden ser fácilmente incorporados en el modelo, y las interacciones individuales entre agentes no producen la misma *volatilidad* que un sistema de ecuaciones diferenciales interdependientes.

En la última década, los sistemas basados en agentes han sido aplicados al estudio y análisis de distintos sistemas biológicos: comunidades de insectos (M. Holcombe y otros, 2003 [73], M. Gheorghe y otros, 2003 [63], D. Jackson y otros, 2004 [79], [78]), tejido epitelial (D. Walker y otros, 2004 [155]), rutas señalizadoras (M. Pogson y otros, 2006 [129]), migración de células tumorales (L. Dib y otros, 2005 [50]), etc. El *Computational Systems Biology Group* del departamento de Ciencias de la Computación de la Universidad de Sheffield (U.K.) ha sido pionero en este campo.

2.4.3. Redes de Petri

Un sistema bioquímico y una población de individuos pueden ser representados a través de un sistema de eventos discretos cuyas propiedades estructurales pueden resultar de utilidad para la obtención de conclusiones tanto acerca del comportamiento y de la estructura del sistema como de la población original [136].

Las redes de Petri (K.A. Petri, 1962) constituyen una herramienta matemático-computacional para la modelización y el análisis de sistemas de eventos discretos con un comportamiento concurrente. Dichas redes proporcionan un marco formal para la representación de la estructura de un sistema de sucesos discretos, a la vez que permiten simular su comportamiento y establecer ciertas propiedades del mismo.

Una red de Petri consiste en un grafo formado por dos tipos de nodos llamados *lugares* y *transiciones*, respectivamente. Los lugares se conectan a las transiciones y, a su vez, las transiciones a los lugares mediante arcos dirigidos con un peso asociado. Dada una transición se distingue entre lugares de entrada, que son aquellos nodos con arcos que llegan hasta la transición, y lugares de salidas que son aquellos nodos con arcos que salen de la transición y llegan a ellos. Normalmente una red de Petri se representa gráficamente dibujando los lugares como círculos y las transiciones como rectángulos.

Recientemente, el marco de modelización de las redes de Petri ha sido aplicado en diferentes campos de ingeniería de sistemas y en ciencias de la computación. La variante específica de redes de Petri, denominada *place-transition net* (PT-net) ha sido utilizada para modelar sistemas de interacciones moleculares [66, 136].

En este marco computacional sólo se puede realizar un estudio cualitativo de los sistemas moleculares. Con la finalidad de obtener un marco cuantitativo se desarrollaron las redes de Petri estocásticas en las que el tiempo de espera para el disparo de las transiciones se determina según una distribución exponencial cuyo parámetro se calcula, a su vez, de acuerdo con una constante asociada a cada transición y al número de elementos en los lugares de entrada.

2.4.4. Álgebra de procesos, π -cálculo

El π -cálculo fue introducido por R. Milner, J. Parrow y D. Walker como un lenguaje formal para describir procesos móviles que se comunican a través de canales de comunicación [97]. Está considerado como un modelo para sistemas que interaccionan y que poseen una topología dinámica de comunicación. El π -cálculo permite que los canales pasen, como si fueran datos, a través de otros canales y este hecho proporciona una movilidad a los canales, lo cual es un hecho importante que aumenta su potencia expresiva. La semántica del π -cálculo es relativamente simple y está basada en una teoría algebraica tratable. Partiendo de acciones atómicas y procesos más simples, se pueden construir procesos más complejos de muy diversas formas. La evolución de un proceso es descrita en π -cálculo mediante una relación de reducción entre procesos que contienen a aquellas transiciones que pueden ser inferidas a partir de un conjunto de reglas.

Diferentes variantes han sido usadas para modelizar interacciones moleculares [137], redes de genes y para integrar redes moleculares y de genes [127].

En los últimos años se han producido diversos intentos de establecer puentes entre el π -cálculo y los sistemas P. En [89], se han descrito paso por paso los mecanismos de transferencia de la bomba de sodio-potasio (*Na-K pump*) y se han usado herramientas de verificación formal para chequear la validez de esta aproximación. En [18], el funcionamiento de la misma bomba se describe y analiza en el marco de los sistemas P. Para ello, se han introducido nuevos ingredientes tales como una variable de etiquetado de membranas, unas condiciones de activación para las reglas, membranas con doble capa y unas reglas de comunicación específicas.

Al igual que sucede con las redes de Petri ordinarias, este nuevo marco es sólo cualitativo. Por ello, para introducir información cuantitativa se asocia una constante a cada canal de comunicación y, entonces, el tiempo de espera para realizar una comunicación se determina mediante una distribución exponencial cuyo parámetro se calcula utilizando la constante citada y el número de procesos que intentan comunicarse a través de dicho canal.

2.5. Modelos estocásticos versus modelos probabilísticos

La aleatoriedad inherente a los procesos biológicos y a la dinámica de poblaciones, en general, así como el ruido externo y la incertidumbre asociada a los mismos, se captura en los modelos computacionales a través del uso de estrategias estocásticas basadas en el *algoritmo de Gillespie* o de estrategias probabilísticas con algoritmos de simulación *ad hoc*.

2.5.1. Modelos estocásticos

Las descripciones continuas y deterministas son adecuadas sólo en determinadas situaciones, en particular, si es *suficientemente grande* el número de individuos o actores que intervienen en el sistema (el proceso o la población) que se estudia. Como ilustración de esta aproximación presentamos la modelización mesoscópica asociada a ciertos procesos celulares.

A nivel microscópico, el funcionamiento de tales procesos sigue las leyes de la física. Un resultado fundamental de la física teórica es la famosa *ley raíz de n* que viene a decir que el nivel de aleatoriedad o fluctuación de un sistema es inversamente proporcional a la raíz cuadrada del número de individuos que intervienen en el mismo. En consecuencia, en un sistema bioquímico de células vivas con un número pequeño de moléculas de un cierto reactante, el enfoque estocástico y discreto puede ajustarse más y mejor que el enfoque continuo y determinista.

El primer paso para el análisis de descripciones estocásticas de reacciones químicas consiste en definir un conjunto de variables de estados suficientemente complejo de tal manera que los cambios sólo dependan del estado actual. Supondremos que el sistema objeto de estudio ocupa un volumen fijo, constante, V y que se encuentra en equilibrio a una cierta temperatura. Supondremos,

además, que el sistema consta de moléculas de n tipos o especies $\{s_1, \dots, s_n\}$ tales que interactúan entre ellas a través de una serie de reacciones químicas $\{r_1, \dots, r_q\}$

De esta manera, es posible representar la descripción instantánea de un sistema a partir del número de moléculas de cada especie reactante; es decir, el estado del sistema en un instante t se puede representar mediante un vector $\mathbf{X}(t) = (X_1(t), \dots, X_n(t))$, en donde $X_i(t)$ representa el número de moléculas de la especie molecular i en ese instante. El objetivo consiste en estudiar la evolución del vector de estado $\mathbf{X}(t)$ a partir de un estado inicial conocido $\mathbf{X}(t_0)$.

Cada interacción molecular r_j está caracterizada por un *vector de cambio de estado* $\mathbf{v}_j = (v_{1j}, \dots, v_{nj})$ y una constante de *propensidad* $p_j(\mathbf{X}(t))$ asociada a cada estado en un instante determinado. Las componentes v_{ij} ($1 \leq i \leq j$) representan el cambio que se produce en la población molecular de la especie s_i debido a la aplicación de la reacción r_j . Así pues, si en un instante determinado el estado del sistema es $\mathbf{X}(t)$ y se ejecuta una regla r_j , entonces el nuevo estado será $\mathbf{X}(t) + \mathbf{v}_j$.

La propensidad $p_j(\mathbf{X}(t))$ de una regla r_j en un instante t se define de tal manera que $p_j(\mathbf{X}(t))dt$ sea la probabilidad de que tenga lugar una interacción molecular del tipo r_j en el intervalo de tiempo $[t, t + dt]$. Para calcular la propensidad $p_j(\mathbf{X}(t))$, se parte de una *constante cinética* k_j usada en la teoría cinética convencional, como ODE's, la cual depende de las propiedades físicas de las moléculas involucradas así como de otros parámetros físicos. La constante cinética asociada a r_j se suele calcular experimentalmente y, a partir de ella, se calcula una *constante estocástica* c_j que permite hallar la propensidad, a través un proceso que depende del tipo de la reacción química r_j basado en la *ley de acción de masas*. Esta aproximación se denomina *modelización mesoscópica*.

El algoritmo de Gillespie está siendo utilizado con éxito para explorar el espacio de estados asociado a un sistema de ecuaciones diferenciales con tantas ecuaciones como posibles estados (*Chemical Master Equation*, CME). La idea es la siguiente: en lugar de resolver analíticamente la CME, se generan trayectorias de $\mathbf{X}(t)$ usando un algoritmo de Monte Carlo para la simulación estocástica de interacciones moleculares que tienen lugar en un determinado volumen [65] (ver también [64] para algunas mejoras recientes). El algoritmo proporciona un método exacto para la simulación estocástica de sistemas de reacciones bioquímicas. La validez del método ha sido demostrada y, además, este algoritmo se ha usado con éxito en la simulación de una amplia gama de procesos bioquímicos [96]. Además, el algoritmo de Gillespie ha sido usado en

la implementación del π -cálculo estocástico [127, 10], así como en su aplicación a la modelización de procesos biológicos [132].

Supongamos que en un determinado medio acotado m (con un volumen determinado) existe un cierto multiconjunto de sustancias químicas que están sometidas a un conjunto de posibles reacciones químicas r_1, \dots, r_q , de las que se conocen los valores h_1, \dots, h_q (número de posibles combinaciones de los reactantes), así como los valores k_1, \dots, k_q (constantes cinéticas de las reacciones). Entonces, el algoritmo (que, a partir de ahora, denominaremos clásico) de Gillespie nos permite seleccionar la reacción química a ejecutar y su correspondiente *tiempo de espera* necesario para ejecutar esa reacción. El agoritmo puede ser descrito como sigue:

Entrada: Un volumen homogéneo m que contiene unas sustancias químicas sometidas a q reacciones químicas r_1, \dots, r_q .

1. Calcular $a_0 = \sum_{j=1}^q p_j$, siendo $p_j = h_j \cdot k_j$ las propensidades de las reglas.
2. Generar dos números aleatorios b_1 y b_2 uniformemente distribuido sobre el intervalo unidad $(0, 1)$.
3. Elegir el índice j_0 , de la siguiente reacción química, que verifica:

$$\sum_{k=1}^{j_0-1} p_k < b_1 \cdot a_0 \leq \sum_{k=1}^{j_0} p_k.$$

4. Calcular el tiempo de espera de la siguiente reacción: $\tau_{r_{j_0}} = \frac{1}{a_0} \ln\left(\frac{1}{b_2}\right)$.

Salida: el par $(r_{j_0}, \tau_{r_{j_0}})$ compuesto por la reacción química a ejecutar y el tiempo de espera que consume su ejecución.

A partir de aquí, la dinámica del sistema continúa aplicando *una vez* la regla r_{j_0} y actualizando el número de moléculas del medio m . El tiempo de cada paso de transición en la simulación/evolución del sistema se puede considerar no constante, y será determinado en cada iteración en función de la configuración del propio sistema.

2.5.2. Modelos probabilísticos

En la especificación sintáctica de un modelo computacional de tipo probabilístico se suele asociar una constante o una función probabilística cuya variable independiente es el tiempo, a ciertos elementos básicos que controlan la dinámica del sistema (transiciones en el caso de las redes de Petri, canales de comunicación en el π cálculo, o reglas de evolución en el caso de los sistemas basados en agentes o en los sistemas P).

Los valores numéricos asociados a esos elementos del modelo en un instante determinado, representan la probabilidad que cada uno de ellos tiene de ser utilizados para que el sistema evolucione en ese momento. Dichas constantes o funciones son obtenidas experimentalmente o bien son calculadas o estimadas a través de una cierta distribución de probabilidad mediante un proceso de calibración. Posteriormente, se diseña un algoritmo *ad hoc* que ha de describir la semántica del modelo computacional en el que se está trabajando. Esta semántica debe incorporar el tratamiento del carácter aleatorio que está materializado mediante constantes o funciones probabilísticas.

El algoritmo de simulación deberá ser validado experimentalmente contrastando los valores producidos por el algoritmo con aquellos datos que han sido obtenidos a través de experimentos o trabajos de campo. Obviamente, ese contraste deberá realizarse simulando el algoritmo en los escenarios específicos en los que han sido generados los datos. Para ese menester es necesario que el algoritmo sea eficiente hasta el punto de poder ser ejecutado en tiempo prudencial, en los distintos escenarios de interés. En el capítulo siguiente se describirán dos algoritmos de simulación probabilísticos para sistemas P.

3

Modelización computacional basada en Membrane Computing

Desde hace varios años, la disciplina *Membrane Computing (Computación Celular con Membranas)* está siendo utilizada como marco formal para la modelización computacional de fenómenos y sistemas de la vida real que engloba tanto a procesos que tienen lugar “a nivel micro” (molecular, celular, etc) como otros que tienen lugar “a nivel macro” (dinámica de poblaciones, ecosistemas reales, etc.)

En el presente capítulo se presenta un marco formal uniforme que permite describir y representar sistemas complejos de muy diversa índole a través de modelos computacionales dentro de la *Computación Celular con Membranas*. Ahora bien, para realizar análisis y predicciones acerca de los sistemas objeto de estudio es indispensable disponer de herramientas/aplicaciones/plataformas software que implementen o simulen los modelos formales. De esa forma se facilita el manejo de los modelos de computación a través de programas que pueden ser ejecutados en ordenadores convencionales.

El capítulo está estructurado en tres secciones. En la primera de ellas se presenta un marco formal de modelización matemática basado en la Computación Celular con Membranas, que denominamos genéricamente *sistemas P multien-*

tornos. La segunda sección está dedicada a la presentación de dos orientaciones del marco general: una orientación de tipo estocástica (*sistemas P multicompartimentales: multicompartmental P systems*) y una orientación de tipo probabilística (*sistemas P de dinámica de poblaciones: population dynamics P systems*, PDP systems). En la última sección se describen unos algoritmos de simulación que tratan de capturar la semántica de los modelos computacionales con las dos orientaciones antes mencionadas, a saber, el algoritmo multicompartimental de Gillespie para la orientación estocástica y los algoritmos DNDP (*Direct Non-Deterministic distribution with Probabilities*) y DCBA (*Direct distribution based on Consistent Blocks Algorithm*) para la orientación probabilística.

3.1. Sistemas P multientorno

Definición 3.1. Un sistema P multientorno (*multienvironment P system*) de grado (q, m, n) ($q, m, n \geq 1$) y con $T \geq 1$ unidades de tiempo es una tupla

$$(G, \Gamma, \Sigma, \mu, T, \Pi_1, \dots, \Pi_n, \mathcal{R}, E_1, \dots, E_m, \mathcal{R}_E)$$

donde:

- $G = (V, S)$ es un grafo dirigido. Sea $V = \{e_1, \dots, e_m\}$.
- Γ y Σ son alfabetos tales que $\Sigma \subsetneq \Gamma$.
- μ es un árbol enraizado con q nodos (denominados membranas) etiquetados de forma inyectiva por elementos del conjunto $\{1, \dots, q\} \times \{0, +, -\}$. Si la etiqueta de una membrana es (i, α) , entonces notaremos dicha membrana por $[]_i^\alpha$ y diremos que la membrana tiene etiqueta i y carga eléctrica α . La raíz del árbol tiene asociada la etiqueta 1.
- T es un número natural.
- Para cada k , $1 \leq k \leq n$, $\Pi_k = (\Gamma, \mu, \mathcal{M}_{1,k}, \dots, \mathcal{M}_{q,k}, \mathcal{R})$ son sistemas P de grado q tales que: (a) todos ellos tienen la misma estructura de membranas μ ; (b) $\mathcal{M}_{1,k}, \dots, \mathcal{M}_{q,k}$ son multiconjuntos finitos sobre Γ (los multiconjuntos iniciales); y (c) \mathcal{R} es un conjunto finito de reglas de la forma $r \equiv u[v]_i^\alpha \xrightarrow{f_r} u'[v']_i^{\alpha'}$, donde $u, v, u', v' \in M_f(\Gamma)$, $u + v \neq \emptyset$, $1 \leq i \leq q$ y $\alpha, \alpha' \in \{0, +, -\}$. Cada regla r del sistema P tiene asociada una función computable f_r cuyo dominio es $\{1, \dots, T\}$.

- E_1, \dots, E_m son multiconjuntos finitos sobre Σ .

- \mathcal{R}_E es un conjunto finito de reglas del tipo

$$(x)_{e_j} \xrightarrow{p_r} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}} \quad y \quad (\Pi_k)_{e_j} \xrightarrow{p_{r'}} (\Pi_k)_{e_{j'}}$$

donde $x, y_1, \dots, y_h \in \Sigma$, $(e_j, e_{j_l}) \in S$, $1 \leq l \leq h$, $(e_j, e_{j'}) \in S$, $1 \leq j, j' \leq m$, $1 \leq k \leq n$ y $p_r, p_{r'}$ son funciones computables cuyo dominio es $\{1, \dots, T\}$. Además:

- Si $(x)_{e_j} \xrightarrow{p_r} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ es una regla de \mathcal{R}_E , entonces no existe una regla de \mathcal{R} cuya parte izquierda sea del tipo $u[v]_1^\alpha$ con $x \in u$.

Un sistema P multientorno de grado (q, m, n) y con T unidades de tiempo se puede considerar como un conjunto de m entornos e_1, \dots, e_m interconectados por los arcos de un grafo dirigido G y un conjunto de n sistemas P, Π_k , que tienen el mismo “esqueleto” (el mismo alfabeto de trabajo Γ , la misma estructura de membranas μ , y las mismas reglas).

Cada entorno e_j contiene inicialmente un multiconjunto finito E_j de objetos del alfabeto Σ (que denominaremos objetos del entorno e_j). Además, cada sistema Π_k debe estar contenido en algún entorno e_j .

Si un sistema Π_k está contenido en un entorno e_j , entonces inicialmente contiene los multiconjuntos $\mathcal{M}_{1,k}, \dots, \mathcal{M}_{q,k}$ que también dependerán del entorno donde se encuentre. Además, todas las membranas de dicho sistema están inicialmente con carga eléctrica neutra.

En un sistema P multientorno se pueden distinguir dos tipos de reglas: las que están asociadas a los distintos sistemas P contenidos en los entornos, cuyo conjunto notaremos por \mathcal{R} , y las que permiten la comunicación de los objetos entre entornos, así como el movimiento de un sistema P de un cierto entorno a otro, cuyo conjunto notaremos por \mathcal{R}_E . Por tanto, el conjunto total de reglas del sistema P multientorno es $\bigcup_{j=1}^n \mathcal{R}_{\Pi_j} \cup \mathcal{R}_E$. Todas las reglas del sistema tienen asociadas unas funciones computables $f_r, p_r, p_{r'}$ que dependerán del entorno correspondiente. Esas funciones indican la afinidad que tienen para ser aplicadas, en el caso de que sean aplicables y que varias reglas compitan por objetos.

El número natural T representa el tiempo de simulación.

La *semántica* de los sistemas P multientorno está definida a través de un modelo paralelo, no determinista y sincronizado, en el sentido de que admitimos la existencia de un reloj que marca cada uno de los instantes o transiciones

propiciando la dinámica del sistema. A continuación describimos los aspectos semánticos de estos sistemas multientornos.

Una *descripción instantánea* o *configuración* del sistema en un instante t es una tupla formada por: (a) los multiconjuntos de objetos sobre Σ presentes en los m entornos; (b) los sistemas P que están contenidos en cada entorno; (c) los multiconjuntos de objetos sobre Γ contenidos en cada una de las regiones de dichos sistemas P ; y (d) las polarizaciones de las membranas en cada sistema P .

Una regla $r \in \mathcal{R}$ del tipo $u[v]_i^\alpha \xrightarrow{f_r} u'[v']_i^{\alpha'}$ asociada a un sistema Π_k es *aplicable* a una configuración \mathcal{C} en un instante t si en ella la membrana de Π_k etiquetada por i tiene carga eléctrica α y de tal manera que contiene al multiconjunto v y su membrana padre (el entorno en el caso de que la membrana sea la raíz del árbol) contiene el multiconjunto u . La ejecución de dicha regla produce los siguientes efectos: (a) los multiconjuntos de objetos v y u se eliminan de la membrana i y de su membrana padre, respectivamente; (b) de manera simultánea, el multiconjunto u' se añade a la membrana padre de i y el multiconjunto v' se añade a la membrana i ; y (c) la nueva carga eléctrica de la membrana i pasa a ser α' (que puede coincidir con α). Es interesante observar que a la hora de aplicar un cierto conjunto de reglas a una cierta membrana, las cargas eléctricas de la parte derecha de esas reglas deben coincidir; es decir, las reglas aplicadas en un mismo paso de transición deben ser *consistentes*. El valor de f_r en un cierto instante t indica la afinidad que tiene la regla para ser aplicada.

Una regla $r \in \mathcal{R}_E$ del tipo $(x)_{e_j} \xrightarrow{p_r} (y_1)_{e_{j_1}} \dots (y_h)_{e_{j_h}}$ es aplicable a una configuración \mathcal{C} del sistema en un instante t si en ella, el entorno e_j contiene el objeto x . La ejecución de dicha regla produce los siguientes efectos: (a) el objeto x es eliminado del entorno e_j ; y (b) en los entornos e_{j_1}, \dots, e_{j_h} se añaden los objetos y_1, \dots, y_h respectivamente. Téngase presente que en cualquier instante t , $1 \leq t \leq T$, para cada objeto x en el entorno e_j , si existen reglas de comunicación del tipo $(x)_{e_j} \xrightarrow{p_r} (y_1)_{e_{j_1}} \dots (y_h)_{e_{j_h}}$, entonces alguna de ellas será aplicada. En caso de existir varias cuya parte izquierda sea $(x)_{e_j}$, entonces se aplicarán de acuerdo con las correspondientes probabilidades asociadas en ese instante t . Una regla $r \in \mathcal{R}_E$ del tipo $(\Pi_k)_{e_j} \xrightarrow{p_{r'}} (\Pi_k)_{e_{j'}}$ es aplicable al entorno e_j si éste contiene un sistema Π_k . La ejecución de dicha regla provoca que el sistema Π_k pase del entorno e_j al entorno $e_{j'}$.

A partir de aquí, se define de manera natural qué significa pasar de una configuración \mathcal{C} del sistema P multientorno en un instante determinado a otra

configuración \mathcal{C}' en el instante siguiente, mediante la ejecución de un multiconjunto maximal de reglas de acuerdo con las indicaciones antes señaladas. De esta forma, se obtiene un *paso de computación*; es decir, una *transición* desde una configuración a una configuración *siguiente* del sistema. Esto nos conduce de manera natural al concepto de *computación* como sucesión de configuraciones en la forma definida en la sección 2 del capítulo primero.

3.2. Tipos de sistemas P multientorno

Entre los sistemas P multientorno podemos destacar, básicamente, dos tipos que corresponden a marcos diferentes de modelización computacional: los sistemas que tienen una orientación estocástica, denominados *sistemas P multicompartmentales*, y aquellos que tienen una orientación probabilística, denominados *sistemas P de dinámica de poblaciones*. Seguidamente vamos a detallar las características sintácticas específicas de cada uno de ellos, así como unos algoritmos de simulación que permiten implementar las correspondientes semánticas.

3.2.1. Sistemas P multicompartmentales

Los *sistemas P multicompartmentales* constituyen una variante de sistemas P multientorno que proporcionan un marco para la modelización de procesos relacionados con mecanismos moleculares y celulares. En estos sistemas, cada entorno contiene un determinado número de sistemas P que son distribuidos, inicialmente, de manera aleatoria y que trabajan a modo de células. Además, en cada entorno, los sistemas P que contiene tendrán unos multiconjuntos iniciales específicos del entorno.

En estos sistemas la aleatoriedad inherente a los procesos moleculares y celulares se captura a través de una semántica basada en reglas de reescritura que pueden ser aplicadas en determinadas circunstancias y con una cierta propensidad.

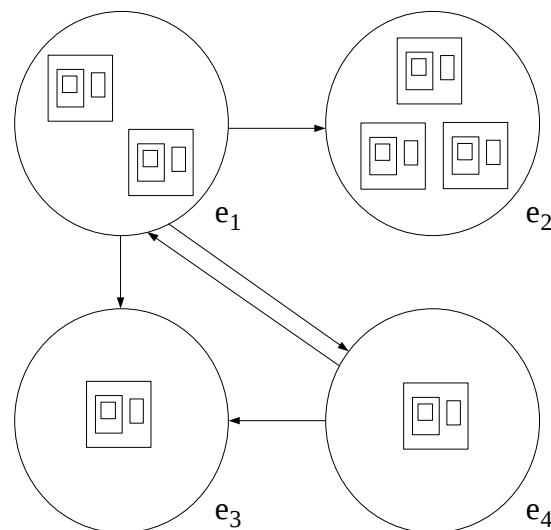
Definición 3.2. *Un sistema P multicompartmental de grado (q, m, n) ($q, m, n \geq 1$) y con $T \geq 1$ unidades de tiempo es un sistema P multientorno de grado (q, m, n) y con T unidades de tiempo*

$$(G, \Gamma, \Sigma, \mu, T, \Pi_1, \dots, \Pi_n, \mathcal{R}, E_1, \dots, E_m, \mathcal{R}_E)$$

que satisface las condiciones siguientes:

- Las funciones computables asociadas a las reglas del entorno y a las reglas de los sistemas P son las propensidades de dichas reglas. Estas funciones se determinan a partir de las constantes estocásticas aplicando la ley de acción de masas. A su vez, las constantes estocásticas asociadas a cada regla se calculan a partir de las constantes cinéticas que han sido calculadas experimentalmente. Las propensidades son funciones que dependen del tiempo pero, en cambio, no dependen del entorno e_j en donde se encuentre el sistema P correspondiente.
- En el instante inicial, los n sistemas P son distribuidos aleatoriamente entre los m entornos del sistema.
- En las reglas del tipo $(x)_{e_j} \xrightarrow{pr} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$, se verifica que $h = 1$; es decir, un objeto x sólo puede pasar de un entorno a otro, posiblemente transformándose en otro objeto y_1 .

Un sistema P multicompartimental puede ser descrito gráficamente como sigue:



La semántica de un sistema P multicompartimental puede ser simulada a través del *algoritmo multicompartimental de Gillespie* que será presentado en la siguiente sección.

3.2.2. Sistemas P de Dinámica de Poblaciones

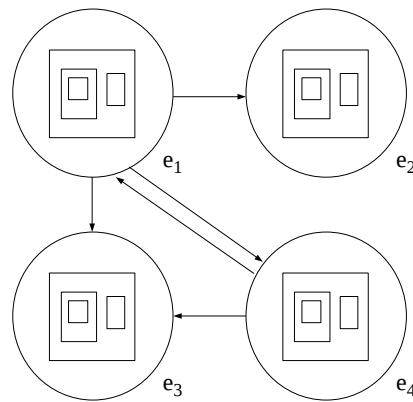
Los *sistemas P de Dinámica de Poblaciones* constituyen una variante de sistemas P multientorno que proporcionan un marco para la modelización de procesos relacionados con la dinámica de poblaciones, en general. Estos sistemas capturan la aleatoriedad inherente a dichos procesos a través de una semántica basada en reglas de reescritura que pueden ser aplicadas en determinadas circunstancias y con una cierta probabilidad. En estos sistemas cada entorno contiene un único sistema P que trabaja a modo de células y los multiconjuntos iniciales de sus membranas son específicos del entorno. Además, cada regla tendrá asociada una función de probabilidad que también será específica del entorno.

Definición 3.3. Un sistema P de dinámica de poblaciones (**sistema PDP**) de grado (q, m) ($q, m \geq 1$) y con $T \geq 1$ unidades de tiempo es un sistema P multientorno de grado (q, m, m) y con T unidades de tiempo

$$(G, \Gamma, \Sigma, \mu, T, \Pi_1, \dots, \Pi_m, \mathcal{R}, E_1, \dots, E_m, \mathcal{R}_E)$$

que satisface las condiciones siguientes:

- En el instante inicial, cada entorno e_j contiene exactamente un sistema P que notaremos por Π_j . Por tanto, el número de sistemas P coincide con el número de entornos.



- Las funciones p_r asociadas a las reglas de \mathcal{R}_E del tipo

$$(x)_{e_j} \xrightarrow{p_r} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$$

tienen rango contenido en $[0, 1]$ y verifican:

- Para cada $e_j \in V$ y $x \in \Sigma$, la suma de las funciones asociadas a las reglas del tipo antes descrito es la función constante 1.
- Las funciones $p_{r'}$ asociadas a las reglas de \mathcal{R}_E del tipo $(\Pi_k)_{e_j} \xrightarrow{p_{r'}} (\Pi_k)_{e_{j'}}$ son constantes e igual a 0; es decir, se puede suponer que este tipo de reglas no existen, propiamente, en el sistema, o lo que es lo mismo, los sistemas P que se encuentran en cada entorno no pueden desplazarse de un entorno a otro.
- Para cada regla $r \in \mathcal{R}$ del sistema Π_j contenido en e_j , $1 \leq j \leq m$, la función computable f_r depende también del entorno (por ello, la notaremos $f_{r,j}$) y su rango está contenido en $[0, 1]$. Además, verifica que para cada $u, v \in M_f(\Gamma)$, $1 \leq i \leq q$ y $\alpha, \alpha' \in \{0, +, -\}$, la suma de las funciones $f_{r,j}$ con $r \equiv u[v]_i^\alpha \rightarrow u'[v']_i^{\alpha'}$, es la función constante 1.

La semántica de un sistema P de dinámica de poblaciones puede ser simulada a través del *algoritmo DNDP* (*Direct Non-Deterministic distribution with Probabilities*) o el *algoritmo DCBA* (*Direct distribution based on Consistent Blocks Algorithm*) que será presentado en la próxima sección.

3.3. Algoritmos de simulación para sistemas P multientorno

En esta sección vamos a presentar sendos algoritmos de simulación que nos va a permitir manejar modelos computacionales de fenómenos de interés que han sido diseñados en el marco de los sistemas P multientorno.

En el caso de los sistemas P estocásticos nos vamos a inspirar en el algoritmo de Gillespie que ha sido utilizado con gran éxito en la simulación de procesos celulares y moleculares. Para ello, se trata de adaptar dicho algoritmo a las especificaciones de los sistemas P multicompartmentales, lo que dará lugar a una extensión del mismo que denominaremos *algoritmo multicompartmental de Gillespie*.

Para los sistemas P de dinámica de poblaciones se presentarán dos algoritmos de simulación que han sido desarrollados en los últimos años: el *algoritmo DNDP* (*Direct Non-Deterministic distribution with Probabilities*) y el *algoritmo DCBA* (*Direct distribution based on Consistent Blocks Algorithm*).

3.3.1. Un algoritmo de simulación para sistemas P multicompartmentales

En la Sección 5.1 del capítulo anterior, hemos señalado que el algoritmo de Gillespie captura la aleatoriedad inherente a la dinámica de los sistemas complejos, en general, y de los procesos biológicos, en particular, así como el ruido externo y la incertidumbre. No obstante, este algoritmo suele aplicarse en un determinado medio acotado (con un volumen determinado) en donde existe un cierto multiconjunto de sustancias químicas que están sometidas a un conjunto de posibles reacciones químicas.

Se trata de usar la estructura del algoritmo de Gillespie en el marco de la computación celular con membranas. Para ello, hemos de extender dicho algoritmo a fin de hacerlo compatible con la estructura compartimentalizada que caracteriza los dispositivos del paradigma computacional citado.

Supongamos que tenemos un sistema P multicompartmental que modeliza un cierto proceso biológico. Específicamente, se dispone de una serie de entornos debidamente interconectados que, a su vez, contienen una serie de estructuras jerarquizadas mediante árboles enraizados (todos ellos idénticos) en la que cada membrana del sistema delimita una región que incluye, propiamente, un volumen específico. Cada uno de esos compartimentos (entornos y membranas) contiene su propio conjunto de reglas (que son abstracciones de las interacciones moleculares) y un multiconjunto de objetos (que son abstracciones de las sustancias químicas).

Por ello, si se desea *adaptar* el algoritmo de Gillespie al marco de los sistemas P multicompartmentales, será necesario desarrollar una extensión del mismo, de tal manera que cada compartimento (entorno/membrana) del sistema pueda evolucionar de acuerdo con el algoritmo de Gillespie clásico, el cual nos devolverá la regla que se va a aplicar en ese compartimento, así como su correspondiente *tiempo de espera*.

En [139], los autores han desarrollado una extensión del algoritmo de Gillespie, (denominado *algoritmo de Gillespie multicompartmental*), que ha sido usado de manera exitosa para la simulación de sistemas P multicompartmentales que modelizan sistemas complejos de interés. El pseudocódigo de dicho algoritmo es el siguiente:

Entrada: Un sistema P multicompartimental II con tiempo de simulación T

- Inicialización

- Poner el tiempo de simulación a $t = 0$.
- Para cada compartimento i en II aplicar el algoritmo clásico de Gillespie a ese compartimento devolviendo una terna $(r_{j_i}, \tau_{r_{j_i}}, i)$, formada por el par de salida del algoritmo citado $(r_{j_i}, \tau_{r_{j_i}})$ y la etiqueta del compartimento.
- Construir una lista formada por todas las ternas anteriormente obtenidas.
- Ordenar la lista de ternas $(r_{j_i}, \tau_{r_{j_i}}, i)$ con relación al tiempo de espera τ (en orden ascendente), formando una pila de reglas a aplicar;

- Iteración

- Elegir la primera terna de la pila, $(r_{j_{i_0}}, \tau_{r_{j_{i_0}}}, i_0)$.
- Poner el tiempo de simulación a $t = t + \tau_{j_{i_0}}$.
- Actualizar el tiempo de espera para el resto de las ternas, restando el valor $\tau_{j_{i_0}}$.
- Aplicar una vez la regla $r_{j_{i_0}}$ al compartimento i_0 modificando el número de objetos en los compartimentos afectados.
- Para cada compartimento i' afectado por la aplicación de la regla $r_{j_{i_0}}$: (a) eliminar la terna correspondiente $(r_{j_{i'}}, \tau_{r_{j_{i'}}}, i')$ de la pila; y
(b) volver a ejecutar el algoritmo de Gillespie clásico en el nuevo contexto, obteniendo una nueva terna $(r'_{j_{i'}}, \tau'_{r_{j_{i'}}}, i')$.
- Añadir la terna $(r'_{j_{i'}}, \tau'_{r_{j_{i'}}}, i')$ a la lista y ordenarla de nuevo de acuerdo con los tiempos de espera, formando una nueva pila.
- Iterar el proceso.

- Finalización

- Si el tiempo de la simulación t alcanza o excede a un tiempo maximal prefijado, finalizar el proceso.

Salida: La configuración del sistema II en el instante T

En esta aproximación, el proceso se inicializa ejecutando el algoritmo de Gillespie clásico en cada uno de los compartimentos (membranas/entornos) que conforman el sistema P multicompartimental. De esta forma se obtiene una lista de ternas (regla, tiempo de espera, compartimento), una para cada

compartimento, que se ordenará según los valores de los tiempos de espera (en orden creciente). De esa lista se selecciona una terna cuyo tiempo de espera asociado sea mínimo (la primera de la lista ordenada). Entonces, en el siguiente paso de computación se ejecutará sólo la regla de la terna elegida y en el compartimento que corresponda. A continuación se actualizarán aquellos valores afectados por la ejecución de esa regla y que están colocados en los compartimentos afectados por esa ejecución. Seguidamente se vuelve a aplicar el algoritmo de Gillespie clásico en cada una de esos compartimentos afectados, a fin de obtener las nuevas ternas asociadas a ellos. Entonces el proceso se itera hasta que se alcance el tiempo de simulación prefijado.

Este generalización del algoritmo de Gillespie clásico se puede aplicar a sistemas biológicos compartmentalizados en donde aparecen distintos volúmenes claramente diferenciados. El número de individuos que intervienen en el sistema objeto de estudio puede o no ser elevado.

El algoritmo multicompartimental de Gillespie ha sido utilizado con éxito en algunos casos interesantes, entre los que destacan: (a) estudio de los mecanismos moleculares del *quorum sensing* en la bacteria *Vibrio Fischeri* [139]; (b) análisis de una ruta señalizadora apoptótica mediatizada por la proteína FAS [34]; y (c) demostración de la robustez de una cascada de señales relacionadas con el factor de crecimiento epidérmico [123].

Es interesante hacer notar que el algoritmo multicompartimental de Gillespie tiene problemas de eficiencia debido a que el algoritmo clásico debe aplicarse en reiteradas ocasiones. Ahora bien, existen casos en los que la aproximación continua y determinista puede ser considerada, proporcionando resultados aceptables. Entonces, en estos casos es posible usar una especie de “versión determinista” del algoritmo multicompartimental de Gillespie, denominada *algoritmo determinista de tiempo de espera*. Este nuevo algoritmo mejora notablemente su eficiencia (para más detalles acerca del algoritmo citado, ver [133])

3.3.2. Dos algoritmos de simulación para sistemas P de dinámica de poblaciones

Un algoritmo de simulación de un sistema P de dinámica de poblaciones (escribiremos brevemente **sistema PDP**) debe ser capaz de representar los distintos ingredientes sintácticos (estructura de membranas, multiconjunto de objetos, reglas, etc.) y de reproducir fielmente la semántica (cómo se aplican

las reglas) del modelo computacional. Entre las propiedades semánticas que deberían de ser reflejadas en un algoritmo de simulación de sistemas PDP destacamos las siguientes: (a) el comportamiento probabilístico; es decir, la reproducción de la aleatoriedad inherente a los procesos de la naturaleza; (b) la competición de recursos; es decir, las reglas dictaminan cómo evolucionan grupos de individuos, y además, cómo éstos compiten entre sí; (c) la maximilidad del modelo; es decir, la capacidad de ejecutar “todas las reglas posibles” en cada paso de computación (tras esa aplicación no deben quedar objetos que puedan evolucionar por alguna regla del sistema); y (c) la consistencia de reglas en cuanto a las cargas de las partes derecha de las reglas aplicadas.

A continuación, vamos a introducir algunos conceptos previos que serán utilizados en la descripción de los algoritmos de simulación. Estos conceptos están asociados a un sistema PDP arbitrario de grado (q, m) con T unidades de tiempo $(G, \Gamma, \Sigma, \mu, T, \Pi_1, \dots, \Pi_n, \mathcal{R}, E_1, \dots, E_m, \mathcal{R}_E)$

Definición 3.4. *La parte izquierda de las reglas se define como sigue:*

(a) *Dada una regla $r \in \mathcal{R}$ de la forma $u[v]_i^\alpha \rightarrow u'[v']_i^{\alpha'}$ donde $1 \leq i \leq q$, $\alpha, \alpha' \in \{0, +, -\}$ y $u, v, u', v' \in M_f(\Gamma)$:*

- *La parte izquierda de r es $LHS(r) = (i, \alpha, u, v)$. La carga de $LHS(r)$ es $charge(LHS(r)) = \alpha$.*
- *La parte derecha de r es $RHS(r) = (i, \alpha', u', v')$. La carga de $RHS(r)$ es $charge(RHS(r)) = \alpha'$.*

La carga de $LHS(r)$ es la segunda componente de la tupla (análogamente para $RHS(r)$).

(b) *Dada una regla $r \in \mathcal{R}_E$ de la forma $(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ donde $e_j \in V$ y $x, y_1, \dots, y_h \in \Sigma$:*

- *La parte izquierda de r es $LHS(r) = (e_j, x)$.*
- *La parte derecha de r es $RHS(r) = (e_{j_1}, y_1) \cdots (e_{j_h}, y_h)$.*

Definición 3.5. *Dados $x \in \Gamma$, $l \in H$, y $r \in \mathcal{R}$ tales que $LHS(r) = (i, \alpha, u, v)$, diremos que (x, l) aparece en $LHS(r)$ con multiplicidad k en cualquiera de los siguientes casos:*

- *$l = i$, y la multiplicidad de x en el multiconjunto v es k.*
- *l es la etiqueta de la membrana padre de i , y la multiplicidad de x en el multiconjunto u es k.*

Definición 3.6. Se dice que un conjunto A de reglas de \mathcal{R} es consistente si para cada $i \in \{1, \dots, q\}$ y cada $\alpha \in \{0, +, -\}$ existe $\alpha' \in \{0, +, -\}$ verificando la siguiente condición: para cada regla $r \in A$ tal que la parte izquierda sea del tipo (i, α, u, v) se tiene que $\text{charge}(\text{RHS}(r)) = \alpha'$.

Dicho con otras palabras, un conjunto de reglas se dice que es consistente si todas las reglas de ese conjunto pueden ser aplicadas de manera simultánea en un paso de computación; es decir, en esa aplicación no se produce ningún conflicto a la hora de asignar nuevas cargas eléctricas a las membranas afectadas.

Definición 3.7. Se dice que una regla $r \in \mathcal{R}$ es consistente con un conjunto A consistente de reglas de \mathcal{R} si el conjunto $A \cup \{r\}$ es un conjunto consistente de reglas.

Es decir, una regla $r \in \mathcal{R}$ es consistente con un conjunto A consistente de reglas de \mathcal{R} si r se puede aplicar simultáneamente con las reglas del conjunto A en un paso de computación, sin que exista conflicto alguno con la asignación de cargas a las membranas afectadas.

3.3.3. Direct Non-Deterministic distribution with Probabilities (DNDP)

En esta sección describimos el primero de los algoritmos propuestos en este trabajo para la simulación de sistemas PDP. Se denomina algoritmo DNDP [44, 45] (Direct Non-Deterministic distribution with Probabilities) y está inspirado en el algoritmo DND [107] con extensión para probabilidades. La entrada es un sistema PDP de grado (q, m) con T unidades de tiempo.

El Algoritmo 3.3.1 describe la rutina principal del algoritmo DNDP. Como se puede observar, el algoritmo está estructurado, básicamente, en tres fases: (a) inicialización; (b) selección; y (c) ejecución.

La fase de inicialización de estructura de datos se describe en el siguiente Algoritmo 3.3.2.

La fase selección se compone de dos “micro fases”: la primera fase (Algoritmo 3.3.3) y la segunda fase (Algoritmo 3.3.4). La primera fase de selección realiza un recorrido de las reglas (sin considerar bloques) de forma aleatoria. Cada vez que se elige una regla, se selecciona un número aleatorio de veces según la distribución binomial con el número de aplicaciones posibles, así como

Algoritmo 3.3.1 PROCEDIMIENTO PRINCIPAL DNDP

Entrada: Un sistema PDP de grado (q, m) con $q \geq 1$, $m \geq 1$, con T unidades de tiempo, $T \geq 1$.

- 1: $C_0 \leftarrow$ la configuración inicial del sistema.
 - 2: **para** $t \leftarrow 0$ to $T - 1$ **hacer**
 - 3: $C'_t \leftarrow C_t$
 - 4: **INICIALIZACIÓN** ▷ (Algoritmo 3.3.2)
 - 5: **PRIMERA FASE DE SELECCIÓN:** consistencia ▷ (Algoritmo 3.3.3)
 - 6: **SEGUNDA FASE DE SELECCIÓN:** maximalidad ▷ (Algoritmo 3.3.4)
 - 7: **EJECUCIÓN** ▷ (Algoritmo 3.3.5)
 - 8: $C_{t+1} \leftarrow C'_t$
 - 9: **fin para**
-

Algoritmo 3.3.2 INICIALIZACIÓN DNDP

- 1: **para** $j \leftarrow 1$ a m **hacer**
 - 2: $R_{E,j} \leftarrow$ conjunto ordenado de reglas de \mathcal{R}_E asociadas con el entorno j
 - 3: $A_j \leftarrow$ conjunto ordenado de reglas de $R_{E,j}$ con probabilidad > 0 en paso t
 - 4: $LC_j \leftarrow$ conjunto ordenado de pares $\langle label, charge \rangle$ para todas las membranas de C_t contenidas en el entorno j
 - 5: $B_j \leftarrow \emptyset$
 - 6: **para cada** $\langle h, \alpha \rangle \in LC_j$ (siguiendo el orden establecido) **hacer**
 - 7: $B_j \leftarrow B_j \cup$ conjunto ordenado de reglas de R_{Π_j} cuya probabilidad es > 0 en el paso t para el entorno j
 - 8: **fin para**
 - 9: **fin para**
-

la probabilidad asociada. Es importante resaltar que la configuración de partida (en realidad se trabajará con una copia de ella) se irá actualizando en el sentido de consumir los objetos que corresponden a la aplicación de las reglas seleccionadas. De esta manera, en ulteriores consultas se sabrá exactamente con qué elementos se dispone a la hora de seleccionar nuevas reglas. Las reglas son elegidas según un orden aleatorio, y cada una se selecciona si su aplicación es consistente con el conjunto de reglas ya seleccionadas. Al final de esta primera fase se obtiene un multiconjunto consistente de reglas aplicables a la configuración actual.

La segunda fase de selección tiene como objetivo garantizar que el conjunto de reglas a aplicar para realizar un paso de computación del sistema, sea maximal. Para ello, se recorre el conjunto de reglas seleccionadas, una a una, de tal manera que si sigue siendo aplicable a la configuración actual, entonces

Algoritmo 3.3.3 PRIMERA FASE DE SELECCIÓN DNDP: CONSISTENCIA

```

1: para  $j \leftarrow 1$  a  $m$  hacer
2:    $R_j \leftarrow \emptyset$ 
3:    $D_j \leftarrow A_j \cup B_j$  con un orden aleatorio
4:   para cada  $r \in D_j$  (siguiendo el orden considerado) hacer
5:      $M \leftarrow$  máximo número de veces que  $r$  es aplicable a  $C'_t$ 
6:     si  $r$  es consistente con las reglas en  $R_j \wedge M > 0$  entonces
7:        $N \leftarrow$  máximo numero de veces que  $r$  es aplicable a  $C_t$ 
8:        $n \leftarrow \min\{M, F_b(N, p_{r,j}(t))\}$ 
9:        $C'_t \leftarrow C'_t - n \cdot LHS(r)$ 
10:       $R_j \leftarrow R_j \cup \{<r, n>\}$ 
11:    fin si
12:  fin para
13: fin para

```

se aplicará el máximo número de veces que sea posible. Al final de esta fase se obtiene un multiconjunto consistente y maximal de reglas aplicables.

Algoritmo 3.3.4 SEGUNDA FASE DE SELECCIÓN: MAXIMALIDAD

```

1: para  $j \leftarrow 1$  a  $m$  hacer
2:    $R_j \leftarrow R_j$  con un orden según la probabilidad de reglas, de mayor a menor
3:   para cada  $<r, n> \in R_j$  (siguiendo el orden seleccionado) hacer
4:     si  $n > 0 \vee (r \text{ es } \textit{consistente} \text{ con las reglas en } R_j)$  entonces
5:        $M \leftarrow$  máximo número de veces que  $r$  es aplicable a  $C'_t$ 
6:       si  $M > 0$  entonces
7:          $R_j \leftarrow R_j \cup \{<r, M>\}$ 
8:          $C'_t \leftarrow C'_t - M \cdot LHS(r)$ 
9:       fin si
10:     fin si
11:   fin para
12: fin para

```

Finalmente, en la fase de ejecución se actualiza la configuración produciendo los objetos que correspondan de acuerdo con las partes derechas de las reglas que se están aplicando en ese paso (y con las multiplicidades correspondientes). A la vez, en este paso se actualizan las cargas de las membranas afectadas por la aplicación de las reglas de acuerdo con las cargas que aparecen en las partes derechas de las reglas aplicadas.

Algoritmo 3.3.5 EJECUCIÓN DNDP

```

1: para cada  $< r, n > \in R_j$ ,  $n > 0$  hacer
2:    $C'_t \leftarrow C'_t + n \cdot RHS(r)$ 
3:   Actualizar la carga eléctrica de  $C'_t$  de acuerdo a  $RHS(r)$ 
4: fin para

```

3.3.4. Direct distribution based on Consistent Blocks Algorithm (DCBA)

En el algoritmo DNDP, las reglas se seleccionan de acuerdo con sus probabilidades de forma uniforme y, por tanto, adolece de un sesgo hacia aquellas que tienen menor probabilidad. Esta sección introduce un algoritmo reciente, conocido como *Direct distribution based on Consistent Blocks Algorithm (DCBA)* [43, 42] que permite subsanar la deficiencia antes mencionada, realizando una distribución uniforme de los recursos a aquellos bloques (conjuntos de reglas que tienen la misma parte izquierda) que compiten por ellos (con intersección en las partes izquierdas). Para ello, se hará uso de un nuevo concepto denominado *bloque consistente*.

El mecanismo de selección del DCBA parte del supuesto de que las reglas en \mathcal{R} y \mathcal{R}_E pueden ser clasificadas en bloques consistentes; es decir, en conjuntos de reglas que, además de tener la misma parte izquierda, son conjuntos consistentes en el sentido de que las reglas de tales bloques pueden ser ejecutadas todas ellas de manera simultánea en un paso de computación.

Definición 3.8. *Las reglas del sistema pueden ser clasificadas en bloques:*

- (a) *Para cada $i \in \{1, \dots, q\}$, $\alpha \in \{0, +, -\}$ y $u, v \in M_f(\Gamma)$, el bloque asociado a la tupla (i, α, u, v) se define como sigue:*

$$B_{i,\alpha,u,v} = \{r \in \mathcal{R} : LHS(r) = (i, \alpha, u, v)\}$$

- (b) *Para cada entorno e_j y cada objeto $x \in \Sigma$, el bloque asociado al par (e_j, x) se define como sigue:*

$$B_{e_j,x} = \{r \in \mathcal{R}_E : LHS(r) = (e_j, x)\}$$

Definición 3.9. *Se dice que un bloque $B_{i,\alpha,u,v}$ es consistente si y solo si existe α' tal que, para cada $r \in B_{i,\alpha,u,v}$, $charge(RHS(r)) = \alpha'$.*

Es decir, un bloque consistente (de reglas de \mathcal{R}) es un conjunto consistente de reglas. Es obvio que todo conjunto de reglas del entorno es consistente en el sentido de que todas ellas pueden ser aplicadas de manera simultánea debido a la no existencia conflicto de cargas.

Definición 3.10. *Dada una tupla $(i, \alpha, \alpha', u, v)$, con $1 \leq i \leq q$, $\alpha, \alpha' \in \{0, +, -\}$, $u, v \in \Gamma^*$, el bloque asociado a dicha tupla es el conjunto siguiente:*

$$B_{i,\alpha,\alpha',u,v} = \{r \in \mathcal{R} : LHS(r) = (i, \alpha, u, v) \wedge charge(RHS(r)) = \alpha'\}$$

Por abuso del lenguaje, diremos que la parte izquierda de un bloque B , que notaremos por $LHS(B)$, es la parte izquierda de cualquier regla del bloque.

Definición 3.11. *Diremos que dos bloques $B_{i_1, \alpha_1, \alpha'_1, u_1, v_1}$ y $B_{i_2, \alpha_2, \alpha'_2, u_2, v_2}$ son mutuamente consistentes si y solo si $(i_1 = i_2 \wedge \alpha_1 = \alpha_2) \Rightarrow (\alpha'_1 = \alpha'_2)$.*

Definición 3.12. *Un conjunto de bloques $\mathcal{B} = \{B^1, B^2, \dots, B^s\}$ es mutuamente consistente si y solo si $\forall i, j$ (B^i y B^j son mutuamente consistentes).*

3.3.5. Pseudocódigo del DCBA

El Algoritmo 3.3.6 describe la rutina principal del algoritmo DCBA. Se observa que el algoritmo está estructurado, básicamente, en tres fases: (a) inicialización; (b) selección; y (c) ejecución.

Algoritmo 3.3.6 PROCEDIMIENTO PRINCIPAL DCBA

Entrada: Un sistema PDP de grado (q, m) con $q \geq 1$, $m \geq 1$, con T unidades de tiempo, $T \geq 1$ y $A \geq 1$ (*Precisión*).

- 1: $C_0 \leftarrow$ la configuración inicial del sistema
 - 2: *INICIALIZACION* ▷ (Algoritmo 3.3.7)
 - 3: **para** $t \leftarrow 1$ **a** T **hacer**
 - 4: Calcular funciones de probabilidad $f_{r,j}(t)$ y $p(t)$.
 - 5: $C'_t \leftarrow C_{t-1}$
 - 6: *SELECCIÓN* de reglas:
 - *FASE 1*: distribución ▷ (Algoritmo 3.3.8)
 - *FASE 2*: maximalidad ▷ (Algoritmo 3.3.9)
 - *FASE 3*: probabilidades ▷ (Algoritmo 3.3.10)
 - 7: *EJECUCIÓN* de reglas. ▷ (Algoritmo 3.3.11)
 - 8: $C_t \leftarrow C'_t$
 - 9: **fin para**
-

Por cuestiones técnicas, en la entrada del algoritmo se considera un parámetro A cuyo objetivo es repetir A veces un determinado ciclo a fin de reducir al mínimo la fase de maximalidad que puede introducir desajustes en los resultados.

En la fase de inicialización (Algoritmo 3.3.7) se construye una *tabla estática* (\mathcal{T}), en cuyas filas se disponen los bloques y en las columnas se disponen los objetos del alfabeto asociado a cada compartimento (membrana/entorno) del sistema. La tabla asocia bloques y objetos a cada membrana según las partes izquierdas de los bloques. Esta información se complementa con los bloques asociados a las reglas de comunicación entre entornos.

Algoritmo 3.3.7 INICIALIZACIÓN

- 1: Construcción de la tabla de *distribución estática* \mathcal{T} :
 - Etiquetas de columna: bloques $B_{i,\alpha,\alpha',u,v}$ de \mathcal{R} .
 - Etiquetas de fila: pares (x, i) , para todo objeto $x \in \Gamma$, y $0 \leq i \leq q$.
 - En cada celda de la tabla poner $\frac{1}{k}$ si el objeto de la fila aparece en la LHS del bloque de la columna con multiplicidad k .
 - 2: **para** $j = 1$ **a** m **hacer** ▷ (Construir las tablas *estáticas expandidas* \mathcal{T}_j)
 - 3: $\mathcal{T}_j \leftarrow \mathcal{T}$. ▷ (Inicializar la tabla con la original \mathcal{T})
 - 4: Añadir una fila para cada bloque $B_{e_j,x}$ de \mathcal{R}_E , con el valor 1 en la fila $(x, 0)$.
 - 5: Inicializar los multiconjuntos $B_{sel}^j \leftarrow \emptyset$ y $R_{sel}^j \leftarrow \emptyset$
 - 6: **fin para**
-

Observación 3.1. *Cada columna de las tablas \mathcal{T}_j contiene la información correspondiente a la parte izquierda de un bloque.*

Observación 3.2. *Cada fila de las tablas \mathcal{T}_j contiene la información relacionada con la competición por objetos: dado un objeto, la fila indica cuáles son los bloques que hacen uso de él para su aplicación.*

En el algoritmo DCBA, la fase de selección se estructura en tres micro fases: (a) fase 1 de distribución (donde se calculan las aplicaciones de los bloques mediante el uso de la tabla de distribución); (b) fase 2 de maximalidad (donde se asegura una aplicación maximal de los bloques seleccionados); y (c) fase 3 de probabilidad (donde se traduce la selección de bloques a selección de reglas mediante el uso de números aleatorios con distribución normal). Refiérase a [43] para una explicación mas detallada con ejemplos de cómo aplicar el algoritmo.

En la fase 1 de distribución, se calcula el número de veces que se aplicará cada bloque, utilizando, para ello, una tabla de distribución.

Esta fase utiliza tres procedimientos de filtros auxiliares. El FILTRO 1 descarta las columnas de la tabla correspondientes a bloques no aplicables por discrepancia de cargas en la parte izquierda y la configuración C'_t . El FILTRO 2 descarta las columnas con objetos en la parte izquierda que no aparecen en C'_t . Finalmente, a fin de ahorrar espacio en la tabla, el FILTRO 3 descarta filas vacías. Estos tres filtros son aplicados al comienzo de la Fase 1, y el resultado es una *tabla dinámica* \mathcal{T}_j^t .

Algoritmo 3.3.8 FASE DE SELECCIÓN 1: DISTRIBUCIÓN

```

1: para  $j = 1$  a  $m$  hacer ▷ (Por cada entorno  $e_j$ )
2:   Aplicar filtros a la tabla  $\mathcal{T}_j$ , usando  $C'_t$  y obteniendo  $\mathcal{T}_j^t$ :
    

- $\mathcal{T}_j^t \leftarrow \mathcal{T}_j$
- FILTRO 1 ( $\mathcal{T}_j^t, C'_t$ ).
- FILTRO 2 ( $\mathcal{T}_j^t, C'_t$ ).
- Comprobar la mutua consistencia para los bloques restantes en  $\mathcal{T}_j^t$ .  
Si existe al menos una inconsistencia entonces reportar el error, y acabar la ejecución (en caso de no activar el paso 3).
- FILTRO 3 ( $\mathcal{T}_j^t, C'_t$ ).


3:   (OPCIONAL) Generar un conjunto  $S_j^t$  de sub-tablas de  $\mathcal{T}_j^t$ , formado por los conjuntos de bloques mutuamente consistentes, de forma maximal en  $\mathcal{T}_j^t$ . Reemplazar  $\mathcal{T}_j^t$  con una tabla elegida aleatoriamente de  $S_j^t$ .
4:    $a \leftarrow 1$ 
5:   repeat
6:     para cada fila  $X$  en  $\mathcal{T}_j^t$  hacer
7:        $RowSum_{X,t,j} \leftarrow$  suma total de valores no nulos en la fila  $X$ .
8:       fin para
9:        $\mathcal{TV}_j^t \leftarrow \mathcal{T}_j^t$  ▷ (Una copia temporal)
10:      para cada posición no nula  $(X, Y)$  en  $\mathcal{T}_j^t$  hacer
11:         $mult_{X,t,j} \leftarrow$  multiplicidad en  $C'_t$  y  $e_j$  del objeto en fila  $X$ .
12:         $\mathcal{TV}_j^t(X, Y) \leftarrow \lfloor mult_{X,t,j} \cdot \frac{(\mathcal{T}_j^t(X, Y))^2}{RowSum_{X,t,j}} \rfloor$ 
13:      fin para
14:      para cada columna no filtrada, etiquetada por el bloque  $B$ , en  $\mathcal{T}_j^t$  hacer
15:         $N_B^a \leftarrow \min_{X \in rows(\mathcal{T}_j^t)}(\mathcal{TV}_j^t(X, B))$  ▷ (El mínimo de la columna)
16:         $B_{sel}^j \leftarrow B_{sel}^j + \{B^{N_B^a}\}$  ▷ (Acumular el valor al total)
17:         $C'_t \leftarrow C'_t - LHS(B) \cdot N_B^a$  ▷ (Eliminar el LHS del bloque)
18:      fin para
19:      FILTRO 2 ( $\mathcal{T}_j^t, C'_t$ )
20:      FILTRO 3 ( $\mathcal{T}_j^t, C'_t$ )
21:       $a \leftarrow a + 1$ 
22:    until ( $a > A$ )  $\vee$  (Todos los mínimos del paso 15 son 0)
23:  fin para

```

En la fase 2 de maximalidad se trata de asegurar que se ejecute un multi-conjunto maximal de reglas del sistema, a través de una aplicación maximal de los bloques seleccionados.

Algoritmo 3.3.9 FASE DE SELECCIÓN 2: MAXIMALIDAD

```

1: para  $j = 1$  a  $m$  hacer                                 $\triangleright$  (Para cada entorno  $e_j$ )
2:   Elegir un orden aleatorio de los bloques restantes en la tabla  $\mathcal{T}_j^t$ .
3:   para cada bloque  $B$ , siguiendo el orden impuesto hacer
4:      $N_B \leftarrow$  número de aplicaciones posibles de  $B$  en  $C'_t$ .
5:      $B_{sel}^j \leftarrow B_{sel}^j + \{B^{N_B}\}$             $\triangleright$  (Acumular el valor al total)
6:      $C'_t \leftarrow C'_t - LHS(B) \cdot N_B$   $\triangleright$  (Eliminar la LHS del bloque  $B$ ,  $N_B$  veces.)
7:   fin para
8: fin para

```

En la fase 3 de probabilidad se traduce la selección de bloques a selección de reglas mediante el uso de números aleatorios con distribución normal.

Algoritmo 3.3.10 FASE DE SELECCIÓN 3: PROBABILIDAD

```

1: para  $j = 1$  a  $m$  hacer                                 $\triangleright$  (Para cada entorno  $e_j$ )
2:   para cada bloque  $B^{N_B} \in B_{sel}^j$  hacer
3:     Calcular  $\{n_1, \dots, n_l\}$ , con una multinomial  $M(N_B, g_1, \dots, g_l)$  según
       las probabilidades de las reglas  $r_1, \dots, r_l$  del bloque.
4:     para  $k = 1$  a  $l$  hacer
5:        $R_{sel}^j \leftarrow R_{sel}^j + \{r_k^{n_k}\}$ .
6:     fin para
7:   fin para
8:    $B_{sel}^j \leftarrow \emptyset$ .                            $\triangleright$  (Útil para la siguiente iteración)
9: fin para

```

Finalmente, el Algoritmo 3.3.11 describe la fase de ejecución

Algoritmo 3.3.11 EJECUCIÓN

```

1: para  $j = 1$  a  $m$  hacer                                 $\triangleright$  (Para cada entorno  $e_j$ )
2:   para cada regla  $r^n \in R_{sel}^j$  hacer           $\triangleright$  (Aplicar las RHS de las reglas)
3:      $C'_t \leftarrow C'_t + n \cdot RHS(r)$ 
4:     Actualizar las cargas eléctricas de  $C'_t$  con  $RHS(r)$ .
5:   fin para
6:    $R_{sel}^j \leftarrow \emptyset$ .                          $\triangleright$  (Útil para la siguiente iteración)
7: fin para

```

En la fase de ejecución se actualiza la configuración produciendo los objetos que correspondan de acuerdo con las partes derechas de las reglas que se están aplicando en ese paso (y con las multiplicidades correspondientes). A la vez, en este paso se actualizan las cargas de las membranas afectadas por la aplicación de las reglas de acuerdo con las cargas que aparecen en las partes derechas de las reglas aplicadas.

Para más información y detalles de los algoritmos introducidos, refiérase a los artículos correspondientes. El algoritmo DNDP se describe en [44], en donde se realiza una comparativa de comportamiento y rendimiento con el algoritmo BBB (Binomial Block Based simulation algorithm), utilizando pLinguaCore y una implementación paralela en Java. De manera adicional, en [45] se presentó una verificación formal del algoritmo DNDP, demostrando la corrección del objetivo de cada módulo (o fase) que forma el algoritmo. En [43, 42] se presentó el algoritmo DCBA. En el mismo se incluyó una validación experimental, tanto del algoritmo DNDP como del DCBA, con el modelo de un ecosistema real relacionado con el quebrantahuesos en el Pirineo Catalán.

Parte II

Cuerpo

4

MeCoSim: Un entorno visual de simulación de propósito general para la computación celular con membranas

El presente capítulo se centra en la descripción del entorno de simulación MeCoSim, software de propósito general en el ámbito de la computación celular con membranas. La primera sección presenta una introducción a la herramienta, partiendo de sus orígenes y estableciendo los objetivos que se pretenden alcanzar. La siguiente sección se centra en la explicación detallada de la estructura y las funcionalidades presentes en MeCoSim. Finalmente, en la tercera sección se establecen los puntos principales de una metodología propuesta para la solución de problemas en el ámbito de sistemas P a través de la modelización y simulación, que se complementa con una batería de funcionalidades adicionales para diseñadores de sistemas P y usuarios finales de las aplicaciones basadas en MeCoSim.

4.1. Introducción y objetivos

Los sistemas P son los dispositivos computacionales del modelo de computación celular con membranas. En la actualidad, estos sistemas no han sido implementados aún ni en medios electrónicos ni en medios bioquímicos; es decir, no existe una máquina real que capture las características semánticas los sistemas P. Se precisaría para ello de una máquina, en principio biológica, que fuera capaz de llevar a cabo las operaciones, las primitivas concebidas dentro del modelo de computación.

La ausencia de máquinas reales capaces de ejecutar las instrucciones en la forma descrita en el modelo teórico no impide, por ejemplo, el estudio detallado de la potencia computacional de los dispositivos, de la eficiencia computacional de distintas variantes del modelo, ni de la verificación y validación formal de determinadas propiedades que eventualmente satisfacen ciertos modelos basados en sistemas P. Tampoco es óbice para conocer el resultado de una computación “ejecutada” sobre el modelo teórico, abstracto. De este modo, dada una cierta variante de sistema P, se podrá realizar la traza de una computación en papel; en caso de variantes no deterministas, probabilísticas o estocásticas, lógicamente, esta computación no será única. En el caso de sistemas P confluentes, el resultado final dependerá exclusivamente de la configuración inicial y, por tanto, será suficiente ejecutar la traza de una computación.

Llevar a cabo manualmente una traza de cada posible computación de un sistema P concreto que suscite nuestro interés resultaría no solamente tedioso sino que, en el caso de instancias de una entidad significativa, con un tamaño grande, la tarea sería inviable en la práctica, por consumir un tiempo excesivo, inabordable. Por ello, tanto si estamos interesados en analizar una variante de sistemas P como si queremos estudiar las propiedades de una solución específica, o la evolución y/o resultado de una computación asociada a una instancia de un problema, deberíamos ejecutar esa computación sobre una máquina real que, al menos, se comporte de acuerdo a los preceptos teóricos del modelo de computación considerado. Para ello, hemos de acudir a las máquinas actualmente existentes, a saber, los ordenadores electrónicos. Sobre estos dispositivos desarrollamos simuladores de las máquinas teóricas tratando de reproducir el comportamiento que tendría la máquina real en términos de computación aunque sin la ventaja de la inspiración biológica de ésta, que implica un paralelismo inherente real, ejecución simultánea de tareas/reacciones químicas, comunicación entre distintos compartimentos jerarquizados, etc.

Un simulador debe reproducir el mismo comportamiento que una máquina virtual, en términos de computación. Ahora bien, es muy diferente plantearse como objetivo mimetizar una computación concreta de la que se haya estudiado la traza, o bien tratar de buscar un diseño genérico que pueda servir para cualquier sistema P de una cierta variante. En este sentido, en el ámbito de los sistemas P se han venido siguiendo dos enfoques principales:

1. Desarrollar un simulador ad-hoc para una solución específica a un problema, bien para una instancia o para distintas instancias del mismo.
2. Desarrollar un simulador general para una determinada variante de sistemas P, capturando todos los ingredientes del modelo de computación teórico.

La mayor parte de los desarrollos realizados por distintos grupos de investigación se ha centrado fundamentalmente en el primero de los enfoques presentados, proporcionando simuladores capaces de resolver problemas concretos, simular soluciones específicas de problemas o incluso instancias concretas. Otras herramientas han estado enfocadas al análisis o la verificación de una solución concreta en el ámbito de los sistemas P.

El segundo enfoque ha recibido con carácter general menor atención fuera del grupo de investigación en Computación Natural de Sevilla, debido a que el esfuerzo de otros grupos se ha focalizado en la consecución de otros objetivos, desde el estudio teórico de las variantes al desarrollo de soluciones que resuelvan problemas de interés. Por nuestra parte, estamos interesados en proporcionar las mejores soluciones posibles siguiendo el segundo enfoque, tratando de desarrollar simuladores del modelo de computación con todos sus ingredientes. Otras iniciativas en este sentido serían aplicaciones como Infobiotics o Meta P-Lab.

Nuestra visión va encaminada incluso un poco más allá, hacia un tercer enfoque: desarrollar simuladores que cubran un espectro de variantes de sistemas P lo más amplio posible. Estas variantes deben cubrir un amplio rango, tanto en términos de estructura (disposición jerárquica –Cell-Like–, grafo plano –Tissue-Like, Spiking, kernel– o grafo cuyos nodos contienen, a su vez, árboles –multientorno, PDP systems–); como en términos de ingredientes contemplados (disolución, división, creación de membranas, catalizadores, cargas, guardas, comunicación symport/antiport, parada, etc.); y también en lo que respecta a la semántica (ejecución secuencial, con prioridades, no determinista maximal/minimal, probabilística, estocástica, etc.).

Para lograr este objetivo, se desarrollaron en el seno del grupo citado un amplio conjunto de simuladores para distintas variantes de sistemas P, todos ellos siguiendo la filosofía antes comentada, que permite la simulación de muchas variantes de sistemas P que operen a modo de células, tejidos o neuronas. La mayor parte de estos simuladores se encuentran disponibles en la biblioteca pLinguaCore.

Dada la naturaleza de propósito general de nuestro enfoque, a la hora de emplear el simulador para realizar la computación de una solución de un problema mediante una variante de sistema P, deberemos comenzar especificando el sistema P a simular y analizar. Para llevar a cabo esta tarea, contamos con el lenguaje P-Lingua, que nos permite especificar una gran cantidad de variantes (ver [8]).

Ambos elementos comentados, tanto el lenguaje de especificación P-Lingua como sus intérpretes/parsers y simuladores incluidos en pLinguaCore, constituyen conjuntamente un completo entorno de programación para *Membrane Computing*, que permite especificar sistemas P también en formato xml, y está abierto a posibles extensiones a formatos adicionales.

Dado un problema abstracto, por ejemplo el problema *SAT*, el sistema P (correspondiente a una determinada variante) que describe una solución al problema se escribirá típicamente en un archivo con extensión *.pli*, contenido la especificación del sistema P. Por ejemplo, en el caso de un sistema P que trabaja a modo de célula, se especificará:

- Su estructura de membranas.
- Su conjunto de reglas.
- Sus multiconjuntos iniciales presentes en las distintas regiones.

Este archivo especificaría el sistema P asociado a una solución del problema y que deberá contener los parámetros de una instancia genérica del problema, a saber, en el caso del problema *SAT*, el número n de variables y el número m de cláusulas concreto, es decir, una instancia de la solución al problema planteado. A partir de esta especificación, nuestro entorno permitirá simular una computación dando valores concretos a los parámetros (en el ejemplo, a n y a m). De esta manera, se proporciona una familia de sistemas P en los que, en nuestro caso, el conjunto de reglas, el alfabeto, los multiconjuntos iniciales e incluso la estructura de membranas dependería de los valores asignados a unos parámetros iniciales.

El procedimiento llevado a cabo habitualmente en el entorno de P-Lingua consiste en la definición de la familia de sistemas P en un fichero con extensión *.pli* que contenga, además, los posibles parámetros, tanto si se trata de parámetros asociados al tamaño de instancia, como *n* y *m* en el ejemplo anterior, como parámetros que especifiquen una y sólo una instancia del problema. De esta forma, la simulación de instancias de distinto tamaño, o de instancias distintas del mismo tamaño, requiere la instanciación del fichero de modelo *.pli*, mediante la asignación de valores a los parámetros. Para cada instancia, se podrá lanzar mediante una línea de comandos un programa que ejecute la simulación de una computación.

A través del entorno descrito, el diseñador del sistema P tiene el control tanto de la definición de la familia, como de la introducción de los valores específicos de los parámetros para cada instancia del problema. Este enfoque ha demostrado ser muy útil para la definición de sistemas P o familias en las que el usuario del simulador es precisamente la persona que ha diseñado el sistema P; es decir, casos en los que la persona que va a usar el simulador para resolver su problema es la misma que quien ha diseñado una solución mediante sistemas P.

No obstante, se podría plantear, como de hecho ocurre con frecuencia, que un usuario final necesitara resolver un problema (por ejemplo, la satisfactibilidad de una fórmula proposicional, la posibilidad de colorear un determinado grafo, o analizar la evolución de un ecosistema real) que podría ser abordado exitosamente en el marco de los sistemas P. Este usuario no tiene por qué conocer nada acerca de dicho marco, pero necesita proporcionar los datos específicos de la instancia concreta del problema en el que está interesado (la fórmula proposicional o los vértices y aristas del grafo, en los ejemplos anteriores), de forma que se generen internamente de forma transparente al usuario los parámetros correspondientes para instanciar un sistema P que resuelva la instancia.

Por otra parte, sería deseable que el propio diseñador de sistemas P pudiera disponer con carácter general de facilidades para dotar de distintas instancias del problema a los simuladores, más allá de los archivos de texto con formato *.pli*, facilitando así la prueba y depuración de los modelos. Por tanto, sería interesante disponer de entornos de simulación de más alto nivel, en lugar de acudir a la línea de comandos o tener que modificar el archivo *.pli* para cada instancia. Asimismo, también sería conveniente aportar mecanismos para el análisis visual de los sistemas P, al margen de la salida textual proporcionada por pLinguaCore.

El doble objetivo planteado, proporcionar un entorno integrado de más alto nivel para la modelización, edición, depuración, simulación, análisis y visualización de sistemas P, así como un entorno de simulación para que los usuarios finales puedan introducir distintas instancias de los problemas, constituye el punto de partida para el desarrollo de un entorno visual de propósito general conocido como **MeCoSim** (*Membrane Computing Simulator*).

La detección de esta necesidad se produjo en el contexto de la simulación de modelos analizando la evolución de ecosistemas reales. Los sistemas P diseñados para simular y estudiar la dinámica de poblaciones del quebrantahuesos en el área pirenaico catalana, especie en peligro de extinción en nuestro país, así como la evolución de la población de larvas e individuos adultos de mejillón cebra en el embalse de Ribarroja, gestionado por la compañía Endesa S.A., precisaron del desarrollo de aplicaciones software que permitieran la introducción de distintos escenarios iniciales y la visualización de determinadas salidas procesadas, mostrando objetos concretos en determinadas regiones de las configuraciones del sistema P a lo largo de las computaciones. Estas aplicaciones de propósito específico requirieron un importante esfuerzo de análisis, diseño y desarrollo de software, y eran necesarias para atacar los citados problemas desde un enfoque de modelización y simulación, tanto para el diseñador de sistemas P como para el usuario final que lanzaba las simulaciones y analizaba las salidas del sistema ante los posibles escenarios de interés para el problema objeto de estudio.

Una vez diseñados diversos modelos de ecosistemas, se identificaron una serie de necesidades comunes. Dado el esfuerzo requerido de desarrollo y mantenimiento adaptativo de software, se concluyó que no era viable asumir el coste de desarrollo y mantenimiento que requería cada problema específico en el ámbito de los ecosistemas. El objetivo planteado, por tanto, no fue otro que diseñar y desarrollar una aplicación software capaz de proporcionar los mecanismos necesarios para la definición de aplicaciones de simulación para cada ecosistema a estudiar, incluyendo interfaces de usuario adaptadas al problema y evitando la necesidad de desarrollos a medida. En este sentido, más que un entorno de simulación, MeCoSim podría ser considerada como un meta-entorno o meta-aplicación de simulación que permite la definición de aplicaciones de simulación para cada problema particular. De forma resumida, podemos considerar que nuestra meta-aplicación, nuestro entorno integrado, nace para proporcionar (de la forma más automática y amigable posible) los mecanismos que permitan definir para cada problema, para cada aplicación de simulación concreta, las necesidades identificadas en los distintos ecosistemas.

1. Mecanismos de *entrada* para indicar los datos del problema, que darán lugar a parámetros específicos para distintas instancias.
2. Mecanismos de generación de *parámetros* del sistema P para cada instancia, a partir de los datos del problema correspondientes a cada instancia.
3. Mecanismos de *salida* para mostrar al usuario la información deseada relativa a cada aplicación en cuestión (tablas de resultados, gráficas, etc.)

Estos primeros requisitos generales para nuestro meta-entorno general de simulación permitirían la definición de aplicaciones de simulación concretas. Toda aplicación de simulación resultante de este enfoque debería, por tanto, incluir las citadas entradas, parámetros y salidas fruto de la correspondiente definición, evitando así la necesidad de desarrollar aplicaciones software a medida para cada problema, y limitando este proceso a la definición anterior haciendo uso de los mecanismos proporcionados.

Más concretamente, para generar estas funcionalidades comunes adaptables a cualquier aplicación, es necesario desarrollar mecanismos para especificar, en cada aplicación particular:

- Qué estructura/organización visual se desea.
- Qué datos de entrada, en forma de tablas visuales en las que el usuario pueda introducir la información.
- Qué parámetros del modelo se generan a partir de las entradas.
- Qué salidas se desean mostrar (qué datos proporcionar, con qué criterios, a partir de los resultados de la simulación) y en qué formato (tablas de salida, gráficos, etc.).

Al igual que un sistema P se puede definir por parte del diseñador mediante un archivo *.pli*, se ha diseñado un mecanismo para la definición de aplicaciones visuales de simulación mediante un archivo *.xls*, es decir, formato de hoja de cálculo que puede generarse desde paquetes ofimáticos como Microsoft Office, Open/Libre Office, etc. Este archivo definirá, por tanto, una aplicación basada en ***MeCoSim***, presentando las entradas, salidas y estructura deseada, de forma que al ejecutarla y arrancar nos permita cargar el sistema P definido en un *.pli*, introducir los datos específicos de una instancia a través de las tablas de entrada (pudiendo guardar estos datos como archivo de escenario *.ec2*, o

cargar uno previamente guardado) y simular el sistema P generado a partir del archivo de modelo *.pli*, junto con los parámetros específicos del escenario particular instanciados a partir de las tablas de entrada. Además, podemos editar el archivo de modelo *.pli* y depurarlo simulando paso a paso.

Uno de los objetivos primordiales de la filosofía subyacente de nuestro enfoque es la separación de los posibles roles implicados en un proceso de modelización y simulación, de tal manera que:

- El *usuario final* proporciona los datos de campo sobre el problema en cuestión, interactúa con el diseñador de sistemas P y le traslada las necesidades de información tanto de entrada (datos a proporcionar sobre el fenómeno objeto de estudio) como de salida (información que estamos interesados en conocer). También se encarga de contrastar los resultados devueltos con los datos experimentales obtenidos en los mismos escenarios, permitiendo así que se puedan validar los modelos basados en sistemas P. Este usuario necesita contar con una aplicación que le permita introducir datos sobre determinados escenarios de interés y obtener resultados. Para este usuario, la aplicación actúa como una caja negra ya que no está interesado en los detalles propios del sistema P.
- El *diseñador* de sistemas P es el encargado de seleccionar la variante de modelo de computación dentro de los sistemas P a emplear, y de especificar el modelo a partir de la abstracción del mundo real, según el fenómeno en el que estamos interesados. Debe interactuar con el usuario final para recabar información sobre el problema, seleccionar una variante adecuada y trabajar intensamente en el diseño de un modelo que aborde el problema estudiado y reproduzca lo más fielmente posible el fenómeno objetivo.

Este diseñador deberá trasladar el modelo a una representación entendible por la máquina; en este caso, mediante un archivo *.pli* en lenguaje P-Lingua. Este modelo deberá ser adaptado de forma que permita recibir distintas instancias de la solución específica al problema en cuestión. Por lo tanto, se necesita recibir de algún modo distintas instancias del problema, para lo que deberá definir una aplicación de simulación en MeCoSim adaptada a la solución al problema a tratar.

Los datos correspondientes a distintas soluciones o distintos problemas serán generalmente diferentes, implicarían distintas necesidades de información, por lo que requerirían la definición de distintas aplicaciones de simulación.

- El *desarrollador* de aplicaciones de simulación debe proporcionar el software capaz de recibir entradas (modelo y datos/parámetros de escenarios de interés para los expertos), simular el sistema P instanciado en el modelo con los datos suministrados y devolver la salida deseada. Este proceso es diferente en cada problema a tratar y, por tanto, requeriría potencialmente el desarrollo de una aplicación virtual para cada problema particular a tratar. Este era el funcionamiento de la familia de aplicaciones Ecosim 1.0, con distintas herramientas desarrolladas para la simulación de la dinámica de distintos ecosistemas reales. En la práctica esto resultaría muy tedioso, y provocaría un *time to market* excesivo desde el diseño del modelo hasta llevar a cabo las simulaciones y obtener los datos depurados. Por lo tanto, la idea es evitar la necesidad de desarrollos a medida, eliminando así la necesidad de contar con un desarrollador de software en el proceso de modelización y simulación.

Para cumplir con este objetivo, se requiere una herramienta de propósito general que pueda adaptarse a las entradas, a la disposición general y a la salida de cada problema particular. Esa es la filosofía subyacente en esta tesis, así como en el resultado fundamental (en términos de su aplicación práctica): **MeCoSim**. Esta herramienta puede ser considerada un meta-simulador, en el sentido de que nos permite generar aplicaciones de simulación adaptadas a cada problema particular. Para ello, debe dotarse de la suficiente flexibilidad para que el diseñador de sistemas P pueda definir una aplicación basada en MeCoSim, conteniendo las entradas, salidas, etc., determinadas por el usuario final, que la empleará para llevar a cabo las simulaciones sobre los escenarios que considere de interés. Junto a ese primer objetivo, MeCoSim pretende proporcionar una batería de funcionalidades para todo tipo de sistemas P, incluyendo la modelización, edición, depuración, simulación, análisis y visualización de resultados. Estas funcionalidades principales pueden ser (y de hecho han sido) extendidas.

4.2. El software 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: (a) desarrollador de software; (b) diseñador de sistemas P; y (c) usuario final de una aplicación de simulación. ¿Qué proporciona el software MeCoSim dentro de este esquema?

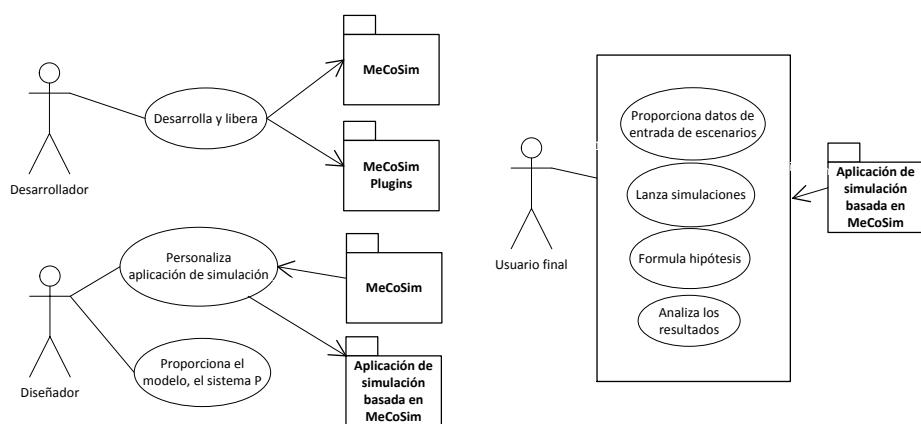


Figura 4.1: Roles y usos de MeCoSim

Como se ilustra en la figura 4.1, el desarrollador de software desarrolla y libera versiones de MeCoSim y/o sus plugins, de forma que tales versiones queden a disposición de cualesquier diseñadores de sistemas P y usuarios finales potenciales.

El diseñador de sistemas P parte de una versión de MeCoSim, acompañada de los plugins que han sido desarrollados. 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. La personalización de aplicaciones de simulación no requiere conocimientos de programación, sino de sistemas P, así como de sus computaciones y simulaciones, de ahí que la lleve a cabo el diseñador.

Por su parte, el usuario final 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 tal manera 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, cuya principal representación será 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, lo cual 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.
- Junto con la disposición/organización en pestañas, las tablas y los gráficos, componentes básicos de la aplicación personalizada en cuan-

to a su interfaz, se deben definir los dos elementos 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, como se describe en el apartado 4.2.1.
- 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 4.2.

Simulación	Ciclo	Paso	Entorno	Etiqueta	Membrana	Objeto	Multiplicidad
------------	-------	------	---------	----------	----------	--------	---------------

Figura 4.2: Salidas - Datos de partida

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 estar interesados en la densidad de larvas de mejillón cerebra, que se puede calcular haciendo uso de datos de simulación correspondiente a una larva con una cierta multiplicidad, junto con un parámetro asociado al volumen de un determinado compartimento. 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.

Al final del proceso, el resultado de aplicar este mecanismo conllevará la generación y ejecución de una consulta **SQL** contra la base de datos. Los detalles de este lenguaje se describen en el apartado 4.2.2.

La aplicación general MeCoSim almacena todas las aplicaciones cargadas previamente en una base de datos interna SQLite, conteniendo la información inicialmente importada del archivo de configuración con extensión *.xls*. Esta base de datos se compone de un sencillo fichero con extensión *.db*.

Cada vez que ejecutamos MeCoSim, aparece el listado de aplicaciones basadas en MeCoSim que tenemos disponibles en nuestra base de datos. Además, existe la opción de añadir una nueva o de actualizar la configuración de una existente a partir de un archivo de configuración *.xls*. La información que se almacena en la base de datos es de la naturaleza que podemos ver en el esquema relacional de la figura 4.3.

Como se puede observar, la aplicación actual en ejecución se almacena, con objeto de facilitar el uso de las funciones posteriores sobre la misma. Además, se almacena la última configuración cargada (haciendo así más eficiente la implantación de una nueva aplicación de simulación sobre un nuevo problema, de tal manera que no haya que buscar el archivo conforme se vayan realizando pequeños cambios hasta tener lista la aplicación).

Al ejecutar cualquiera de las aplicaciones, MeCoSim busca la información asociada a la misma (por *appId*) y genera la aplicación visual correspondiente; es decir, presentando la estructura de pestanas, tablas de entrada, salidas, etc. establecidas en su configuración. Por tanto, cada aplicación visual de simulación será distinta, adaptada a la personalización requerida y reflejada en el archivo de configuración. Esta aplicación estará preparada para cargar un archivo de modelo y los datos propios de un escenario.

A la hora de cargar un modelo en MeCoSim, mediante la opción *Model > Set Model*, los modelos aceptados son archivos basados en P-Lingua (con extensión *.pli*), y por tanto la sintaxis debe ser acorde con la incluida en la versión 4.0 de la biblioteca de software pLinguaCore. Además, la variante de sistemas P especificada en el archivo tiene que ser conforme a los tipos de modelos aceptados en P-Lingua. La biblioteca pLinguaCore contiene un mapeo de los simuladores asociados a cada variante, a través de ficheros xml que asocian ambos.

Así, por ejemplo, si tenemos un PDP system, probabilístico, en P-Lingua se expresará como *@model<probabilistic>*. Al cargarlo en MeCoSim, éste leerá el archivo, lo parseará, y si es un modelo válido no mostrará mensaje de error, de forma que en la ventana aparecerá la ruta del modelo cargado. Además, buscará un archivo que se denomine *probabilistic.xml* contenido los simuladores disponibles para el tipo de modelo seleccionado.

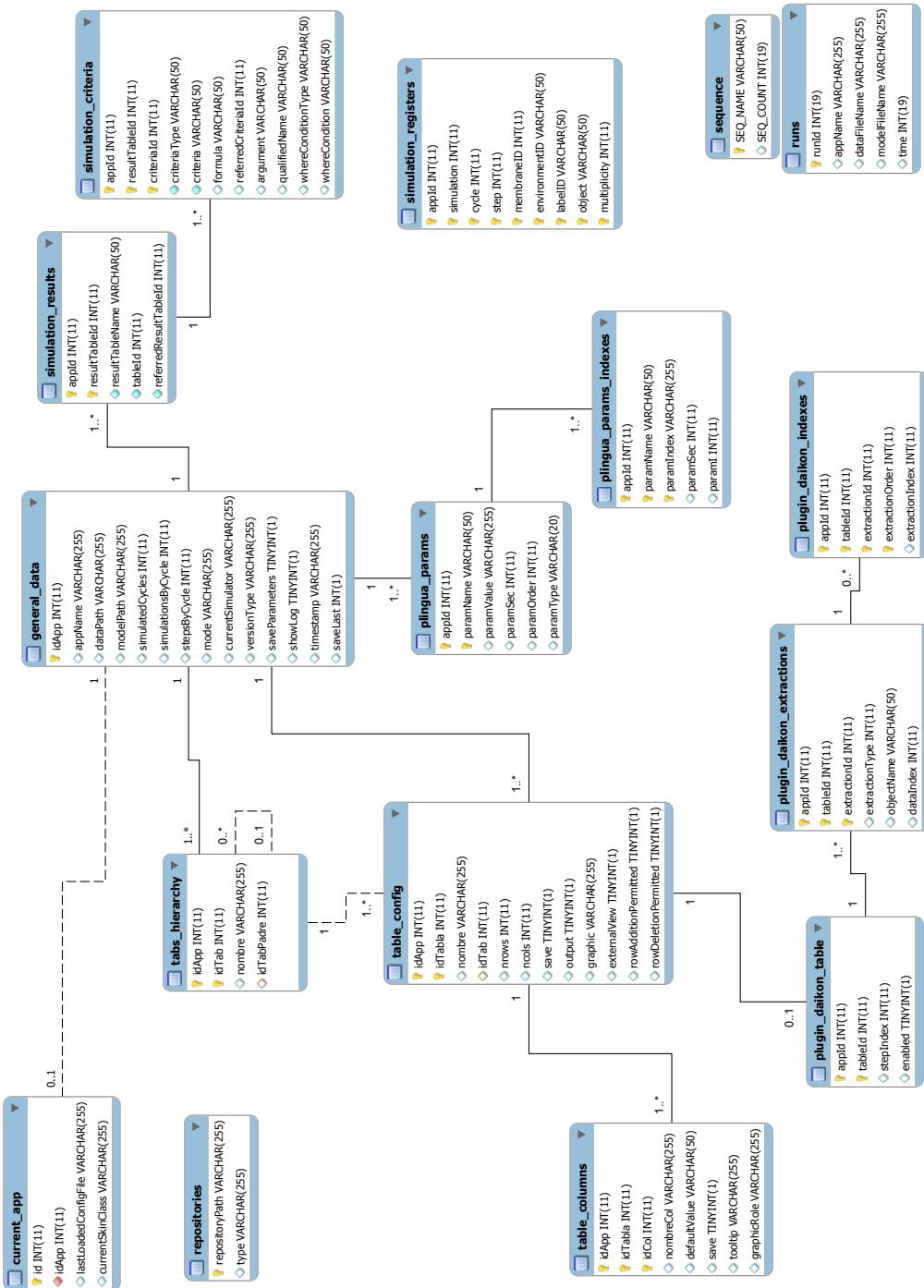


Figura 4.3: Base de datos interna de MeCoSim

El archivo tendrá una forma similar a la siguiente:

```
<?xml version="1.0" encoding="utf-8">
<simulators>
<simulator id="binomial" class=
"org.gcn.plinguacore.simulator.celllike.probabilistic.BinomialProbabilisticSimulator">
</simulator>
...
<simulator id="dcba" class=
"org.gcn.plinguacore.simulator.celllike.probabilistic.DCBAProbabilisticSimulator">
</simulator>
</simulators>
```

Al recorrer el archivo asociado al tipo de modelo cargado, se recopilan los simuladores asociados al mismo, interpretando el último de ellos como simulador preseleccionado, por defecto. Así, para un modelo válido, se dispondrá en el menú (*Simulation > Options > Simulation Algorithm*) de la lista de simuladores posibles, estando preseleccionado el último de ellos; cuando se cambia la opción seleccionada, ésta será recordada como parte de la información de la aplicación. Concretamente, se almacenará en la tabla *general_data*, en el campo *current_simulator*.

Para que la simulación pueda llevarse a cabo, será necesario introducir previamente la instancia concreta a tratar, para el modelo especificado en el archivo con extensión *.pli*. Si éste no depende de datos de entrada concretos, bastará con almacenar los datos de entrada vacíos mediante *Scenario > Save As....* De este modo, la aplicación de simulación entiende que ya se le han proporcionado tanto el modelo como los datos de entrada necesarios, permitiendo por tanto que se lance la simulación a partir de dicha situación de partida.

Así, el primer paso al ejecutar la acción de simular será generar el sistema P correspondiente al escenario y modelo proporcionados, incluyendo la estructura de membranas, la configuración inicial y los conjuntos de reglas asociados a cada compartimento. Esta información dependerá o podrá depender de la instancia particular. Por tanto, será necesario un proceso que incorpore los datos del escenario al archivo del modelo. Este proceso está basado en dos aspectos complementarios:

1. El archivo *.pli* del modelo puede presentar estructuras, reglas y configuración inicial parametrizadas, dependientes de valores de parámetros como *n*, *m*, *p_i*, etc. De este modo, distintos valores de los parámetros instanciarán distintos sistemas P.
2. La aplicación de simulación basada en MeCoSim presenta una entrada personalizada con la información necesaria para un escenario concreto.

Así pues, se deberá establecer un mecanismo que convierta los datos de entrada en valores concretos para los parámetros del modelo, posiblemente empleando transformaciones sobre los datos de entrada durante el proceso. Los detalles de este mecanismo se encuentran explicados en la sección 4.2.1.

A modo de ejemplo breve, podemos ver una parte de una solución del problema de la coloración de grafos con tres colores, mediante sistemas P que trabajan a modo de tejidos [47]. Concretamente, nos centraremos en la fase de división celular, que tiene como finalidad la generación de todas las posibles coloraciones con 3 colores en un grafo de n vértices. Podemos resumir la parte de la solución en la que estamos interesados como sigue:

Sea $\Pi(n) = (\Gamma(n), w_1, w_2(n), \varepsilon, R(n), i_0)$ una familia de sistemas P de tejidos con división celular de grado 2, donde:

1. $\Gamma(n) = \{A_i, R_i, T_i, B_i, G_i : 1 \leq i \leq n\}$
2. $w_1 = \emptyset, w_2(n) = \{A_1, \dots, A_n\}$
3. $R(n)$ es un conjunto de reglas de división:
 - $r_{1,i} \equiv [A_i]_2 \rightarrow [R_i]_2[T_i]_2$ para $i = 1, \dots, n$
 - $r_{2,i} \equiv [T_i]_2 \rightarrow [B_i]_2[G_i]_2$ para $i = 1, \dots, n$

Así, el archivo de modelo *.pli* representando a la familia de sistemas P correspondiente contiene instrucciones como:

```
/* Multiconjunto de la región 2 */
@ms(2) += A{i} : 1<=i<=n;

/* Reglas de división: */
/* r1 */ [A{i}]'2 --> [R{i}]'2 [T{i}]'2 : 1<=i<=n;
/* r2 */ [T{i}]'2 --> [B{i}]'2 [G{i}]'2 : 1<=i<=n;
```

Nótese que el archivo de modelo *.pli* representa una familia de sistemas P que trabajan a modo de tejidos, si bien los diseñadores de sistemas P o los usuarios finales estarán interesados en analizar y simular cada instancia particular del problema abstracto (en nuestro caso, esto se correspondería con analizar el sistema P asociado a cada valor distinto para n).

Tab Id	Tab Name	Tab Parent Id				
1	Tissue Example	0				
2	Input	1				
3	Size (n)	2				
Table Id	Table Name	Tab Id	Columns	Init Rows	Save To File	Input / Output
5	Size (n)		3	1	1	VERDADERO Entrada
Column Id	Column Name	Default Value	Editable	Tooltip	Param Name	Param Value
1	Size		VERDADERO	n	n	<5,1,1>

Figura 4.4: Generación de parámetros - Configuración

¿Cómo se proporciona la capacidad para disponer de una aplicación que acepte distintos valores de n y que, además, genere y simule la solución anterior según el valor de n especificado? Definimos una tabla de entrada mediante el archivo de configuración .xls conteniendo la información que aparece en la figura 4.4, a nivel de jerarquía de pestañas, configuración de tabla, columnas de tabla y generación de parámetros.

Como se observa en la figura 4.4, se define el parámetro n , con el mismo nombre que se emplea en el fichero de P-Lingua. En este caso, n tomará el valor del dato introducido en la primera columna de la primera fila de la tabla con id 5. Con esta configuración anterior almacenada en nuestra aplicación, al ejecutar la simulación del modelo y escenario cargados, si hemos introducido un 3 en la tabla de entrada, el primer paso que realiza la aplicación basada en MeCoSim es generar los parámetros (en este caso n , el único definido en la pestaña *SimulationParams*) a partir del mecanismo de generación. Con este valor ya se podrá instanciar el sistema P parametrizado dado por el archivo de modelo .pli, y comenzar la simulación de una computación del mismo.

Cabe destacar en este punto que MeCoSim proporciona distintos modos de simulación:

- Hasta alcanzar un número prefijado de pasos de computación. Estos pasos se estructuran en simulaciones, cada una compuesta por cierto número de ciclos que, a su vez, constan de un cierto número de pasos. Las distintas simulaciones nos permiten obtener medias, desviaciones, etc. en sistemas probabilísticos, estocásticos o no deterministas, en los que distintas simulaciones pueden arrojar resultados diferentes. Los ciclos se emplean en esta modalidad de simulación para estructurar la dinámica de los sistemas en distintas iteraciones (pueden simbolizar años, días, etc.) sobre procesos que no terminan, y que se encuentran estructurados en una serie de pasos en cada iteración, manteniendo cierta sincronización en el modelo. La interfaz de la aplicación basada en MeCoSim permitirá la modificación del número de simulaciones y el número de ciclos, y, en el caso del usuario diseñador, también se podrá modificar el número de pasos por ciclo.

- b) Hasta alcanzar una configuración de parada. Es decir, hasta que no haya reglas aplicables sobre una determinada configuración del sistema. Se prevé la inclusión de nuevos criterios de terminación adicionales, basados en condiciones específicas que deba cumplir la configuración actual para ser considerada de parada, si bien esto quedaría como una posible ampliación de las capacidades actuales de MeCoSim.
- c) Hasta encontrarnos con un objeto de parada de la computación. Junto a los modos de simulación anteriores, existe la posibilidad de incluir en los modelos algunos objetos de tipo *halt*, cuya aparición provoca la interrupción de la ejecución de la simulación.

Además del mecanismo general de simulación, en las aplicaciones basadas en MeCoSim existe un módulo de depuración, que puede ser ocultado si se considera necesario para su uso por usuarios finales. Se trata de una pestaña denominada “Debug”, que aparece en cada aplicación de simulación, y que proporciona las siguientes funcionalidades:

- *Set Model*: establece el modelo sobre el que trabajar, igual que la opción de menú Model >Set Model.
- *Init Model*: carga los parámetros para el escenario específico, desenrolla las reglas de P-Lingua y genera la configuración inicial. Según el resultado de este proceso, puede ser que tengamos:
 - a) Un modelo junto con el escenario válido, listo para ejecutar paso a paso o bien un número de pasos.
 - b) Un modelo junto con el escenario no válido, en cuyo caso se muestran los errores detectados en la pestaña *Errors*.

En ambos casos se proporcionará, junto con la información anterior, la información concerniente a las pestañas:

- *ParsingInfo*: muestra los parámetros que se van generando, y las reglas desenrolladas que se van obteniendo del modelo.
- *Warnings*: muestra avisos relacionados con el parsing del archivo de modelo para el escenario actual, que no impide la simulación y depuración del modelo.
- *Step*: da un paso de computación a partir de la configuración actual, una vez inicializado el modelo.

- *Run steps*: da un número de pasos a partir de la configuración actual, una vez inicializado el modelo.

Ambas opciones muestran en la pestaña *SimulationInfo* la información asociada a cada configuración en cada paso, así como a cada selección y aplicación de reglas en cada membrana.

Nota: durante la depuración del modelo paso a paso, el estado actual del sistema P puede ser consultado en la pestaña *SimulationInfo*, así como a través de diversos plugins que muestran el alfabeto, la estructura de membranas y los multiconjuntos presentes en cada una de ellas. El mecanismo de plugins se explicará más adelante en la sección 4.2.3.

Se ha explicado el mecanismo general de simulación y de depuración, si bien no se han detallado las capacidades de generación de parámetros ni de obtención de las salidas mediante un mecanismo lo suficientemente flexible para que se adapte a cada problema particular. Estos mecanismos se verán en las secciones 4.2.1 y 4.2.2, respectivamente.

4.2.1. Generación de parámetros

Esta sección detalla el mecanismo de generación de parámetros a partir de los datos específicos de cada escenario particular.

Como se ha comentado en la sección anterior, un archivo de modelo puede contener definiciones de reglas, estructura de membranas y/o configuración inicial dependientes de parámetros que tomen diferentes valores según la instancia concreta del problema a tratar. En el caso de nuestro enfoque, los datos se proporcionan mediante tablas de entrada, de tal manera que el usuario final, ajeno a la existencia de los sistemas P, pueda definir distintos escenarios a simular simplemente modificando los valores introducidos en las tablas visuales de entrada. Por tanto, el mecanismo detallado en este apartado debe establecer la forma de generar los parámetros.

La información sobre la configuración de los parámetros forma parte del archivo *.xls* de la aplicación de simulación, y serán cargados naturalmente en la base de datos. Al inicializar el modelo como parte de la simulación o la depuración, el primer paso será la lectura de la información de la base de datos, desencadenando la generación de los parámetros a partir de la configuración de los mismos, leída de la base de datos.

La forma de los parámetros generados para el modelo P-Lingua se corresponde con la sintaxis de identificadores válidos en lenguaje P-Lingua: por ejemplo, *var*, *x*, *i*. Además de ello se establecen posibles arrays de valores, o variables indexadas, disponiendo de tantos índices numéricos como sea necesario, hasta un máximo de 4

(esto podría ampliarse en el futuro para contemplar un número de índices tan grande como sea necesario). Los índices deben ser expresiones numéricas de tipo entero separadas por comas, y se colocan encerradas entre llaves a la derecha del nombre de la variable. Por ejemplo: $a\{3\}$, $a\{7, 10\}$, $s\{a\{i\}, 2\}$. Estos parámetros se generarán a partir de la configuración de la base de datos, que contiene los siguientes datos en la pestaña *SimulationParams*:

- *Param name*: nombre del parámetro a generar. Debe ser un identificador válido. Es obligatorio.
- *Param value*: expresión numérica simbolizando el valor del parámetro. Debe ser una expresión válida en un lenguaje de generación de parámetros que describiremos a continuación.
- *Index*: expresión que devuelve un entero o un rango, y que determina los valores que podrá adoptar un determinado índice del parámetro, como se detallará a continuación.

Así, MeCoSim generará un parámetro por cada combinación de valores de los índices, pudiendo éstos estar fijados o constituir rangos. Por ejemplo, se podrían generar los parámetros de tipo: $name = value$, $name\{i\} = value$, $name\{i, j, k, l\} = value$.

Cuando se generan los valores de los parámetros, si se define un índice t como un rango de la forma $[initial_t..final_t]$, entonces se genera un bucle de la forma:

- El valor actual para el índice t va de $initial_t$ a $final_t$ con paso 1; es decir, se generan como valores del índice todos los números enteros entre $initial_t$ y $final_t$.
- En cada paso se genera la siguiente asignación:
 $name\{i, \dots, t, \dots, l\} = expr(i, \dots, t, \dots, l)$, siendo ésta una expresión numérica válida donde el valor actual sobre el índice t puede accederse a través de la expresión t .

Cualesquiera de los índices indicados (i, j, k, l), en caso de ser rango, podría incluir un paso para no generar todos los índices dentro del rango sino cada n pasos. Por ejemplo, un valor de $[1..10, 2]$ para un índice generaría los valores impares entre 1 y 10 para ese índice, dado el paso 2 indicado y el comienzo en 1. Además, teniendo presente que varios índices pueden presentar rangos, pueden presentarse bucles anidados generando parámetros para todas las posibles combinaciones de valores de índices.

Las configuraciones de parámetros, valores e índices deben presentar la sintaxis adecuada, reconocida dentro del lenguaje creado a tal efecto como parte de esta tesis. En la figura 4.5 se resume la gramática del lenguaje, partiendo de tres posibles elementos raíz: valor de parámetro, valor de índice y valor de parámetro de tipo cadena (este último elemento reconocido es muy similar al valor de parámetro, pero no tiene como finalidad la generación de parámetros para el modelo y su simulación sino para la generación de parámetros de tipo cadena, que podrán ser usados posteriormente por determinados plugins como veremos en el apartado 4.2.3).

Esta es la gramática de los parámetros numéricos generados en MeCoSim junto con sus índices, de tal manera que se pueda instanciar el sistema P. Adicionalmente, incorpora los parámetros de texto que también se pueden generar, no siendo empleados en P-Lingua pero pudiendo emplearse tras la simulación (por ejemplo, por algunos plugins como *GraphsPlugin*).

Para generar las herramientas necesarias para el análisis sintáctico conforme a esta gramática, se ha hecho uso en MeCoSim, implementado bajo Java1.6, de JavaCC, que a partir de un archivo con extensión *.jj* nos genera el analizador léxico y sintáctico, y nos permite generar el sistema P asociado al modelo, tras aplicarlo a una instancia concreta proporcionada como entrada.

Hemos visto formalmente la gramática. Analicemos a continuación su funcionamiento de forma concisa:

- Los parámetros del modelo son tuplas $\langle nombre, valor, ind_1, ind_2, ind_3, ind_4 \rangle$, de tal manera que:
 - El nombre es una variable en P-Lingua, y debe corresponderse con un identificador válido.
 - El valor se obtiene a partir de una expresión válida en la gramática anterior, partiendo como raíz de *raiz*, y de ahí a *valor*.
 - Los índices se generan a partir de expresiones válidas en la gramática anterior, a partir del elemento raíz *raiz*, y a partir de ahí a *indice*. Como se aprecia en la gramática, los índices pueden venir representados por expresiones numéricas o por rangos de valores numéricos.
- Los identificadores válidos en P-Lingua son los constituidos por cualquier sucesión finita de caracteres en mayúsculas o minúsculas, dígitos o carácter de subrayado, que no comience por un carácter numérico y que no coincida con una palabra reservada de P-Lingua. Así, se pueden considerar nombres de parámetros válidos: *R*, *P*, *Q*. Si además *P* y *Q* son arrays, *P{\$1\$, \$2\$}* y *Q{\$1\$, \$2\$}* también podrían ser parámetros válidos.

```

    raiz   →   valor
              |   indice
              |   valorDeCadena
    valor  →   number
    indice →   expresion
    expresion →   number
                  |   LSQUARE number POINTS number RSQUARE
    number  →   number2(op2 number2) *
    op2    →   PLUS
              |   MINUS
    number2 →   number1(op1 number1) *
    op1    →   MUL
              |   DIV
              |   MOD
    number1 →   number0(POW number0) *
    number0 →   (PLUS|MINUS)?
                  (constant|cycles|sims|steps|rows|columns|cellValue|
                   var|pLinguaVar|normal|random|mx|func|excel|
                   LPARnumber RPAR)
    constant →   NUMBER
    cycles  →   CYCLES
    sims   →   SIMS
    steps   →   STEPS
    rows    →   ROWS COMMA number GT
    columns →   COLUMNS COMMA number GT
    cellValue →   LT number COMMA number COMMA number GT
    var     →   DOLAR NUMBER dolar
    pLinguaVar →   ID (LBRACE number (COMMA number) * RBRACE)?
    normal  →   NORMAL COMMA number COMMA number COMMA number GT
    random →   RANDOM COMMA number GT
    max    →   MAX COMMA number COMMA number GT
    func   →   FUNC COMMA ID (COMMA number) *
    excel  →   EXCEL COMMA string0 GT
    string0 →   STRING
                  |   stringVariable
    stringVariable →   DOLARIID
    valorDeCadena →   devStringValue
    devStringValue →   (stringCellValue|string0|stringfunc)(PLUS devStringValue)?
    stringCellValue →   LT COMMA number COMMA number COMMA number GT
    stringfunc →   SFUNC COMMA ID (COMMA (number | devStringValue))*

```

Figura 4.5: Resumen de la gramática del lenguaje

- Los valores de los parámetros numéricos podrían ser:

- Números, enteros o reales, posiblemente empleando notación exponencial.
- Expresiones aritméticas habituales como multiplicación (*), división (/), módulo (%), adición (+) o sustracción (-). Naturalmente, los operandos de una operación de este tipo podrían ser a su vez expresiones aritméticas o cualesquiera otras devolviendo valores numéricos. Por ejemplo: 3e-5, $(3^{(4-x)+17})/23$.
- Parámetros definidos previamente, como en este caso x . De este modo, se pueden referenciar valores de variables (o de arrays de variables) previamente calculados.
- Valores numéricos distinguidos, como @cycles, @sims y @steps, simbolizando ciclos, simulaciones y pasos, respectivamente.
- Información propia de las tablas de entrada o salida:
 - Filas de la tabla: $<@r, ID>$.
 - Columnas de la tabla: $<@c, ID>$.
 - Celdas de la tabla, como números: $<ID, i, j>$; devuelve el valor contenido en la celda de la tabla con id ID en la fila i , columna j .
 - Celdas de la tabla, como texto: $<, ID, i, j>$; devuelve el valor contenido en la celda de la tabla con id ID en la fila i , columna j , pero interpretando el valor de forma textual, como texto.
- Funciones predefinidas: se cuenta con una serie de funciones predefinidas, preexistentes, que se pueden emplear en las expresiones numéricas o textuales. Por ejemplo:
 - $<@random, N>$: devuelve un número aleatorio entre 1 y N , según una distribución aleatoria uniforme, siendo N una expresión numérica.
 - $<@max, X, Y>$: devuelve el máximo de las expresiones numéricas X e Y .
 - $<@ncdf, mu, sigma, x>$: devuelve el valor de la función de distribución de probabilidad normal (normal cumulative distribution function) para una distribución normal de parámetros mu , $sigma$ y x , siendo éstas sendas expresiones numéricas.
- Junto con las funciones predefinidas anteriores, se ha dotado al lenguaje de un mecanismo que permite la extensibilidad del mismo para incorporar nuevas funcionalidades, que puedan ser llamadas a través de expresiones del tipo:

- $<@\text{func}, ID, \text{par}_1, \dots, \text{par}_n >$: toma una lista de expresiones numéricas $\text{par}_1, \dots, \text{par}_n$, las evalúa, y llama a la función denominada por ID pasándole los valores de las expresiones numéricas evaluadas como argumentos. Esta función debe devolver un número.
- $<@\text{sfunc}, ID, \text{par}_1, \dots, \text{par}_n >$: muy similar al anterior, pero recibe una lista de elementos, numéricos o no, posiblemente cadenas de texto, y llama a la función denominada ID , que debe devolver una cadena de texto.
- $<@\text{excel}, formula_excel >$: permite escribir como valor del parámetro o expresión una fórmula en Excel. En ese caso, se trataría de evaluar la expresión al generar los parámetros mediante el mecanismo de evaluación de funciones de la biblioteca Jakarta POI, y si no se incluye esta función en dicha biblioteca y nos encontramos en Windows, se calcula mediante un script de Visual Basic.

Así, el mecanismo de extensibilidad basado en el uso de `@func` dota al desarrollador, diseñador o usuario final de la posibilidad de extender la funcionalidad inicial del lenguaje aceptado por MeCoSim para la generación de los parámetros. Las funciones que se pueden aportar deben tomar una serie de argumentos numéricos y devolver igualmente un valor numérico. Para ello, habría que hacer lo siguiente:

1. Desarrollar una clase Java que implemente la interfaz `IAlgorithm`:

```

1 package algorithms;
2
3 import java.util.List;
4
5 public interface IAlgorithm {
6     public Number algorithm (List<Number> params);
7 }
```

Para ello, el único método a implementar será `algorithm`, respetando la firma mostrada. Por ejemplo, la clase asociada a la función *floor* tendría el código:

```

1 package algorithms;
2
3 import java.util.List;
4
5 public class Floor implements IAlgorithm {
6     public Number algorithm (List<Number> params){
7         Number result = 0;
8
9         Number n = params.get(0);
10
11         result = (int) Math.floor(n.doubleValue());
12 }
```

```

12
13         return result;
14     }
15 }
```

2. Dotar a la nueva funcionalidad de un nombre reconocible por el mecanismo de generación de parámetros de MeCoSim, mediante la asociación del nombre de función a la clase en el archivo `ecosim-properties`. Así, para una función como `floor` bastará con indicar `func-floor = algorithms.Floor`.

Un proceso similar se lleva a cabo para las funciones derivadas del mecanismo de extensibilidad basado en el uso de `@sfunc`, pero implementando en este caso la interfaz `IStringAlgorithm`:

```

1 package algorithms;
2
3 import java.util.List;
4
5 public interface IStringAlgorithm {
6     public String stringalgorithm (List<Object> params);
7 }
```

Hasta la fecha, una serie de funciones numéricas están disponibles a través de este mecanismo en la versión estándar de MeCoSim: *floor*, *ceil*, *min*, *eq*, *if*, *abs*, *g*, *log*, *exp*, *initialize*, *caudal*, *renovacion*, *gammanext*, *normalnext*, *poissonnext*, *acumstats*. Además, están disponibles las funciones de texto: *conc*, *color*, *toString*, *grm*.

4.2.2. Generación de salidas

Como se ha descrito en el apartado 4.2, tras instanciar el sistema P a partir de los datos del modelo y del escenario concreto, complementando al anterior con el mecanismo de generación de parámetros, al lanzar la simulación se ejecutará la computación hasta alcanzar la condición de parada establecida (número fijado de pasos o configuración de parada, según lo que corresponda). Las distintas configuraciones se irán almacenando en una base de datos *on-memory*, de tal manera que podamos establecer criterios flexibles para la obtención de salidas a partir de tales datos. Como se indicó también anteriormente, la información almacenada en la base de datos tiene la forma de la figura 4.6.

Simulación	Ciclo	Paso	Entorno	Etiqueta	Membrana	Objeto	Multiplicidad
------------	-------	------	---------	----------	----------	--------	---------------

Figura 4.6: Registros de la simulación en la base de datos.

Teniendo en cuenta esta estructura, en función de los elementos del problema en los que estemos interesados se podrán definir distintas salidas, tanto en forma de tablas como de gráficos, que informen acerca de determinada salida en la configuración final o la evolución de determinados elementos a lo largo de la computación. En el caso de lanzar un número determinado de simulaciones para el mismo escenario, podemos estar interesados en obtener no ya los resultados de una computación sino más bien datos estadísticos tales como medias y desviaciones de determinados elementos entre las distintas simulaciones.

Como se explicara anteriormente, la configuración de las salidas se realiza para cada aplicación de simulación en su archivo .xls asociado, que será el que se cargará en MeCoSim para dar de alta la aplicación. Para lograr la mayor flexibilidad posible de salidas a partir de la información de los registros de la computación en la base de datos, se ha diseñado un mecanismo basado en el lenguaje estándar SQL, inspirado en los trabajos de Codd, Chamberlain y Boyce [32, 37]. Con objeto de abstraer a los diseñadores de sistemas P y usuarios finales de los detalles propios del lenguaje, se diseñó una capa de abstracción superior que permitiera generar las salidas en base a la selección de los datos a mostrar, las condiciones a cumplir y la agrupación de los datos, pero sin llegar a los detalles de sintaxis requeridos por SQL. A partir de la selección del usuario (diseñador o usuario final), a la hora de mostrar las salidas se genera internamente la consulta SQL asociada a la configuración definida en el .xls y cargada en la base de datos de la aplicación. Veamos el mecanismo diseñado para generar salidas a través de una aproximación *top – down*:

- Definir una tabla de salida que contendrá los datos a mostrar. Esto se realiza a través de la pestaña *TablesConfig* que ya se ha visto y que contiene, entre otras cosas, un *IdTabla*, un nombre o el número de columnas de la tabla, junto con el dato Input/Output fijado como “Salida”.
- Declarar un *resultado* en la pestaña *SimulationResults*, indicando un *IdResultado*, un nombre para el mismo, un *IdTabla* en la que deberá mostrarse el resultado obtenido y un *IdResultadoReferenciado*. De este modo, un resultado puede hacer referencia a otro, de tal manera que establezca un nuevo nivel de agregación sobre el anterior (por ejemplo, un resultado puede obtener los datos de distintas simulaciones y otro puede hallar las medias y desviaciones de determinados datos existentes en el resultado anterior). Así, se puede contar con resultados anidados a partir de la estructura básica de la base de datos subyacente. En caso de que un resultado no dependa de ningún resultado previo, el campo *IdResultadoReferenciado* valdrá 0. Además, si un resultado referenciado no se va a mostrar en ninguna tabla de salida sino que sirve únicamente como auxiliar de otro/s, el campo *IdTabla* valdrá 0.

- Definir los criterios para la generación del resultado anterior. Si el resultado tiene como nombre “Simulation”, la configuración se deberá proporcionar en una pestaña de la hoja de cálculo denominada “SD_Simulation”, con una serie de filas para selección, filtrado o agrupación, conteniendo cada una de ellas algunos de los datos dados por el conjunto de la figura 4.7:

[Criteria Id](#) | [Select/Where/Group](#) | [Criteria](#) | [Formula](#) | [ReferredCriteria Id](#) | [Argument](#) | [Qualified Name](#) | [Where/Select condition type](#) | [Where condition](#)

Figura 4.7: Criterios asociados a un resultado.

En resumen, los criterios a establecer deberán decirnos **qué** datos mostrar (*Select*), cumpliendo **qué restricciones** (*Where*) y cómo **agruparlos** (*Group*). Veamos la forma de indicar estos criterios:

- Select*: podemos indicar como objeto de la selección alguno de los datos de partida de la base de datos (ver figura 4.6), algún dato de un *resultado* previo o bien una operación sobre un criterio anterior que actúa como auxiliar (*Auxiliary*). Estos criterios auxiliares deberán haber sido definidos previamente dentro de la misma pestaña de criterios (de la misma forma que los criterios de tipo *Select*, pero cambiando *Select* por *Auxiliary* en el campo correspondiente), de tal manera que podamos hacer referencia a su Id de criterio. Cuando se indica operación sobre un criterio anterior, hacemos referencia a la posibilidad, heredada de SQL, de emplear operadores aritméticos y funciones sobre los datos de la base de datos. Concretamente, podemos emplear los operadores y funciones definidos en la referencia del SGBD subyacente, HSQLDB, como aparece en [3]. Podemos ver un ejemplo de uso de criterios de selección y auxiliares en la figura 4.8.

Criteria Id	Select/Where/Group	Criteria	Formula	Referred Criteria Id	Argument	Qualified Name	Where/Select condition type	Where condition
1 Select		simulation						
2 Select		cycle						
3 Select		environmentID						
4 Select		step						
5 Auxiliary		object						
6 Select	formula	LEFT		5	1 type			
7 Auxiliary	formula	LOCATE		5 }				
8 Auxiliary	formula	-		7	1			
9 Auxiliary	formula	LEFT		5	8			
10 Auxiliary	formula	SUBSTR		9	3			
11 Select	formula	CONVERT		10 INT	speciel			
12 Auxiliary	multiplicity							
13 Select	formula	SUM		12	mult			

Figura 4.8: Criterios de selección asociados a un resultado.

Los cuatro primeros criterios son de tipo *Select*, accediendo a datos básicos existentes en la tabla que contiene los datos de la simulación. El sexto obtiene el primer carácter de cada objeto de la simulación (haciendo uso de la función *LEFT* con *Argument* a 1), apoyándose en el

criterio auxiliar 5, *object*. La referencia al criterio auxiliar viene dada, como vemos, por el valor indicado en el campo *Referred Criteria Id*. El campo *Qualified name* permite asociar un nombre al dato calculado con el criterio *Select*, en este caso *type*. La anidación de criterios auxiliares 7 al 10 permiten la generación del dato de tipo *Select* nombrado *specieI*, obteniendo un número entero tras la conversión de la subcadena de texto que va desde el tercer carácter del objeto hasta llegar al carácter '}'. El dato *condition type* permite establecer el tipo de *Argument* en el caso de emplearse en un criterio de tipo *Select*. Así, los valores *Integer* o *String* hacen mención a los tipos conocidos correspondientes. El valor *Reference* provoca que el argumento se interprete como una referencia al *Id* de otro criterio, comportándose en ese caso de la misma forma que el campo *Referred Criteria Id*. El valor *DirectString* indica que el argumento se debe interpretar como una cadena literal, sin incluir caracteres como comillas en el resultado.

- *Where*: permite el filtrado de los datos a obtener; es decir, no establece qué se va a mostrar sino restricciones sobre los registros de la base de datos a partir de los que los obtendremos. Así, por ejemplo, se puede tener muchos objetos en nuestra computación pero estar interesados en aquellos que empiecen por *C* o por *H*; igualmente, se pueden requerir ciertos datos de todas las configuraciones o solamente cada cierto número de pasos. Esta sección *Where* es la encargada de establecer estos criterios de filtrado, estas restricciones, como se observa en la figura 4.9.

							String	String	String	C%	H%
							Integer	Integer	Integer	0	0
14	WhereAux	object									
15	WhereAux	object									
16	WhereAux	step									
17	Auxiliary	step									
18	WhereAux	formula	NOT		16						
19	WhereAux	formula	MOD		17	14					
20	WhereAux	formula	AND		18	19					
21	WhereAux	formula	OR		14	15					
22	Where	formula	AND		21	20					

Figura 4.9: Criterios de filtrado.

Como se puede observar, existe un único criterio *Where*, el cual debe establecer una condición sobre los datos de la simulación que queremos que aparezcan. Esa condición puede ser simple, como por ejemplo obtener las configuraciones únicamente cada 15 pasos (es decir, $MOD(step, 15) = 0$), pero también puede ser una fórmula lógica empleando operadores lógicos AND, OR, NOT. En ese caso será necesario emplear criterios con condiciones auxiliares parairlas combinando, lo cuál se lleva a cabo mediante criterios del tipo *WhereAux*. Si en alguno de los criterios es necesario emplear una función, como en el caso de *MOD*, sobre algún campo o criterio previo, este criterio previo debe tener el tipo *Auxiliary*.

como en el caso de empleo de funciones en *Select*. En el ejemplo tenemos una condición *Where* (22), conjunción de los criterios 21 y 20. El criterio 21 constituye la disyunción de las condiciones 14 y 15, o lo que es lo mismo, restringirá los registros a mostrar a aquellos cuyo objeto comience por *C{* o bien por *H{*.

- *Group*: establece el nivel de agrupación de los datos a mostrar. Así, suponiendo que se va a lanzar una única simulación, si se desea mostrar el total de individuos de una determinada especie (supongamos representados por objetos *X*) por cada membrana (cada valor de *membraneID*), se deberá indicar como criterios *Select* los datos a mostrar (en este caso, *membraneID* y *COUNT(object)*), de tal manera que el nivel de agrupación sería a nivel de membrana, y por tanto incluiría únicamente un criterio *Group* para *membraneID*. En función del orden en que aparezcan los datos de tipo *Group* en la definición, se ordenarán los datos de la salida obtenida (a modo de *Orderby* en SQL). El ejemplo anterior se completa con la agrupación recogida en la figura 4.10.

23 Group	simulation
24 Group	cycle
25 Group	environmentID
26 Group	step
27 Group	type
28 Group	speciel

Figura 4.10: Criterios de agrupación.

Para una referencia completa de los elementos que se pueden emplear en el mecanismo de salidas se puede consultar la guía rápida de construcción de aplicaciones basadas en MeCoSim, disponible en [4].

Un caso particular de salidas a mostrar al usuario son las salidas gráficas. El mecanismo proporcionado por MeCoSim se basa en la definición de salidas previamente descrita, indicando en caso necesario que la salida dé lugar a un gráfico. Por tanto, si se desea mostrar un gráfico de salida involucrando determinados datos, antes habrá que definir la salida correspondiente que obtenga esos datos. A partir de esa tabla de salida definida, se llevarán a cabo los siguientes pasos para definir el gráfico asociado:

1. Declaración de la tabla como salida: se indica en la pestaña *TablesConfig*, asignando en el campo *Input/Output* como *Output* o como *Salida*, ambos aceptados.
2. Declaración de la salida como gráfico: se indica en la pestaña *TablesConfig*, asignando en el campo *OutputGraphic* algún valor distinto de vacío y de

None. El valor indicado debe ser uno de los nombres de gráfico indicados como válidos en MeCoSim, o bien *Default* (que asigna el gráfico por defecto según el número de columnas presentes en la tabla de salida). Hasta la fecha existen los siguientes tipos de gráficos:

- *PieChart*: gráfico de tarta, representando una tabla de dos columnas a través de un gráfico circular en el que cada *categoría* aparece con un tamaño proporcional a su *valor* en la salida correspondiente.
- *LineChart*: gráfico de líneas, representando a partir de una tabla de tres columnas la evolución de distintas *series* de datos presentando un *valor* para cada *categoría*; por ejemplo, se podría ver la evolución de distintas especies (*serie*) con valores de número de individuos (*valor*) para cada año (*categoría*). Podemos ver un ejemplo en la figura 4.11.

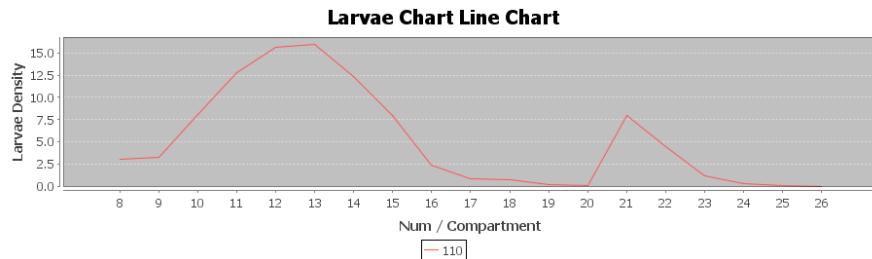


Figura 4.11: Gráfico de líneas.

- *BarChart*: gráfico de barras, representando a partir de una tabla de tres columnas distintos grupos en función de la *categoría* y, dentro de ella, barras verticales de distintos colores identificando diferentes *series* con sus respectivos *valores* para cada valor de la *categoría*. En particular, como en el caso de los gráficos de líneas, se podría ver la evolución de distintas especies (*serie*) con valores de número de individuos (*valor*) para cada año (*categoría*); pero en lugar del año podríamos tener cualquier otro criterio para agrupar, atendiendo a criterios no necesariamente temporales. La figura 4.12 muestra un ejemplo.

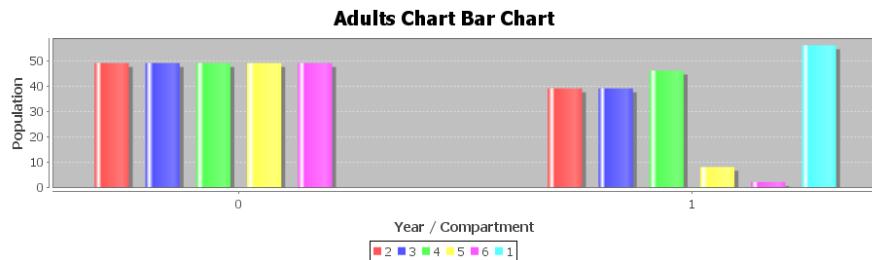


Figura 4.12: Gráfico de columnas.

- *StackedBarChart*: gráfico de columnas apiladas, representando a partir de una tabla de cuatro columnas los datos correspondientes a *valores* agrupados en *categorías* (grupos) divididas en *subcategorías* (barras verticales), cada una incluyendo los valores para distintas *series* de datos a modo de sectores de distintos colores. La figura 4.13 muestra un ejemplo.

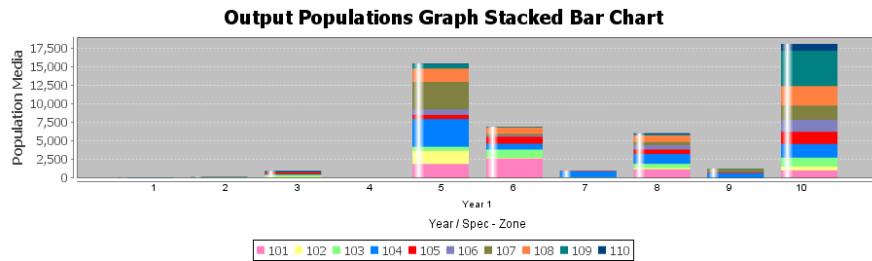


Figura 4.13: Gráfico de columnas apiladas.

- *MultiLineChart*: a partir de una tabla de cuatro columnas, presenta una lista de elementos con los distintos valores de *subcategoría* (siendo ésta una de las columnas de la tabla), de tal manera que al seleccionar cualquiera de ellos nos muestre el gráfico de líneas para esa subcategoría, donde los datos *categoría*, *serie* y *valor* juegan el mismo papel que en el gráfico de tipo *LineChart*. La figura 4.14 muestra un ejemplo.

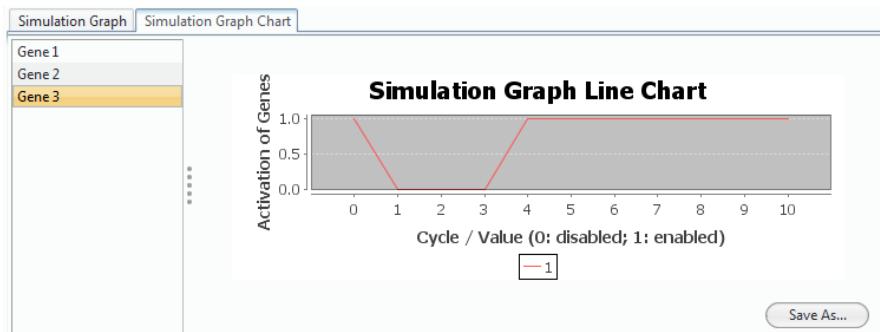


Figura 4.14: Lista de gráficos de líneas.

- *MultiBarChart*: como en el caso anterior, a partir de una tabla de cuatro columnas, presenta una lista de elementos con los distintos valores de *subcategoría* (siendo ésta una de las columnas de la tabla), de tal manera que al seleccionar cualquiera de ellos nos muestre el gráfico de líneas para esa subcategoría, donde los datos *categoría*, *serie* y *valor* juegan el mismo papel que en el gráfico de tipo *BarChart*. La figura 4.15 muestra un ejemplo.

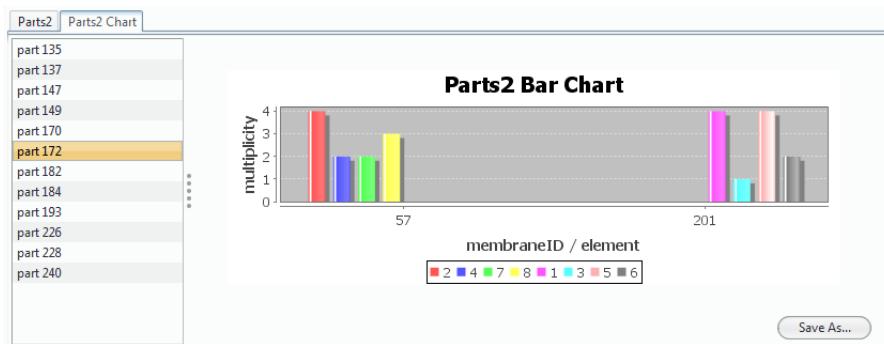


Figura 4.15: Lista de gráficos de columnas.

Estos tipos de gráficos son los existentes a día de hoy, y para que queden disponibles para MeCoSim han de estar incluidos en el archivo de configuración `ecosim-properties`, mediante propiedades del tipo:

$$\text{valid_chart_tipo de gráfico} = \text{True}$$

Como se ha mencionado anteriormente, a la hora de especificar el tipo de gráfico a mostrar, una posibilidad era indicar *Default*. En ese caso, se mostrará el gráfico por defecto asociado según el número de columnas de la tabla. Este gráfico por defecto se indica en el mismo fichero mediante propiedades del tipo:

$$\text{default_chart_textit{número de columnas} = tipo de gráfico}$$

En las definiciones anteriores se puede apreciar una serie de elementos destacados, *categoría*, *subcategoría*, *serie* y *valor*, que desempeñan determinados roles a la hora de visualizar los gráficos a partir de la información de las tablas de salida. La asignación de estos roles se lleva a cabo en la definición de la tabla de salida, asignando en el campo *GraphicRole* las roles *Category*, *Subcategory*, *Series* y *Data*, respectivamente. Podemos ver un ejemplo en la figura 4.16.

Column Id	Column Name	Default Value	Editable	Tooltip	GraphicRole
1	Cycle	VERDADERO	Cycle		Category
2	Gene	VERDADERO	Gene		Subcategory
3	Value (0: disabled; 1: enabled)	VERDADERO	Value (0: disabled; 1: enabled)		Series
4	Activation of Genes	VERDADERO	Activation		Data

Figura 4.16: Definición de gráficos

4.2.3. Plugins

Las funcionalidades generales descritas al comienzo de la sección 4.2 constituyen el punto de partida, el CORE de MeCoSim. A partir de ahí, otras muchas funcionalidades dotan a la aplicación de contenidos adicionales y capacidades de interoperabilidad.

La arquitectura de plugins y el mecanismo para su personalización hacen que se puedan enriquecer fácilmente las funcionalidades disponibles. Permite incorporar funcionalidad adicional implementada en lenguaje Java, de tal manera que estos programas sean lanzados a través de opciones de menú de MeCoSim. También permite incorporar funcionalidades mediante programas externos, con alguno de los mecanismos citados a continuación. Las siguientes funcionalidades están disponibles haciendo uso del mecanismo de extensibilidad articulado por la arquitectura de plugins empleada:

- *MeCoSimBasics plugin*: incluye una ventana para la edición del archivo de modelo P-Lingua (*.pli*). Junto a ello, proporciona ventanas de visualización del alfabeto del sistema P, la estructura de membranas y los multiconjuntos existentes en cada región del sistema.
- *Processes plugin*: sirve para conectar MeCoSim con otros programas externos. Para ello, el programa externo debe ser reconocido por el sistema operativo, bien mediante el motor de ejecución de programas o bien desde línea de mandos. También se puede configurar para disponer de opciones de menú que envíen correos o abran direcciones web determinadas.
- *Graphs plugin*: nos permite definir grafos a partir de información contenida en las entradas o salidas tabuladas. Tiene opciones para mostrar grafos, árboles e incluso árboles de grafos o de árboles bicolor.

Los plugins se activan para su posterior uso mediante un archivo de configuración (**plugins-properties**). Estos plugins contienen funcionalidades que están disponibles para ser llamadas desde puntos de entrada. Para conectar MeCoSim con los puntos de entrada deseados de los plugins es necesario proporcionar en **plugins-properties** lo siguiente:

plugin – nombre del plugin = nombre de la clase Java
pluginname – nombre del plugin = nombre de la opción de menú
pluginmethod – nombre del plugin = nombre del punto de entrada
pluginparam – nombre del plugin – n^o param = parámetro a pasar
pluginjar – nombre del plugin – n^o jar = nombre del jar
pluginorder – nombre del plugin = n^o de orden en el menú

Junto con los plugins básicos mencionados, de carácter general, se han desarrollado algunos plugins que proporcionan funcionalidades algo más específicas:

- *Daikon plugin*: plugin para la extracción de propiedades invariantes sobre los modelos y las configuraciones asociadas a sus computaciones. Daikon es una implementación de la detección dinámica de los probables invariantes y, por tanto, su labor es informar de invariantes que sean probables en un programa. Daikon posee varios mecanismos para extraer propiedades sobre la ejecución de programas escritos en varios lenguajes de programación. En nuestro caso particular, se desarrolló un programa Java incluyendo un paquete con las clases de Daikon y un nuevo paquete, *daikonapplication*, que se encarga de llamar a Daikon con los parámetros de entrada que corresponda. Así, la interfaz de usuario cargará un fichero de traza de datos y generará una salida con los invariantes detectados. Por tanto, la detección se realizará a partir de trazas de programa.

De esta manera, tomando ventaja del mecanismo de configuración de salidas de MeCoSim, se diseñaría una salida adecuada que generara la traza de salida deseada. Además, algunas pestañas adicionales de configuración permiten definir el formato de la traza Daikon a generar. Estas pestañas contienen la siguiente información:

- Información general:
 - *StepIndex*: índice de la columna de la tabla de salida que representa el dato “paso de computación”.
 - *Enabled* (TRUE/FALSE): activa o desactiva la generación de la traza de Daikon.
- Datos de la traza: información sobre cada dato de la traza a generar.
 - *Extraction Id*: identificador asignado al dato o datos a extraer.
 - *Extraction Type*: puede ser *Named*, en cuyo caso se extrae un dato de la salida correspondiente y se le pone un nombre, o bien *Generated*,

en cuyo caso se generan distintos datos de traza a partir de una de las columnas de la tabla de salida. De esta manera, se genera un tipo de dato de la traza por cada valor posible de la columna.

- *Object Name*: dota al dato de un nombre en caso de que éste sea de tipo Named.
- *Extraction Data Index*: establece un orden de salida de los datos dentro de cada fila.
- Datos para la generación de datos: información con los detalles para la extracción de datos de tipo *Generated*. Entre las columnas a aportar con el detalle de la extracción *Generated* tenemos:
 - *Extraction Id*: identificador asignado al dato o datos a extraer.
 - *Extraction Order*: orden de la porción de dato (subdato) dentro del dato a generar.
 - *Extraction Index*: índice del dato de la tabla de salida que ocupará esa posición (orden) en la extracción generada.

Por ejemplo, si se desea obtener una traza para Daikon con el paso y con los datos de cada objeto en cada membrana, partiendo de la tabla de la figura 4.17, se deberá indicar la configuración establecida en la figura 4.18.

Column Id	Column Name	Default Value	Editable	Tooltip
1	step	1	VERDADERO	step
2	membraneID		VERDADERO	membraneID
3	object		VERDADERO	object
4	multiplicity		VERDADERO	multiplicity

Figura 4.17: Daikon - Datos de salida - TableConfig

General Daikon Data		ExtractionDetails	
StepIndex	Enabled	ExtractionId	ExtractionType
1	VERDADERO	1	Named
		2	Generated
ExtractionDetails		ExtractionIndex	
ExtractionId	ExtractionOrder	ExtractionIndex	
2	1	3	1
2	2	2	4

Figura 4.18: Daikon - Extracción - DaikonConfig

Como se observa en la figura 4.18, el paso ocupa la primera posición en la salida, por lo que se indica un 1. La figura 4.17 muestra que se trata efectivamente del dato *step*. El dato *Enabled* a *VERDADERO* indica que esta extracción está activa, luego cuando se ejecute la simulación se generará el fichero de traza correspondiente para que pueda ser seleccionado por el plugin de Daikon para la extracción de propiedades a partir del mismo. Los datos a extraer en la traza son el nombrado como

step, ocupando la primera posición de la tabla de salida (1), y los generados a partir del dato de multiplicidad de objetos (como se indica con un 4, al ser *multiplicity* la cuarta columna de la tabla de salida correspondiente). Cuando se usa el término “los generados”, se está haciendo referencia al hecho de que se generará un dato diferente por cada nombre generado a partir del criterio indicado en el bloque *Extraction details*. En este caso, el dato de *multiplicity* anterior se generará para cada objeto de cada membrana, en el orden objeto - membrana, ya que el *extractionOrder* 1 se corresponde con el *extractionIndex* 3 (dato de la tercera columna de la tabla de salida, *obj*) y el *extractionOrder* 2 se corresponde con el *extractionIndex* 2 (dato de la segunda columna, *memb*). Así, los datos generados comprenderán todas las combinaciones posibles de pares *objeto_membrana*, con nombres como: *A{12}_2*, dado un objeto *A{12}* en la membrana 2. Para cada uno de esos datos, se mostrará en cada fila resultante la multiplicidad de ese objeto en esa membrana.

Al cargar la aplicación en MeCoSim (el archivo de configuración con formato *.xls*), esta información sobre extracciones de Daikon se vuelca sobre la base de datos local. Al lanzar la aplicación, cargar un modelo y proporcionar datos de un escenario específico, se está procediendo a la simulación del mismo. A la finalización de ésta, se obtienen las salidas correspondientes configuradas para la aplicación. Además, si tenemos extracciones para Daikon activas (*Enabled*), se generan ficheros de traza, uno por cada extracción.

Mediante la opción de menú configurada “Daikon”, se puede lanzar una primera ventana que muestra un listado contenido los ficheros extraídos de la simulación. Al seleccionar el deseado, se lanzará una nueva ventana mostrando el contenido del archivo de traza, la traza resultante transformada para Daikon y una salida con las invariantes detectadas.

4.2.4. Repositorios

Tanto las distintas aplicaciones de simulación liberadas como los plugins, modelos y escenarios que hayan sido empleados o desarrollados podrían ser interesantes para otros diseñadores de sistemas P y usuarios de los modelos y sus aplicaciones asociadas. Por ello, se ha diseñado, desarrollado y liberado un sistema de repositorios de:

- *Plugins (.jar)*.
- *Apps (.xls)*.
- *Models (.pli)*.
- *Scenarios (.ec2)*.

Un usuario de MeCoSim puede proporcionar un repositorio de plugins o del tipo que sea del modo siguiente:

- *Plugins*: generando un archivo xml denominado *plugins.xml* similar al siguiente:

```
<plugins>
    <plugin name="MeCoSimBasicsPlugin" jnlp="MeCoSimBasicsPlugin.jnlp"
        path="http://p-lingua.org/mecosim/jnlp/plugins/MeCoSimBasicsPlugin.jnlp"/>
    ...
</plugins>
```

- *Apps*: generando un archivo xml denominado *apps.xml* similar al siguiente:

```
<apps>
    <app name="Tricolor Simple Kernel 1" config="TricolorConfigFile.xls"
        path="http://www.gcn.us.es/redmine/dmsf/files/2951/download"/>
    ...
</apps>
```

- *Models*: generando un archivo xml denominado *models.xml* similar al siguiente:

```
<models>
    <model name="Tricolor Simple Kernel 1" pli="tricolor.pli"
        path="http://www.gcn.us.es/redmine/dmsf/files/2952/download"/>
    ...
</models>
```

- *Scenarios*: generando un archivo xml denominado *scenarios.xml* similar al siguiente:

```
<scenarios>
    <scenario name="Tricolor Simple Kernel 1" pli="3color.ec2"
        path="http://www.gcn.us.es/redmine/dmsf/files/2953/download"/>
    ...
</scenarios>
```

A través de los submenús disponibles en la opción de menú “Manage repositories” se pueden añadir nuevos repositorios de los tipos mencionados, proporcionando la ruta donde está accesible el documento xml correspondiente. Una vez seleccionado uno de los repositorios dados de alta en MeCoSim, se despliegan los archivos correspondientes en forma de listado. De este modo, al instalar el plugin, app, model o scenario, se descargarán e instalarán los archivos correspondientes.

El objetivo de estos mecanismos es poner a disposición de la comunidad, de forma fácil y rápida, los modelos en los que se esté trabajando. Igualmente, permite que otros miembros de la comunidad pongan sus modelos a disposición del resto, fortaleciendo así las posibilidades de trabajo colaborativo.

4.3. Metodología

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 la 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 [87] un primer esbozo de esta metodología para la simulación de modelos, extracción de propiedades y verificación de sistemas P, extendida posteriormente en [62]. 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 análisis del problema a tratar, se lleva a cabo el proceso de abstracción para capturar los datos relevantes para el fenómeno objeto de estudio, hasta diseñar un modelo basado en sistemas P que responda a las cuestiones planteadas en dicho análisis, 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 [38].

Una vez se dispone del modelo sobre un determinado fenómeno, éste puede ser traducido al lenguaje P-Lingua [58] y almacenarse como archivo con extensión *.pli*. Introduciendo en el mismo archivo los valores concretos de los parámetros para un escenario de interés, el modelo estaría listo para trabajar con pLinguaCore o MeCoSim[122].

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 Neural o Simple kernel, pudiendo encontrar un desglose completo en la página de P-Lingua [8]. Así, cuando se carga un modelo, dependiendo de la variante de sistemas P a la que pertenezca dicho modelo, la interfaz de MeCoSim permite elegir entre una serie de posibles simuladores asociados al mismo, como se observa en la figura 4.19.

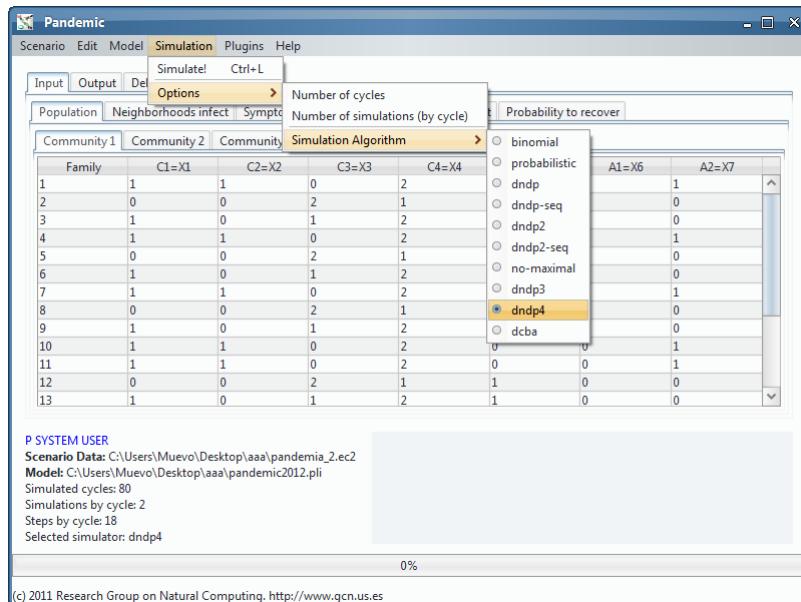


Figura 4.19: Simuladores asociados a un modelo

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, se puede diseñar un único modelo para toda una familia de sistemas P incluyendo un importante número de parámetros, de forma que las distintas instancias del problema y 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, se tiene la posibilidad de establecer el modelo (*Set Model*) si no se ha hecho previamente, y de inicializar el modelo (*Init Model*) 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, entonces se mostrarán los valores de los parámetros generados y las reglas detectadas en la pestaña *ParsingInfo*. En ese momento se podrá hacer uso de las funcionalidades de simular

un paso (*Step*) o un número determinado de pasos (*Run steps*), mostrando todos los datos de las sucesivas configuraciones en la pestaña *Simulation Info*. El aspecto de estas funcionalidades aparece en la figura 4.20.

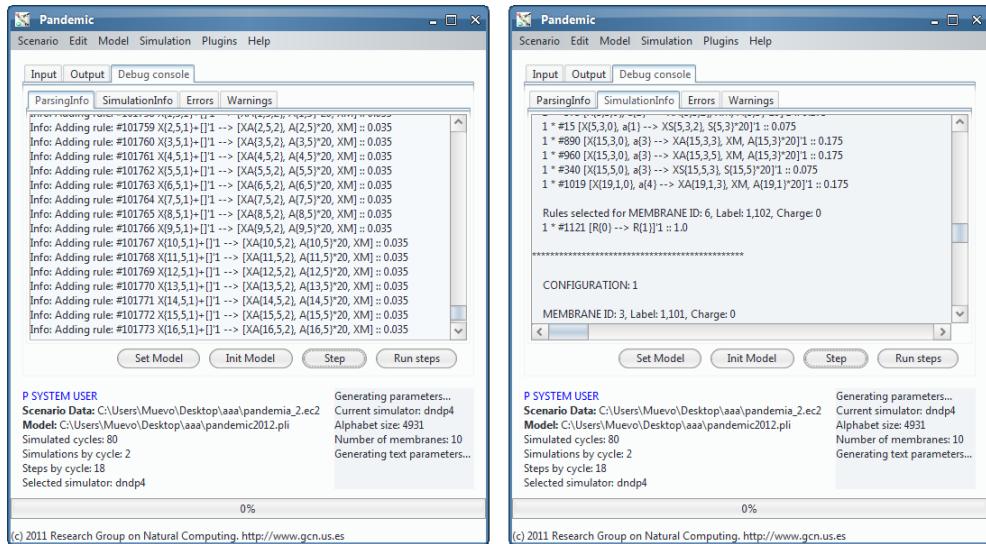


Figura 4.20: Depuración de un modelo

La depuración de los modelos permite detectar posibles errores en los mismos, respecto a sus restricciones sintácticas y semánticas, así como desajustes en los valores de los parámetros del modelo, según el escenario.

Visualización y análisis de datos

- Definición de salidas personalizadas y post-procesadas: los simuladores 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 permite mostrar información específica en forma de tablas o gráficos. Esto da una mayor batería de funcionalidades de visualización de la parte de la información en la que estemos interesados. Además, permite mostrar salidas procesadas, aplicando algún post-procesamiento a los datos de las configuraciones y, mediante técnicas de filtrado y agrupación, 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 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 4.21, izquierda), otro de la estructura de membranas (figura 4.21, derecha) y otro de los multiconjuntos incluidos en cada membrana (figura 4.22).

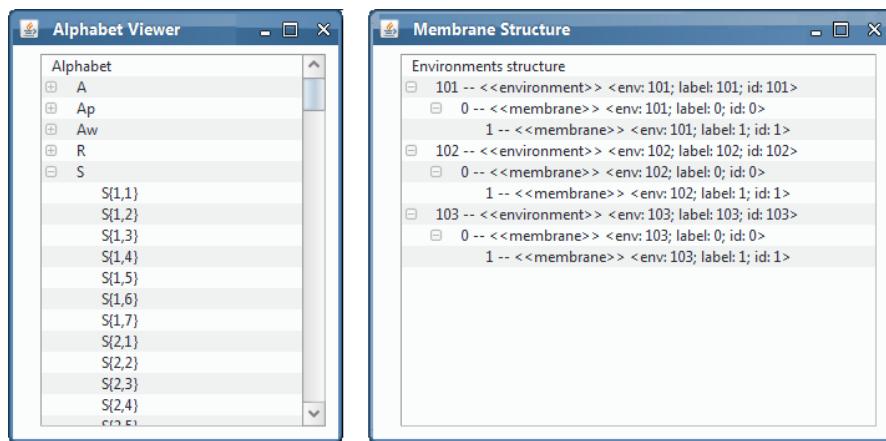


Figura 4.21: Visualizadores de alfabeto (izquierda) y estructura de membranas (derecha)

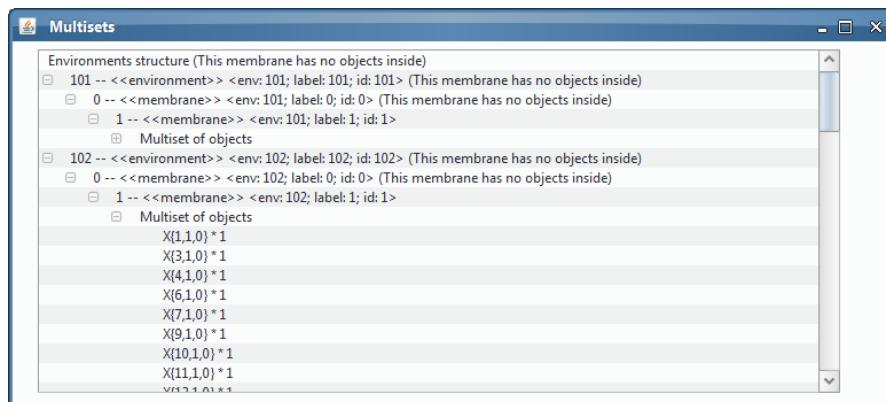


Figura 4.22: Visualizador de los multiconjuntos de cada membrana

- Visualización de grafos: además de las estructuras anteriores, se ha desarrollado un plugin adicional que permite definir en MeCoSim parámetros (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 4.23.

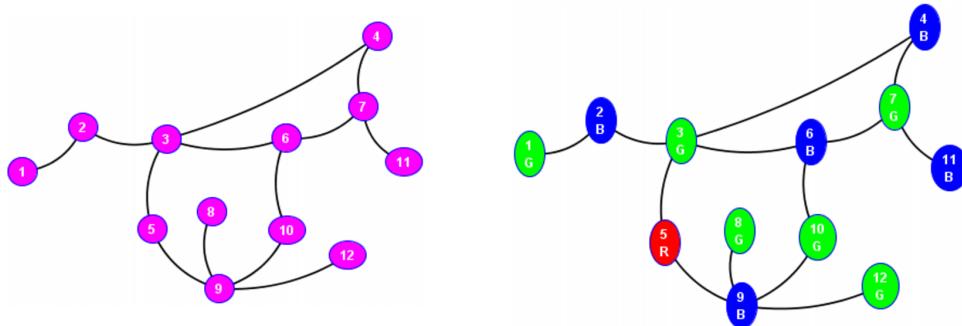


Figura 4.23: Plugin para la visualización de grafos

Mediante el mecanismo de definición de parámetros de MeCoSim se pueden establecer parámetros para el número de vértices (por defecto n), número de aristas (por defecto ne) y aristas específicas del grafo (por defecto $e\{k, 1\}$ se asocia al primer nodo de la arista k , y $e\{k, 2\}$ denota el segundo). De esta manera, 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, como podemos apreciar en la figura 4.24.

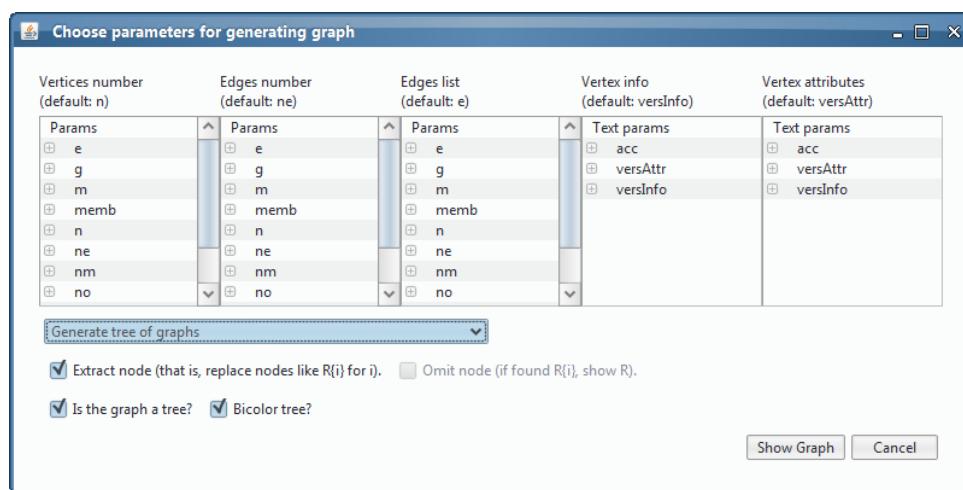


Figura 4.24: GraphsPlugin - Parámetros para la visualización

Con esta información se podrá visualizar un grafo mediante parámetros, que fundamentalmente se obtienen de la información de tablas de entrada, bien a partir de un número de vértices más una lista de aristas, o bien únicamente a partir de la lista de aristas (numéricas o textuales). Las opciones disponibles aparecen en la figura 4.25.

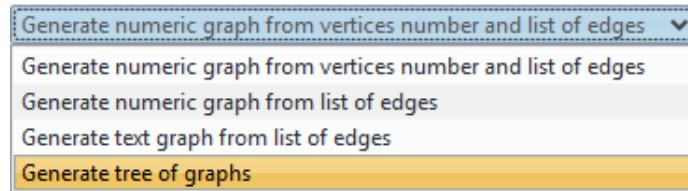


Figura 4.25: GraphsPlugin - Opciones de generación

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 antes mencionados. En la figura 4.26 se observan dos ejemplos de este tipo de visualizaciones de árboles de grafos.

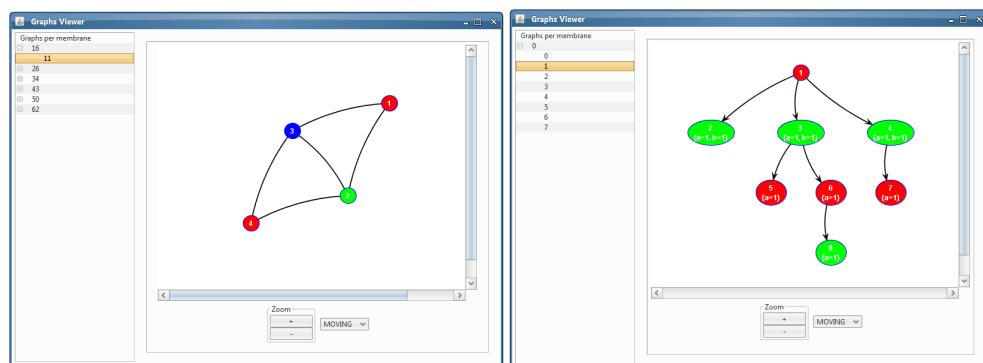


Figura 4.26: GraphsPlugin - Visualización de árboles de grafos

Además, de existir parámetros de tipo *versAttr*, éstos incluirán información adicional acerca de los nodos. Esta información adicional o meta-information 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 la meta-information. 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 en la figura 4.24.

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 [55, 67, 68]), permiten extraer posibles propiedades invariantes en determinados elementos de los modelos diseñados. La definición de salidas a medida nos permite seleccionar los elementos de salida que queremos obtener (por ejemplo, la presente en la figura 4.27), y el plugin de Daikon permite extraer de la salida seleccionada (figura 4.28) las propiedades que la herramienta Daikon pueda encontrar (figura 4.29). Este proceso está descrito en mayor detalle en [87].

Step	n	mult
0	2	1
1	2	2
2	2	4
3	2	7
4	2	9

Figura 4.27: Un salida a medida para extraer invariantes con Daikon

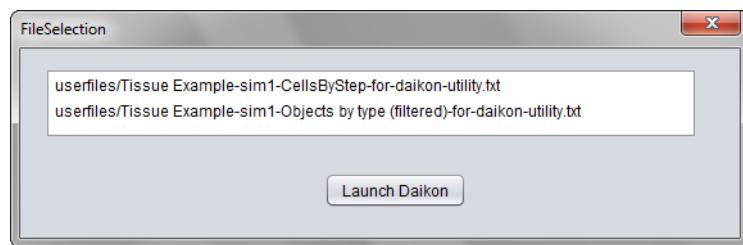


Figura 4.28: Selección de salida de la que extraer invariantes con Daikon

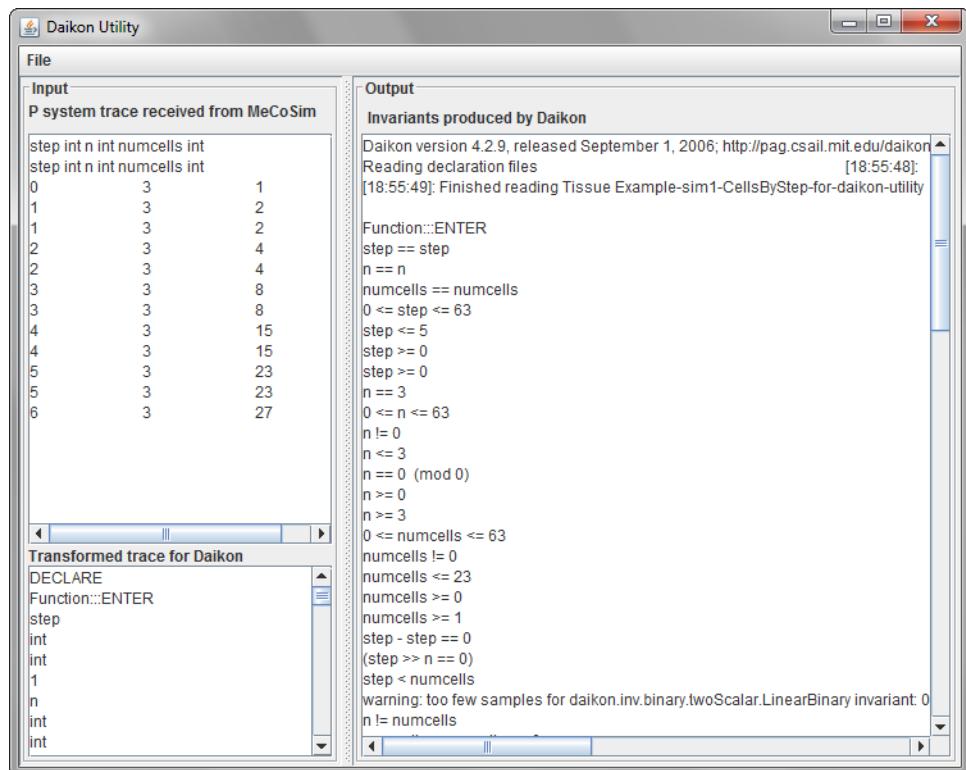


Figura 4.29: Extracción de invariantes con Daikon

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 colaborado¹ 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 [17, 75, 9]. Algunos detalles de este proceso son comentados en [62], teniendo en cuenta las bases para la verificación automatizada de sistemas P mediante Spin sentadas en [77, 88].

¹Se ha colaborado con el Department of Mathematics and Computer Science, University of Pitesti, y el Department of Computer Science, University of Sheffield

El aspecto de la interfaz gráfica del plugin para generación de código Promela y ejecución con Spin puede observarse en la figura 4.30

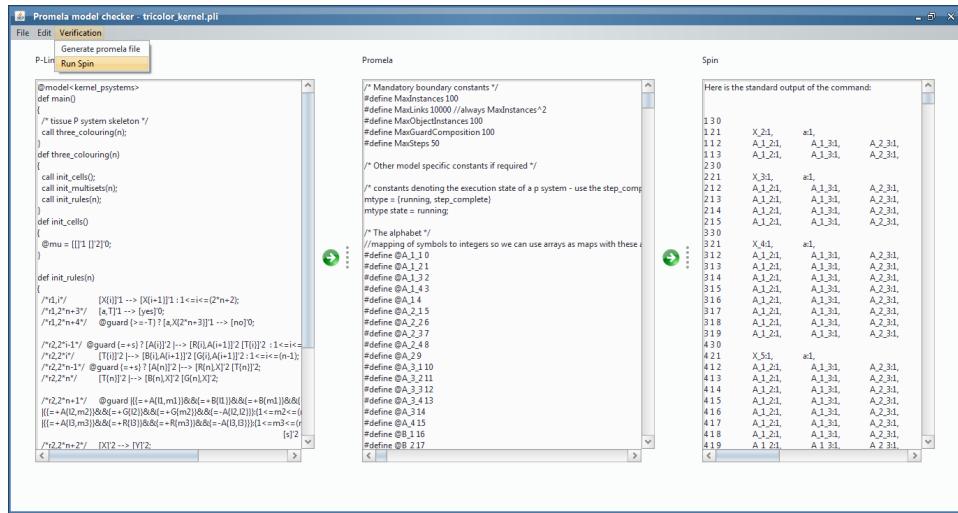


Figura 4.30: Plugin para generación de código Promela y verificación mediante Spin

Dado que el model checker Spin también presenta capacidades de simulación, además del plugin citado 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.

5

Simple Kernel P systems

Como ya se ha comentado a lo largo de esta memoria, la computación celular con membranas es una rama emergente de la Computación Natural que ha dado lugar a múltiples variantes de “sistemas celulares”: unas trabajan a modo de células (sistemas P de transición, sistemas P con membranas activas, etc.); otras lo hacen a modo de tejidos (sistemas P de tejido con división celular, con separación celular, sin entorno, etc.) y otras a modo de neuronas (spiking neural P systems en sus múltiples versiones). Los *kernel P systems* surgen de la necesidad de desarrollar un modelo integrador que incluya un amplio espectro de sistemas celulares que incorpore los ingredientes sintácticos y semánticos más relevantes.

Con objeto de simplificar este tipo de sistemas y facilitar así un avance más gradual hasta los objetivos pretendidos por los kernel P systems, en cuanto a la homogeneización de distintas variantes previas en un único marco, estudio de sus propiedades y verificación de las soluciones empleando tales sistemas, se diseñó una variante inicial, denominada *simple kernel P systems*, que fue presentada en [62]. A través de un caso de estudio, el problema de decisión 3-COL [47], se estudió la potencia expresiva y la eficiencia computacional de los kernel P systems. De hecho, para el citado problema de decisión se diseñaron dos soluciones basadas en *simple kernel P systems* que fueron analizadas en términos de complejidad computacional. Los modelos basados en *simple kernel P systems* demuestran ser más sencillos, en términos del número de reglas, objetos, número de células y pasos de ejecución, que los correspondientes sistemas P de tejidos, disponibles en la literatura, que resuelven el mismo problema, a cambio de una mayor longitud de las reglas.

Este capítulo tiene como objetivo la presentación este nuevo marco integrador y esá estructurado como sigue. En la Sección 5.1 se presenta una introducción que justifica la conveniencia de introducir los *simple kernel P systems*, explicando el papel que vienen a desempeñar en el contexto de la computación celular con membranas y sus dispositivos computacionales, los sistemas P. La Sección 5.2 describe con todo detalle una formalización del modelo de computación antes citado, incluyendo tanto su sintaxis como su semántica. La Sección 5.3.1 está dedicada a exponer el trabajo realizado en el ámbito de las herramientas software para trabajar con *simple kernel P systems*, indicando los objetivos acometidos así como las aportaciones tecnológicas y metodológicas realizadas. En la última sección de este capítulo 5.4 se presenta una solución eficiente del problema *3-COL* a través de una familia de *simple kernel P systems*.

5.1. Introducción

Desde el nacimiento de la computación celular con membranas a finales de 1998, se han introducido y estudiado diferentes clases/variantes de sistemas P. Uno de los objetivos de este estudio ha sido determinar qué tipo de ingredientes son relevantes a la hora de analizar la potencia computacional (capacidad para resolver todos aquellos problemas que resuelven las máquinas de Turing deterministas) o la eficiencia computacional (capacidad para dar soluciones “eficientes” a problemas computacionalmente duros).

Los dispositivos del paradigma computacional citado han sido usados en contextos bien diferentes que van desde la modelización computacional de fenómenos de la vida real, hasta la “implementación” de algoritmos eficientes relativamente simples [31, 16] en este marco, pasando por la resolución “eficiente” de problemas NP-completos [48]. Más recientemente, se han estudiado varios algoritmos distribuidos y algunos problemas relacionados con los mismos [108] haciendo uso de una nueva variante de sistemas P.

En muchos casos, la especificación del sistema estudiado requiere características, restricciones o tipos de comportamientos que no siempre son proporcionados por el modelo en su definición inicial. Por tanto, sería interesante tener cierta flexibilidad dentro de las aproximaciones o mecanismos de modelización, especialmente en las etapas tempranas de la misma, ya que podría simplificar el modelo, acortar los procesos asociados y clarificar aspectos más complejos o desconocidos del sistema.

Como contrapartida nos podríamos encontrar con la carencia de un marco coherente y bien definido que nos permita analizar, verificar y probar este comportamiento y simular el sistema.

Los *simple kernel P systems* consta, básicamente, de un sistema de especificación de sistemas P de relativamente bajo nivel cuyo objetivo esencial consiste en combinar características e ingredientes de interés presentes en muchas de las variantes de sistemas P de diversos tipos estudiados hasta la fecha. Tales características han aparecido en modelos basados en dichas variantes y proporcionan soluciones a un buen número de problemas relevantes tanto desde el punto de vista teórico como desde el punto de vista práctico.

Estos sistemas emergen desde su concepción con la intención de ser definidos formalmente en un estilo operacional y ser, finalmente, implementados en un model checker (SPIN [17], Maude [36]), además de integrar estos mecanismos en la plataforma proporcionada por el marco de P–Lingua.

La idea fundamental que subyace en este formalismo es la de actuar como un modelo integrador que pueda ser empleado con un doble objetivo:

- Proporcionar a los diseñadores de sistemas P un dispositivo de computación lo suficientemente flexible que permita dar solucionar a problemas diversos, a partir de un núcleo de elementos básicos en términos estructurales y de dinámica.
- Proporcionar a los desarrolladores de posibles plataformas de modelización computacional, simulación y, fundamentalmente, de verificación de modelos o diseños basados en sistemas P, un marco unificado sobre el que desarrollar las herramientas necesarias.

El concepto anterior parte de la dificultad para desarrollar herramientas de verificación adaptadas a un sinfín de variantes de sistemas P, cada uno potencialmente con diferentes estructuras, reglas, ingredientes sintácticos, en general, y que suelen usar diversas estrategias de ejecución de sus reglas.

Con la introducción de este nuevo formalismo se pretenden aunar, en un mecanismo integral, las capacidades de los distintos modelos de una forma general, describiendo dentro del formalismo los elementos que nos permitan, de algún modo, reemplazar la diversidad de estructuras, notaciones y dinámicas de sus predecesores, sin perder la potencia de los mismos.

Todo ello proporcionaría, en definitiva, una fachada exterior que facilitaría la labor de los diseñadores en lo que respecta a aprender un único formalismo (con un conjunto acotado pero flexible de ingredientes) y propiciaría el desarrollo de herramientas generales capaces de simular y verificar todos ellos dentro del modelo de manera uniforme.

5.2. Simple Kernel P Systems

El objetivo de esta sección es la introducción de un nuevo modelo de computación dentro del paradigma de *Membrane Computing*, titulado *simple kernel P systems*. Para ello, en primer lugar se van a realizar una serie de consideraciones generales que haga más fácil la lectura del formalismo asociado al modelo que se va a introducir.

Recordemos que todo modelo de computación tienen una especificación sintáctica y otra semántica. En lo que respecta a la parte sintáctica hemos de decir que su objetivo consiste en identificar con toda precisión los dispositivos (procedimientos *mecánicos*) del modelo que se presenta. La parte semántica consiste en definir la dinámica de los dispositivos; es decir, cómo éstos evolucionan a lo largo del tiempo, a partir de una situación o configuración de partida.

Los *simple kernel P systems* consta de una serie de unidades o componentes básicas de procesamiento que se denominarán *compartimentos*. Esas componentes están estructuradas según un grafo no dirigido dinámico, similar a la estructura que presentan los sistemas P que trabajan a modo de tejidos (*tissue-like*), si bien en este modelo la estructura es definida de manera explícita. Cada componente del sistema contendrá un multiconjunto finito de objetos (eventualmente vacío), así como un conjunto finito reglas de diversos tipos que serán aplicadas de acuerdo con una estrategia de ejecución perfectamente definida. Las reglas serán las responsables de la evolución y el transporte de objetos entre distintas componentes, así como de los cambios en la propia estructura (grafo no dirigido) asociada al modelo. Además, como suele ser usual en las variantes de los sistemas P, todo *simple kernel P system* tiene asociado una componente “especial” que denominaremos *entorno* del sistema.

En los *simple kernel P system* existen unos determinados *tipos* de tal manera que cada compartimento será de un cierto tipo. Ademas, cada *tipo de compartimento* tiene asociado un conjunto finito de reglas que pueden ser *reglas de procesamiento de objetos* (también denominadas de *reescritura y comunicación*), o bien *reglas de modificación de la estructura* (reglas de *división*). Las primeras son las que permiten la transformación y el transporte de objetos entre distintos compartimentos, o bien el intercambio de objetos entre un compartimento y el entorno. Las reglas de división son responsables de alterar la topología del sistema (el grafo subyacente), además de posibilitar la creación de una cantidad de espacio exponencial (en términos del número de compartimentos y de objetos) en tiempo polinomial.

Otro hecho novedoso en este nuevo marco computacional es que cada regla tiene asociada una fórmula (que denominaremos *guarda*) que posee una única variable libre cuyos valores son multiconjuntos de objetos del sistema. Dicha fórmula actúa de forma similar a los activadores e inhibidores asociados a ciertas variantes “clásicas” de sistemas P, en el sentido de proporcionar condiciones necesarias para la aplicabilidad de las reglas a una determinada configuración del sistema.

Los simple kernel P systems van a ser utilizados para proporcionar soluciones eficientes de problemas computacionalmente duros, en particular de problemas **NP**–completos.

5.2.1. Formalización

Definición 5.1. Una guarda atómica sobre un alfabeto Γ es la fórmula TRUE , la fórmula FALSE , o bien una expresión del tipo $w\gamma w_0$, siendo w una variable sobre $M_f(\Gamma)$, $w_0 \in M_f(\Gamma)$ y $\gamma \in \{\leq, \geq, =, \neq, >, <\}$.

Es decir, una guarda atómica $g(w)$ (no trivial) sobre un alfabeto Γ es una formula booleana cuya variable toma valores en el conjunto de los multiconjuntos sobre Γ . Ejemplos de guardas atómicas son los siguientes: $g_1(w) \equiv w \geq a^3b^2$ y $g_2(w) \equiv w < a^4$.

Definición 5.2. Una guarda sobre un alfabeto Γ es una sucesión finita de guardas atómicas sobre Γ conectadas mediante los operadores lógicos $\{\neg, \wedge, \vee\}$.

Es decir, una guarda $g(w)$ (no trivial) sobre un alfabeto Γ es una formula booleana cuya variable toma valores en el conjunto de los multiconjuntos sobre Γ . Un ejemplo de guarda sería el siguiente: $g(w) \equiv \neg(w \geq a^3b^2) \wedge (w < a^4) \vee (w \leq abc^2)$.

Recordemos que los operadores lógicos se ejecutarán de acuerdo con el orden de prioridad siguiente: primero la negación, \neg , después la conjunción \wedge y, finalmente, la disyunción \vee .

A continuación se va a definir el concepto de *tipo de comportamiento* tratando de conservar la notación de los *kernel P systems*. Recordemos que, básicamente, un tipo de comportamiento consta, básicamente, de un conjunto de reglas, R , y de una estrategia de ejecución, σ , de las mismas. En el caso de los *simple kernel P systems* la estrategia de ejecución será siempre la misma y, por tanto, podría eliminarse propiamente de la definición. No obstante, nosotros vamos a conservarla para hacer la presentación más cercana a los *kernel P systems* de donde proceden.

Definición 5.3. Un tipo de comportamiento t sobre un alfabeto Γ es un par ordenado (R_t, σ_t) , en donde R_t es un conjunto de reglas sobre Γ etiquetadas de manera inyectiva por elementos de un conjunto $\text{Lab}(R_t)$ y σ_t es una estrategia de ejecución de las reglas del conjunto R_t .

Las reglas de R_t están asociadas a un conjunto $T = \{t_1, \dots, t_s\}$ de tipos de comportamientos tal que $t \in T$ y pueden ser de una de las formas que a continuación se relacionan:

- Regla de reescritura y comunicación: $u \rightarrow v \{g\}$, en donde $u \in M_f(\Gamma)$, $u \neq \emptyset$, $v = (a_1, t_{i_1}) \dots (a_h, t_{i_h})$, con $a_j \in \Gamma$, $t_{i_j} \in T$ y g es una guarda sobre Γ . La longitud de esta regla se define así: $|u| + |v|$.
- Reglas de división: $[a]_t \rightarrow [v_1]_{t_{j_1}} \dots [v_p]_{t_{j_p}} \{g\}$, en donde $a \in \Gamma$, $v_j \in M_f(\Gamma)$, $t_{j,p'} \in T$ y g es una guarda sobre Γ . La longitud de esta regla se define así: $1 + |v_1| + \dots + |v_p|$.

Definición 5.4. Un *compartimento* sobre un alfabeto Γ es un par ordenado (t, w) , en donde t es un tipo de compartimento sobre Γ y w es un multiconjunto de objetos sobre Γ .

Definición 5.5. Un simple Kernel P systems de grado $n \geq 1$ es una tupla

$$\Pi = (\Gamma, \mathcal{E}, T, G, C_1, \dots, C_n, i_{out})$$

donde

- Γ es un alfabeto finito cuyos elementos se denominan *objetos*.
- \mathcal{E} es un alfabeto contenido en Γ (denominado *alfabeto de salida*).
- $T = \{t_1, \dots, t_s\}$ es un conjunto de tipos de compartimentos.
- $G = (V, E)$ es un grafo no dirigido.
- $V = \{C_1, \dots, C_n\}$ es el conjunto de nodos del grafo, denominados *compartimentos del sistema*, de tal manera que $C_i = (t_i, w_i)$, $1 \leq i \leq n$, con $t_i \in T$ y $w_i \in M_f(\Gamma)$.
- $i_{out} \in \{0, 1, \dots, n\}$ es un número que representa o bien el entorno (en el caso $i_{out} = 0$) o bien un compartimento del sistema, en el caso en que $i_{out} \in \{1, \dots, n\}$.

Un simple kernel P systems de grado n puede ser considerado como un conjunto de n compartimentos interconectados por un grafo no dirigido G de tal manera que cada compartimento C_i contiene inicialmente un multiconjunto de objetos y, además, pertenece a un cierto conjunto T de “tipos de compartimentos”. Este conjunto T especifica el conjunto de reglas del sistema, si bien en los *simple kernel P systems* la estrategia de ejecución siempre será la misma (en cada tipo de compartimento): estrategia de paralelismo maximal con la restricción siguiente: en un paso de computación sólo se puede aplicar, a lo sumo, una regla de división a cada compartimento.

En el sistema existe una zona distinguida representada por el número i_{out} que va a representar el entorno o un compartimento especial del sistema en donde se codificará la "salida" de las computaciones.

A continuación nos vamos a centrar en la semántica de dichos modelos. Como es usual, se definirán los conceptos siguientes: (a) configuración; (b) aplicabilidad de una regla a una configuración; (c) resultado de ejecutar una regla aplicable a una configuración; (d) paso de transición o de computación; y (e) computación.

Definición 5.6. Una configuración o descripción instantánea de un simple kernel P system, es una tupla formada por $(G', C'_{i_1}, \dots, C'_{i_r})$, en donde G' es un grafo no dirigido cuyos nodos son los compartimentos $(C'_{i_1}, \dots, C'_{i_r})$.

Dado el simple kernel P system $\Pi = (\Gamma, \mathcal{E}, T, G, C_1, \dots, C_n, i_{out})$, la configuración (G, C_1, \dots, C_n) se denomina configuración inicial del sistema Π .

A continuación se va a definir el concepto de *aplicabilidad* de una regla a una configuración, así como el efecto que tiene la ejecución de una regla aplicable a una configuración. Para ello, recordemos que toda regla del sistema está asociada a un tipo de comportamiento (**no**, propiamente, **a un comportamiento**) y, además, tiene asociado una guarda.

Definición 5.7. Diremos que una regla de reescritura asociada a un tipo de comportamiento t , $u \rightarrow v \{g\}$, con $u \in M_f(\Gamma)$, $u \neq \emptyset$ y $v = (a_1, t_{i_1}) \dots (a_h, t_{i_h})$, es aplicable a una configuración $\mathcal{C} = (G', C'_{i_1}, \dots, C'_{i_r})$ y a un comportamiento $d = (t, w)$, si se verifican las condiciones siguientes:

- (a) Existe j ($1 \leq j \leq r$), tal que $C'_{i_j} = d$; es decir el comportamiento d es uno de los que aparecen en la configuración \mathcal{C} .
- (b) $u \subseteq w$; es decir, el multiconjunto u que define la parte izquierda de la regla está contenida en el multiconjunto w que está en el comportamiento C'_{i_j} de la configuración \mathcal{C} .
- (c) La fórmula $g(w)$ es verdadera; es decir, se satisface la guarda para el multiconjunto contenido en el comportamiento concreto donde se aplica.
- (d) Existen comportamientos d_1, \dots, d_h de tipos $t_{i_1} \dots t_{i_h}$, respectivamente, tales que $\{d, d_1\}, \dots, \{d, d_h\}$ son aristas del grafo G'

Definición 5.8. Diremos que una regla de división asociada a un tipo de comportamiento t , $[a]_t \rightarrow [v_1]_{t_{j_1}} \dots [v_p]_{t_{j_p}} \{g\}$, con $a \in \Gamma$, $v_j \in M_f(\Gamma)$, es aplicable a una configuración $\mathcal{C} = (G', C'_{i_1}, \dots, C'_{i_r})$ y a un comportamiento $d = (t, w)$, si se verifican las condiciones siguientes:

- (a) Existe j ($1 \leq j \leq r$), tal que $C'_{i_j} = d$; es decir el compartimento d es uno de los que aparecen en la configuración \mathcal{C} . Además, d no es el compartimento de salida del sistema.
- (b) $a \in w$; es decir, el objeto a es un elemento del multiconjunto w que está en el compartimento C'_{i_j} de la configuración \mathcal{C} .
- (c) La fórmula $g(w)$ es verdadera; es decir, se satisface la guarda para el multiconjunto contenido en el compartimento concreto donde se aplica.

En el caso en que la guarda asociada a una regla sea la fórmula TRUE, entonces omitiremos la escritura de dicha guarda ya que la correspondiente “condición” (TRUE) siempre se verificará.

Seguidamente se define cuál es el efecto que tiene la ejecución de una regla aplicable a una configuración.

Definición 5.9. Si una regla de reescritura asociada a un tipo de compartimento t , $u \rightarrow (a_1, t_{i_1}) \dots (a_h, t_{i_h}) \{g\}$, es aplicable a una configuración $\mathcal{C} = (G', C'_{i_1}, \dots, C'_{i_r})$ y a un compartimento $d = (t, w)$, el efecto que produce su aplicación es el siguiente:

- (a) El multiconjunto u que define la parte izquierda de la regla es eliminado del contenido del compartimento d .
- (b) Para cada k ($1 \leq k \leq h$), el objeto a_k se añade a un compartimento de G' de tipo t_{i_k} (si hubiera más de uno, se haría una elección no determinista).

Definición 5.10. Si una regla de división asociada a un tipo de compartimento t , $[a]_t \rightarrow [v_1]_{t_{j_1}} \dots [v_p]_{t_{j_p}} \{g\}$, es aplicable a una configuración $\mathcal{C} = (G', C'_{i_1}, \dots, C'_{i_r})$ y a un compartimento $d = (t, w)$, el efecto que produce su aplicación es el siguiente:

- (a) El objeto a que dispara la regla se elimina del contenido del compartimento d .
- (b) Se crean p nuevos compartimentos $d_1 \dots d_p$ de tipos $t_{j_1} \dots t_{j_p}$, respectivamente, cuyo contenido es idéntico al del contenido del compartimento d con la excepción de que, para cada k ($1 \leq k \leq p$), en el compartimento d_k el objeto a es sustituido por el multiconjunto v_k .
- (c) Al grafo G' se le añaden las aristas $(d, d_1) \dots (d, d_p)$.

Dadas dos configuraciones \mathcal{C} y \mathcal{C}' de un simple kernel P system, Π , diremos que \mathcal{C}' se obtiene de \mathcal{C} tras la ejecución de un *paso de computación*, y notaremos $\mathcal{C} \Rightarrow_{\Pi} \mathcal{C}'$, si la configuración \mathcal{C}' se obtiene a partir de la configuración \mathcal{C} al aplicar un multiconjunto maximal de reglas, con la restricción antes indicada en lo que respecta a las reglas de división.

Como es usual, diremos que una configuración \mathcal{C} de un *simple kernel P system* es de *parada* si no existe ninguna regla del sistema que sea aplicable a dicha configuración.

Una *computación* en un *simple kernel P system* es una sucesión (finita o infinita) de configuraciones tal que: (a) el primer término de la sucesión es una configuración inicial del sistema; (b) para cada $n \geq 2$, el n -ésimo término de la sucesión se obtiene a partir del anterior mediante la ejecución de un paso de computación; y (c) si la sucesión es finita (en cuyo caso se denominará *computación de parada*), entonces el último término de la sucesión es una configuración de parada. Sólo las computaciones de parada proporcionan un resultado, el cual estará codificado por el multiconjunto de objetos del comportamiento de salida de la correspondiente configuración de parada que, eventualmente, puede ser el entorno del sistema.

Al igual que en otras variantes de sistemas P, en el nuevo modelo de computación admitiremos que toda computación ejecuta un proceso sincronizado; es decir, se supone que hay una especie de *reloj universal* que marca las actuaciones de todos los elementos que integran el sistema celular.

5.3. Herramientas software para modelización y simulación

En el primer apartado de esta sección (5.3.1) se incluye una metodología similar a la ilustrada en el capítulo 4, si bien ahora está aplicada al caso particular de los problemas a resolver, analizar y verificar mediante simple kernel P systems, con el fin de dar una visión de conjunto de la propuesta metodológica y de las herramientas que nos ayudan a ponerla en práctica. Posteriormente se añaden apartados explicando con mayor nivel de detalle el papel de cada uno de los componentes involucrados.

5.3.1. Un marco para la modelización, simulación y verificación

En este apartado se presenta una metodología para la modelización, simulación y verificación (mediante técnicas de model checking), para simple kernel P systems (apareció publicada por primera vez en [87]). La metodología, esbozada en la figura 5.1, está apoyada por el entorno software MeCoSim [122, 153] descrito en el capítulo anterior, conteniendo el lenguaje de especificación P-Lingua [58, 120] y que engloba como hemos visto muchas variantes de sistemas P incluyendo los simple kernel P systems cuyos intérpretes, simuladores y herramientas asociadas han sido desarrollados a lo largo de la presente tesis.

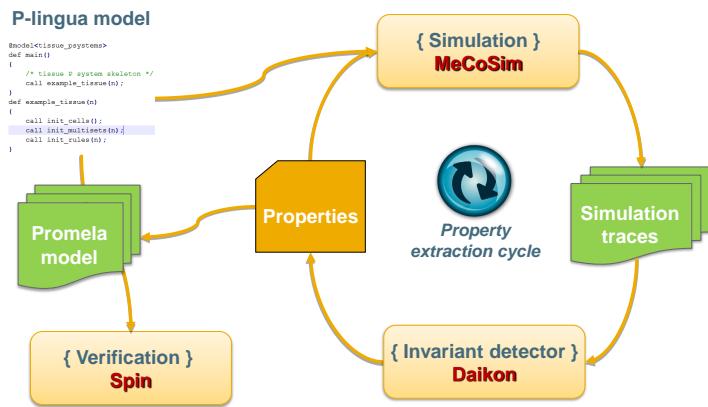


Figura 5.1: Especificación, análisis y verificación de simple kernel P systems

interfaces personalizadas, diseñadas específicamente mediante configuración en MeCoSim [122, 153], nos permite manejar los datos de entrada para cada problema, proporciona resultados de visualización y extrae los datos de salida apropiados. Las simulaciones se llevan a cabo dentro de MeCoSim empleando el motor de simulación para simple kernel P systems de pLinguaCore [49]. Entonces se pueden obtener propiedades invariantes a partir del conjunto de datos de salida seleccionado, o formularse de acuerdo con el comportamiento observado del sistema. Los resultados pueden ser verificados empleando técnicas de model checking (por ejemplo, usando Spin para un modelo escrito en lenguaje Promela [17]). Para una descripción más detallada ver [87].

De acuerdo con esta metodología y sus pasos asociados, se describirá en la última sección de este capítulo, una solución de un problema NP-completo basada en simple kernel P systems, desde su diseño hasta su especificación en P-Lingua incluyendo parámetros que toman distintos valores para diferentes instancias de los problemas siguiendo el modelo de MeCoSim.

La simulación se lleva a cabo comenzando con la configuración inicial del sistema P, y la computación se ejecuta hasta que se alcanza una configuración de parada. En ese proceso existe una serie de opciones adicionales, algunas de las cuales han visto la luz como consecuencia del desarrollo de esta tesis; así por ejemplo, el sistema P puede ser simulado paso a paso o durante un tiempo de ejecución prefijado, o bien una selección de objetos y compartimentos pueden ser producidos o, de forma alternativa, todos ellos pueden ser generados. Además, las tablas de salida de la interfaz personalizada muestran la información seleccionada para ser visualizada.

Todo lo anterior se ilustrará en los casos de estudio incluidos en la última sección de este capítulo, así como en el capítulo 6. Finalmente, como complemento a las salidas tabuladas y con objeto de ayudar a interpretar el resultado, se han incluido algunas herramientas de visualización en MeCoSim, incluyendo visores de distintos elementos del sistema P, gráficas y grafos. Estos últimos permiten visualizar la respuesta, la solución al problema para cada instancia del mismo, mediante una representación visual cercana al problema en cuestión en lugar de proporcionar una respuesta árida textual o tabular en forma de objetos del alfabeto dentro de cada compartimento del sistema diseñado.

Asimismo, en el capítulo 6 se comentarán las simulaciones realizadas sobre simple kernel P systems para la resolución de problemas NP-completos que pondrán de manifiesto los beneficios de usar la aproximación basada en simple kernel P systems con respecto a otros formalismos usando sistemas P. Por una parte, el uso de simple kernel P systems como marco de resolución, disminuye el número de reglas con relación a los sistemas P de tejidos, si bien estas reglas pueden ser más complejas, de modo que hay una cierta compensación entre esos dos aspectos. En general, se observa una diferencia en el número de compartimentos y tiempo de ejecución entre los simple kernel P systems y los sistemas P de tejidos. Esto no se puede considerar como una sorpresa teniendo en cuenta que en estos sistemas se emplea un número menor de reglas, así como el hecho de que los procesos de división de los compartimentos cesan cuando la condición para producir una solución correcta deja de cumplirse; por ejemplo, en el caso del problema 3-COL la existencia de dos vértices adyacentes con el mismo color.

Es interesante hacer una observación relacionada con una mejora adicional sustancial, con mínimos costes de implementación. Se refiere a un mecanismo que permite evitar el uso de compartimentos que, con toda seguridad, no van a proporcionar una solución correcta y en los que no quedan reglas aplicables. Si bien teóricamente el tiempo consumido por un compartimento en el que ninguna regla es aplicable no debería siquiera ser contemplado, el mero chequeo o análisis de la aplicabilidad de las reglas podría introducir un sobrecoste en tiempo que podría llegar a ser significativo conforme aumenta el número de tales compartimentos y, además, las reglas tienen asociadas guardas con alta complejidad. Para mitigar las consecuencias negativas de este efecto, se ha considerado y desarrollado como parte de P-lingua, un mecanismo para evitar la evaluación de un compartimento donde no hay reglas aplicables. Una consecuencia de este enfoque es la posibilidad de eliminar por completo tales compartimentos y, por tanto, reducir el número total de compartimentos que se procesa. Esta opción ha permitido simular en tiempo razonable instancias mucho mayores de los problemas estudiados, debido a la liberación de espacio que evita el desbordamiento de memoria ocasionada por la cantidad exponencial de espacio construido si no se adoptan medidas en relación con los compartimentos que se van creando a través de las reglas de división.

5.3.2. Especificación de simple kernel P systems

La metodología propuesta se sustenta en el desarrollo de herramientas que permitan llevar a cabo la definición, personalización, simulación, análisis, visualización, extracción de propiedades y verificación de soluciones a problemas basadas en sistemas P. El primer paso de este proceso consiste en la especificación de sistemas P del tipo *simple kernel P system* en un lenguaje entendible por la máquina. Para ello se ha adaptado/extendido el lenguaje P-Lingua con el fin de incorporar los ingredientes necesarios para procesar soluciones basadas en *simple kernel P systems*.

A continuación se describen los elementos del lenguaje que representan a sus homólogos en el diseño de un *simple kernel P system*.

5.3.2.1. Estructuras en P–Lingua

En primer lugar conviene indicar que de acuerdo con la primera definición de los *simple kernel P systems* ([62]) las guardas asociadas a las reglas tienen una expresión específica que vamos a describir a continuación. No obstante esa expresión puede ser deducida de la definición general dada en 5.2.

Una guarda subatómica no trivial $g(w)$ sobre un alfabeto Γ es una guarda atómica sobre Γ del tipo $w(a) \gamma a^n$, donde $a \in \Gamma$, $n \geq 1$ y $w(a)$ indica la multiplicidad del objeto a en el multiconjunto w . Entonces una guarda atómica no trivial $g(w) \equiv w\gamma w_0$ sobre Γ , siendo w una variable sobre $M_f(\Gamma)$, $w_0 \in M_f(\Gamma)$ y $\gamma \in \{\leq, \geq, =, \neq, >, <\}$, se expresará de la siguiente forma:

$$\bigwedge_{x \in supp(w_0)} w(x) \gamma w_0(x)$$

Es decir, la guarda atómica $\{w \geq a^3 b^2 c^5\}$ se reescribirá como conjunción de guardas subatómicas: $\{w(a) \geq a^3 \wedge w(a) \geq b^2 \wedge w(a) \geq c^5\}$ (el número de objetos a en el multiconjunto w es mayor o igual que 3, el número de objetos b en el multiconjunto w es mayor o igual que 2 y el número de objetos c en el multiconjunto w es mayor o igual que 5). Más aún, la expresión anterior se escribirá más brevemente así $\{\geq a^3 \wedge \geq b^2 \wedge \geq c^5\}$ e, incluso, eliminando \wedge como sigue $\{\geq a^3 \geq b^2 \geq c^5\}$.

En lo que resta de capítulo, en lugar de trabajar con el alfabeto Γ lo haremos con el alfabeto $\Gamma \cup \bar{\Gamma}$, en donde $\bar{\Gamma} = \{\bar{a} : a \in \Gamma\}$. Es decir, $\bar{\Gamma}$ es un conjunto disjunto con Γ cuyos elementos son los mismos de Γ a los que les vamos a asignar un papel distinto. Concretamente, la aparición de un objeto del tipo \bar{a} en una guarda significará que la condición asociada es la negación de la correspondiente al objeto a ; es decir, la guarda $\{\geq a^3 \wedge \geq \bar{b}^2 \wedge \geq c^5\}$ es equivalente a la siguiente $\{\geq a^3 \wedge < b^2 \wedge \geq c^5\}$, que escribiríamos brevemente como sigue: $\{\geq a^3 < b^2 \geq c^5\}$.

Además, nos vamos a restringir al uso de guardas que sean disyunciones de conjunciones de las guardas subatómicas (recordemos sobre el alfabeto $\Gamma \cup \bar{\Gamma}$).

Las siguientes estructuras, incluyendo características relacionadas con simple kernel P systems, fueron añadidas durante la preparación de esta tesis en la versión 4.0 de P-Lingua, disponible en [120] y como parte de MeCoSim. La versión actual de P-Lingua permite definir *simple kernel P systems* y tanto el intérprete como el simulador asociado soporta las características propias de dichos modelos, no incluyendo naturalmente otras propias del marco más general de los *kernel P systems*, como reglas de creación/destrucción de enlaces o definición de estrategias de ejecución ya que solamente incorpora paralelismo maximal con la restricción adicional asociada al uso de las reglas de división.

Guardas:

Una guarda subatómica $g(w) \equiv w(a) \gamma a^n$, se escribe como $\{\gamma' a * n\}$, donde γ' es una representación de γ empleando $<$ (para $<$), \leq (para \leq), $=$ (para $=$), \neq (para \neq), \geq (para \geq) y $>$ (para $>$). Como ejemplos ilustrativos, $\{\leq +c*2\}$ representa la guarda $\leq c^2$; mientras $\{>= -b\}$ representa $\geq \bar{b}$. Los operadores booleanos involucrados en expresiones booleanas, \wedge y \vee , se representan como $\&\&$ and $\|$, respectivamente. Por ejemplo, $\{\leq +a*2\} \&\& \{\leq -b\}$ representa la guarda $\{\leq a^2 \wedge \leq \bar{b}\}$. Del mismo modo, $\{\leq -a*2\} \| \{\leq +b\}$ representa la guarda $\{\leq \bar{a}^2 \vee \leq b\}$. Los operadores \wedge y \vee pueden combinarse para describir guardas complejas, tales como $\{\leq +a*2\} \&\& \{\leq -b\} \| \{\leq -a*3\} \&\& \{\leq +c*3\}$.

Una regla $r \{g\}$ se define como `@guard g ? r`. Por ejemplo, la regla $a \rightarrow b \{= a^2\}$ se define como `@guard {=+a*2} ? [a --> b]`.

Inicialización de componentes (membranas/células/compartimentos):

La inicialización de las componentes básicas del sistema permite al diseñador definirlas en la configuración inicial. Según la estructura subyacente del sistema considerado, la sintaxis difiere ligeramente. Así, se emplea

`mu(label1) += [multiset] 'label2;`

para estructuras de membranas de sistemas que trabajan a modo de células (cell-like), mientras que se emplea

`mu(0) *= [multiset] 'label;`

para estructuras de membranas de sistemas que trabajan a modo de tejidos, tales como las correspondientes a los *simple kernel P systems*.

En el primer caso, se añade una nueva membrana etiquetada como $label_2$ con el multiconjunto asociado *multiset* como membrana hija de la etiquetada como $label_1$. En el segundo caso, a la configuración inicial se le añade una

nueva componente (célula/compartimento) etiquetada como *label* con su multiconjunto *multiset*. Recuérdese que en los sistemas que trabajan a modo de tejidos no existe, propiamente, una estructura jerárquica sino un grafo no dirigido por el que todas las componentes básicas se encuentran al mismo “nivel” enmarcadas dentro de un entorno común, etiquetado por 0.

Definición de multiconjuntos iniciales:

Se ha visto que en la inicialización de las componentes básicas del sistema se han definido multiconjuntos asociados a cada componente individual. Pues bien, también es posible definir el multiconjunto asociado a todos aquellos compartimentos (componentes especiales) que pertenezcan a un determinado tipo. Esto se puede implementar mediante la expresión `ms(label) = multiset;`.

Así pues, según el alcance que se le quiera dar a la definición (compartimento o tipo) podemos tener:

- Multiconjunto asociado a cada compartimento específico: se incorpora a la estructura inicial:

```
@mu = [[ a*2 ]'1 [ b ]'2 [ c*2 ]'2] '0;
```

- Multiconjunto asociado a todos los compartimentos de un determinado tipo:

```
@ms(1) = x;
@ms(2) = y*3;
```

Nuevas reglas en P-Lingua:

Para definir los *simple kernel P systems*, se han incorporado en P-Lingua algunas nuevas reglas, con la descripción comentada en la Sección 5.2.1:

Reglas de reescritura y comunicación:

Una regla de reescritura y comunicación $u \rightarrow v \{g\}$, en donde $u \in M_f(\Gamma)$, $u \neq \emptyset$, $v = (a_1, t_1) \dots (a_h, t_h)$, con $a_j \in \Gamma$, $t_h \in T$ y g es una guarda sobre Γ , se representa en P-Lingua así:

```
@guard g ? [a] 't0 --> [a1]'t1, ..., [ah]'th,
```

siendo t_0 el tipo del compartimento actual. A diferencia de la definición dada en la sección 5.2.1, la implementación en P-Lingua de estas reglas no requiere que t_0 esté conectado por una arista a cada compartimento de tipos $t_i, 1 \leq i \leq h$ de forma explícita, sino que queda implícito al existir una regla que comunica los tipos de compartimentos.

Reglas de división:

Una regla de división $[a]_t \rightarrow [v_1]_{t_{j_1}} \dots [v_p]_{t_{j_p}} \{g\}$, en donde $a \in \Gamma$, $v_j \in M_f(\Gamma)$, $t_{j,p} \in T$ y g es una guarda sobre Γ , se representa en P-Lingua así:

$$@guard g ? [a]'t |--> [v_1]'t_{j_1}, \dots, [v_p]'t_{j_p};.$$

Es posible que algún multiconjunto v_j , $1 \leq j \leq p$, contenga el símbolo especial @d. Este símbolo se usa para implementar el mecanismo citado anteriormente que permite la disminución del coste en espacio al eliminar compartimentos que no van a producir una solución correcta y en los que no quedan reglas aplicables.

Tanto las reglas de reescritura como las de división se pueden emplear en conjunción con iteradores internos al compartimento (se explican más adelante), resultando que en dichas reglas pueden aparecer etiquetas, parametrizadas por algún índice, tales como

$$[a]'1 |--> [b]'2 &{[c,d{i}]}'{i}: {3 <= i <= n};.$$

Iteradores internos:

Estos iteradores permiten a los usuarios definir multiconjuntos, estructuras de membranas y guardas dependientes de parámetros. Esos iteradores expanden las posibilidades a la hora de definir reglas y configuraciones iniciales. La sintaxis para los iteradores internos es:

$$&\{items\}: \{index_ranges\},$$

excepto para guardas unidas por \vee , que es

$$| \{items\}: \{index_ranges\}.$$

A menos que se indique lo contrario, se pueden combinar distintos iteradores internos en la misma regla o sentencia, pero no pueden ser anidados. Los iteradores internos pueden emplearse en tres contextos diferentes:

1. Iteradores internos sobre multiconjuntos:

La sintaxis para este tipo de iteradores es

$$&\{multiset\}: \{index_ranges\}$$

Estos iteradores permiten la extensión de multiconjuntos en la definición de reglas. Por ejemplo, dado un valor para n y un conjunto de valores para e_i , $1 \leq i \leq n$, la regla $[a \rightarrow b, c_i, d_i^{e_i}, 1 \leq i \leq n]_1$ se pueden escribir como:

$$[a \rightarrow b, &\{c{i}, d{i}*e{i}\}: \{1 <= i <= n\}'1;$$

2. Iteradores internos sobre membranas:

La sintaxis para este tipo de iteradores es

$$\& \{ [multiset] ' \{label\} \} : \{index_ranges\}$$

Nos permiten especificar comunicaciones a varios compartimentos en la parte derecha de las reglas (no pudiendo iterarse mediante este mecanismo sobre elementos en la parte izquierda de las mismas). Por ejemplo, dado un valor de n y valores para $e_i, 1 \leq i \leq n$, la regla de reescritura y comunicación $x \rightarrow (a_i^{e_i}, t_i), 1 \leq i \leq n$, con $x, a_i \in A, 1 \leq i \leq n$ y t_j indicando el tipo de compartimento perteneciente a T , se escribiría como:

$$[x] ' t_0 \dashrightarrow \& \{ [a\{i\}*e\{i\}] ' \{i\} \} : \{1 \leq i \leq n\} ; ,$$

siendo t_0 el compartimento actual.

3. Iteradores internos sobre guardas:

Estos iteradores constituyen un caso especial ya que pueden aparecer dos tipos distintos de elementos según los operadores booleanos involucrados. Estas formas son:

$$\& \{guard\} : \{index_ranges\}, \text{ para conjunción, y}$$

$$| \{guard\} : \{index_ranges\}, \text{ para disyunción.}$$

De este modo, las expresiones del primer tipo construyen guardas unidas por el operador \wedge (\wedge -joined guards), mientras que las del segundo tipo generan guardas unidas por operadores \vee (\vee -joined guards). Además, son los únicos iteradores internos que pueden ser anidados, empleando una “ \wedge -joined guard” en el interior de una “ \vee -joined guard” (forma normal disyuntiva). Por el contrario, no se permite definir “ \vee -joined guards” en el interior de “ \wedge -joined guards”. Por ejemplo, fijados un valor para n y otro para m , la regla

$$@guard | \{ \{ \& \{ \{ \leq + B\{i,j\} * 2 \} : \{1 \leq j \leq m\} \} : \{1 \leq i \leq n\} ? [a \dashrightarrow b] ' 1 ;$$

puede ser aplicada si y sólo si antes de aplicarse la regla, existe al menos un valor i , con $1 \leq i \leq n$, tal que la cardinalidad de cada objeto $B_{i,j}$, con $1 \leq j \leq m$, en el interior de la membrana 1, es mayor o igual a 2.

Existen algunas restricciones sobre los índices en los iteradores internos ya que éstos no se pueden incluir en expresiones numéricas, tales como $i + 1$ en la expresión $\& \{a\{i+1\}\} : \{1 \leq i \leq 10\}$, ni se pueden emplear para acceder a constantes indexadas, tales como $g\{i\} \& \{a\{g\{i\}\}\} : \{1 \leq i \leq 10\}$. Además, los nombre usados para índices de iteradores internos no pueden emplearse en ningún otro lugar, incluyendo otro iterador interno.

5.4. Una solución para 3-COL basada en simple kernel P systems

En esta última sección del capítulo, vamos a presentar una solución eficiente del problema *3-COL* a través de una familia de *simple kernel P systems*. Para más detalle ver [62].

A continuación se definen dos funciones computables, s y cod , sobre el conjunto de instancias del problema 3-COL como sigue:

$$\begin{cases} s(\mathcal{G}) &= |\mathcal{V}| \\ cod(\mathcal{G}) &= \{A_{i,j} : \{i,j\} \in \mathcal{E}, 1 \leq i < j \leq n\} \end{cases}$$

siendo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ una instancia arbitraria (no trivial) del problema *3-COL*; es decir, \mathcal{G} es un grafo no dirigido que contiene al menos 2 vértices ($|\mathcal{V}| = n \geq 2$).

Se considera la familia de *simple kernel P systems*: $\Pi = \{\Pi(n) : n \in \mathbb{N}\}$, en donde $\Pi(n) = (\Gamma, IO, T, G, C_1, C_2, i_{out}, i_{in})$ se define como sigue:

- El alfabeto de trabajo Γ es el siguiente conjunto:

$$\begin{aligned} \Gamma = & \{A_1, \dots, A_n\} \cup \{A_{i,j} \mid 1 \leq i < j \leq n\} \cup \{T_1, \dots, T_n\} \cup \\ & \{B_1, \dots, B_n\} \cup \{R_1, \dots, R_n\} \cup \{G_1, \dots, G_n\} \cup \\ & \{a, s, X, Y, Z, yes, no\} \cup \{X_1, \dots, X_{2n+3}\} \end{aligned}$$

En donde:

- A_i , $1 \leq i \leq n$, son símbolos que representan los n vértices de \mathcal{G} .
- $A_{i,j}$, $1 \leq i < j \leq n$, son símbolos que permitirán representar las posibles aristas de \mathcal{G} .
- T_i , $1 \leq i \leq n$, son símbolos que se usarán en el proceso de división de los compartimentos C_2 .
- B_i, R_i, G_i , $1 \leq i \leq n$, son símbolos que codifican los tres colores (como en el caso de los sistemas P de tejido).
- El símbolo a se emplea únicamente en el compartimento C_1 para seleccionar una única respuesta que sería, posteriormente, enviada al entorno.
- s, X, Y son símbolos que se emplean en el compartimento C_2 .
- Z es un símbolo que se envía al compartimento C_1 .
- **yes, no** son las posibles respuestas: llegando una de ellas al entorno procedente de C_1 en el último paso de computación.
- Los símbolos X_1, \dots, X_{2n+3} se usan como contador para saber el máximo número de pasos ($2n + 2$) requerido por la única entrada posible desde C_2 .

- $IO = \{\text{yes}, \text{no}\}$.
 - $T = \{t_1, t_2\}$, siendo $t_1 = (R_1, \sigma_1)$ y $t_2 = (R_2, \sigma_2)$; es decir, existen dos tipos de compartimentos t_1 y t_2 . Las estrategias de ejecución $\sigma_1 = \sigma_2$ es la de maximal paralelismo con la restricción de que a cada compartimento en un paso de computación se le aplicará, a lo sumo, una regla de división.

Los conjuntos de reglas del sistema son los que se relacionan a continuación.

- R_1 es el siguiente conjunto de reglas:
 - ★ $r_{1,i} : X_i \rightarrow X_{i+1}, 1 \leq i \leq 2n + 2.$
 - ★ $r_{1,2n+3} : aZ \rightarrow (yes, 0).$
 - ★ $r_{1,2n+4} : aX_{2n+3} \rightarrow (no, 0) \quad \{ \geq \overline{Z}$

Las reglas $r_{1,i}, 1 \leq i \leq 2n + 2$ se encargan de contar los primeros $2n + 2$ pasos; en dichos pasos, por cada solución encontrada se envía un objeto Z de C_2 a C_1 ; cuando se recibe uno o más objetos Z de los compartimentos de tipo C_2 , es decir, hay al menos una solución, entonces el compartimento C_1 envía *yes* al entorno; en otro caso, cuando no se haya recibido ningún objeto Z , tras $2n + 3$ steps se envía un objeto *no*;

- R_2 es el siguiente conjunto de reglas:

Reglas de división de membranas:

- ★ $r_{2,2i-1} : [A_i]_2 \rightarrow [R_i A_{i+1}]_2 [T_i]_2 \{ = s \}$.
 - ★ $r_{2,2i} : [T_i]_2 \rightarrow [B_i A_{i+1}]_2 [G_i A_{i+1}]_2, 1 \leq i \leq n-1$.
 - ★ $r_{2,2n-1} : [A_n]_2 \rightarrow [R_n X]_2 [T_n]_2 \{ = s \}$.
 - ★ $r_{2,2n} : [T_n]_2 \rightarrow [B_n X]_2 [G_n X]_2$.

Estas reglas se aplican como máximo en $2n$ pasos y se obtienen todas las posibles coloraciones para los n vértices.

Reglas de reescritura y comunicación:

La guarda que aparece en la regla $r_{2,2n+1}$ contiene $3n(n - 1)/2$ términos y comprueba, para cada par $1 \leq i < j \leq n$, si el color de los nodos i y j es el mismo. En tal situación, si el compartimento actual contiene el objeto s , entonces se elimina dicho objeto y, a partir de ese instante, ninguna regla se aplicará al compartimento.

La regla $r_{2,2n+2}$ transformará X en Y una vez que se han completado todos los cálculos.

La regla $r_{2,2n+3}$ se aplicará cuando haya una solución en el compartimento actual C_2 y, además, enviará el objeto Z al compartimento C_1 .

- $G = (\{1, 2\}, \{\{1, 2\}\})$.
- $C_1 = (t_1, w_1), C_2 = (t_2, w_2)$, donde $w_1 = a X_1$, $w_2 = A_1 s$.
- $i_{out} = 0$; es decir, la salida del sistema está codificada en el entorno.
- $i_{in} = 2$; es decir, el compartimento de entrada es C_2 .

Téngase presente que la instancia $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ del problema 3-COL será procesada por el *simple kernel P system* $\Pi(s(\mathcal{G}))$ con multiconjunto de entrada $cod(\mathcal{G})$.

Para una descripción detallada de la verificación de la familia de *simple kernel P systems* presentada, se puede consultar el artículo [62]. En el mismo se propone, además, una segunda solución alternativa involucrando reglas de reetiquetado y se analiza en profundidad diversas propiedades relacionadas con la complejidad de las distintas soluciones, tomando como referencia la solución mediante sistemas P de tejido presentada en la sección 1.5.3 del capítulo 1.

6

Caso de estudio 1: Modelización de un ecosistema real

En el capítulo 2 se ha justificado la necesidad de diseñar modelos formales que capturen de alguna forma procesos relevantes que suceden en la vida real, con objeto de extraer de ellos algún tipo de información que, en última instancia, pueda proporcionar nuevo conocimiento acerca de los fenómenos objetos de estudio.

Los dos próximos capítulos de esta memoria están dedicados a analizar la evolución de las aplicaciones de simulación de modelos de la computación celular con membranas, basadas en P-Lingua, y que van desde las aplicaciones visuales precursoras *ad-hoc*, hasta algunas aplicaciones importantes a las que ha dado lugar MeCoSim, fundamentalmente en el campo de los ecosistemas y la dinámica de poblaciones, pasando por una incursión interesante en la resolución de problemas computacionalmente duros. Más concretamente, nos vamos a centrar en la aportación realizada dentro del entorno de simulación software MeCoSim, así como en la metodología propuesta en la Sección 4.3 para la modelización matemática (en particular, computacional) basada en la computación celular con membranas.

El objetivo principal de este capítulo es presentar con todo detalle una aplicación de la plataforma MeCoSim para el diseño de un plan de gestión del embalse de Ribarroja en la zona del río Ebro, gestionado por la compañía Endesa S.A., con relación a una especie exótica invasora: el *mejillón cebra* (trabajo recientemente publicado, [104]). Esta especie está causando estragos tanto a nivel medioambiental, desplazando a muchas especies autóctonas, como a nivel económico, en tanto que provocan obstrucciones importantes en maquinarias, sistemas de conducción, etc.

El presente capítulo está estructurado como sigue. En la primera sección se trata de dar una visión general de algunos sistemas complejos analizados en el campo de la dinámica de poblaciones y, en particular, en algunos ecosistemas reales centrados en una especie en peligro de extinción: el *quebrantahuesos*. La Sección 6.2 está dedicada a presentar las características singulares del ecosistema real objeto de estudio, con especial atención a la biología de la especie del *mejillón cebra* y al embalse de Ribarroja. En la Sección 6.3 se plantea el marco de modelización basado en los *Population Dynamics P systems* y en la Sección 6.4 se presenta el diseño de modelo propuesto. La última sección del capítulo está dedicada al código P-Lingua que corresponde al modelo diseñado y a la presentación de una aplicación basada en MeCoSim adaptada al caso objeto de estudio.

6.1. Modelización computacional de ecosistemas reales basada en sistemas PDP

Desde el año 2007, el Grupo de Investigación en Computación Natural (RGNC) de la Universidad de Sevilla ha venido estudiando diversos sistemas dinámicos complejos que se dan en la vida real, fundamentalmente en el campo de los ecosistemas y la dinámica de poblaciones, desde la perspectiva de la modelización computacional basada en sistemas P. El estudio se ha desarrollado tanto en el ámbito de biología de la conservación como de la gestión de recursos naturales y ecosistemas, generalmente afectados por una serie de factores naturales y humanos que tienen un gran impacto sobre la dinámica de las especies y organismos presentes en este tipo de sistemas. De manera simultánea, dentro del marco de la Computación Celular con Membranas, el RGNC ha analizado otros sistemas complejos de interés que abarcan un amplio espectro que va desde fenómenos a nivel que podríamos denominar *micro* (molecular, celular, tisular, etc.) hasta procesos a nivel que podríamos denominar *macro* tales como pandemias, fenómenos de índole económico, etc.

El objetivo de esta sección es ilustrar la aportación realizada por el RGNC en lo concerniente al estudio de sistemas dinámicos complejos, mediante la modelización, simulación y análisis dentro del marco de la computación celular con membranas, apoyados en el empleo de la plataforma proporcionada por P-Lingua [58, 120] y MeCoSim [122, 153], ilustrada en el capítulo 4, y haciendo uso de la metodología que en él se describe.

La mayor parte de los trabajos desarrollados en el ámbito de los sistemas complejos a nivel *macro*, se han llevado a cabo en el ámbito de la modelización formal de ecosistemas. Ésta ha experimentado un cambio sustancial e interesante en las últimas décadas, desde un punto en el que los modelos formales eran desarrollados por investigadores de *Computer Science* con pocas implicaciones prácticas, hasta

6.1. Modelización computacional de ecosistemas reales basada en sistemas PDP173

llegar a una situación en la que ecólogos de campo, gestores de la fauna silvestre, así como el personal encargado del mantenimiento y conservación de ecosistemas, adoptan una serie de decisiones apoyadas y justificadas por las hipótesis plausibles que proporcionan los modelos diseñados. Esta deriva interesante en el rol de los modelos computacionales de fenómenos y procesos interesantes de la vida real, debe ser tenida muy en cuenta por los diseñadores de los modelos, tratando de proporcionar a los usuarios de los mismos (los expertos en cuestión) no solamente un modelo que represente de alguna forma una abstracción de los fenómenos de su interés sino también los mecanismos adecuados para proporcionar herramientas de ayuda para el análisis de los procesos objeto de estudio, la formulación de hipótesis y la posible toma de decisiones, una vez analizados los escenarios de interés (para los expertos) que han sido introducidos y simulados mediante las herramientas correspondientes.

El primer trabajo de modelización de ecosistemas que se desarrolló en el marco de la computación celular con membranas [28, 29] vio la luz en 2008, y trataba acerca de la dinámica poblacional del *quebrantahuesos* en el Pirineo catalán.

El *quebrantahuesos* es una de las especies menos frecuentes en Europa, con una consideración de vulnerable a nivel europeo dentro de la lista de especies amenazadas, en un estado de conservación desfavorable (SPEC 3), con una población de unas 80 a 100 parejas en la Unión Europea [11], con una serie de amenazas conocidas (algunas derivadas de leyes que obligan a la retirada del monte de los ungulados domésticos muertos) y una baja cobertura de zonas de especial protección (SPAs) (tan solo el 2.5 % de las parejas se encuentran en dichas zonas, a 2004).

La *biología de la conservación* estudia la Naturaleza y la biodiversidad en la Tierra con el objetivo de proteger a las especies, sus hábitats y ecosistemas de tasas de extinción excesivas y la erosión de interacciones bióticas. De este modo, el foco principal se sitúa en dos objetivos principales:

1. Evaluar el impacto del ser humano sobre la biodiversidad biológica.
2. Desarrollar aproximaciones prácticas para prevenir la extinción de las especies.

En el trabajo publicado en [28, 29] se presentaba un novedoso modelo matemático/computacional aplicado al ecosistema citado, estudiando en la zona objetivo la dinámica de población del *quebrantahuesos* en conjunción con otras cinco especies/subfamilias que proporcionan los huesos de los que se alimentan. Como se pone de relevancia en el artículo, la aportación de los sistemas P para el estudio de este problema se traduce en la provisión de un marco de modelización computacional de alto nivel, a diferencia de otros marcos clásicos no computacionales, de los que ha sido necesario llevar a cabo aproximaciones para poder realizar simulaciones mediante ordenador.

El marco citado permite la integración de los aspectos estructurales y dinámicos de los ecosistemas objetos de estudio, de forma comprensible y relevante. Los modelos que se desarrollan en ese marco proporcionan sistemas de carácter discreto, representando cada individuo a través de objetos presentes en el sistema con una cierta multiplicidad, tanto para animales vivos en la población como para huesos. La estocasticidad e incertidumbre inherente a esta dinámica poblacional de los ecosistemas se captura a través del empleo de estrategias probabilísticas. En consecuencia, no es posible realizar una validación formal de esos modelos.

Para poder llevar a cabo una validación experimental de un modelo diseñado en ese marco, inicialmente se hizo uso de un simulador completamente *ad-hoc* [1] que incorpora en código fuente C++ toda la información sobre los datos específicos del escenario bajo estudio: (a) las reglas del sistema; (b) los parámetros y las constantes; (c) el resto de elementos del modelo; y (c) el propio motor de simulación. De este modo, no existe una separación de roles y responsabilidades en el software, incluyendo datos, modelo y simulador de forma monolítica en el código fuente. Por tanto, no se dispone de diferentes visiones de programador, diseñador y usuario final. El software proporcionado devuelve los datos de la evolución del escenario incluido en el código, con el modelo dado y el simulador desarrollado. Así pues, no existe la posibilidad de que el usuario final introduzca distintos escenarios de interés, modifique levemente el modelo o introduzca cierta variabilidad en la configuración de la simulación. La salida proporciona un fichero con el número de animales de cada especie por edades al final de cada año de un rango específico indicado en el propio código fuente.

Dentro de la misma línea de investigación abierta por el trabajo anterior, en [27] se presentó un modelo solventando algunas carencias del anterior. Dicho modelo incorporaba la regulación de la densidad de población, las limitaciones sobre la alimentación en caso de escasez, la tasa de crecimiento (“reproducción”) del quebrantahuesos era variable y la competición por determinados recursos como espacio y alimento, debido a la inclusión de varias especies de aves carroñeras y de nuevas especies de ungulados. La modelización de este ecosistema requiere una serie de parámetros biológicos obtenidos experimentalmente (por los expertos) y de la literatura, de tal modo que incorporen los datos necesarios a los procesos básicos, las poblaciones iniciales y el entorno físico (superficie, orografía, etc.)

A medida que los modelos van incorporando más parámetros relacionados con la dinámica de las especies y se van añadiendo más ingredientes que acorte las distancias entre modelo y realidad, se altera la visión acerca de los mismos en tanto que comienzan a ser considerados por los expertos como herramientas que pueden asistir y ayudar en la mejora en la gestión de los fenómenos objeto de estudio. Por ello, se hace necesario poder disponer de plataformas software más potentes y flexibles, capaces de facilitar el desarrollo de los simuladores, el diseño de los modelos y, por último, su utilización posterior por usuarios finales que puedan validar los

6.1. Modelización computacional de ecosistemas reales basada en sistemas PDP175

resultados y realizar experimentos virtuales; es decir, simulaciones bajo diferentes condiciones/escenarios iniciales (de interés para los expertos), que le ayuden en su gestión.

En el trabajo antes citado, se hace uso del framework **P-Lingua** [58] para llevar a cabo tanto la especificación como la simulación del modelo. Así, mientras que en el primer artículo relativo al quebrantahuesos, se trabajaba con un software monolítico que no distinguía roles ni dotaba de flexibilidad a la hora de modificar los datos de entrada o el modelo, en [58] se proporciona una clara separación entre: (a) el intérprete del lenguaje de especificación y el motor de simulación, proporcionados por los desarrolladores de **P-Lingua**; (b) el modelo realizado por los diseñadores de sistemas P, especificando la estructura, reglas, multiconjuntos, etc. del sistema; y (c) los datos de entrada, proporcionados por los usuarios finales.

Las primeras versiones de los modelos diseñados para un ecosistema real del quebrantahuesos en la zona pirenaico-catalana, así como sus plataformas software fueron las precursoras de las herramientas y la metodología detallada en el capítulo 4, construyendo un simulador que permita analizar la evolución del ecosistema bajo diferentes condiciones iniciales. Durante la realización del trabajo plasmado en el artículo [58] no se contaba aún con el entorno visual de simulación **MeCoSim**. Sólo estaba disponible el framework **P-Lingua**, donde el usuario era un diseñador de modelos basados en sistemas P, lejano al ámbito del ecólogo (que podría aportar generalmente mayor riqueza de conocimientos sobre el fenómeno objeto de estudio pero no tendría por qué estar interesado o tener conocimientos acerca del diseño de los sistemas P).

En consecuencia, para permitir trabajar al usuario final (ecólogo) con los sistemas P subyacentes a modo de caja negra, la visión del simulador debería ocultarles los detalles del modelo y centrarse en el dominio del problema, interactuando este tipo de usuarios con escenarios en los que introducir los datos de entrada del problema, a la vez que les permitiera visualizar la evolución de la población. Para ello, se desarrolló una primera aplicación visual *ad-hoc* para proporcionar al usuario ecólogo una interfaz en la que introducir datos de su escenario, modificar fácilmente los parámetros de entrada y visualizar la evolución del ecosistema mediante salidas, mostrando las multiplicidades de los objetos y trabajando internamente con el modelo proporcionado por los diseñadores de sistemas P.

En vista del éxito cosechado con las modelizaciones computacionales del quebrantahuesos en el Pirineo catalán, a requerimiento de expertos de la compañía Endesa S.A., se comenzó a trabajar en la modelización de un nuevo ecosistema que afectaba a la citada compañía. Se trataba del embalse de Ribarroja en la zona del Rio Ebro y, concretamente, del ecosistema de dicho embalse relacionado con una especie exótica invasora: el *mejillón cebra*. Dicha especie ha producido importantes daños tanto desde el punto de vista ecológico como económico en los últimos años.

Este trabajo contó con el interés y participación activa de expertos de la compañía citada, al ser afectada directamente por la corrosión de depósitos y tuberías que la especie ocasiona en el embalse gestionado por dicha empresa.

El objetivo consistió en diseñar un modelo computacional basado en sistemas P que proporcionara a la compañía una visión de conjunto en forma de modelos que pudieran de algún modo predecir la evolución de la densidad de larvas de esta especie. De esta manera, en base a este mayor conocimiento y las simulaciones realizadas a partir del modelo, la compañía podría realizar las pertinentes actuaciones para tratar de erradicar la especie en el embalse o, en su defecto, minimizar el número de individuos y, por ende, los daños ocasionados en el embalse, mediante la implementación de un plan de control. La primera mención a este trabajo apareció en [26], donde se indicaban también las herramientas software desarrolladas para la experimentación virtual de los modelos, tanto del quebrantahuesos como del mejillón cebra.

Naturalmente, el mejillón cebra requería distinto tipo de información, diferentes parámetros, distintos tipos de información de entrada, distintos elementos de interés en el modelo y diferentes salidas para el usuario. Por ello, se hizo necesario el desarrollo de una nueva aplicación software visual para sus diseñadores y usuarios finales, con el esfuerzo de desarrollo de software correspondiente.

Tras los trabajos anteriormente citados, aparecieron diferentes expertos ecólogos interesados en que se pudieran realizar estudios usando el marco de modelización computacional basado en sistemas P. En este contexto, se han llevado a cabo (y se continúa en la actualidad) diversas colaboraciones científicas entre las que destacamos los siguientes estudios: (a) ecosistemas relacionados con aves carroñeras en el pirineo navarro y en Swaziland (South Africa); (b) ecosistemas afectados por cambios aleatorios en el entorno relacionados con la temperatura, la lluvia, etc. (desarrollo y crecimiento de ciertos anfibios en estanques); (c) reintroducción de una especie de pájaros en la zona pirinaico-catalana (el gregol, *hazel grouse*); (d) ecosistemas muy sensibles a inundaciones y altas precipitaciones (predicción posibles escenarios de extinción en especies como el tritón) y (d) efecto de los pestivirus en la dinámica de la población de rebecos.

De este modo comenzó a crecer el número de aplicaciones software a desarrollar y mantener, siendo imprescindible implementar un proceso de adaptación correspondiente a cada nuevo cambio en los modelos o bien en las entradas y las salidas requeridas. Como consecuencia de todo ello, el mantenimiento de todas esas aplicaciones software para los diseñadores y usuarios finales comenzó a ser inviable. Se hacía imprescindible buscar una solución y ésta pasó por la identificación de los elementos comunes en toda aplicación de gestión de ecosistemas, satisfaciendo las necesidades de diseñadores y usuarios finales, trabajando con distintos modelos, distintas entradas y salidas y poniendo el foco en distintos elementos de los modelos.

6.1. Modelización computacional de ecosistemas reales basada en sistemas PDP177

Éste sería el punto de partida para el desarrollo de una aplicación de propósito general configurable que nos permitiera definir cada aplicación de simulación específica. Por tanto, en cierto sentido se trata de un meta-simulador, un entorno de simulación general que nos permite seguir la secuencia:

1. Definir mediante configuración, la aplicación software específica de simulación a generar, evitando así la necesidad de desarrollar un modelo basado en sistemas P, a medida para cada problema.
2. Trabajar con nuestros modelos en la aplicación de simulación generada en el paso anterior, de tal modo que el diseñador de sistemas P pueda cargar y depurar el modelo en dicha aplicación.
3. Realizar una validación experimental del modelo y experimentación virtual por parte del usuario final, a partir de la aplicación generada en el paso 1 y con el modelo cargado y depurado en el paso 2. De esta manera se proporciona a este usuario los datos finales, mediante las pestañas de entrada definidas en el paso 1 para la aplicación particular, permitiendo la simulación del modelo cargado para esos datos particulares del escenario de interés y obtener las salidas igualmente definidas en el paso 1 aplicadas a los datos introducidos en este paso 3.

Así, para cualquier modelo de ecosistemas que se venían estudiando hasta ahora, los componentes básicos de una interfaz para que el ecólogo pudiera trabajar de forma transparente con los ingredientes de su problema debían incluir, al menos, los siguientes:

- Mecanismo de entrada de datos para introducir los parámetros del modelo: esperanzas de vida de cada especie, ratios de mortalidad, fertilidad, necesidades alimenticias, etc. Estos parámetros son manipulados por los ecólogos para permitir la introducción de diferentes escenarios de interés, y sirven de entrada al modelo basado en sistemas P proporcionado por los diseñadores. Los parámetros correspondientes se pueden emplear tanto en la generación de la configuración inicial del sistema P como en la confección de las reglas probabilísticas correspondientes, según se haga uso de los parámetros en el fichero P-Lingua asociado.
- Mecanismo de entrada de datos para introducir las poblaciones iniciales de cada especie presente en el ecosistema. Como en el caso anterior, los usuarios ecólogos pueden proporcionar esos datos y dotar así al sistema P subyacente de los datos poblacionales iniciales para el modelo.

- Mecanismo de entrada de datos para introducir las condiciones del entorno, pudiendo estar subdividido en distintas zonas con distintas condiciones de suelo, clima, etc.
- Mecanismo de modificación de los parámetros de simulación generales: número de ciclos a simular (años, meses, días, etc. dependiendo del problema objeto de estudio), y número de simulaciones a realizar (dado que se trata de sistemas probabilísticos, no estaremos interesados en una simulación concreta sino en hallar una imagen más global a través de un conjunto de simulaciones de las que extraer las poblaciones medias y sus desviaciones estándar).
- Mecanismo de visualización de las salidas, mostrando al usuario final aquella información del modelo en la que está interesado, y de manera entendible por el mismo, abstrayéndolo de los detalles propio del modelo subyacente basado en sistemas P y centrándose en la información útil para la gestión del ecosistema. Podría incluir tablas de salida, gráficos, etc.

Estas ideas constituyeron el punto de partida de **MeCoSim**, cuyo espectro se amplió para cubrir las necesidades mucho más generales de cualquier otro modelo basado en sistemas P, incluyendo las distintas variantes del modelo de computación, así como los algoritmos de simulación asociados, incluidos en el marco **P-Lingua**. De esta manera se propició la definición de aplicaciones de simulación para todo tipo de problemas, con los mecanismos flexibles y extensibles detallados en el capítulo 4.

Para clarificar la aportación realizada por **MeCoSim** en el ámbito de los ecosistemas y, en general, de algunos problemas relevantes de la vida real, ilustraremos en el siguiente apartado 6.2 el proceso llevado a cabo mediante el segundo caso de estudio comentado anteriormente: el mejillón cebra en el embalse de Ribarroja.

6.2. El mejillón cebra en el embalse de Ribarroja

El mejillón cebra es una especie exótica invasora que, en los últimos años, ha producido importantes daños tanto desde el punto de vista medioambiental como del económico, allá por donde ha aparecido. La presencia de esta especie se detectó en España por primera vez en 2001, en el embalse de Ribarroja, lo que alertó a la compañía Endesa S.A., encargada de la gestión del mismo, al verse afectada seriamente por la corrosión de depósitos y tuberías que la especie ocasionaba en el mismo.

El diseño de modelos computacionales y el desarrollo de herramientas robustas, basadas en esos modelos, que permitan simular tendencias espaciales y poblacionales, es fundamental para la toma de decisiones en el manejo y la gestión de ecosistemas

[81, 103]. Ello se debe a que los modelos pueden proporcionar ese conocimiento requerido para mitigar los efectos negativos que produce la especie y, a la vez, permiten reproducir la dinámica del sistema ante posibles escenarios de interés para los expertos. Muchos de esos escenarios no podrían ser reproducidos en la vida real debido al impacto que supondría sobre las especies del embalse, la realización de ese tipo de experimentos reales.

En este contexto, existen diversos paradigmas de modelización para la simulación de dinámica de poblaciones. Sin lugar a dudas, la más ampliamente utilizada está basada en sistemas de ecuaciones diferenciales ordinarias o parciales. Se trata de una aproximación continua y determinista, con las restricciones a las que suelen estar sometidas este tipo de aproximaciones. En ella se estudian y simulan las concentraciones medias y los flujos de determinadas variables en flujos de control. Sin embargo, la complejidad de algunos fenómenos y los modelos derivados de éstos para poder estudiarlos en profundidad, suelen involucrar un elevado número de parámetros, algunos de cuyos valores son desconocidos o no se concen con una gran precisión, siendo los modelos muy sensibles a la variación de los mismos.

En lo que respecta al volumen de parámetros e imprecisión acerca del conocimiento de algunos de ellos, la problemática planteada ha conducido a la búsqueda de nuevos horizontes de modelización basados en paradigmas computacionales alternativos. En este sentido se han desarrollado modelos computacionales basados en autómatas celulares, agentes, redes neuronales y otros que permiten estudiar con éxito ciertos problemas complejos no lineales.

Dentro del paradigma genérico de modelos computacionales bioinspirados, la computación celular con membranas y sus dispositivos, los sistemas P, han emergido con fuerza en los últimos años como una buena alternativa para la modelización de sistemas complejos y dinámica de poblaciones. Los sistemas P presentan algunas propiedades interesantes, como su estructura compartimentalizada, su modularidad y la facilidad que ello conlleva para adaptar el modelo de un determinado fenómeno objeto de estudio cuando se introducen nuevos ingredientes en el mismo. Esto no suele suceder en las aproximaciones basadas en sistemas de ecuaciones diferenciales, en donde pequeños cambios en un fenómeno modelizado puede implicar comenzar de nuevo el proceso de modelización.

El paradigma computacional desarrollado en la computación celular con membranas proporciona modelos discretos basados en individuos, que suelen ser más intuitivos para los ecólogos que otros paradigmas. Ello se debe a la relativa cercanía entre la interacción entre individuos en un entorno de un ecosistema y en su correspondencia en la interacción de objetos en un comportamiento de un entorno de un sistema P en el marco de los PDP systems.

El potencial y la flexibilidad de adaptación presentada por los sistemas P para el estudio de dinámicas poblacionales, hace pensar que, por ejemplo, puedan resultar

especialmente útiles en la modelización de ecosistemas fluviales, formados por un mosaico de hábitats dinámicos en los que intervienen múltiples disciplinas, lo que requiere la participación de expertos en diferentes áreas. Junto con un conocimiento exhaustivo de la biología y del comportamiento de la especie a estudiar y las especies que interactúan con ella, es fundamental entender la influencia del movimiento de los fluidos en la dinámica y distribución de los organismos acuáticos, con el fin de poder recabar los parámetros hidráulicos más relevantes e introducirlos en los modelos que han de permitir la toma de decisiones acerca de la gestión de ríos o embalses [40].

Ahora bien, con vista al empleo de este paradigma, como ocurriría con cualquier paradigma de modelización y simulación de forma práctica en la gestión de un ecosistema, en este caso ubicado en el embalse de Ribarroja, una vez diseñado el modelo computacional y validado experimentalmente con la ayuda de los expertos, éstos deberían tener la posibilidad de interactuar fácilmente con el sistema, planteando escenarios de interés y recibiendo resultados que faciliten su gestión y la toma de decisiones más informada/justificada.

Para poder llevar a cabo lo dicho anteriormente, sería muy importante poder disponer de un equipo permanente formado por expertos ecólogos, diseñadores de modelos basados en sistemas P y desarrolladores de software que pongan en producción las herramientas adecuadas. De ese modo, cada pequeño cambio en un escenario de interés planteado por los usuarios finales requerirá una acción inmediata de actualización del modelo por parte de los diseñadores y del software por parte de los desarrolladores.

Sin embargo, esta visión no nos parece del todo realista. Por ello, se debería dotar de herramientas flexibles y personalizables que se pudieran adaptar al estudio de distintos ecosistemas, según las necesidades de los usuarios finales. Este es el enfoque que se le ha dado a nuestro trabajo con el fin de poner el esfuerzo invertido por nuestro grupo de investigación, a disposición de cualesquiera otros diseñadores de modelos y otros posibles usuarios finales que requieran de este tipo de herramientas de modelización y simulación, sin necesidad de contar, específicamente, con un equipo de desarrollo de software expertos en Java, C++, etc.

Mediante este enfoque, en el que la metodología descrita en el capítulo 4 y las herramientas P-Lingua y MeCoSim detalladas en el capítulo 4 juegan un papel crucial, se obtienen una serie de ventajas que nos permitirán ser más efectivos desde el punto de vista operativo y económico:

- El planteamiento de los distintos escenarios por parte de los **gestores** únicamente implicará a los mismos, introduciendo los datos de su escenario de una forma visual y recibiendo los resultados de sus simulaciones. De esta manera no será necesario recurrir ni a los diseñadores de los modelos ni a los desarrolladores del software, debido al entorno amigable proporcionado por MeCoSim.

- El diseño de modelos y la introducción de posibles cambios en los mismos únicamente requerirá a los gestores contar con los **diseñadores** de modelos basados en sistemas P, no siendo necesaria el desarrollo de herramientas de modelización y simulación ya proporcionadas por P-Lingua.
- El desarrollo de la nueva aplicación de gestión adaptada a las necesidades del gestor empleadas en el primer punto mencionado, no requerirá tampoco la presencia de los desarrolladores, ya que no será necesario tal desarrollo al llevarse a cabo mediante la simple definición de la aplicación configurando un pequeño fichero xls para la plataforma MeCoSim.

En la sección 6.2.1 se analizará la complejidad de la problemática relacionada con el desarrollo de un modelo formal del mejillón cebra en un ecosistema real. En la sección 6.4 se describirá con detalle el modelo diseñado empleando el marco de los sistemas PDP y en la sección 6.4.2 se ilustrará cómo poner a disposición de los gestores herramientas adecuadas basadas en MeCoSim sin necesidad de desarrollar software a medida, mediante la simple definición de nuestra necesidad (entradas, salidas y mecanismos específicos) y su carga junto con el modelo en MeCoSim.

6.2.1. Descripción del problema

Como ya hemos comentado, el mejillón cebra es una especie exótica invasora. Al parecer, fue descrita originalmente en un afluente del río Ural en la cuenca del Mar Caspio y la especie se ha ido expandiendo rápidamente a lo largo de Norteamérica y Europa durante las últimas décadas [83, 98, 147]. Se puede considerar una especie “oportunista” en el sentido de colonizar una amplia variedad de tipos de hábitats.

Cuando el mejillón cebra invade un nuevo sistema acuático, provoca cambios ecológicos significativos ya que se trata de una especie extremadamente agresiva. Los individuos de esta especie son auténticos ingenieros del ecosistema, capaces de alterar tanto la estructura como el funcionamiento del mismo [83]. A través de acciones coordinadas, modifican el hábitat para el resto de organismos, afectando a las interacciones tróficas e influyendo sobre cuestiones tan importantes como son las características ópticas de la columna de agua, la composición de las especies autóctonas, la mineralización de los nutrientes y la materia orgánica, la disponibilidad de oxígeno y las tasas de sedimentación, entre otras [90, 109, 83, 112, 157, 101, 41].

Conviene, también, destacar los efectos devastadores de esta especie en lo concerniente al empleo del agua tanto para uso doméstico como agrícola, así como en la explotación de centrales eléctricas. Así por ejemplo, produce cambios en la composición del agua, incrementando sustancialmente los costes operativos y de mantenimiento de los trabajos hidráulicos para limpieza y tratamiento de aguas, así como para el desbloqueo de desagües y tuberías, en los que quedan fijados de forma masiva los individuos [95, 80, 39].

Una consecuencia negativa nada desdeñable es la pérdida de atracción y, en algunos casos, incluso el rechazo turístico de zonas recreativas asociadas con la pesca, la vela o la natación, ya que puede producir lesiones, cortes, daños en embarcaciones y alteraciones en la fauna local [154, 53].

Por tanto, el impacto económico causado por la invasión del mejillón cebra es a menudo difícil de cuantificar, pero se han realizado estimaciones que lo sitúan cerca del millardo de dólares anuales en USA [128] o algunos millones de libras en Gran Bretaña [110], simplemente por hacernos una idea de su magnitud. Si centramos nuestra atención en el área que nos ocupa, los estudios realizados en la cuenca del Ebro arrojan unos costes económicos derivados de esta invasión (tanto de los problemas de funcionamiento de las instalaciones afectadas como de los gastos añadidos por la limpieza y tratamientos de control de las mismas, afectando a los sectores energético, industrial, agrícola, lúdico-deportivo, administraciones públicas y abastecimientos) en torno a 2,6 millones de euros entre 2001 y 2005 [86], alcanzando los 11,6 de euros entre 2005 y 2009 [54], con el mayor coste centrado en las Administraciones Pùblicas con más del 50%, seguidas de las empresas energéticas con un 26,4%.

La rápida expansión de esta especie y sus importantes efectos asociados a nivel medioambiental y socio-económico han llevado a la realización de numerosos estudios, centrados en el estudio de la biología de la especie, su dispersión geográfica y su influencia en el entorno que invaden. Muchos de ellos incluyen ensayos de métodos para tratar de controlar o erradicar la especie. [141, 99, 152, 147, 71, 72, 148, 140]

6.2.2. Biología de la especie

Las características de la biología de la especie, la variedad de factores que intervienen en la evolución del hábitat ocupado y el funcionamiento de los embalses, convierten al estudio de la dinámica de población del mejillón cebra en el ecosistema acuático constituido por el embalse de Ribarroja, en un sistema complejo con gran cantidad de factores y procesos involucrados.

A continuación se analizarán los principales elementos del sistema para ilustrar en la siguiente sección el modelo presentado en este caso de estudio. Este modelo pretende capturar el comportamiento, la dinámica del sistema con suficiente precisión, especialmente en lo que respecta a las larvas del mejillón. De esta forma, el empleo de aplicaciones de experimentación virtual basadas en los sistemas PDP, dotando de la posibilidad de plantear y simular los escenarios que resulten de interés y analizar sus hipotéticas consecuencias, podrá servir como herramienta de ayuda a la toma de decisiones, proporcionando información adicional a los gestores de la reserva para que puedan diseñar actuaciones encaminadas al control o erradicación de la especie invasora.

El mejillón cebra (*Dreissena polymorpha*) es un molusco bivalvo acuático que presenta un rápido ciclo vital con un alto potencial reproductivo, alta movilidad de las larvas y una gran capacidad de dispersión[92, 14, 134].

No hay un consenso establecido en cuanto a su esperanza media de vida, aunque el dato más aceptado sitúa su media en torno a 2-3 años. La esperanza de vida máxima puede llegar a unos 4-5 [91, 149, 152] años, en determinadas zonas hasta 3-9 [91] e, incluso, hasta 15 años o más [90, 84]. El adulto llega a un tamaño medio máximo de unos 35 mm [112], si bien otras fuentes llegan a hablar de 40mm [91, 159] y hasta 50mm [152].



Figura 6.1: Mejillón cebra

Su ciclo vital presenta dos fases: la primera, la fase larvaria, tiene lugar en suspensión en la columna de agua en forma planctónica; tras un periodo determinado pasa a otra sésil, sujetada al sustrato (forma bentónica, en el fondo de la cuenca en la que se encuentre), forma en la que permanece durante toda su etapa juvenil y adulta.

El mejillón cebra es una especie dioica y la proporción de sexos se sitúa en torno a 1:1. El número de óvulos que emite cada hembra por cada ciclo reproductivo depende de su tamaño, que viene influido por la edad [143, 33].

En [149] se analiza la influencia que pueden tener diferentes factores sobre las medidas del mejillón, incluyendo el crecimiento de su concha para cada instante (año) $t + 1$ en relación al tamaño en el instante t . Ajustando un modelo de regresión logarítmica sobre los valores correspondientes se obtiene la ecuación:

$$Y = 6.6649 \ln x - 13.235 \quad (6.1)$$

en donde la variable Y representa el tamaño en milímetros y la variable X representa la edad en años.

En [144, 156, 149] se estima el número de huevos en función de la longitud de la concha mediante la expresión:

$$Z = 0.4 \cdot Y^{4.4} \quad (6.2)$$

Utilizando las expresiones 6.1 y 6.2 se obtiene que el número de huevos para mejillones de 20 a 53 semanas se puede estimar mediante la expresión siguiente:

$$Z = 12.15 d^2 + 117.81 d - 3391.5, R^2 = 0.9999 \quad (6.3)$$

en donde la variable d representa las semanas de vida.

Para más de 53 semanas tendríamos:

$$Z = 1920.5 d - 79065, R^2 = 0.9993 \quad (6.4)$$

Ségun se ha estudiado, las hembras pueden producir entre 30 o 40.000 [91] y 1.000.000 de gametos [109] o incluso 1.610.000 según [22]

Las larvas permanecen en la columna de agua de 15-28 días [14, 80] o según otros autores de 3-5 semanas [134], en función de la temperatura del agua u otras condiciones ambientales. Parte de las larvas que se encuentran en la columna de agua son depredadas por los propios adultos, de tal manera que si la densidad de adultos es muy grande, entonces va decreciendo el número de larvas que sobreviven. Después de este período larvario, éstas caen por el efecto de la gravedad sobre el substrato.

Ségun [146], el reclutamiento de jóvenes mejillones en función de la densidad de mejillones adultos viene dado por la ecuación:

$$R = -2 \cdot X + 1, X \leq 0.5 \quad (6.5)$$

siendo R el tanto por ciento de larvas que se reclutan y la variable X representa el tanto por ciento de población respecto a la carga máxima. Si la población supera el 50 % de la carga máxima el reclutamiento es el 0 %.

La mortalidad entre el periodo de desove hasta la etapa juvenil inicial puede alcanzar el 98 %, si bien algunos autores llegan a situarlo en el 99,8 %, fundamentalmente en invierno [91]. Pese a la elevada mortalidad, su enorme tasa de natalidad consigue mantener una capacidad de crecimiento muy alta.

6.2.3. La especie en el embalse de Ribarroja

En el entorno objeto de nuestro estudio se suelen observar dos periodos reproductivos anuales: uno que va desde finales de marzo hasta junio, y otro de mediados de octubre a principios de noviembre. Esto coincide con los observados por otros autores en el hemisferio norte [145, 21]. De estos periodos, el más importante es el primero; en él la hembra libera, aproximadamente, el 80 % de los gametos, pudiendo éstos llegar a la madurez sexual antes de que comience el segundo ciclo reproductivo. La producción de larvas es poco activa cuando la temperatura del agua está entre 12-15 °C y es mayor entre los 15 y los 17 °C [80].

Los movimientos de las larvas tienen lugar por la acción de la gravedad hacia zonas más profundas, o bien a favor de la corriente de agua pudiendo colonizar otras zonas. Este movimiento de las larvas, así como la activación del período reproductivo en función de la temperatura del agua, provoca una desincronización total en el ciclo biológico de la especie en el embalse, ya que la temperatura no es uniforme; es decir, la misma en cada punto del embalse: habitualmente suele ser más alta en superficie en verano y más baja en invierno. Todo lo anterior es habitual en ecosistemas fluviales y marinos. En definitiva, es importante tener en cuenta que, como consecuencia de todo lo anterior, existe la posibilidad de que en un determinado momento, los mejillones se pueden encontrar en diferentes etapas de desarrollo en un mismo punto del embalse.

Las larvas que caen en substratos arenosos o limosos no pueden sobrevivir, pero sí sobreviven si tienen la “suerte” de caer sobre substratos rocosos o más duros donde quedan fijadas. Sobre ellas se van apilando a su vez otros individuos adultos. La cantidad de movimientos de los mejillones jóvenes es despreciable en condiciones naturales [13], aunque puede haber un “desplazamiento artificial” por embarcaciones (al pegarse a la quilla) u otras intervenciones humanas.

El crecimiento de los mejillones guarda relación con la temperatura del agua registrada desde su fijación al sustrato. Es óptimo entre 18 y 20 °C y disminuye gradualmente conforme nos vamos alejando de este intervalo. El tiempo necesario para la madurez sexual depende, también, de la temperatura del agua.

El mejillón cebra puede formar densos agregados de individuos, formando varias capas, una encima de otra. Se han observado densidades máximas superiores a los 100.000 individuos por m^2 [98] y en el embalse de Ribarroja hasta de 250.000 [112]. Existe algún registro más antiguo en Norteamérica de 779.000 [115].

El embalse de Ribarroja se encuentra en la cuenca del Ebro (figura 6.2), situado al Nordeste de España. Este embalse tiene una longitud de 35km con una profundidad variable que llega a alcanzar los 28m en algunas zonas. El embalse recibe las aguas del río Ebro procedente del embalse Mequinenza y de los ríos Segre y Matarraña.

El embalse presenta una organización hidrodinámica [102] con una estratificación columnar del agua debido al contacto entre la más fría y mineralizada (procedente del embalse de Mequinenza), que se sitúa en la parte inferior, y aguas más cálidas con menos contenido de minerales pero con mayor concentración de nutrientes (procedentes del río Segre) y que fluye a través de la superficie. Debido a esta diferente temperatura de las aguas superiores e inferiores, así como al gran tamaño del embalse, es necesario tener en cuenta las diferentes condiciones térmicas y características de los sustratos, no pudiendo considerarse homogéneos [52].

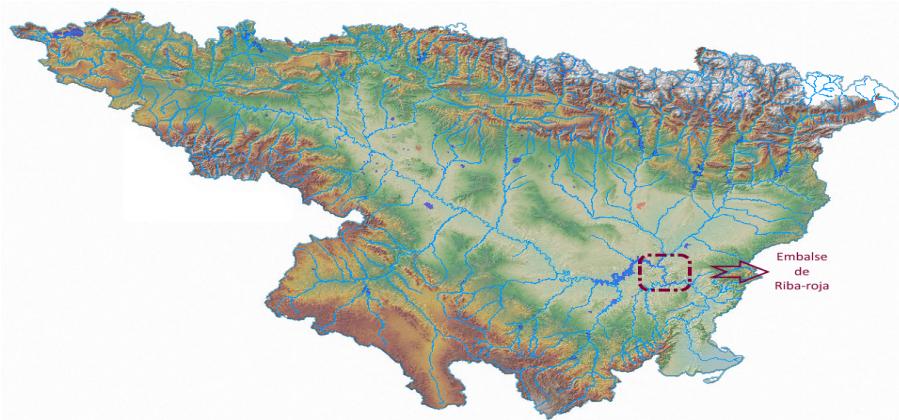


Figura 6.2: Cuenca del Ebro

En definitiva, vamos a considerar la división del embalse longitudinalmente y en profundidad, reconociendo dos capas bien diferenciadas en la columna de agua del embalse durante todo el año y, especialmente, durante el verano, cuando las aguas entrantes del Segre son más cálidas. Por tanto podemos estructurar el embalse en 9 zonas diferentes longitudinalmente, cada una dividida en 2 partes según la profundidad (U, upper y L, lower), totalizando 17 áreas como se muestra en la figura 6.3 (no se consideran 18 dada la escasa profundidad del río Segre al llegar al embalse). Esta división es muy relevante ya que el ciclo de vida del mejillón cebra se puede desarrollar a diferentes tasas y con diferente intensidad en cada zona.

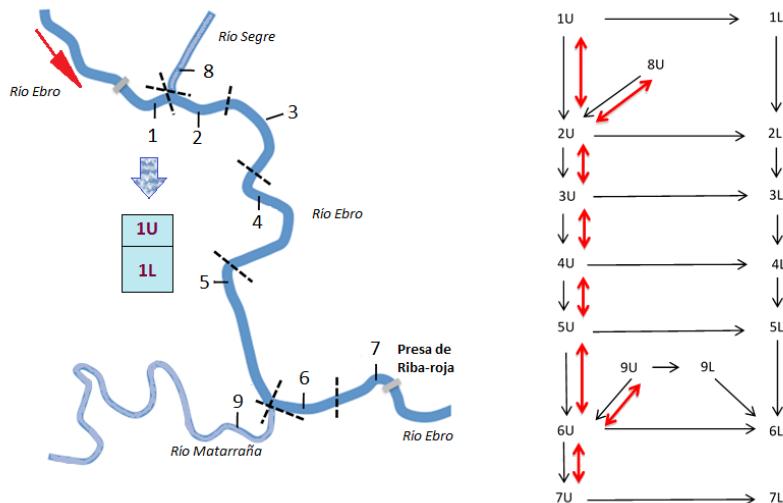


Figura 6.3: División en zonas del embalse

6.3. Planteamiento y marco de modelización

El ecosistema antes descrito presenta numerosas características (el ciclo biológico del mejillón cebra, el tamaño del embalse y la heterogeneidad de elementos que presenta, el flujo de agua o los factores derivados de la intervención humana, entre otras) cuya combinación hacen que su modelización sea muy compleja. En este contexto, la aplicación de técnicas de modelización convencionales, especialmente la aproximación continua y determinista basada en sistemas de ecuaciones diferenciales, resulta poco factible. Por ello, puede ser interesante el empleo de modelos formales alternativos, a ser posible de aproximación discreta, como es el caso de los sistemas P de dinámica de poblaciones, presentados en la sección 3.2.2.

Naturalmente, el hecho de optar por uno u otro paradigma de modelización computacional no exime de la dificultad para representar la esencia de los fenómenos que se tratan de estudiar. Un proceso de abstracción adecuado debe ser capaz de describir aquellos aspectos que se consideren más relevantes para el objetivo del estudio, desecharando otros que pudieran ser considerados menos relevantes de una realidad que nunca podrá ser capturada íntegramente, hasta el más mínimo detalle, sin evitar que el modelo sea intratable desde el punto de vista de la complejidad computacional.

En el caso que nos ocupa, para llevar a cabo el estudio de la dinámica de poblaciones del mejillón cebra en el embalse de Ribarroja se han considerado los siguientes factores:

1. Los procesos biológicos básicos de la especie, determinados por las condiciones térmicas y la idoneidad del sustrato en cada zona estudiada.
2. Las características especiales del hábitat objeto de estudio: un embalse artificial con corrientes de agua y cambios en la renovación de la misma dependiendo de la profundidad y la época del año, en función, además, de la gestión del embalse para la generación de energía hidroeléctrica, y teniendo en cuenta las características del agua entrante.
3. La posibilidad de entrada de larvas externas desde embalses y afluentes cercanos, y la transferencia de individuos desde el exterior al embalse a través de la presencia de embarcaciones.

Dentro de los factores del hábitat objeto de estudio cabe destacar la influencia de las condiciones térmicas del agua dentro de cada una de las 17 zonas, determinando completamente el comienzo y duración del ciclo biológico. Cuando se alcanza un cierto umbral de temperatura dentro de un área, los individuos comienzan a reproducirse progresivamente conforme alcanzan la madurez sexual. La influencia

de la temperatura, que va cambiando cada semana, se plasma directamente en la probabilidad de que un individuo se reproduzca, lo que vemos reflejado en la tabla 6.1. Aquellos individuos jóvenes que alcancen la madurez sexual más tarde, si las condiciones de temperatura siguen haciéndolo factible, también comienzan a liberar huevos. Como consecuencia de ello, cada área puede contener individuos en distintos estadios reproductivos y de crecimiento, evolucionando de manera simultánea.

Week	1	2	3	4	5	6	7	8	9	10	11
Cycle 1 (%)	4	4	7	15	20	20	15	10	3	1	1
Cycle 2 (%)	80	15	5								

Tabla 6.1: Porcentaje de mejillones que comienzan la reproducción por semana, dadas las condiciones de temperatura

Además, la naturaleza del sustrato y, fundamentalmente, su dureza y porosidad afectará drásticamente a la densidad de mejillones y mortalidad de larvas como muestra la tabla 6.2.

Área	Tipo I	Tipo II	Tipo III	Tipo IV	Capacidad
1U	19	24	50	7	2202
2U	10	9	5	76	777
3U	11	7	10	72	851
4U	24	11	10	55	1593
5U	20	5	15	60	1325
6U	30	15	10	45	1985
7U	25	11	18	46	1755
1L	9	55	26	10	2079
2L	8	15	11	66	899
3L	12	19	5	64	1107
4L	15	23	4	58	1335
5L	17	20	8	55	1422
6L	15	12	14	59	1222
7L	20	16	12	52	1536
8U	5	50	20	25	1680
9U	35	13	7	45	2147
8L	0	0	0	100	0
9L	22	28	14	36	1940

Tabla 6.2: % de terreno de cada tipo de suelo en cada zona: tipo I roca (hasta 5000 individuos/ m^2), tipo II guijarro (2300 ind/ m^2), tipo III grava (1400 ind/ m^2), tipo IV gravilla, arena, limos (0 ind/ m^2), y capacidad de albergar mejillones por zona (mussels/ m^2).

Junto con la desincronización en el ciclo de los diversos individuos y las características propias de la especie y del hábitat, existe otro tipo de factores que son determinantes en este estudio; por ejemplo, el efecto de las acciones humanas sobre la dinámica del flujo de agua, afectando al movimiento de larvas entre zonas o incluso hacia fuera del embalse, cuya estructura se muestra en la figura 6.3.

Para llevar a cabo un estudio sistemático de este tipo de problemas de modelización computacional de sistemas complejos en el ámbito de este tipo de herramientas bio-inspiradas, se presentó en [38] un protocolo estandarizado consistente en la consecución de una serie de fases que van desde el objetivo hasta el empleo de un simulador software para la realización de experimentos virtuales. Seguidamente, se describen de manera sucinta cada una de dichas fases.

Fase 1: Objetivo

El objetivo fundamental del estudio es la modelización de la dinámica de poblaciones del mejillón cebra en el embalse de Ribarroja y el desarrollo de herramientas que permitan realizar experimentos virtuales para medir la efectividad/fiabilidad de posibles acciones de control bajo distintos escenarios hipotéticos. La salida principal del modelo es la evolución de la distribución de larvas en cada zona del embalse, así como el número de individuos adultos, a lo largo del año.

Fase 2: Procesos a modelizar

Los procesos principales a modelizar comprenden aquellos que son propios de la biología de la especie, los relacionados con el entorno en que se realiza el estudio, así como aquellos derivados de la gestión realizada por los humanos, tratándose de procesos interrelacionados.

- *Biología de la especie*: liberación de huevos, estado larvario, fijación al sustrato, mortalidad asociada a sus distintos estadios y depredación debida al proceso de filtración de larvas y huevos llevado a cabo por los individuos adultos.
- *Procesos medioambientales*: generación de condiciones térmicas (introduciendo elementos de aleatoriedad para simular la fluctuación de la temperatura), características del sustrato en cada zona y control de la capacidad de carga en función de la idoneidad del mismo.
- *Intervención humana*: desplazamiento de larvas entre áreas causado por la gestión del agua para generar energía hidroeléctrica, desplazamiento de individuos adultos debido a movimiento de botes, e introducción externa de larvas.

La figura 6.4 muestra los distintos procesos e interacciones que involucran a dos áreas y reflejan la posibilidad de movimientos entre zonas colindantes en un cierto sentido (flechas unidireccionales) o en ambos (flechas bidireccionales).

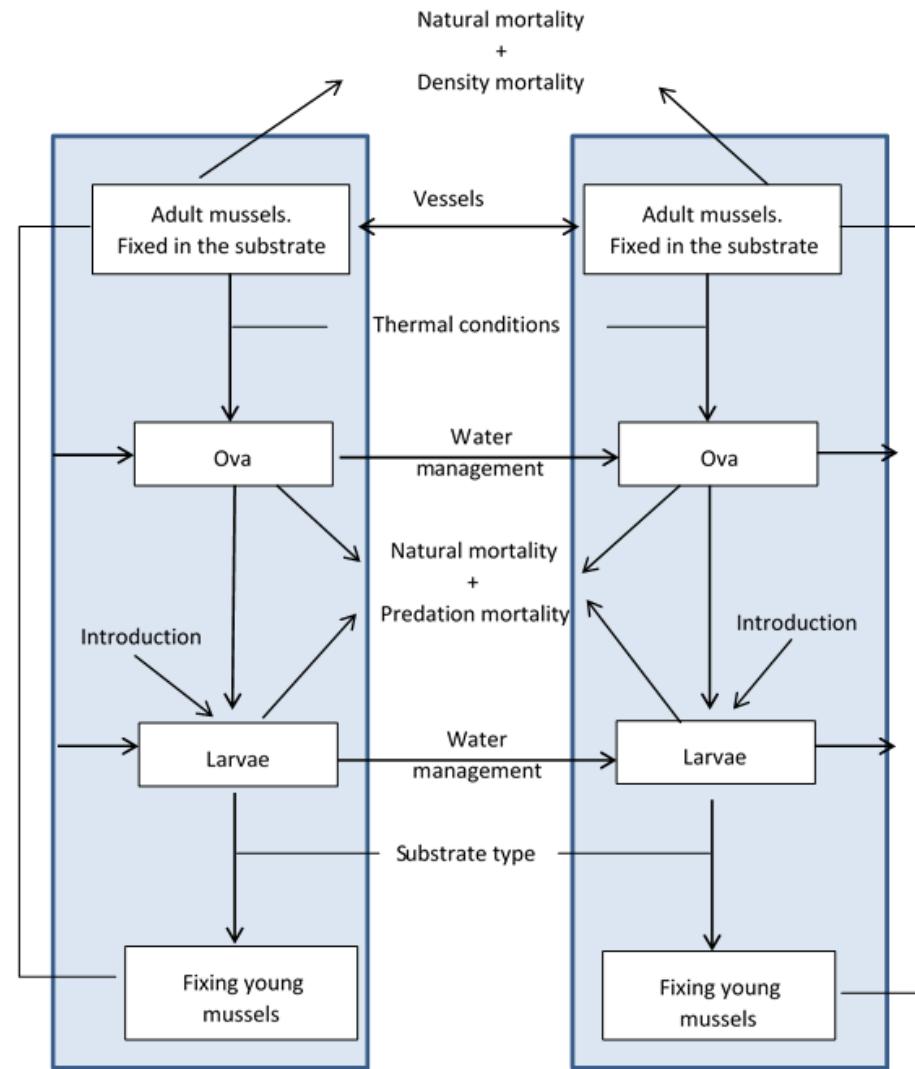


Figura 6.4: División en zonas del embalse

La complejidad del modelo completo es mayor, debido a la existencia de 17 zonas diferentes intercomunicadas y con distintas características y áreas próximas. Los procesos modelizados se producen simultáneamente en el interior y/o entre las distintas zonas.

Fase 3: Entrada del modelo y parámetros a considerar

Cuando se trata de analizar un determinado escenario haciendo uso del modelo, la entrada del mismo es el tamaño de la población inicial, junto con una serie de parámetros asociados a la biología de la especie, al entorno y a la actuación humana.

Conviene observar que parte de la información correspondiente a los parámetros de entrada para el modelo, no está disponible para el embalse de Ribarroja. Por ello, se han tomado algunos datos de las correspondientes variables, que proceden de estudios realizados en otras áreas. Sin embargo, se ha contrastado su adecuación al caso concreto que se estudia, mediante su validación por investigadores del área. Hay parámetros representando datos sobre la naturaleza del embalse y su entorno físico, como las dimensiones, el tipo de sustrato y las condiciones térmicas. Algunos otros dependen de la gestión humana del embalse, con interacciones humanas derivadas del movimiento de flujos de agua, de embarcaciones y la introducción externa de larvas.

Fase 4: Secuenciación y paralelización de los procesos

Para un mayor control del modelo a estudiar, habitualmente se secuencian los procesos que tratan de ser representados. En el caso que nos ocupa sería muy complejo secuenciar el problema completamente, por ello se propone la secuenciación parcial que aparece en la figura 6.5 y que corresponde a un determinado ciclo de ejecución t .

Como se ha comentado en apartados anteriores y se ilustra en la figura citada, los procesos son muy sensibles a cambios de temperatura. Cuando se alcanza un determinado umbral, se produce el inicio de la fase de reproducción. La duración de ese proceso dependerá del ciclo. Además, una vez comenzada la liberación de los huevos, se lanzan simultáneamente varios procesos, como son la liberación de huevos propiamente, el movimiento de larvas de diferentes edades, así como la fijación al sustrato de individuos en una fase más avanzada de desarrollo, dependiendo de la población existente y del tipo de sustrato. Se contempla en el modelo que algunos individuos adultos podrían, además, ser objeto de movimientos debido al tráfico de embarcaciones.

La presencia de individuos jóvenes que, obviamente, alcanzarán la madurez sexual más tarde que los adultos, incrementa la dificultad de una secuenciación de los procesos. En este sentido, se dice que los procesos se encuentran desincronizados. No obstante, tras un proceso de mortalidad dependiente de la densidad de individuos, el proceso quedará finalmente sincronizado, terminando el ciclo en el último bloque, el módulo que permite la restauración de la configuración inicial para comenzar el nuevo ciclo, $t + 1$. En este caso, un año natural conlleva la ejecución de dos iteraciones, ciclos de ejecución del modelo, que corresponden a las dos etapas que tienen lugar en el ciclo reproductivo del mejillón cebra a lo largo del año.

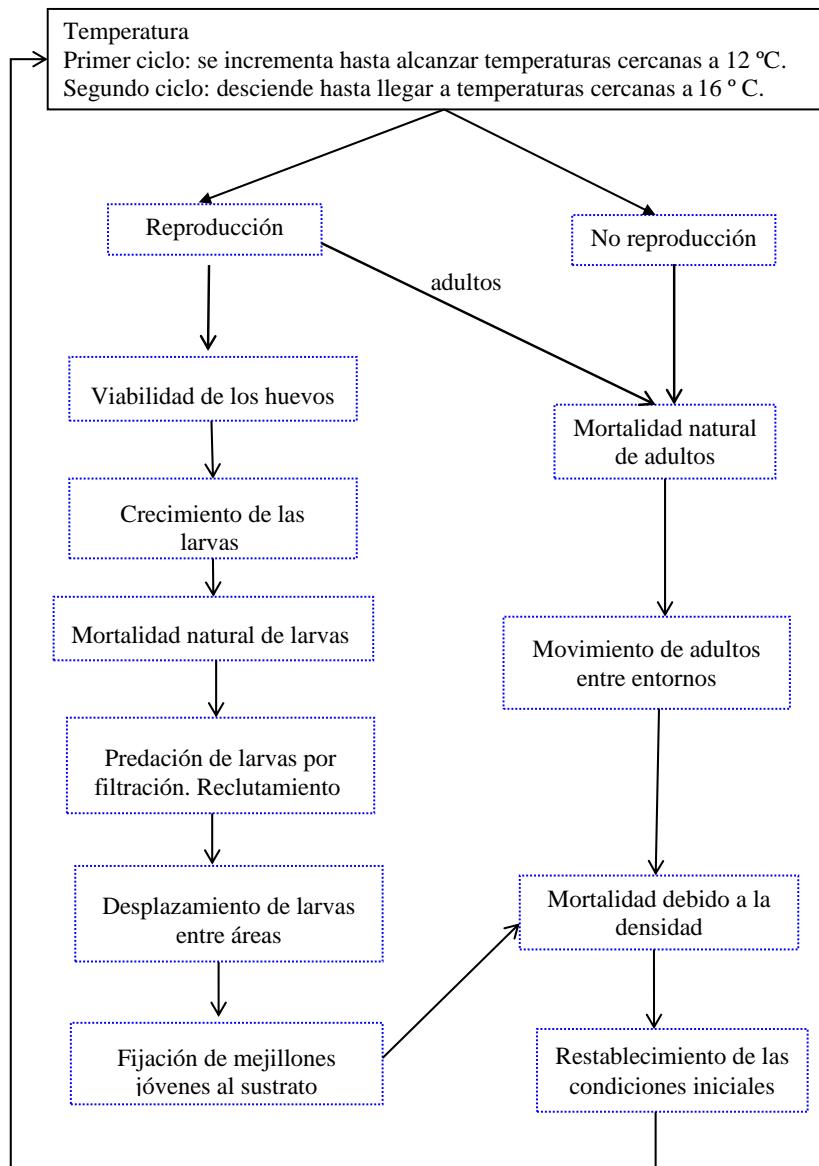


Figura 6.5: Bloques del modelo

Para simplificar el movimiento del flujo hidráulico en el embalse, se considera un flujo turbulento, con número de Reynolds $36 * 10^6$, y se han considerado las velocidades aproximadas que aparecen en [52]. La sección central del embalse se divide en 7 partes, a través de las cuales se distribuye el flujo de agua, adoptando las diferentes velocidades del agua dependiendo del flujo entrante. Como se detallará en

la fase 7 del protocolo que se está aplicando aquí, se ha diseñado e implementado un nuevo algoritmo para estimar la distribución final del agua dentro de cada área con carácter semanal, influyendo de forma decisiva sobre la probabilidad del movimiento de las larvas entre las distintas áreas.

Fase 5: el modelo

La fase 5 establece el diseño del modelo, reflejando todos los procesos y elementos de entrada que se han comentado en los apartados anteriores, haciendo uso de un sistema P de dinámica de poblaciones. El diseño del modelo se detalla en la sección 6.4.

Fase 6: Análisis gráfico de un ciclo de ejecución del modelo

Para complementar la información detallada sobre el modelo presentada en la fase 5, se aporta en la sección 6.4.1 la traza de ejecución de un ciclo del modelo, ilustrando la evolución de los objetos del sistema a lo largo de ese ciclo. Esta fase nos permite analizar en profundidad la corrección del modelo y su adecuación al objetivo perseguido, facilitando la detección de errores antes de proceder a la adaptación del modelo a una plataforma software donde poder simularlo.

Fase 7: diseño del simulador

Una vez diseñado el modelo y realizado el análisis descrito en la fase 6, el siguiente paso consiste en traducir el modelo a un lenguaje entendible por una máquina capaz de simularlo. Afortunadamente, disponemos del marco de **P-Lingua**, que presenta un lenguaje de especificación de sistemas P incluyendo los PDP, así como las herramientas de parsing y simulación correspondientes para este tipo de sistemas. De este modo, sólo se necesita traducir directamente de nuestra especificación del modelo al muy cercano lenguaje **P-Lingua**.

Por otra parte, el modelo especificado no contiene toda la información necesaria para simular un escenario de la vida real. Además del modelo en sí, el problema contiene, propiamente, una gran cantidad de información específica de cada escenario concreto que se quiera analizar, lo que conllevará la necesidad de desarrollar herramientas para poder introducir los datos de los escenarios, generar la codificación $cod(s)$ de esa entrada en forma de parámetros a emplear dentro del modelo traducido a **P-Lingua**, establecer el número de ciclos a simular y un número de simulaciones a realizar y, finalmente, devolver las salidas del modelo con un determinado significado para el usuario final. Estos y algunos otros aspectos serán cubiertos por el software **MeCoSim**, como se explicará en la sección 6.4.2.

6.4. Diseño del modelo

Para detallar el modelo que se propone, comenzamos describiendo las notaciones utilizadas en el mismo.

(a) Índices comunes

- s (*semana*): representa las semanas que son de interés para el estudio (generalmente el rango es $\{1, \dots, 36\}$).
- j (*compartimento*): representa las distintas zonas en las que estructuramos el embalse y su rango es $\{1, \dots, 18\}$, según el orden en que aparecen en las columnas de la tabla 6.3).
- c (*ciclo reproductivo*): representa los ciclos de reproducción que, como se ha dicho, en principio son dos (notados por 1 y 2, respectivamente); el primero consta de 20 semanas y el segundo consta de 16.
- sem (*semestre*): representa los semestres y se emplean para distinguir valores de parámetros afectando de forma distinta a los individuos según su edad medida en semestres. En principio de 1 a 6, puesto que no se han encontrado en Ribarroja individuos de edad superior a 3 años.
- i (*tipo de suelo*): representa el tipo de suelo y se distinguen cuatro tipos que enumeramos de 1 a 4 (que denominamos *bueno*, *medio-alto*, *medio-bajo* y *malo*, respectivamente, entendiendo que un suelo es tanto mejor cuanta más capacidad tenga para la fijación y preservación de los mejillones).

(b) Parámetros

Condiciones medioambientales y su influencia sobre la especie:

- $T_{s,j}$: temperatura en la semana s para el compartimento j .
- IT_c : temperatura con la que comienza el ciclo reproductivo c .
- $PI_{s,c}$: proporción de larvas liberadas en la semana s del ciclo reproductivo c .
- mo_c : mortalidad de los óvulos en el ciclo reproductivo c (probabilidad de que no lleguen a ser fecundados).
- N_j : número de individuos por metro cuadrado inicialmente situados en el compartimento j .

Fecha	Sem	1U	2U	3U	4U	5U	6U	7U	1L	2L	3L	4L	5L	6L	7L	8U	9U	8L	9L	
8-14 Mar	1	9.4	11.2	11.8	10.3	9.5	8.5	9.0	9.0	9.1	8.2	8.5	8.5	8.5	11.8	9.7	0.0	8.9		
15-21 Mar	2	10.1	11.7	11.7	12.0	11.5	10.3	9.1	10.0	9.6	8.5	9.7	9.6	12.9	10.7	0.0	9.9			
22-28 Mar	3	10.8	12.2	12.2	12.1	12.6	11.0	11.1	9.6	10.9	10.0	8.7	10.9	10.7	13.9	11.7	0.0	10.9		
29 Mar - 4 Abr	4	11.0	12.7	12.7	12.3	12.8	12.0	10.3	11.0	11.0	10.6	9.4	11.4	11.2	14.4	12.4	0.0	11.4		
5-11 Abr	5	11.2	13.1	13.1	12.5	13.0	12.9	10.9	11.1	11.1	10.1	11.9	11.7	14.9	13.1	0.0	11.8			
12-18 Abr	6	11.9	13.5	13.5	13.0	13.2	13.3	13.1	11.7	11.6	11.8	10.7	12.1	12.1	15.5	13.6	0.0	12.2		
19-25 Abr	7	12.5	13.8	13.8	13.5	13.3	13.5	13.2	12.5	12.0	12.0	11.3	12.2	12.5	16.1	14.0	0.0	12.6		
26 Abr-2 May	8	13.1	14.3	14.3	14.3	13.9	13.8	13.7	13.1	12.8	12.8	13.2	11.9	12.7	13.1	16.7	14.6	0.0	13.3	
3-9 May	9	13.6	14.8	14.8	15.0	14.5	14.0	14.1	13.6	13.5	13.5	13.8	12.5	13.2	13.6	17.2	15.1	0.0	14.0	
9-16 May	10	14.2	15.4	15.4	15.4	15.2	15.0	15.2	14.2	14.2	14.2	14.4	13.2	14.1	14.4	17.5	14.4	0.0	14.6	
17-23 May	11	14.8	16.0	16.0	15.8	15.8	15.8	15.9	16.2	14.8	14.8	14.8	15.0	13.9	15.0	15.2	17.8	16.9	0.0	15.1
24-30 May	12	15.9	16.7	16.7	16.4	16.5	16.8	16.8	15.9	15.9	15.9	15.8	14.3	16.0	16.1	17.8	17.6	0.0	16.0	
31 May-6 Jun	13	16.9	17.3	17.3	16.9	17.3	17.6	17.4	16.9	16.9	16.9	16.7	17.0	17.0	17.1	17.8	18.2	0.0	16.8	
7 - 13 Jun	14	18.0	18.0	18.0	17.5	18.0	18.5	18.0	18.0	18.0	18.0	17.5	15.1	18.0	18.0	17.8	18.9	0.0	17.7	
14-20 Jun	15	18.9	19.4	19.4	19.5	19.5	19.5	19.3	18.3	18.4	18.4	18.3	16.5	18.3	18.4	19.4	20.0	0.0	18.0	
21-27 Jun	16	19.8	20.7	20.7	21.5	21.0	20.5	20.5	18.5	18.7	18.7	19.0	17.9	18.5	18.8	21.0	21.0	0.0	18.2	
28 Jun-4 Jul	17	20.4	21.9	21.9	23.2	22.3	21.4	21.5	18.3	18.5	18.5	19.2	18.3	18.8	18.9	22.4	22.0	0.0	18.6	
5-11 Jul	18	21.0	23.0	23.0	24.8	23.5	22.2	22.4	18.1	18.2	18.2	18.2	19.3	18.6	19.0	19.0	23.8	0.0	18.9	
12-18 Jul	19	22.2	24.0	24.0	25.3	24.2	23.1	23.1	18.5	18.4	18.4	18.7	18.7	20.4	20.0	24.0	23.6	0.0	20.5	
19-25 Jul	20	23.3	24.9	24.9	25.7	24.9	23.9	23.8	18.9	18.5	18.5	20.2	18.7	21.8	20.9	24.1	24.1	0.0	22.0	
26 Jul-1 Ago	21	23.5	25.0	25.0	25.7	24.0	23.8	19.1	18.8	18.8	18.8	20.6	19.4	21.9	21.0	24.1	24.4	0.0	22.2	
2-8 Ago	22	23.8	25.1	25.1	25.7	25.1	24.1	23.9	19.2	19.2	19.2	19.2	20.9	20.1	21.9	21.1	24.1	24.7	0.0	22.4
9-15 Ago	23	24.0	25.2	25.2	25.7	25.2	24.2	23.9	19.4	19.5	19.5	19.5	21.3	20.8	22.0	21.2	24.1	25.0	0.0	22.6
16-22 Ago	24	23.6	25.1	25.1	25.5	25.2	24.2	23.6	19.7	20.3	20.3	21.9	21.3	21.5	21.2	24.1	25.1	0.0	22.0	
23-29 Ago	25	23.2	24.9	24.9	25.2	25.1	24.1	23.2	20.0	21.0	21.0	22.5	21.7	21.0	21.2	24.0	25.2	0.0	21.3	
30 Ago - 5 Sep	26	22.7	24.5	24.5	25.0	25.0	24.1	23.6	20.7	21.3	21.3	22.5	21.9	21.4	21.7	24.0	25.1	0.0	21.8	
6-12 Sep	27	22.2	24.0	24.0	24.8	24.9	24.1	23.9	21.3	21.5	21.5	22.5	22.1	21.8	22.2	24.0	24.9	0.0	22.2	
13-19 Sep	28	21.6	23.2	23.2	24.4	24.4	23.8	23.7	21.2	20.7	20.7	21.9	22.2	21.8	22.1	23.4	24.3	0.0	22.2	
20-26 Sep	29	21.0	22.3	22.3	24.0	23.8	23.4	23.5	21.0	19.9	19.9	21.3	22.2	21.8	22.0	22.7	23.6	0.0	22.1	
27 Sep - 3 Oct	30	20.7	21.7	21.7	23.3	23.0	22.7	22.8	20.6	19.4	19.4	20.7	22.0	21.2	21.5	21.8	22.8	0.0	21.5	
4-10 Oct	31	20.5	21.1	21.1	22.7	22.2	22.0	20.3	19.0	19.0	20.1	21.8	20.6	21.0	20.8	22.0	0.0	20.9		
11-17 Oct	32	20.2	20.5	20.5	22.0	21.4	21.3	21.5	19.9	18.5	18.5	19.5	21.6	20.0	20.5	19.9	21.2	0.0	20.3	
18-24 Oct	33	19.9	19.7	19.7	21.0	20.3	20.4	19.1	17.4	17.4	18.7	20.7	19.2	19.7	18.4	20.1	0.0	19.3		
25- 31 Oct	34	19.5	18.9	18.9	20.0	19.1	19.3	18.3	16.2	16.2	17.9	19.7	18.3	18.9	16.8	19.0	0.0	18.3		
1 - 7 Nov	35	19.1	17.9	17.9	19.0	17.9	18.2	18.2	17.6	15.1	17.0	18.7	17.4	18.0	15.3	18.0	0.0	17.2		
8 - 14 Nov	36	18.6	16.8	16.8	18.0	16.7	17.0	17.0	16.9	14.0	16.0	17.6	16.5	17.0	13.8	16.9	0.0	16.0		

Tabla 6.3: Temperatura media del agua cada semana, por compartimento.

- $q_j: N_j/7$, número de individuos de cada franja de edad; a partir del número N_j se considera una distribución uniforme en cuanto a las edades de los individuos contenidos en el compartimento, incluyendo siete grupos de edad, cinco correspondientes a los X_s con $2 \leq s \leq 6$, más otros dos grupos de individuos jóvenes de 13 y 39 semanas.
- $p_{s,j}$: probabilidad de que se active la reproducción de los individuos del compartimento j durante la semana s . Su valor viene dado por la probabilidad acumulada para IT_1 en el caso de las semanas 1 a 20 e IT_2 en el de las semanas 21 a 36, a partir de la distribución de probabilidad normal $N(T_{s,j}, 0.5)$.

Parámetros biológicos:

- g_1 : porcentaje de óvulos emitidos en el primer ciclo.
- g_2 : reducción de óvulos.
- g_3 : mortalidad de larvas.
- m_{sem} : mortalidad de los individuos de edad sem (en semestres).

Propiedades del compartimento:

- $SURF_j$: anchura (en metros) del compartimento j .
- $A2_j$: semi-anchura (en metros) del compartimento j .
- L_j : longitud (en metros) del compartimento j .
- DS_j : profundidad (en metros) del compartimento j .
- C_j : capacidad (en metros cúbicos) del compartimento j .
- $S_{i,j}$: porcentaje del suelo del compartimento j que es del tipo i . Al haber cuatro tipos de suelo, tendremos desde $S_{1,j}$ hasta $S_{4,j}$.
- AS_i : capacidad de albergar mejillones por parte del tipo de suelo i .
- φ_j : capacidad del compartimento j , calculado a partir de la proporción de cada tipo de suelo presente en el compartimento y la capacidad de los mismos. A partir de ese número, se estaría excediendo la capacidad máxima del compartimento según su composición de suelos y se empezaría a producir la muerte de los mejillones de la parte inferior debido al exceso de densidad (no confundir con la máxima capacidad de carga).

Intervención humana:

- $LE_{j,c}$: inoculación externa de larvas en el compartimento j durante el ciclo reproductivo c .
- $PA_{j,j'}$: probabilidad de que un mejillón adulto sea transportado del compartimento j al compartimento j' por el movimiento de embarcaciones.
- $PR_{s,j,j'}$: probabilidad de que las larvas se vean desplazadas en la semana s , del compartimento j al compartimento j' a causa del régimen de renovación del agua.
- CV_s : factor multiplicador de la probabilidad de que una larva situada en un área superior pase a su correspondiente área inferior en la semana s . La probabilidad correspondiente es $0.2 * CV_s$.

(c) Símbolos correspondientes a los objetos del modelo

- X_{sem} : mejillones adultos de edad sem (en semestres) al inicio del año; es decir, al inicio del primero de los dos ciclos reproductivos en los que dividimos el año.
- Q_s : mejillones jóvenes de edad s semanas al inicio del año.
- α, α_i (con $1 \leq i \leq 7$), β_i (con $0 \leq i \leq 9$) : objetos auxiliares empleados para habilitar y sincronizar procesos de regulación por máxima capacidad de carga y por exceso de densidad en función del tipo de suelo, que tienen lugar en las membranas 37 a 39 al final del ciclo.
- T_s : objeto auxiliar para disparar mediante T_0 el proceso de generación pseudoaleatoria de temperaturas al inicio del ciclo reproductivo y, a partir de ahí, controla el comienzo o no del ciclo reproductivo cada semana s , en función de las condiciones de temperatura.
- I_c : objeto auxiliar empleado para permitir la inoculación externa de larvas al comienzo del ciclo reproductivo c .
- γ_s : objeto auxiliar que representa la existencia de condiciones favorables de temperatura para comenzar el ciclo reproductivo en la semana s .
- γ : objeto auxiliar que activa el ciclo reproductivo en la membrana en la que se encuentre, correspondiente a la semana s , donde habrá sido generado a partir de un objeto γ_s .
- I' : objetos provenientes de la inoculación externa de larvas.

- VR : objetos que representan mejillones jóvenes o adultos, con el fin de controlar la capacidad de carga al final del ciclo.
- O_s : huevos producidos en la semana s del ciclo reproductivo.
- $Q'_{s,s2}$: mejillones jóvenes de edad s semanas en la semana $s2$.
- Y_{sem} : mejillones adultos de edad sem tras la fase de reproducción.
- V_{sem} : mejillones adultos de edad sem tras la fase de mortalidad.
- V'_{sem} : mejillones adultos de edad sem durante la fase de posible desplazamiento por el movimiento de embarcaciones. Posteriormente vuelven a ser V_{sem} .
- $L1_s$: larvas generadas a partir de los huevos liberados en la semana s , una vez pasada la fase de viabilidad de los mismos.
- $L2_s$: larvas de la semana s que han superado la fase de mortalidad.
- $L3_s$: larvas liberadas en la semana s . A partir de los objetos $L2_{s1}$ procedentes de los huevos producidos en la semana $s1$ que sobrevivieron a la fase de mortalidad, se van liberando objetos $L3_{s1+s2-1}$ para $s2$ de 1 a 11 en el primer ciclo y de 1 a 3 en el segundo (la liberación final de las larvas no se produce en un único momento sino progresivamente, cada semana se van liberando algunos individuos).
- $L_{s,i,j,j'}$: larvas liberadas en la semana s y que lleva $i+1$ semanas, con $0 \leq i \leq 3$, en la columna de agua, habiéndose originado en el compartimento j y que deberán desplazarse al compartimento j' .
- L'_s : larvas liberadas en la semana s que ya han pasado su periodo en la columna de agua.
- $\delta, \delta_{20}, \delta_{35}, \delta_{36}$: objetos auxiliares encargados de la sincronización de los objetos que representan individuos, cuando no se han alcanzado las condiciones térmicas para disparar el proceso de reproducción en el primer y segundo ciclo reproductivo.
- $\rho, \rho_0, \rho_1, \rho_2, \omega_i$ (con $0 \leq i \leq 15$): objetos auxiliares involucrados en la sincronización de los procesos de control de la capacidad de carga y del reclutamiento de individuos jóvenes.
- X'_{sem} : mejillones adultos de edad sem cuando no se produce reproducción, pasando posteriormente a Y_{sem} .

- Q''_{s+20} : mejillones jóvenes de edad $s + 20$ generados a partir de los objetos que entraron con Q_s en el principio del primer ciclo reproductivo; una vez que no se ha dado la reproducción, aumenta su edad en 20 semanas. Darán lugar a objetos $Q'_{s+20,20}$ para indicar que, finalmente, los mejillones jóvenes han alcanzado la edad $s + 20$ en la semana 20.
- Q''_{s+16} : análogo al anterior, para mejillones jóvenes de edad $s + 16$ generados a partir de los objetos que entraron con Q_s en el segundo ciclo reproductivo; una vez que no se ha dado la reproducción, aumenta su edad en 16 semanas. Darán lugar a objetos $Q'_{s+16,36}$ para indicar que, finalmente, los mejillones jóvenes han alcanzado la edad $s + 16$ en la semana 36.
- A, A_i (con $1 \leq i \leq 2$): objetos auxiliares para la regulación del exceso de densidad en cada compartimento.
- AR : objetos encargados de la regulación de la capacidad de carga máxima, generados en número correspondiente a la máxima carga a nivel de cada compartimento y que se neutralizarán con los objetos VR surgidos de los correspondientes individuos jóvenes y adultos, de modo que si se excede el nivel se active la eliminación de individuos por exceso de carga.
- DP_i (con $0 \leq i \leq 39$): objetos auxiliares representando la disponibilidad en tanto por ciento de capacidad sobrante tras comprobar el posible exceso sobre la carga máxima; estos objetos se generan a partir de los objetos AR que no se neutralizaron con sus correspondientes objetos VR , siempre que sobre capacidad.
- $QQ_{s,k}$ (con $1 \leq s \leq 52, 37 \leq k \leq 38$): objetos auxiliares que terminan dando lugar a objetos Q_s y X_1 , en caso de que lo permita la capacidad de carga.
- $XX_{2,k}$ (con $37 \leq k \leq 38$): objetos auxiliares que terminan dando lugar a objetos X_2 , en caso de que lo permita la capacidad de carga.
- T'_{20} : objeto auxiliar que facilita la sincronización en la fase de actualización para establecer las condiciones iniciales, concretamente permitiendo la generación de I_2 , que permitirá al inicio del próximo ciclo la inoculación externa de larvas.

(d) Formalización del sistema PDP

En este apartado se presenta el modelo basado en sistemas PDP para tratar el problema de la dinámica de poblaciones del mejillón cebra en el embalse de Ribarroja. El modelo presenta los módulos mostrados en la Figura 6.5 y, por ello, los bloques de reglas serán descritos siguiendo la misma estructura modular, teniendo

en cuenta la naturaleza secuencial inherente al diagrama anterior. La simulación de un año natural conlleva la realización de dos ciclos del modelo.

El modelo consiste en un sistema PDP de grado $(18, 40)$ tomando $T \geq 1$ unidades de tiempo (pasos),

$$\Pi = (G, \Gamma, \Sigma, \mu, T, \Pi_1, \dots, \Pi_{18}, \mathcal{R}, E_1, \dots, E_{40}, \mathcal{R}_E)$$

en donde:

- $G = (V, S)$ es el grafo dirigido completo compuesto por los nodos $V = \{e_1, \dots, e_{18}\}$. Todos los movimientos entre entornos son formalmente posibles, pero muchos de los movimientos se darán muy poco, en función del régimen de renovación del agua. Los movimientos más habituales se muestran en la Figura 6.6.

- El alfabeto de trabajo, Γ , es el conjunto:

$$\begin{aligned} \Sigma \cup & \{O_m, L_{1m}, L_{2m} \mid 1 \leq m \leq 36\} \cup \{\gamma, VR, \delta_{20}, \delta, \delta_{36}, \rho\} \cup \\ & \{\rho_i \mid 3 \leq i \leq 11\} \cup \{X_s, X'_s, Y_s, V_s, V'_s \mid 1 \leq s \leq 6\} \cup \{Q_d \mid 0 \leq d \leq 87\} \cup \\ & \{Q'_{d,m} \mid -19 \leq d \leq 72, 1 \leq m \leq 40\} \cup \{\omega_i \mid 1 \leq i \leq 15\} \cup \\ & \{Q''_d \mid 16 \leq d \leq 52\} \cup \{A, \alpha, \eta\} \cup \{AR_i \mid 1 \leq i \leq 5\} \cup \{A_s \mid 3 \leq s \leq 5\} \cup \\ & \{\alpha_i \mid 1 \leq i \leq 7\} \cup \{DP_i \mid 0 \leq i \leq 39\} \cup \{\beta_i \mid 0 \leq i \leq 9\} \cup \\ & \{QQ_{d,k} \mid 1 \leq d \leq 52, 37 \leq k \leq 38\} \cup \{XX_{2,k}, \mid 37 \leq k \leq 38\} \cup \{T'_{20}, D\} \end{aligned}$$

siendo Σ (alfabeto de los entornos) el siguiente conjunto:

$$\begin{aligned} \Sigma = & \{I, I', I_1, I_2\} \cup \{T_d \mid 0 \leq d \leq 35\} \cup \{\gamma_m \mid 1 \leq m \leq 36\} \cup \\ & \{L_{3m} \mid 1 \leq m \leq 39\} \cup \{L'_m \mid 5 \leq m \leq 43\} \cup \\ & \{L_{m,i,j,j'} \mid 1 \leq m \leq 43, 0 \leq i \leq 4, 1 \leq j \leq 18, 1 \leq j' \leq 18\} \cup \\ & \{\rho_0, \rho_1, \rho_2, \omega_0, \delta_{20}, \delta_{35}, \delta_{36}, A_1, A_2, AR\} \end{aligned}$$

- $\mu = [\]_1 \dots [\]_{39}]_0$ es la estructura de membranas (un árbol enraizado cuya raíz se etiqueta por 0 y los restantes nodos son hijos de la raíz), y los correspondientes multiconjuntos iniciales para el entorno j (con $1 \leq j \leq 18$) son:

- $\mathcal{M}_{0,j} = \{X_i^{q_{j,i}}, Q_d^{q_{j,d}} \mid 1 \leq j \leq 18, 2 \leq i \leq 6, 1 \leq d \leq 52\}$.

Los objetos X_i representan mejillones adultos de edad i en semestres y $q_{j,i}$ es el número de tales individuos que existen en el entorno j . Los objetos Q_d representan mejillones más jóvenes de un año de edad, con d la edad en semanas y $q_{j,d}$ es el número de tales individuos que existen en el entorno j .

- $\mathcal{M}_{1,j} = \dots = \mathcal{M}_{36,j} = \emptyset$.
- $\mathcal{M}_{37,j} = \mathcal{M}_{38,j} = \mathcal{M}_{39,j} = \{\alpha\}$.

- $T = 51 \cdot 2 \cdot Anyos$, donde $Anyos$ es el número de años a simular. Por tanto, cada año del escenario real se simula mediante 2 ciclos de 51 pasos.

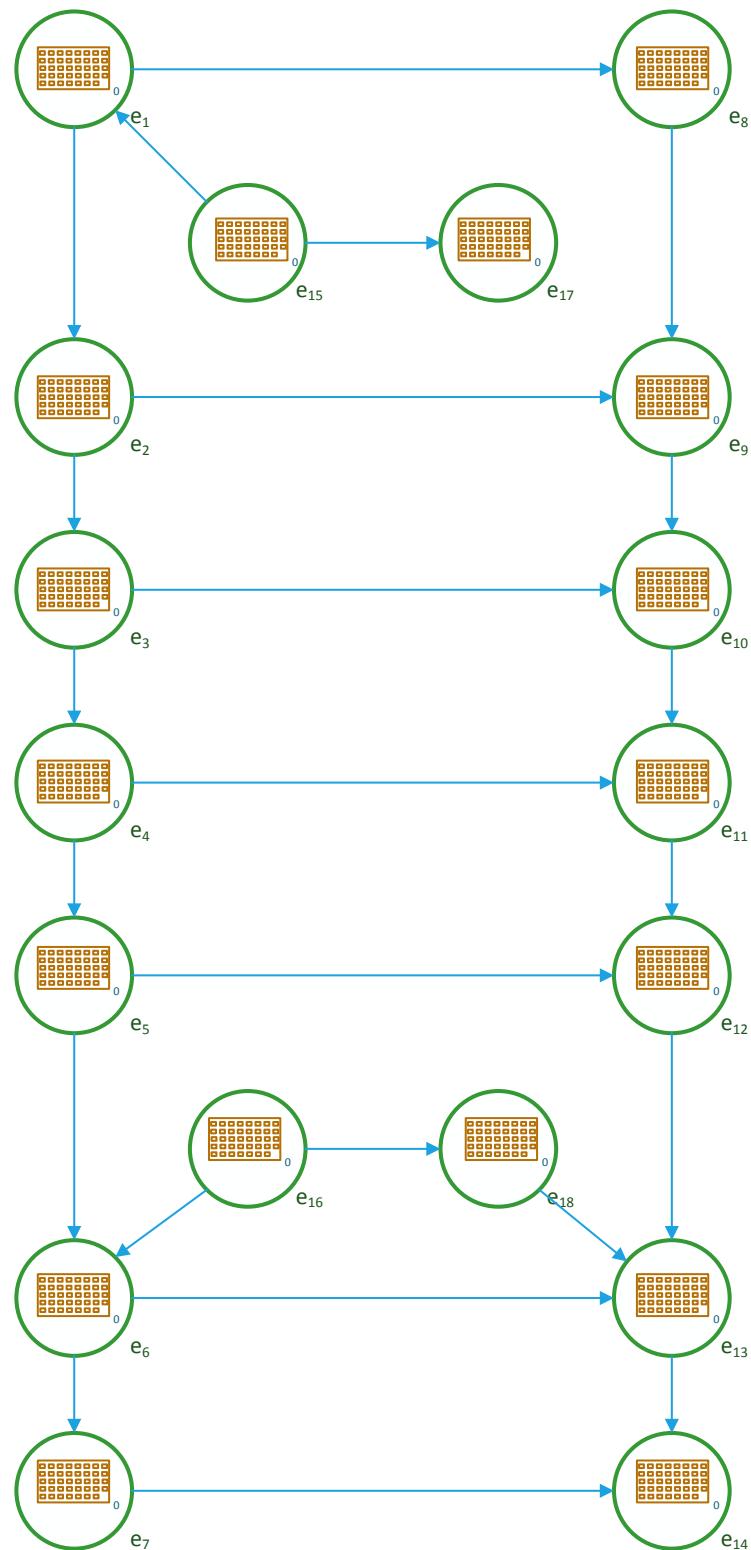


Figura 6.6: Aristas principales del grafo de entornos del sistema PDP

- $E_1 = \dots = E_{18} = \{T_0, I_1\}$
- Los conjuntos de reglas \mathcal{R} de los sistemas P y \mathcal{R}_E de comunicación entre entornos, son las que se relacionan a continuación.

Inoculación externa

$$r_{e_{1,j}} \equiv (I_1 \rightarrow I^{LE(j,1)} I')_{e_j}, \quad 1 \leq j \leq 18$$

$$r_{e_{2,j}} \equiv (I_2 \rightarrow I^{LE(j,2)} I')_{e_j}, \quad 1 \leq j \leq 18$$

$$r_1 \equiv I []_0^0 \rightarrow [I]_0^0$$

$$r_2 \equiv I' []_0^0 \rightarrow [I']_0^0$$

La multiplicidad $LE(j,i)$ para los objetos I representa el número de larvas introducidas externamente en el ciclo i por los humanos.

Nota: en esta sección, las reglas de comunicación entre entornos, del tipo $(x)_{e_j} \xrightarrow{pr} (y_1)_{e_j} \dots (y_h)_{e_j}$, las escribiremos de forma abreviada como: $(x \xrightarrow{pr} y_1 \dots y_h)_{e_j}$.

Temperaturas

$$r_{e_{3,j,d}} \equiv (T_d \xrightarrow{p(d+1,j)} \gamma_{d+1} T_{d+1})_{e_j} \quad \begin{cases} 1 \leq j \leq 18 \\ 0 \leq d \leq 34, d \neq 19 \end{cases}$$

$$r_{e_{4,j,d}} \equiv (T_d \xrightarrow{1-p(d+1,j)} T_{d+1})_{e_j} \quad \begin{cases} 1 \leq j \leq 18 \\ 0 \leq d \leq 34, d \neq 19 \end{cases}$$

$$r_{3,m} \equiv \gamma_m []_m^0 \rightarrow [\gamma_m]_m^0, \quad 1 \leq m \leq 36$$

$$r_{4,m} \equiv \gamma_m []_m^0 \rightarrow [\gamma]_m^-, \quad 1 \leq m \leq 36$$

$$r_{5,m} \equiv [\gamma]_m^- \rightarrow []_m^0, \quad 1 \leq m \leq 36$$

Ciclo reproductivo y liberación de huevos

$$r_{6,m} \equiv I []_m^- \rightarrow [I]_m^0, \quad 1 \leq m \leq 36$$

$$r_{7,m} \equiv X_1 []_m^- \rightarrow []_m^0, \quad 1 \leq m \leq 36$$

$$r_{8,m,s} \equiv X_s []_m^- \rightarrow VR [X_s]_m^0 \quad \begin{cases} 1 \leq m \leq 36 \\ 2 \leq s \leq 6 \end{cases}$$

$$r_{9,m,d} \equiv Q_d []_m^- \rightarrow VR [Q_{d+m}]_m^0 \quad \begin{cases} 1 \leq m \leq 20 \\ 20 - m \leq d \leq 52 - m \end{cases}$$

$$r_{10,m,d} \equiv Q_d []_m^- \rightarrow VR [Q_{d+m-20}]_m^0 \quad \begin{cases} 21 \leq m \leq 36 \\ 40 - m \leq d \leq 72 - m \end{cases}$$

$$r_{11,m,d} \equiv Q_d []_m^- \rightarrow VR [X_1]_m^0 \quad \begin{cases} 1 \leq m \leq 20 \\ 53 - m \leq d \leq 72 - m \end{cases}$$

$$r_{12,m,d} \equiv Q_d []_m^- \rightarrow VR [X_1]_m^0 \left\{ \begin{array}{l} 21 \leq m \leq 36 \\ 53 - m + 20 \leq d \leq 72 - m + 20 \end{array} \right.$$

Las siguientes reglas formalizan la generación de objetos representando huevos que se mueven a la membrana piel. Como se ha explicado en el apartado 6.2.1, su número depende del tamaño y, por tanto, de la edad del mejillón emisor. El 50 % de los individuos son hembras que se pueden reproducir. Las fórmulas que regulan la producción de huevos son las siguientes:

$$\Theta(s) = 1920.5 \cdot \left(\frac{52}{4} + \frac{52}{2} \cdot s \right) - 79065, \text{ con } 2 \leq s \leq 6 \text{ la edad en semestres.}$$

$$\Psi(d) = 12.15 \cdot d^2 + 117.81 \cdot d - 3392, \text{ con } 20 \leq s \leq 52 \text{ la edad en semanas}$$

Las reglas correspondientes son las siguientes:

$$r_{13,m} \equiv [I]_m^0 \rightarrow O_m []_m^0, \quad 1 \leq m \leq 36$$

$$r_{14,m,d} \equiv [Q_d]_m^0 \rightarrow Q'_{d,m} O_m^{g(1) \cdot \Psi(d)} []_m^0 \left\{ \begin{array}{l} 1 \leq m \leq 20 \\ 20 \leq d \leq 52 \end{array} \right.$$

$$r_{15,m,d} \equiv [Q_d]_m^0 \xrightarrow{0.5} Q'_{d,m} []_m^0 \left\{ \begin{array}{l} 1 \leq m \leq 20 \\ 20 \leq d \leq 52 \end{array} \right.$$

$$r_{16,m,d} \equiv [Q_d]_m^0 \xrightarrow{0.5} Q'_{d,m} O_m^{(1-g(1)) \cdot \Psi(d)} []_m^0 \left\{ \begin{array}{l} 21 \leq m \leq 36 \\ 20 \leq d \leq 52 \end{array} \right.$$

$$r_{17,m,d} \equiv [Q_d]_m^0 \xrightarrow{0.5} Q'_{d,m} []_m^0 \left\{ \begin{array}{l} 21 \leq m \leq 36 \\ 20 \leq d \leq 52 \end{array} \right.$$

$$r_{18,m,s} \equiv [X_s]_m^0 \xrightarrow{0.5} Y_s O_m^{g(1) \cdot \Theta(s)} []_m^0 \left\{ \begin{array}{l} 1 \leq m \leq 20 \\ 2 \leq s \leq 6 \end{array} \right.$$

$$r_{19,m,s} \equiv [X_s]_m^0 \xrightarrow{0.5} Y_s O_m^{(1-g(1)) \cdot \Theta(s)} []_m^0 \left\{ \begin{array}{l} 21 \leq m \leq 36 \\ 2 \leq s \leq 6 \end{array} \right.$$

$$r_{20,m,s} \equiv [X_s]_m^0 \xrightarrow{0.5} Y_s []_m^0 \left\{ \begin{array}{l} 1 \leq m \leq 36 \\ 2 \leq s \leq 6 \end{array} \right.$$

$$r_{21,m} \equiv [X_1]_m^0 \xrightarrow{0.5} Y_1 O_m^{g(1) \cdot \Theta(\frac{3}{2})} []_m^0, \quad 1 \leq m \leq 20$$

$$r_{22,m} \equiv [X_1]_m^0 \xrightarrow{0.5} Y_1 O_m^{(1-g(1)) \cdot \Theta(\frac{3}{2})} []_m^0, \quad 21 \leq m \leq 36$$

$$r_{23,m} \equiv [X_1]_m^0 \xrightarrow{0.5} Y_1 []_m^0, \quad 1 \leq m \leq 36$$

Mortalidad natural

La mortalidad natural depende de la edad y la densidad.

$$r_{24,s} \equiv [Y_s \xrightarrow{m(s)} \#]_0^0, \quad 1 \leq s \leq 6$$

$$r_{25,s} \equiv [Y_s]_0^0 \xrightarrow{1-m(s)} V_s []_0^0, \quad 1 \leq s \leq 6$$

Movimiento de adultos por embarcaciones

Existe una probabilidad $PA(j, j_2)$ de que los mejillones adultos sean transportados por las embarcaciones del compartimento j al compartimento j_2 .

$$r_{e_{5,s,j,j'}} \equiv (V_s)_{e_j} \xrightarrow{PA(j,j')} (V'_s)_{e_{j'}} \begin{cases} 1 \leq s \leq 6 \\ 1 \leq j \leq 18 \\ 1 \leq j' \leq 18, j' \neq j \end{cases}$$

$$r_{e_{6,s,j}} \equiv (V_s)_{e_j} \xrightarrow{PA(j,j)} (V'_s)_{e_j} \begin{cases} 1 \leq s \leq 6 \\ 1 \leq j \leq 18 \end{cases}$$

$$r_{26,s} \equiv V'_s []_0^0 \rightarrow [V_s]_0^0, \quad 1 \leq s \leq 6$$

Viabilidad de los huevos

$$r_{27,m} \equiv [O_m \xrightarrow{g(2)} \#]_0^0, \quad 1 \leq m \leq 36$$

$$r_{28,m} \equiv [O_m \xrightarrow{(1-g(2)) \cdot (1-mo(1))} L_{1m}]_0^0, \quad 1 \leq m \leq 20$$

$$r_{29,m} \equiv [O_m \xrightarrow{(1-g(2)) \cdot mo(1)} \#]_0^0, \quad 1 \leq m \leq 20$$

$$r_{30,m} \equiv [O_m \xrightarrow{(1-g(2)) \cdot (1-mo(2))} L_{1m}]_0^0, \quad 21 \leq m \leq 36$$

$$r_{31,m} \equiv [O_m \xrightarrow{(1-g(2)) \cdot mo(2)} \#]_0^0, \quad 21 \leq m \leq 36$$

Mortalidad de las larvas

$$r_{32,m} \equiv [L_{1m} \xrightarrow{1-g(3)} L_{2m}]_0^0, \quad 1 \leq m \leq 36$$

$$r_{33,m} \equiv [L_{1m} \xrightarrow{g(3)} \#]_0^0, \quad 1 \leq m \leq 36$$

La liberación de huevos se extiende a lo largo del tiempo, y los huevos emitidos por cada hembra se distribuyen sobre las semanas que dura el proceso (11 semanas para el primer ciclo, 3 para el segundo).

$$r_{34,j,m} \equiv [L_{2m}]_0^0 \xrightarrow{PI(j,1)} L_{3m+j-1} []_0^0 \begin{cases} 1 \leq j \leq 11 \\ 1 \leq m \leq 20 \end{cases}$$

$$r_{35,j,m} \equiv [L_{2m}]_0^0 \xrightarrow{PI(j,2)} L_{3m+j-1} []_0^0 \begin{cases} 1 \leq j \leq 3 \\ 1 \leq m \leq 36 \end{cases}$$

Movimiento de larvas

Movimiento vertical

$$\begin{aligned}
 r_{e_{7,j,m}} &\equiv (L_{3m} \xrightarrow{0.2 \cdot CV(m)} L_{m,0,j,j+7})_{e_j} \quad \left\{ \begin{array}{l} 1 \leq j \leq 7 \\ 1 \leq m \leq 39 \end{array} \right. \\
 r_{e_{8,j,m}} &\equiv (L_{3m} \xrightarrow{1-0.2 \cdot CV(m)} L_{m,0,j,j})_{e_j} \quad \left\{ \begin{array}{l} 1 \leq j \leq 7 \\ 1 \leq m \leq 39 \end{array} \right. \\
 r_{e_{9,j,m}} &\equiv (L_{3m} \xrightarrow{0.001} L_{m,0,j,j-7})_{e_j} \quad \left\{ \begin{array}{l} 8 \leq j \leq 14 \\ 1 \leq m \leq 39 \end{array} \right. \\
 r_{e_{10,j,m}} &\equiv (L_{3m} \xrightarrow{1-0.001} L_{m,0,j,j})_{e_j} \quad \left\{ \begin{array}{l} 8 \leq j \leq 14 \\ 1 \leq m \leq 39 \end{array} \right. \\
 r_{e_{11,j,m}} &\equiv (L_{3m} \xrightarrow{0.001} L_{m,0,j,j-2})_{e_j} \quad \left\{ \begin{array}{l} 17 \leq j \leq 18 \\ 1 \leq m \leq 39 \end{array} \right. \\
 r_{e_{12,j,m}} &\equiv (L_{3m} \xrightarrow{1-0.001} L_{m,0,j,j})_{e_j} \quad \left\{ \begin{array}{l} 17 \leq j \leq 18 \\ 1 \leq m \leq 39 \end{array} \right.
 \end{aligned}$$

Movimiento debido al régimen hidráulico

Las larvas permanecen en la columna de agua durante 4 semanas, donde pueden ser desplazadas por el agua que se va renovando. El movimiento de larvas que podrían dejar el embalse se simula con periodicidad semanal.

Áreas centrales de la reserva

- Larvas que permanecen en el embalse:

$$\begin{aligned}
 r_{e_{13,j,m,i,j_1,j_2}} &\equiv (L_{m,i,j,j_1} \xrightarrow{PR(m,j_1,j_2)} L_{m+1,i+1,j,j_2})_{e_j} \quad \left\{ \begin{array}{l} 1 \leq j \leq 18 \\ 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \\ 1 \leq j_1 \leq 7 \\ 1 \leq j_2 \leq 7 \end{array} \right. \\
 r_{e_{14,j,m,i,j_1}} &\equiv (L_{m,i,j,j_1} \xrightarrow{PR(m,j_1,8)} \#)_{e_j} \quad \left\{ \begin{array}{l} 1 \leq j \leq 18 \\ 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \\ 8 \leq j_1 \leq 14 \end{array} \right. \\
 r_{e_{15,j,m,i,j_1,j_2}} &\equiv (L_{m,i,j,j_1} \xrightarrow{PR(m,j_1,j_2)} L_{m+1,i+1,j,j_2})_{e_j} \quad \left\{ \begin{array}{l} 1 \leq j \leq 18 \\ 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \\ 8 \leq j_1 \leq 14 \\ 8 \leq j_2 \leq 14 \end{array} \right.
 \end{aligned}$$

$$r_{e_{16,j,m,i,j_1}} \equiv (L_{m,i,j,j_1} \xrightarrow{PR(m,j_1,15)} \#)_{e_j} \begin{cases} 1 \leq j \leq 18 \\ 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \\ 8 \leq j_1 \leq 14 \end{cases}$$

- Los objetos asociados con larvas que dejan el embalse se disuelven.

Áreas laterales del embalse

$$r_{e_{17,m,i}} \equiv (L_{m,i,15,15} \xrightarrow{0.5} L_{m+1,i+1,15,15})_{e_{15}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{18,m,i}} \equiv (L_{m,i,15,15} \xrightarrow{0.5} L_{m+1,i+1,15,1})_{e_{15}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{19,m,i}} \equiv (L_{m,i,17,17} \xrightarrow{0.5} L_{m+1,i+1,17,17})_{e_{17}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{20,m,i}} \equiv (L_{m,i,17,17} \xrightarrow{0.5} L_{m+1,i+1,17,8})_{e_{17}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{21,m,i}} \equiv (L_{m,i,17,15} \xrightarrow{0.5} L_{m+1,i+1,17,15})_{e_{17}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{22,m,i}} \equiv (L_{m,i,17,15} \xrightarrow{0.5} L_{m+1,i+1,17,1})_{e_{17}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{23,m,i}} \equiv (L_{m,i,16,16} \xrightarrow{0.5} L_{m+1,i+1,16,16})_{e_{16}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{24,m,i}} \equiv (L_{m,i,16,16} \xrightarrow{0.5} L_{m+1,i+1,16,6})_{e_{16}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{25,m,i}} \equiv (L_{m,i,18,18} \xrightarrow{0.5} L_{m+1,i+1,18,18})_{e_{18}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{26,m,i}} \equiv (L_{m,i,18,18} \xrightarrow{0.5} L_{m+1,i+1,18,13})_{e_{18}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{27,m,i}} \equiv (L_{m,i,18,16} \xrightarrow{0.5} L_{m+1,i+1,18,16})_{e_{18}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

$$r_{e_{28,m,i}} \equiv (L_{m,i,18,16} \xrightarrow{0.5} L_{m+1,i+1,18,6})_{e_{18}} \begin{cases} 1 \leq m \leq 42 \\ 0 \leq i \leq 3 \end{cases}$$

En este punto, los objetos asociados con larvas aún no se han movido a otros entornos, pero se les ha asociado un índice representando el desplazamiento que van a experimentar. La siguiente regla los moverá entre los entornos origen y destino.

$$r_{e_{29,j,j',m}} \equiv (L_{m,4,j,j'})_{e_j} \rightarrow (L'_m)_{e'_j} \begin{cases} 1 \leq j \leq 18 \\ 1 \leq j' \leq 18, j' \neq j \\ 5 \leq m \leq 43 \end{cases}$$

$$r_{e_{30,j,m}} \equiv (L_{m,4,j,j'} \rightarrow L'_m)_{e_j} \begin{cases} 1 \leq j \leq 18 \\ 5 \leq m \leq 43 \end{cases}$$

Una vez transcurridas las cuatro semanas de la fase larvaria, los mejillones jóvenes se adherirán al sustrato del área donde estén situados en ese momento. El objeto entra en la célula, donde permanece mientras espera la compleción de otros procesos como el reclutamiento de larvas y el control de la capacidad de carga de los mejillones, que determinará el final satisfactorio del proceso de fijación por parte del joven individuo.

$$r_{36,m} \equiv L'_m []_0^0 \rightarrow [Q'_{0,m-1}]_0^0, \quad 5 \leq m \leq 43$$

Los índices del objeto Q' permiten el cálculo de la edad del individuo.

Reglas de no reproducción

La reproducción no siempre es posible ya que pueden no darse las condiciones térmicas necesarias. Como se ha explicado anteriormente, se considera que el primer ciclo reproductivo dura 20 semanas, mientras que el segundo dura 16. Las condiciones térmicas que habilitan la reproducción podrían no alcanzarse durante algunos de estos períodos. En cualquiera de los casos, los mejillones adultos deben evolucionar siguiendo el ciclo, por lo que es preciso desencadenar el disparo de las correspondientes reglas de evolución encargadas de la sincronización de ambas vías del proceso (reproducción o no reproducción).

$$r_{e_{31,j}} \equiv (T_{19} \xrightarrow{p(20,j)} \gamma_{20} \rho_1 \rho_0)_{e_j}, \quad 1 \leq j \leq 18$$

$$r_{e_{32,j}} \equiv (T_{19} \xrightarrow{1-p(20,j)} \delta_{20} \rho_1 \rho_0)_{e_j}, \quad 1 \leq j \leq 18$$

$$r_{e_{33,j}} \equiv (T_{35} \rightarrow \delta_{36} \rho_2 \omega_0)_{e_j}, \quad 1 \leq j \leq 18$$

- Durante el primer ciclo:

$$r_{37} \equiv \delta_{20} []_0^0 \rightarrow [\delta]_0^0$$

$$r_{38} \equiv \delta_{20} []_{20}^0 \rightarrow [\delta]_{20}^+$$

$$r_{39} \equiv [\delta]_{20}^+ \rightarrow [\#]_{20}^0$$

$$r_{40,s} \equiv X_s []_{20}^+ \rightarrow VR [X'_s]_{20}^0, \quad 2 \leq s \leq 6$$

$$r_{41} \equiv X_1 []_{20}^+ \rightarrow [\#]_{20}^0$$

$$r_{42,d} \equiv Q_d []_{20}^+ \rightarrow VR [Q''_{d+20}]_{20}^0, \quad 1 \leq d \leq 32$$

$$r_{43,d} \equiv Q_d []_{20}^+ \rightarrow VR [X'_1]_{20}^0, \quad 33 \leq d \leq 52$$

$$r_{44,s} \equiv [X'_s]_{20}^0 \rightarrow Y_s []_{20}^0, \quad 1 \leq s \leq 6$$

$$r_{45,d} \equiv [Q''_d]_{20}^0 \rightarrow Q'_{d,20} []_{20}^0, \quad 20 \leq d \leq 52$$

- Durante el segundo ciclo:

$$r_{46} \equiv \delta_{36} []_0^0 \rightarrow [\delta_{36}]_0^0$$

$$r_{47} \equiv \delta_{36} []_{36}^0 \rightarrow [\delta]_{36}^+$$

$$r_{48} \equiv [\delta]_{36}^+ \rightarrow [\#]_{36}^0$$

$$r_{49,s} \equiv X_s []_{36}^+ \rightarrow VR [X'_s]_{36}^0, \quad 2 \leq s \leq 6$$

$$r_{50} \equiv X_1 []_{36}^+ \rightarrow [\#]_{36}^0$$

$$r_{51,d} \equiv Q_d []_{36}^+ \rightarrow [Q''_{d+16}]_{36}^0, \quad 1 \leq d \leq 36$$

$$r_{52,d} \equiv Q_d []_{36}^+ \rightarrow VR [X'_1]_{36}^0, \quad 37 \leq d \leq 52$$

$$r_{53,s} \equiv [X'_s]_{36}^0 \rightarrow Y_s []_{36}^0, \quad 1 \leq s \leq 6$$

$$r_{54,d} \equiv [Q''_d]_{36}^0 \rightarrow Q'_{d,36} []_{36}^0, \quad 16 \leq d \leq 52$$

Dos contadores (ρ_0, ω_0) permitirán la sincronización del proceso de control de la capacidad de carga de los mejillones por área.

$$r_{55} \equiv \rho_0 []_0^0 \rightarrow [\rho_0]_0^0$$

$$r_{56} \equiv \rho_0 []_{37}^0 \rightarrow [\rho_0]_{37}^0$$

$$r_{57,i} \equiv [\rho_i \rightarrow \rho_{i+1}]_{37}^0, \quad 0 \leq i \leq 10$$

$$r_{58} \equiv [\rho_{11}]_{37}^0 \rightarrow [\rho]_{37}^-$$

$$r_{59} \equiv \omega_0 []_0^0 \rightarrow [\omega_0]_0^0$$

$$r_{60} \equiv \omega_0 []_{38}^0 \rightarrow [\omega_0]_{38}^0$$

$$r_{61,i} \equiv [\omega_i \rightarrow \omega_{i+1}]_{38}^0, \quad 0 \leq i \leq 14$$

$$r_{62} \equiv [\omega_{15}]_{37}^0 \rightarrow [\rho]_{38}^-$$

Control del nivel de reclutamiento

Los objetos ρ_i evolucionan generando objetos A y AR ; los primeros permiten el control de la capacidad de carga (< 2500), y los segundos proporcionan información sobre la densidad de mejillones adultos, que se empleará para estimar el reclutamiento de individuos jóvenes.

$$r_{e34,j,i} \equiv (\rho_i \rightarrow A_i^{\varphi_j} AR^{2500})_{ej} \left\{ \begin{array}{l} 1 \leq j \leq 18 \\ 1 \leq i \leq 2 \end{array} \right.$$

$$r_{63,i} \equiv A_i []_0^0 \rightarrow [A_i]_0^0, \quad 1 \leq i \leq 2$$

$$r_{64,i} \equiv A_i []_{36+i}^0 \rightarrow [A]_{36+i}^0, \quad 1 \leq i \leq 2$$

Los objetos AR deben esperar 5 pasos antes de entrar a la membrana etiquetada por 39, con el fin de asegurarse de que los individuos que han comenzado la última semana han finalizado su fase larvaria.

$$r_{65} \equiv AR []_0^0 \rightarrow [AR_1]_0^0$$

$$r_{66,i} \equiv [AR_i \rightarrow AR_{i+1}]_0^0 , \quad 1 \leq i \leq 4$$

$$r_{67} \equiv AR_5 []_{39}^0 \rightarrow [AR]_{39}^+$$

Tras el cambio de polaridad se generan objetos DP_i , cuyo subíndice indicará el ratio de mejillones en relación a la capacidad de carga; un valor de 20 para el mismo representa el 100 % (un valor de 0, naturalmente el 0 %).

$$r_{68} \equiv [\alpha \rightarrow \alpha_1 DP_0]_{39}^+$$

$$r_{69} \equiv VR []_{39}^+ \rightarrow [VR]_{39}^+$$

$$r_{70} \equiv [\alpha_1]_{39}^+ \rightarrow [\alpha_1]_{39}^-$$

$$r_{71} \equiv [ARVR]_{39}^+ \rightarrow [\#]_{36}^-$$

$$r_{72,i} \equiv [\alpha_i \rightarrow \alpha_{i+1}]_{39}^- , \quad 1 \leq i \leq 6$$

El objeto DP_1 representa el 5 % de la capacidad de carga.

$$r_{73} \equiv [AR^{125} \rightarrow DP_1]_{39}^-$$

$$r_{74,i,k} \equiv [DP_i DP_k \rightarrow DP_{i+k}]_{39}^- \left\{ \begin{array}{l} 0 \leq i \leq 20 \\ 1 \leq k \leq 9 \end{array} \right.$$

$$r_{75} \equiv [\alpha_7]_{39}^- \rightarrow [\beta_1]_{39}^0$$

$$r_{76} \equiv [AR \rightarrow \#]_{39}^0$$

$$r_{77} \equiv [VR \rightarrow \#]_{39}^0$$

Cada objeto final DP_i deja la membrana generando un número de objetos mayor que el número de mejillones jóvenes generados en el ciclo reproductivo (con un número máximo de 250000 huevos).

$$r_{78,i} \equiv [DP_i]_{39}^0 \rightarrow DP_i^{1250 \cdot 250000} []_{39}^0 , \quad 1 \leq i \leq 20$$

$$r_{79,i} \equiv [\beta_i \rightarrow \beta_{i+1}]_{39}^0 , \quad 1 \leq i \leq 7$$

$$r_{80} \equiv [\beta_8]_{39}^0 \rightarrow [\beta_9]_{39}^-$$

$$r_{81} \equiv [\beta_9]_{39}^- \rightarrow [\alpha]_{39}^0$$

Mortalidad dependiente de la densidad y reclutamiento de individuos jóvenes

Cuando las membranas 37 y 38 presentan polaridad negativa, todos los objetos que representan mejillones entran en esas membranas para comenzar el proceso de cálculo de individuos supervivientes en función de la densidad.

$$\begin{aligned}
r_{82,k} &\equiv [\rho]_k^- \rightarrow [\eta]_k^0, \quad 37 \leq k \leq 38 \\
r_{83,k} &\equiv [\alpha]_k^- \rightarrow [\beta_0]_k^0, \quad 37 \leq k \leq 38 \\
r_{84,k,s} &\equiv V_s []_k^- \rightarrow [V_s]_k^0 \begin{cases} 37 \leq k \leq 38 \\ 1 \leq s \leq 6 \end{cases} \\
r_{85,m,d} &\equiv Q'_{d,m} []_{37}^- \rightarrow [Q_{20-m+d}]_{37}^0 \begin{cases} 1 \leq m \leq 20 \\ m - 20 \leq d \leq 52 + m - 20 \end{cases} \\
r_{86,m,d} &\equiv Q'_{d,m} []_{37}^- \rightarrow [V_1]_{37}^0 \begin{cases} 1 \leq m \leq 20 \\ 53 + m - 20 \leq d \leq 71 + m - 20 \end{cases} \\
r_{87,m,d} &\equiv Q'_{d,m} []_{38}^- \rightarrow [Q_{52-m+d}]_{38}^0 \begin{cases} 21 \leq m \leq 40 \\ 0 \leq d \leq m \end{cases} \\
r_{88,m,d} &\equiv Q'_{d,m} []_{38}^- \rightarrow [V_1]_{38}^0 \begin{cases} 21 \leq m \leq 40 \\ m + 1 \leq d \leq 83 + m - 52 \end{cases}
\end{aligned}$$

Comienzo del proceso de mortalidad debido al exceso de densidad

$$\begin{aligned}
r_{89,k,s} &\equiv [V_s A]_k^0 \xrightarrow{0.5 \cdot 0.2 \cdot (s+1)} X_{s+1} []_k^+ \begin{cases} 37 \leq k \leq 38 \\ 2 \leq s \leq 5 \end{cases} \\
r_{90,k,s} &\equiv [V_s A]_k^0 \xrightarrow{1 - 0.5 \cdot 0.2 \cdot (s+1)} [V_s A]_k^+ \begin{cases} 37 \leq k \leq 38 \\ 2 \leq s \leq 5 \end{cases} \\
r_{91,k} &\equiv [A]_k^+ \rightarrow [A_0]_k^0, \quad 37 \leq k \leq 38 \\
r_{92,k,s} &\equiv [A_s]_k^+ \rightarrow [A_{s+1}]_k^0 \begin{cases} 37 \leq k \leq 38 \\ 0 \leq s \leq 4 \end{cases} \\
r_{93,k} &\equiv [\beta_0]_k^+ \rightarrow [\beta_1]_k^0, \quad 37 \leq k \leq 38 \\
r_{94,k} &\equiv [\beta_1]_k^+ \rightarrow [\beta_1]_k^0, \quad 37 \leq k \leq 38 \\
r_{95,k,i} &\equiv [\beta_i]_k^0 \rightarrow [\alpha_{i+1}]_k^+ \begin{cases} 37 \leq k \leq 38 \\ 1 \leq i \leq 6 \end{cases} \\
r_{96,k,i} &\equiv [\alpha_i]_k^+ \rightarrow [\beta_i]_k^0 \begin{cases} 37 \leq k \leq 38 \\ 2 \leq i \leq 6 \end{cases} \\
r_{97,k} &\equiv [\alpha_7]_k^+ \rightarrow [\#]_k^-, \quad 37 \leq k \leq 38 \\
r_{98,k,d} &\equiv [Q_d A_0]_k^0 \rightarrow QQ_{d,k} []_k^+ \begin{cases} 37 \leq k \leq 38 \\ 1 \leq d \leq 52 \end{cases} \\
r_{99,k} &\equiv [V_1 A_1]_k^0 \rightarrow XX_{2,k} []_k^+, \quad 37 \leq k \leq 38 \\
r_{100,k,s} &\equiv [V_s A_s]_k^0 \rightarrow X_{s+1} []_k^+ \begin{cases} 37 \leq k \leq 38 \\ 2 \leq s \leq 5 \end{cases}
\end{aligned}$$

$$\begin{aligned}
r_{101,k,d,i} &\equiv QQ_{d,k} DP_i []_k^+ \xrightarrow{0.1 \cdot i - 1} Q_d X_1 []_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 1 \leq d \leq 52 \\ 10 \leq i \leq 20 \end{array} \right. \\
r_{102,k,d,i} &\equiv QQ_{d,k} DP_i []_k^+ \xrightarrow{1 - 0.1 \cdot i - 1} [A_1]_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 1 \leq d \leq 52 \\ 10 \leq i \leq 20 \end{array} \right. \\
r_{103,k,i} &\equiv XX_{2,k} DP_i []_k^+ \xrightarrow{0.1 \cdot i - 1} X_2 []_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 10 \leq i \leq 20 \end{array} \right. \\
r_{104,k,i} &\equiv XX_{2,k} DP_i []_k^+ \xrightarrow{1 - 0.1 \cdot i - 1} [A_2]_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 10 \leq i \leq 20 \end{array} \right. \\
r_{105,k,d,i} &\equiv QQ_{d,k} DP_i []_k^+ \rightarrow [A_1]_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 1 \leq d \leq 52 \\ 0 \leq i \leq 9 \end{array} \right. \\
r_{106,k,i} &\equiv XX_{2,k} DP_i []_k^+ \rightarrow [A_2]_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 0 \leq i \leq 9 \end{array} \right.
\end{aligned}$$

Restauración de la configuración inicial

$$\begin{aligned}
r_{107} &\equiv [\eta]_{37}^- \rightarrow T_{20} T'_{20} [\alpha]_{37}^0 \\
r_{108} &\equiv [\eta]_{38}^- \rightarrow T_0 [\alpha]_{38}^0 \\
r_{109} &\equiv [I' T'_{20}]_0^0 \rightarrow I_2 []_0^0 \\
r_{110} &\equiv [T_{20}]_0^0 \rightarrow T_{20} []_0^0 \\
r_{111} &\equiv [T_0]_0^0 \rightarrow T_0 []_0^0 \\
r_{112} &\equiv T'_{20} []_{36}^+ \rightarrow []_{36}^0 \\
r_{113,k,s} &\equiv [V_s]_k^- \rightarrow D []_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 2 \leq s \leq 6 \end{array} \right. \\
r_{114,k,s} &\equiv [Q_d]_k^- \rightarrow D []_k^0 \left\{ \begin{array}{l} 37 \leq k \leq 38 \\ 1 \leq d \leq 87 \end{array} \right. \\
r_{115,k} &\equiv [A_5]_k^- \rightarrow []_k^0, 37 \leq k \leq 38 \\
r_{116,i} &\equiv DP_i []_{39}^- \rightarrow []_{39}^0, 0 \leq i \leq 20
\end{aligned}$$

6.4.1. Traza del modelo

El modelo presentado abarca un gran número de elementos y reglas de evolución. Para facilitar el acercamiento a la dinámica del modelo, se presenta una traza parcial de la evolución del mismo, estructurada en una serie de años a simular, según el escenario que se quiera estudiar. La simulación de cada año conllevará la ejecución de dos ciclos de 51 pasos de computación; es decir, 102 pasos por año.

Además, el sistema PDP se compone de 18 entornos, cada uno de los cuales presenta un sistema P que contiene 40 membranas (la raíz es la membrana 0 y las restantes membranas son hijas de la raíz). Queda por tanto fuera del objetivo de esta memoria presentar la evolución del sistema completo bajo unas determinadas condiciones (un escenario con sus parámetros y poblaciones iniciales) durante un determinado número de años de forma detallada, en cuanto a la estructura y el número de pasos a considerar.

En su lugar, dado un determinado conjunto de parámetros de entrada, se esboza la traza de un subconjunto de pasos de computación (los primeros 21 pasos), para un determinado entorno j , empezando por la configuración inicial 0. Este esbozo no pretende ser exhaustivo sino, más bien, ilustrar parcialmente la evolución de los objetos de una parte del sistema (sin incorporar su multiplicidad) conforme se va produciendo la aplicación de las reglas correspondientes. Esta traza parcial se muestra en la figura 6.7, y el número de la correspondiente configuración se puede ver por el subíndice del objeto T_i presente en cada paso en el entorno; es decir en el “exterior” de la membrana 0.

En la figura se observa que en el paso 1 aparece el objeto γ_1 , que se habrá producido por la aplicación de la regla $r_{e3,j,0}$ para el entorno j que se muestra en la traza, la cual tenía una probabilidad de $p(1,j)$ de aplicarse. Igualmente podría haberse aplicado su regla “complementaria”, con probabilidad $1 - p(1,j)$. En este caso, al aplicarse la primera regla se asume que se dan las condiciones de temperatura adecuadas para comenzar el ciclo reproductivo, desencadenando este proceso γ_1 al entrar en la membrana 1 y cambiar la carga de neutra (0) a negativa (-).

Como sistema parametrizado, esta computación dependerá de los datos de entrada del escenario concreto introducido y, como sistema probabilístico, no será suficiente con el cumplimiento de las premisas impuestas por la parte izquierda de las reglas para poder asegurar su ejecución, sino que se dará con una cierta probabilidad.

Por este motivo es importante recalcar que la traza anterior no es más que un ejercicio ilustrativo de algunas de las reglas que se van aplicando en una posible computación del sistema, mostrando solamente una parte de éste dado el tamaño que alcanzaría una secuencia de pasos del sistema completo con sus 18 entornos, cada uno con sus 40 membranas y los multiconjuntos presentes en cada una para cada configuración, incluida su multiplicidad.

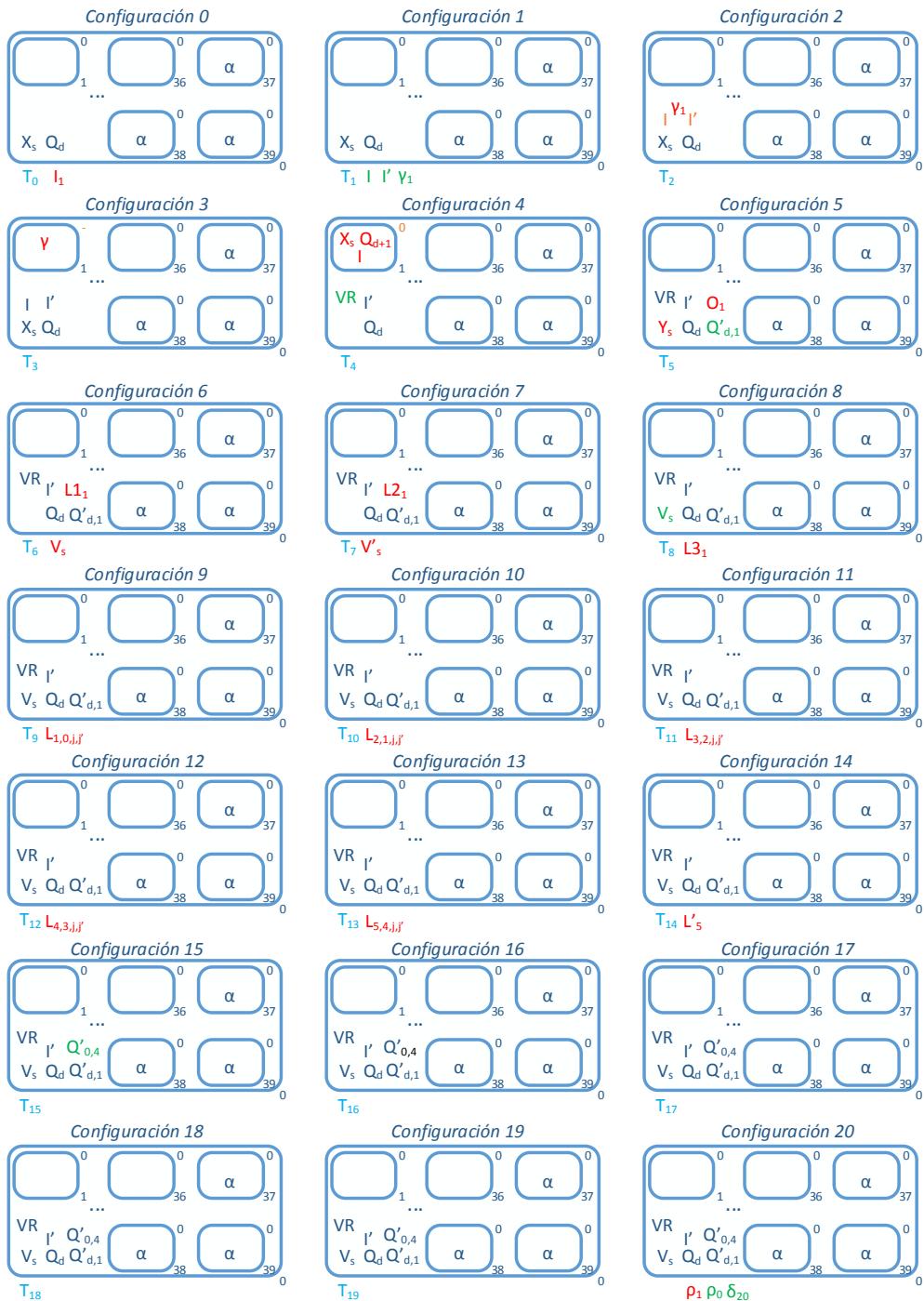


Figura 6.7: Traza de ejecución
{rojo: a eliminar, verde: nuevos, naranja: modificados}

Para el ejemplo mostrado, el primer paso consiste en la simulación de temperaturas que, en este caso, proporciona unas condiciones favorables para el comienzo del ciclo reproductivo. Los pasos 2 a 5 llevan a cabo el proceso de reproducción. El paso 6 aplica la mortalidad natural de adultos (pasando los supervivientes de Y_s a V_s), al mismo tiempo que se comprueba la viabilidad de los óvulos (pasando los óvulos viables de O_1 a L_{11}). El paso 7 ejecuta, por un lado, el proceso de movimiento de adultos por embarcaciones, desplazando objetos V_s o bien entre entornos, o bien hacia el mismo actual, pasando en ambos casos a objetos V'_s . Por otro lado, se aplica la mortalidad de larvas, pasando los supervivientes de L_{11} a L_{21} .

En el paso 8, los mejillones adultos V'_s que llegaron a su entorno destino, se transforman en V_s al pasar al interior de la membrana 0 del entorno. Al mismo tiempo se produce la liberación de huevos. Para ello, se generan objetos indicados en la traza como $L_{3_{m+j-1}}$, $1 \leq m \leq 20$, $1 \leq j \leq 11$, simulando la liberación de los huevos desde la semana m (en la posible computación mostrada en la traza $m = 1$, por tanto desde la primera semana) hasta la semana 11. Por simplificar el diagrama, se muestran estos objetos en la traza como L_{3_s} , entendiendo s como semana.

En el paso 9 se calcula el posible movimiento vertical de las larvas, y en los pasos 10 a 13 se implementan los posibles movimientos entre las áreas del embalse debido al régimen de renovación del agua a lo largo de las 4 semanas que las larvas pueden permanecer en la columna de agua. El paso 14 realiza de forma efectiva el movimiento, pasando de $L_{5,4,j,j'}$ en el entorno e_j a L'_5 en el entorno $e_{j'}$. En el paso 15 se introduce en la membrana piel (etiquetada por 0) el objeto correspondiente al individuo joven $Q'_{0,4}$. Se está realizando el seguimiento a los individuos cuya reproducción se activó como consecuencia de la aparición de condiciones de temperatura favorables ya en el paso 1, al pasar de T_0 a T_1 . Los objetos recién creados $Q'_{0,4}$ que representan individuos jóvenes, quedarán en el interior de la región correspondiente a la membrana 0 hasta el final del ciclo reproductivo (que, como se ha comentado, consta de 20 semanas), no observándose más cambios en la traza mostrada hasta llegar a dicho fin del ciclo. Es entonces cuando se llevarán a cabo los procesos de reclutamiento de larvas o capacidad de carga de los mejillones.

Al final del ciclo, en el paso 20, con una determinada probabilidad $p(20, j)$ se producirá en el entorno el objeto γ_{20} , continuando con la reproducción, o bien δ_{20} para la no reproducción. A partir de aquí continuarían los 32 pasos restantes del primer ciclo y, posteriormente, los otros 51 del segundo ciclo, hasta completar los diferentes procesos involucrados en el modelo.

Sirva este simple fragmento, que se focaliza en una posible evolución de solamente algunos objetos dentro de algunas membranas de un entorno de los 18 del sistema, para ilustrar lo laborioso y complejo que puede resultar el análisis de este tipo de trazas a mano.

Las herramientas plasmadas en el siguiente apartado 6.4.2 permiten facilitar esta labor mediante diversas vistas de la depuración paso a paso, tanto en forma detallada como texto mostrando cada configuración y las reglas aplicadas en cada paso, como en forma de árbol con el contenido de cada entorno y cada membrana en una configuración determinada.

6.4.2. El modelo como herramienta para la gestión del ecosistema

El modelo presentado en la sección 6.4 y del cual se ha analizado un fragmento de una posible traza de ejecución, a partir de un escenario de entrada en la sección 6.4.1, captura muchos de los procesos que afectan a la dinámica de poblaciones del mejillón cebra en el embalse de Ribarroja, incorporando numerosos elementos naturales y humanos del entorno que influyen en su evolución. Hasta aquí podríamos estar hablando de un modelo interesante desde el punto de vista de la investigación, de una serie de fenómenos y de la publicación de los resultados de este trabajo, tras el estudio de una serie de escenarios de interés, la formulación de determinadas hipótesis y la presentación de gráficos acerca de la evolución de la especie bajo las condiciones iniciales planteadas en los escenarios concretos de objeto de estudio.

Sin embargo, una parte importante de la propuesta metodológica que se formula en el presente trabajo va encaminada al empleo de los modelos, no como elementos estáticos sino como entes útiles para los usuarios finales, proporcionándoles herramientas que les permitan profundizar en el estudio de los fenómenos de su interés, permitiéndoles implementar escenarios de su interés y analizar la posible evolución del sistema de acuerdo a las condiciones planteadas. A través de esta experimentación virtual, esos usuarios, por ejemplo los gestores de un ecosistema, pueden plantear situaciones plausibles y observar/analizar la evolución del sistema o bien las posibles consecuencias de una determinada batería de acciones que se podrían ejercer sobre el ecosistema.

Para convertir esa propuesta metodológica en una realidad palpable, es necesario proporcionar herramientas automatizadas que proporcione la infraestructura software necesaria para los usuarios finales. Esto pasa, lógicamente, por facilitar la labor de los diseñadores de los sistemas P correspondientes para que dispongan de marcos adecuados para la modelización, depuración, simulación, análisis introspectivo o verificación de los diseños realizados. Además, una vez validados los modelos, deberá dotarse a estos sistemas de mecanismos lo más sencillos posibles que proporcionen aplicaciones visuales a los usuarios, a partir de las cuales se puedan gestionar los datos del sistema estudiado y visualizar su evolución, sin necesidad de contar con equipos de ingenieros que desarrollen herramientas para cada problema a estudiar.

Con esta filosofía y estos objetivos surgió **MeCoSim**, que se apoya en el potente marco que **P-Lingua** proporciona en cuanto al lenguaje de los sistemas P y su reconocimiento, así como una amplia gama de simuladores para distintas variantes de estos sistemas. A partir de ahí, la atención en el desarrollo de **MeCoSim** se centra en resolver la segunda capa, proporcionar a más alto nivel las herramientas visuales para diseñadores y usuarios, con el fin de vertebrar la metodología a través de herramientas que permitan pasar del estudio del modelo a la liberación de una aplicación software adaptada a las necesidades del problema objeto de estudio.

A continuación se comentan los pasos necesarios para convertir el modelo del mejillón cebra presentado, en una aplicación que sirva a los gestores del embalse como herramienta para el apoyo a una toma de decisiones más informada, al permitirle analizar de antemano las posibles consecuencias y modificaciones en la dinámica de población de la especie y el ecosistema, en función de las posibles actuaciones a realizar sobre el mismo.

El primer paso será traducir el modelo a un lenguaje entendible por la máquina, lo cual se consigue a través de **P-Lingua** y lo hace de una forma muy cercana a la propia sintaxis del marco de modelización correspondiente.

Definición del modelo en **P-Lingua**

A continuación se presenta el código **P-Lingua** correspondiente al modelo descrito en la sección anterior. Gran parte de la información requerida por el modelo dependerá de cada escenario particular. Por ello, el código en cuestión estará parametrizado.

La estructura básica del fichero presenta el tipo de modelo, tras la directiva **@model**, la estructura de entornos y membranas **@mu**, los multiconjuntos iniciales **@ms** y las reglas correspondientes. Los alfabetos de trabajo y de los entornos se infieren a partir de los símbolos empleados en el conjunto de instrucciones dado por el fichero.

```

@model <probabilistic>

def main()
{
    @mu = []'global;
    @mu(global) += [[[]'0]'{j},{j} : 101<= j <= 118;
    @mu(0,{j}) += []'{m}: 1<=m<=39, 101<= j <= 118;

    @ms({0},{j+100}) += X{s}*(q{j}) : 2<= s <= 6, 1<= j <= 18;
    @ms({0},{j+100}) += Q{13}*(q{j}), Q{39}*(q{j}) : 1<= j <= 18;
    @ms({36+m}) += alpha : 1<=m<=3;
    @ms({j+100}) += T{0},I1 : 1<=j<=18;

    /**<Inoculación externa> ***/
    /*re1*/ [I1]'{j+100} --> [I*LE{j,1},Ip]'{j+100} :: 1: 1<=j<=18;
    /*re2*/ [I2]'{j+100} --> [I*LE{j,2}]'{j+100} :: 1: 1<=j<=18;
    /*r1*/ I[]'0 --> [I]'0 :: 1;
    /*r2*/ Ip[]'0 --> [Ip]'0 :: 1;
    /**<Fin Inoculación externa> ***/
}

```

```

/*** <Simulación de temperaturas y orden de activación de la reproducción> ***/
/*re3*/
[T{d}],{j+100} --> [gamma{d+1},T{d+1}],{j+100} :: p{d+1,j} : 0<=d<=34,1<=j<=18,d<>19 ;
/*re4*/ [T{d}],{j+100} --> [T{d+1}],{j+100} :: 1-p{d+1,j} : 0<= d <=34, 1<=j<=18,d<>19;
/*r3*/ gamma{m}[],0 --> [gamma{m}],0 :: 1 : 1<=m<=36;
/*r4*/ gamma{m}[],{m} --> -[gamma]{m} :: 1 : 1<=m<=36;
/*r5*/ -[gamma],[m] --> [],{m} :: 1 : 1<=m<=36;
/**<Fin Simulación de temperaturas y orden de activación de la reproducción>***/

/*** <Preparar para el inicio de la reproducción> ***/
/*r6*/ I -[],{m} --> [I],{m} :: 1 : 1<=m<=36;
/*r7*/ X{1} -[],{m} --> [#],{m} :: 1 : 1<=m<=36;
/*r8*/ X{s} -[],{m} --> VR[X{s}},{m} :: 1 : 2<=s<=6,1<=m<=36;
/*r9*/ Q{d} -[],{m} --> VR[Q{d+m}],{m} :: 1 : 20-m<=d<=52-m, 1<=m<=20;
/*r10*/ Q{d} -[],{m} --> VR[Q{d+m-20}],{m} :: 1 : 20-m+20<=d<=52-m+20, 21<=m<=36;
/*r11*/ Q{d} -[],{m} --> VR[X{1}],{m} :: 1 : 53-m<=d<=72-m, 1<=m<=20;
/*r12*/ Q{d} -[],{m} --> VR[X{1}],{m} :: 1 : 53-m+20<=d<=72-m+20, 21<=m<=36;
/**<Fin Preparar para el inicio de la reproducción> **/


/*** <Reproducción> ***/
/*r13*/ [I],[m]-->0{m} [] ,{m} :: 1 : 1<=m<=36;
/*r14*/ [Q{d}],{m} --> Qp{d,m},0{m}*(g{1}*(12.15*d*d+117.81*d-3392))[] ,{m} :: 0.5 :
20<=d<= 52, 1<=m<=20;
/*r15*/ [Q{d}],{m} --> Qp{d,m}[],{m} :: 0.5 : 20<= d <= 52, 1<= m <= 20;
/*r16*/ [Q{d}],{m} --> Qp{d,m},0{m}*((12.15*d*d+117.81*d-3392)*(1-g{1}))[] ,{m} :: 0.5 :
20<= d<= 52, 21<= m <= 36;
/*r17*/ [Q{d}],{m} --> Qp{d,m}[],{m} :: 0.5 : 20<= d <= 52, 21<= m <= 36;
/*r18*/ [X{s}],[m] --> Y{s},0{m}*((1920.5*(52/4+52/2*s)-79065)*g{1})[] ,{m} :: 0.5 :
2<=s<=6, 1<=m<=20;
/*r19*/ [X{s}],[m] --> Y{s},0{m}*((1920.5*(52/4+52/2*s)-79065)*(1-g{1}))[] ,{m} :: 0.5 :
2<=s<=6, 21<=m<=36;
/*r20*/ [X{s}],[m] --> Y{s}[],{m} :: 0.5 : 2<=s<=6, 1<=m<=36;
/*r21*/ [X{1}],{m} --> Y{1},0{m}*((1920.5*(52)-79065)*g{1})[] ,{m} :: 0.5 : 1<=m<=20;
/*r22*/ [X{1}],{m} --> Y{1},0{m}*((1920.5*(52)-79065)*(1-g{1}))[] ,{m} :: 0.5 : 21<=m<=36;
/*r23*/ [X{1}],{m} --> Y{1}[],{m} :: 0.5 : 1<=m<=36;
/*r24*/ [Y{s}] --> #]^0:: m{s} : 1<=s<=6;
/*r25*/ [Y{s}]^0 --> V{s}[]^0 :: 1-m{s} : 1<=s<=6;
/**<Fin Reproducción> **/


***** <Entorno adulto> ****/
/*re5*/ [[V{s}],[j+100]][],{jp+100}]'global -->[[],{j+100}][Vp{s}],[jp+100]]'global:: PA{j,jp}:
1<=s<=6, 1<=j<=18, 1<=jp<=18,j>jp;
/*re6*/ [[V{s}],[j+100]]'global -->[[Vp{s}],[j+100]]'global:: PA{j,j}: 1<=s<=6, 1<=j<=18;
/*r26*/ Vp{s}[],0 --> [V{s}]^0 :: 1: 1<=s<=6;
/**<Fin Entorno adulto> **/


**** <Viabilidad de los óvulos> ***
/*r27*/ [0{m} --> #]^0 :: g{2}:1<=m<=36;

**** <Temperatura para el éxito en la fecundación> ***
/*r28*/ [0{m} --> L1{m}],{0} :: (1-g{2})*(1-mo{1}):1<=m<=20;
/*r29*/ [0{m} --> #]^0 :: (1-g{2})*mo{1}:1<=m<=20;
/*r30*/ [0{m} --> L1{m}],{0} :: (1-g{2})*(1-mo{2}):21<=m<=36;
/*r31*/ [0{m} --> #]^0 :: (1-g{2})*mo{2}:21<=m<=36;

**** <Mortalidad> ***
/*r32*/ [L1{m} --> L2{m}],{0}: (1-g{3}): 1<=m<=36;
/*r33*/ [L1{m} --> #]^0: g{3} : 1<=m<=36;

**** <Distribución por semanas> ***
/*r34*/ [L2{m}],{0} --> L3{m+k-1}[],{0}: PI{k,1}: 1<=m<=20, 1<=k<=11;
/*r35*/ [L2{m}],{0} --> L3{m+k-1}[],{0}: PI{k,2}: 21<=m<=36, 1<=k<=3;

**** <Desplazamiento vertical de las larvas> ***
/* El 39 proviene de (36 +3) que dura la reproducción del segundo ciclo*/
/*re7*/ [L3{m} --> L{m,0,j,j+7}],{j+100}:: 0.2*CV{m}: 1<=m<=39, 1<=j<=7;
/*re8*/ [L3{m} --> L{m,0,j,j}],{j+100} :: 1-0.2*CV{m}: 1<=m<=39, 1<=j<=7;
/*re9*/ [L3{m} --> L{m,0,j,j-7}],{j+100}:: 0.001: 1<=m<=39, 8<=j<=14;
/*re10*/ [L3{m} --> L{m,0,j,j}],{j+100} :: 1-0.001: 1<=m<=39, 8<=j<=14;
/*re11*/ [L3{m} --> L{m,0,j,j-2}],{j+100}:: 0.001: 1<=m<=39, 17<=j<=18;

```

```

/*re12*/ [L3{m} --> L{m,0,j,j}]'{j+100} :: 1-0.001: 1<=m<=39, 17<=j<=18;
/**<Cambio de entorno debido al sistema de renovación semanal> ***/
/**<Alto> ***/
/*re13*/ [L{m,i,j,j1}--> L{m+1,i+1,j,j2}]'{j+100} :: PR{m,j1,j2} :
   1<=m<=42, 0<=i<=3, 1<=j<=18, 1<=j1<=7, 1<=j2<=7;
/*re14*/ [L{m,i,j,j1}--> #]'{j+100} :: PR{m,j1,8} : 1<=m<=42, 0<=i<=3, 1<=j<=18, 1<=j1<=7;

/**<Bajo> ***/
/*re15*/ [L{m,i,j,j1}--> L{m+1,i+1,j,j2}]'{j+100} :: PR{m,j1,j2} :
   1<=m<=42, 0<=i<=3, 1<=j<=18, 8<=j1<=14, 8<=j2<=14;
/*re16*/ [L{m,i,j,j1}--> #]'{j+100} :: PR{m,j1,15} : 1<=m<=42, 0<=i<=3, 1<=j<=18, 8<=j1<=14;

/**<Ramales laterales> ***/
/*re17*/ [L{m,i,15,15}--> L{m+1,i+1,15,15}]'{115} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re18*/ [L{m,i,15,15}--> L{m+1,i+1,15,1}]'{115} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re19*/ [L{m,i,17,17}--> L{m+1,i+1,17,17}]'{117} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re20*/ [L{m,i,17,17}--> L{m+1,i+1,17,8}]'{117} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re21*/ [L{m,i,17,15}--> L{m+1,i+1,17,15}]'{117} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re22*/ [L{m,i,17,15}--> L{m+1,i+1,17,1}]'{117} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re23*/ [L{m,i,16,16}--> L{m+1,i+1,16,16}]'{116} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re24*/ [L{m,i,16,16}--> L{m+1,i+1,16,6}]'{116} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re25*/ [L{m,i,18,18}--> L{m+1,i+1,18,18}]'{118} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re26*/ [L{m,i,18,18}--> L{m+1,i+1,18,13}]'{118} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re27*/ [L{m,i,18,16}--> L{m+1,i+1,18,16}]'{118} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re28*/ [L{m,i,18,16}--> L{m+1,i+1,18,6}]'{118} :: 0.5 : 1<=m<=42, 0<=i<=3;
/*re29*/ [[[L{m,4,j,jp}]]'{j+100}][jp+100]global-->[[[]]{j+100}[Lp{m}]]'{jp+100}global :: 1 :
   5<=m<=43, 1<=j<=18, 1<=jp<=18, jp>j;
/*re30*/ [L{m,4,j,j}-->Lp{m}]]'{j+100}::1: 5<=m<=43, 1<=j<=18;
/*r36*/ Lp{m} []'0-->[Qp{0,m-1}]]'0::1: 5<=m<=43;
/**<Final del ciclo> ***/
/*re31*/ [T{19}]]'{j+100} --> [gamma{20},ro{1},ro{0}]'{j+100} :: p{20,j} : 1<=j<=18;
/*re32*/ [T{19}]]'{j+100} --> [delta{20},ro{1},ro{0}]'{j+100} :: 1-p{20,j} : 1<=j<=18;
/*re33*/ [T{35}]]'{j+100} --> [delta{36},ro{2},omega{0}]'{j+100} :: 1 : 1<=j<=18;

/**<No reproducción (caso de no darse las condiciones de temperatura)> ***/
/**<Primer ciclo> ***/
/*r37*/ delta{20} []'0-->[delta{20}]'0::1;
/*r38*/ delta{20} []'20-->[delta]'20::1;
/*r39*/ +[delta]'20-->[]'20::1;
/*r40*/ X{s}+[]'20-->VR[Xp{s}]'20::1: 2<=s<=6;
/*r41*/ X{1}+[]'20-->[#]'20::1;
/*r42*/ Q{d}+[]'20-->VR[Qpp{d+20}]'20::1: 1<=d<=32;
/*r43*/ Q{d}+[]'20-->VR[Xp{1}]'20::1: 33<=d<=52;
/*r44*/ [Xp{s}]'20--> Y{s}[]'20::1: 1<=s<=6;
/*r45*/ [Qpp{d}]'20--> Qp{d,20}[]'20::1: 20<=d<=52;

/**<Segundo ciclo> ***/
/*r46*/ delta{36} []'0-->[delta{36}]'0::1;
/*r47*/ delta{36} []'36-->+[delta]'36::1;
/*r48*/ +[delta]'36-->[]'36::1;
/*r49*/ X{s}+[]'36-->VR[Xp{s}]'36::1: 2<=s<=6;
/*r50*/ X{1}+[]'36-->[#]'36::1;
/*r51*/ Q{d}+[]'36-->[Qpp{d+16}]'36::1: 1<=d<=36;
/*r52*/ Q{d}+[]'36-->VR[Xp{1}]'36::1: 37<=d<=52;
/*r53*/ [Xp{s}]'36--> Y{s}[]'36::1: 1<=s<=6;
/*r54*/ [Qpp{d}]'36--> Qp{d,36}[]'36::1: 16<=d<=52;

/**<Sincronización del proceso de control de la capacidad de carga> ***/
/* Reglas con ro y omega */
/*r55*/ ro{0} []'0-->[ro{0}]'0::1;
/*r56*/ ro{0} []'37-->[ro{0}]'37::1;
/*r57*/ [ro{i}-->ro{i+1}]'37::1: 0<=i<=10;
/*r58*/ [ro{11}]'37-->[ro]'37::1;

/*r59*/ omega{0} []'0-->[omega{0}]'0::1;
/*r60*/ omega{0} []'38-->[omega{0}]'38::1;
/*r61*/ [omega{i}-->omega{i+1}]'38::1: 0<=i<=14;
/*r62*/ [omega{15}]'38-->[ro]'38::1;

```

```

*** <Objetos necesarios para el control de densidad> ***
/*re34*/ [ro{i}][j+100] --> [A{i}*Fi{j},AR*2500][j+100]:1: 1<=i<=2,1<=j<=18;
/*r63*/ A{i}[]'0 --> [A{i}]'0::1: 1<=i<=2;
/*r64*/ A{i}[]'36+i --> [A]'{36+i}:1: 1<=i<=2 ;
/*r65*/ AR[]'0 --> [AR{1}]'0::1;
/*r66*/ [AR{i}]'0 --> [AR{i+1}]'0::1:1<=i<=4;
/*r67*/ AR{5}[]'39 -->+[AR]'39::1;
/*r68*/ +[alpha]'39 -->+[alpha{1},DP{0}][39::1;
/*r69*/ VR+[ ]'39 -->+[VR]'39::1;
/*r70*/ +[alpha{1}]'39 -->-[alpha{1}]'39::1;
/*r71*/ +[AR, VR]'39 -->[-] '39::1;
/*r72*/ -[alpha{i}] --> alpha{i+1}]'39::1: 1<=i<=6;
/*r73*/ -[AR*125 --> DP{1}][39::1;
/*r74*/ -[DP{k},DP{k}] --> DP{i+k}]'39::1:0<=i<=20,1<=k<=9;
/*r75*/ -[alpha{7}]'39 --> [beta{1}]'39::1;
/*r76*/ [AR]'39 --> [] '39::1;
/*r77*/ [VR]'39 --> [] '39::1;
/*r78*/ [DP{i}]'39 --> DP{i}*(1250*250000)[ ]'39::1: 0<=i<=20;
/*r79*/ [beta{i}] --> beta{i+1}]'39::1: 1<=i<=7;
/*r80*/ [beta{8}]'39 --> -[beta{9}]'39::1;
/*r81*/ -[beta{9}]'39 --> [alpha]'39::1;
*** <Fin Objetos necesarios para el control de densidad> ***

/*r82*/ -[ro]'{k}-->[eta]'{k}:1: 37<=k<=38;
/*r83*/ -[alpha]'{k} -->[beta{0}]'{k}:1: 37<=k<=38;
/*r84*/ V{s}, A{k} --> [V{s},A]{k}:1: 1<=s<=6, 37<=k<=38;
/*r85*/ Qp{d,m} -[] '37 -->[Q{20-m+d}]'37::1: -20+m<=d<=52-20+m,1<=m<=20;
/*r86*/ Qp{d,m} -[] '37 -->[V{1}]'37::1: 53-20+m<=d<=71-20+m,1<=m<=20;
/*r87*/ Qp{d,m} -[] '38 -->[Q{52-m+d}]'38::1: 0<=d<=m, 21<=m<=40;
/*r88*/ Qp{d,m} -[] '38 -->[V{1}]'38::1: 1+m<=d<=83-52+m, 21<=m<=40;

*** <Mortalidad por exceso de densidad> ***

/*r89*/ [V{s}, A]{k} -->X{s+1} +[] '{k}:0.5*0.2*(s+1) : 2<=s<=5,37<=k<=38;
/*r90*/ [V{s}, A]{k} --> +[V{s},A]{k}:1-0.5*0.2*(s+1) : 2<=s<=5,37<=k<=38;
/*r91*/ +[A]{k}--> [A{0}]{k}:1: 37<=k<=38;
/*r92*/ +[A{s}]{k} -->[A{s+1}]{k}:1:0<=s<=4,37<=k<=38;
/*r93*/ [beta{0}]{k} -->+[beta{1}]{k}:{k}:1: 37<=k<=38;
/*r94*/ +[beta{1}]{k} -->[beta{1}]{k}:{k}:1: 37<=k<=38;
/*r95*/ [beta{i}]{k} -->+[alpha{i+1}]{k}:{k}:1: 1<=i<=6,37<=k<=38;
/*r96*/ +[alpha{i}]{k} -->[beta{i}]{k}:{k}:1: 2<=i<=6,37<=k<=38;
/*r97*/ +[alpha{7}]{k} -->-[] '{k}:1:37<=k<=38;
/*r98*/ [d{d}, A{0}]{k} -->QQ{d,k}+[] 'k : 1:1<=d<=52,37<=k<=38;
/*r99*/ [V{1},A{1}]{k}--> XX{2,k} +[] '{k}:1:37<=k<=38;
/*r100*/ [V{s},A{s}]{k}--> X{s+1} +[] '{k}:1:2<=s<=5,37<=k<=38;
/*r101*/ QQ{d,k}, DP{i} +[] '{k} -->[d],X{1}[]'{k} :: 0.1*i-1:
1<=d<=52, 10<=i<=20,37<=k<=38;
/*r102*/ QQ{d,k}, DP{i} +[] '{k}--> [A{1}]{k} :: 1-(0.1*i-1):
1<=d<=52,10<=i<=20,37<=k<=38;
/*r103*/ XX{2,k}, DP{i} +[] '{k} X{2}[]'{k} :: 0.1*i-1: 10<=i<=20,37<=k<=38;
/*r104*/ XX{2,k}, DP{i} +[] '{k} --> [A{2}]{k} :: 1-(0.1*i-1): 10<=i<=20,37<=k<=38;
/*r105*/ QQ{d,k}, DP{i} +[] '{k}--> [A{1}]{k} :: 1: 1<=d<=52, 0<=i<=9,37<=k<=38;
/*r106*/ XX{2,k}, DP{i} +[] '{k} --> [A{2}]{k} :: 1: 0<=i<=9,37<=k<=38;
/*r107*/ -[eta]'37 -->T{20},Tp{20}[alpha]'37::1;
/*r108*/ -[eta]'38 -->T{0}[alpha]'38::1;
/*r109*/ [Tp{20},Ip]'0--> I2[] '0::1;
/*r110*/ [T{20}][0-->T{20}[] '0::1;
/*r111*/ [T{0}][0-->T{0}[] '0::1;
/*r112*/ Tp{20}+[] '36--> [] '36::1;

*****Update*****
/*r113*/ -[V{s}]{k} -->D[] '{k}:1: 2<=s<=6,37<=k<=38;
/*r114*/ -[Q{d}]{k} -->D [] '{k}:1:1<=d<=87,37<=k<=38;
/*r115*/ -[A{5}]{k}-->[] '{k}:1:37<=k<=38;
/*r116*/ DP{i}[] '39-->[] '39::1: 0<=i<=20;
}

```

En general, no sólo sería muy laborioso proporcionar la gran cantidad de parámetros requeridos por el modelo en el código para cada escenario de estudio, sino que, además, no permitiría una clara distinción de los roles diseñador (experto en sistemas P y la modelización de los fenómenos) y usuario final (encargado de la gestión del ecosistema en cuestión, interesado en proporcionar escenarios de su interés y en analizar la evolución esperada de los mismos en base a simulaciones).

Por ello, una vez traducido el modelo a P-Lingua, se puede emplear la flexibilidad del mecanismo proporcionado por MeCoSim con el fin de evitar la necesidad de desarrollar nuevas herramientas, bastando la configuración de un archivo de hoja de cálculo para tener una aplicación adaptada a sus necesidades en términos de entrada, procesamiento básico de las entradas para generar parámetros y de la computación para producir resultados, y salidas tabuladas y gráficas para presentar los datos al usuario final.

Configuración de una aplicación basada en MeCoSim adaptada al sistema objeto de estudio

El modelo presentado incluye una gran cantidad de parámetros, que deberán generarse a partir de la información proporcionada por el usuario. El primer paso para la definición del fichero de configuración de la aplicación a medida, consiste en la organización de los elementos de la ventana principal. Al margen de la pestaña de depuración (Debug), que siempre se tiene disponible, toda información de entrada o salida se organiza en forma de árbol enraizado de pestañas, donde la raíz es la aplicación, y se van definiendo pestañas a primer nivel (hijas de la aplicación), hijas de éstas, etc. Cada pestaña albergará un conjunto de pestañas hijas, o bien se tratará de una hoja.

- *Estructura visual de la aplicación:* se proporciona en la pestaña TabsHierarchy del archivo de configuración de la Figura 6.8.

Tab Id	Tab Name	Tab Parent Id
1	Zebra Mussel	0
2	Input	1
3	Temperatures	2
4	Medias	3
5	Intervals	3
6	Population	2
7	Biological parameters	2
8	Area properties	2
9	Soil properties	2
10	Human performance	2
11	Inocula	10
12	Movement caused by crafts	10
13	Movement larvae renovation	10
14	Output	1
15	Larvae Medias	14
16	Adults Medias	14
17	Larvae Chart	14
18	Adults Chart	14
19	AdultsPerAge Chart	14

Figura 6.8: Configuración MeCoSim app - Estructura visual

Como muestra la figura 6.8, lo primero que aparece es la aplicación *Zebra Mussel*, con *Tab Id* 1, reconocible por aparecer la primera y tener *Tab Parent Id* a 0. El siguiente nivel serán las pestañas *Input* y *Output* que, como se muestra, tienen a la aplicación como padre (*Tab Parent Id* 1). A partir de ahí, se estructura de forma natural las entradas conforme a grupos funcionales reconocibles por el usuario como son, por un lado, las temperaturas (con hijas diferenciadas para medias e intervalos) y, por otro, las poblaciones, parámetros biológicos, propiedades de las áreas, características de los tipos de suelo y actuaciones llevadas a cabo por los humanos (subdividiéndose en inoculación, movimiento de embarcaciones y régimen de renovación del agua). La estructura de pestañas en forma de árbol de la aplicación configurada queda como se muestra en la figura 6.9.

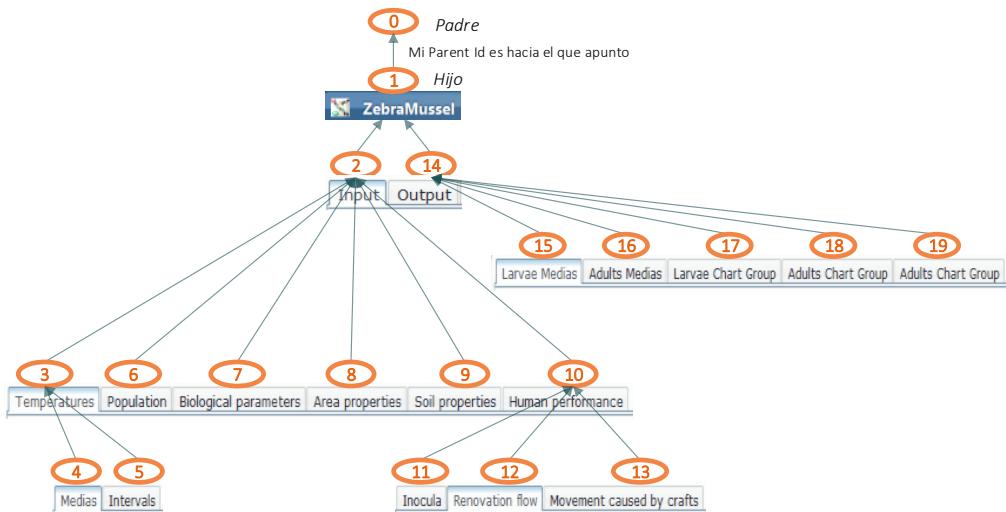


Figura 6.9: MeCoSim app - Árbol de pestañas

Como se puede intuir, esta estructura se puede cambiar de forma muy sencilla, simplemente reestructurándola de acuerdo con los niveles de organización más natural para el usuario en cada caso.

Para las salidas se han dejado 5 pestañas hoja, todas hijas de *Output*, si bien igualmente se podría haber considerado pestañas diferenciando por ejemplo *Larvas* y *Adultos*, y dentro las hojas correspondientes a cada grupo, o bien entre *Salidas tabuladas* y *Gráficos*. Bastará con incluir los niveles correspondientes y hacer que las hojas pasen a depender de los nuevos niveles introducidos.

- *Tablas de entrada y de salida:*

Naturalmente, ningún nivel de organización como el anterior sería interesante si, finalmente, no tuviéramos información con la que trabajar. En MeCoSim la información se introduce en forma de tablas por lo que, para definir la estructura correspondiente, habrá que indicar qué tablas queremos que aparezcan en la aplicación, en qué pestaña deseamos que aparezcan, así como el número de filas y columnas que deben presentar. La figura 6.10 muestra la definición de las tablas en el fichero de configuración para la aplicación *Zebra Mussel*.

Table Id	Table Name	Tab Id	Columns	Init Rows	Save To File	Input / Output	Output Graphic	External View	Row Addition	Row Deletion
1	Medias	4	20	36	VERDADERO	Entrada				
2	Intervals	5	5	14	VERDADERO	Entrada				
3	Population	6	2	18	VERDADERO	Entrada				
4	Biological parameters	7	3	9	VERDADERO	Entrada				
5	Area properties	8	9	18	VERDADERO	Entrada				
6	Soil properties	9	2	4	VERDADERO	Entrada				
7	Inocult	11	3	18	VERDADERO	Entrada				
8	Renovation flow	12	4	50	VERDADERO	Entrada				
9	Movement caused by crafts	13	19	18	VERDADERO	Entrada				
10	Larvae Medias	15	5	100	VERDADERO	Salida				
11	Adults Medias	16	6	100	VERDADERO	Salida				
12	Larvae Chart	17	4	1	VERDADERO	Salida	MultiLineChart	FALSO		
13	Adults Chart	18	4	1	VERDADERO	Salida	MultiLineChart	FALSO		
14	Adults Chart	19	4	1	VERDADERO	Salida	MultiBarChart	FALSO		

Figura 6.10: Configuración MeCoSim app - Tablas

Entre la información de las tablas, un dato fundamental es *Tab Id*, el cual nos indica en qué pestaña de la jerarquía definida en el árbol anterior se situará la tabla correspondiente. Este *Tab Id* deberá coincidir necesariamente con el de alguna pestaña hoja. Así, por ejemplo, se puede observar en la figura 6.10 que la tabla *Area properties* está asociada a la pestaña con *Tab Id* 8 que, como no es de extrañar, pudimos ver en la figura 6.9 que se trata de la pestaña titulada igualmente *Area properties*.

Ahora bien, la configuración anterior no nos dice qué cabecera ha de tener cada una de las tablas, lo que se define de forma separada. La definición de las columnas de la tabla *Area properties* se presenta en la figura 6.11, incluyendo un identificador *id* para cada columna, su nombre, un posible valor por defecto, el indicador de si los datos de la columna serán editables y un *tooltip* que se desplegará cuando estemos situados con el ratón sobre algún dato de la columna. Por ejemplo, la tabla de propiedades del área contendrá los datos de zona (el nombre con que los gestores del embalse, Endesa S. A., tienen catalogada cada área), la anchura, longitud, profundidad, capacidad, y el porcentaje de cada posible tipo de suelo que tiene el área (éstos se presentan en la tabla 6.2). Como se ha explicado al inicio de este capítulo, existen 4 tipos de suelo con distintas tipologías de características, afectando a la capacidad de fijación y a la mortalidad de los mejillones.

Column Id	Column Name	Default Value	Editable	Tooltip
1	Zone		VERDADERO	My Zone is
2	Anchura (m)		VERDADERO	Anchura(m)
3	Length (m)		VERDADERO	Long (m)
4	Depth (m)		VERDADERO	Depth (m)
5	Capacity (m3)		VERDADERO	Capacity (m3)
6	Soil Type 1		VERDADERO	Soil Type 1
7	Soil Type 2		VERDADERO	Soil Type 2
8	Soil Type 3		VERDADERO	Soil Type 3
9	Soil Type 4		VERDADERO	Soil Type 4

Figura 6.11: Configuración MeCoSim app - Definición de tabla

- *Parámetros para instanciar el sistema:*

A partir de la información de las tablas de entrada deberán generarse los parámetros para el modelo presentado. Para ello, es necesario especificar cómo generar cada parámetro, de acuerdo con lo detallado en la sección 4.2.1, donde se aporta la descripción tanto del mecanismo como de la gramática del lenguaje aceptado para la obtención de los parámetros. En el caso que nos ocupa, existen muchas tablas de entrada y muchos parámetros a generar. En la figura 6.12 se muestra un subconjunto de los parámetros.

Param Name	Param Value	Index 1	Index 2	Index 3	Index 4	Comments
T	<1,\$1,\$2\$+>	[1..36]	[1..18]			
IT	<2,1,>	1				
IT	<2,12,>	2				
PI	<2,\$1,\$4>			[1..11]	1	
PI	<2,11+\$1,\$4>			[1..3]	2	
mo	<2,1,>			1		
mo	<2,12,>			2		
N	<3,\$1,\$2>				[1..18]	
q	N(\$1\$)/7				[1..18]	
p	1-<@ncdf,T(\$1,\$2\$),0.5,IT(1)>				[1..20]	[1..18]
p	<@ncdf,T(\$1,\$2\$),0.5,IT(2)>				[21..36]	[1..18]
g	<4,\$1,\$3>			1	[1..3]	
m	<4,\$1+\$3,>				[1..6]	
SURF	<5,\$1,\$2>				[1..18]	
A2	SURF(\$1\$-100)/2				[101..118]	
L	<5,\$1,\$3>				[1..18]	
DS	<5,\$1,\$4>				[1..18]	
C	<5,\$1,\$5>				[1..18]	
S1	<5,\$1,\$6>				[1..18]	
S2	<5,\$1,\$7>				[1..18]	
S3	<5,\$1,\$8>				[1..18]	
S4	<5,\$1,\$9>				[1..18]	
AS	<6,\$1,\$2>				[1..4]	
FI	(S1(\$1\$)*AS(1)+S2(\$1\$)*AS(2)+S3(\$1\$)*AS(3)+S4(\$1\$)*AS(4))/100				[1..18]	
LE	<7,\$1,\$2\$+>				[1..18]	[1..2]
PA	<9,\$1,\$2\$+>				[1..18]	[1..18]
N	<@func,floor,<@r.5>/2.0->					
M	<@func,floor,<@r.3>>					

Figura 6.12: Configuración MeCoSim app - Parámetros

Recordemos, por ejemplo, las reglas del modelo que desencadenan la activación del primer ciclo reproductivo. Dichas reglas están referenciadas como $r_{e_{3,j,d}}$ y $r_{4_{3,j,d}}$, para $1 \leq j \leq 18$, $0 \leq d \leq 19$ (las reglas llegan a $d \leq 34$ porque abarcan también el segundo ciclo, pero nos centraremos en el primero). La ejecución en un instante determinado de una regla u otra se verá influida por su probabilidad, que depende de $p(d + 1, j)$. ¿Cómo se obtiene esta probabilidad? Parece claro que si la activación del ciclo reproductivo dependía de la temperatura, estas probabilidades $p(d + 1, j)$ tengan que ver con ella. Como se establece en la figura 6.12, el esquema paramétrico de nombre T presenta dos índices: *Index 1*, en el rango de 1 a 36, e *Index 2*, en el rango de 1 a 18. Por tanto, se generarán parámetros que van desde $T_{1,1}$ a $T_{36,18}$.

¿Cómo se calcula? Como indica el valor del parámetro, `<1,1, 2+$2>`, para cada parámetro $T_{d,j}$ tomará el valor de la tabla de id 1 (*Medias*, como se especificó en la configuración mostrada en la figura 6.10), para la celda de la fila `1` (es decir, el que tome en cada momento el índice 1, d), columna `2+$2`, ya que las dos primeras columnas indican las fechas y el número de semana respectivamente, de modo que para el primer entorno ($j = 1$), tomaremos los datos de la columna 3 (`2+$2`, o lo que es lo mismo $j + 2$).

La tabla de temperaturas medias con unos posibles valores de prueba a partir de los que obtener los parámetros correspondientes se muestra en la figura 6.13.

Figura 6.13: MeCoSim app - Temperaturas medias por semana

Hasta este momento tenemos parámetros de tipo $T\{d, j\}$, pero se necesitaban del tipo $T\{d+1, j\}$. ¿Cómo se generan dichos parámetros? Como se indica para los parámetros de nombre p , su valor es calculado para el primer ciclo reproductivo (rango para el primer índice de 1 a 20) como:

1-<@ncdf,T{\$1\$,\$2\$},0.5,IT{1}>.

Por tanto, dependerá de los valores de los ya referidos $T\{d, j\}$, además del obviado $IT\{1\}$ que, como se puede observar en la figura 6.10, es directamente el valor de la tabla 2 (*Intervals*), fila 1, columna 2. Ese valor establece el umbral a partir del cual sería viable la reproducción, de tal modo que, en función de la temperatura introducida por el usuario para cada semana en el escenario objeto de estudio, se aplicará la probabilidad que representa la temperatura media respecto a la normal centrada en el intervalo indicado por la fórmula anterior para el parámetro.

En algunos casos especiales necesitaremos funcionalidades demasiado específicas para que sean proporcionadas como funciones básicas en un lenguaje. En el caso del sistema objeto de estudio, resulta que el proceso de renovación de agua sigue los principios de la dinámica de fluidos. Esta funcionalidad no

se proporcionaba por defecto en MeCoSim, pero sus mecanismos de extensibilidad permiten definir nuevas funciones que puedan ser llamadas durante la generación de parámetros, como aparece en la figura 6.14.

Dinf	<8,\$1\$3>	[1..M]		
Cinf1	<@func,caudal,1,Dinf(\$1\$)>	[1..M]	0	[0..N+1,N+1]
Cinf2	<@func,caudal,2,Dinf(\$1\$)>	[1..M]	0	[0..N+1,N+1]
Cinf3	<@func,caudal,3,Dinf(\$1\$)>	[1..M]	0	[0..N+1,N+1]
Cinf4	<@func,caudal,4,Dinf(\$1\$)>	[1..M]	0	[0..N+1,N+1]
Cinf1	<5,N+\$3\$5>	[1..M]	0	[1..N]
Cinf2	<5,N+\$3\$5>	[1..M]	0	[1..N]
Cinf3	<5,N+\$3\$5>	[1..M]	0	[1..N]
Cinf4	<5,N+\$3\$5>	[1..M]	0	[1..N]
Cargainf1	<@func,renovacion,0,N,M,N+1,\$1\$,\$2\$5,Cinf1(\$1\$,\$0,\$2\$5)>	[1..M]	[1..N]	
Praux1	<@func,renovacion,1,N,M,N+1,<@func,caudal,1,Dinf(\$1\$)>,Cinf1(\$1\$,\$0,\$2\$5-(N+1)),\$1\$,\$2\$5-(N+1),\$3\$-(N+1)>	[1..M]	[N+1..2*N+1]	[N+2..2*N+2]
Cargainf2	<@func,renovacion,0,N,M,N+1,\$1\$,\$2\$5,Cinf2(\$1\$,\$0,\$2\$5)>	[1..M]	[1..N]	
Praux2	<@func,renovacion,1,N,M,N+1,<@func,caudal,2,Dinf(\$1\$)>,Cinf2(\$1\$,\$0,\$2\$5-(N+1)),\$1\$,\$2\$5-(N+1),\$3\$-(N+1)>	[1..M]	[N+1..2*N+1]	[N+2..2*N+2]
Cargainf3	<@func,renovacion,0,N,M,N+1,\$1\$,\$2\$5,Cinf3(\$1\$,\$0,\$2\$5)>	[1..M]	[1..N]	
Praux3	<@func,renovacion,1,N,M,N+1,<@func,caudal,3,Dinf(\$1\$)>,Cinf3(\$1\$,\$0,\$2\$5-(N+1)),\$1\$,\$2\$5-(N+1),\$3\$-(N+1)>	[1..M]	[N+1..2*N+1]	[N+2..2*N+2]
Cargainf4	<@func,renovacion,0,N,M,N+1,\$1\$,\$2\$5,Cinf4(\$1\$,\$0,\$2\$5)>	[1..M]	[1..N]	
Praux4	<@func,renovacion,1,N,M,N+1,<@func,caudal,4,Dinf(\$1\$)>,Cinf4(\$1\$,\$0,\$2\$5-(N+1)),\$1\$,\$2\$5-(N+1),\$3\$-(N+1)>	[1..M]	[N+1..2*N+1]	[N+2..2*N+2]
PR	(Praux1(\$1,\$2\$5+\$3\$5+1)*0.1)+(Praux2(\$1,\$2\$5+\$3\$5+1)*0.3)+(Praux3(\$1,\$2\$5+\$3\$5+1)*0.3)+(Praux4(\$1,\$2\$5+\$3\$5+1)*0.1)	[1..M]	[N+1..2*N]	[N+1..2*N+1]

Figura 6.14: Configuración MeCoSim app - Parámetros avanzados

Los detalles del algoritmo quedan fuera del objeto de esta tesis y, por tanto, serán omitidos. No obstante, el algoritmo permite ilustrar los mecanismos de extensibilidad del lenguaje de generación de parámetros, cuando la funcionalidad proporcionada de propósito general no sea suficiente para resolver las necesidades de los usuarios, o bien vengan impuestas por la complejidad de los modelos bajo estudio y sus procesos subyacentes.

- *Configuración de resultados de la simulación:*

Los parámetros calculados, junto con el modelo cargado, permiten generar un sistema PDP concreto y simular el periodo establecido por el usuario. Un diseñador de sistemas P podría analizar la simulación paso a paso mediante su depuración, estudiando la evolución de los objetos presentes en la estructura interna del sistema. No obstante, lo fundamental de este trabajo es servir a los usuarios finales, gestores del embalse, como herramienta útil para su gestión. Por tanto, lo interesante será trabajar a nivel del problema tratado, ajeno a los sistemas PDP subyacentes y centrados en el dominio del problema. Para dar el salto entre el resultado de la computación y la visualización de la información en el dominio del usuario, será necesario definir para la aplicación *Zebra mussel* qué salidas mostrar.

En la sección 4.2.2 se describió la información almacenada en una base de datos empleada internamente por MeCoSim durante la computación, incorporando para cada configuración del sistema, la multiplicidad de cada objeto en cada membrana, célula o neurona según el tipo de sistema. En su caso, dentro de cada entorno y para cada simulación (ya se ha comentado que en estos sistemas probabilísticos es necesario realizar un cierto número de simulaciones bajo el mismo escenario, para obtener medias y desviaciones), por cada ciclo (un año para el caso del mejillón cebra) y cada paso del ciclo.

Asimismo, será necesario establecer para cada salida cuál es la información que se desea mostrar al usuario. Por ejemplo, en el caso que nos ocupa, el usuario requiere una salida tabulada con la media y desviación del número de individuos adultos por cada edad (en años), en cada área del embalse y por cada ciclo reproductivo de cada año.

En la figura 6.15 se muestra la definición de un resultado a partir de la computación, tomando (*Select*) los datos: simulación (*simulation*), año (*cycle*), ciclo reproductivo (calculado en función del paso, como la parte entera del paso –step– dividido entre 51 pasos que dura el ciclo, distinguiendo así ciclos 1 y 2), entorno (*environmentID*), edad (extraída del nombre del objeto X_s , almacenado como $X\{s\}$, extrayendo la s que representa la edad del mejillón en semestres mediante funciones de tratamiento de texto sobre la información almacenada) y suma de la multiplicidad de los elementos seleccionados correspondientes agrupados según todos los criterios anteriores. Solamente se podrá llevar a cabo la obtención de estos datos para los objetos que comienzan por $X\{$ y en el paso de la computación en que se tengan los objetos requeridos en la configuración (los múltiplos de 51, pasos que dura el ciclo); ambas condiciones se aportan como condiciones auxiliares (*WhereAux*) que deben ser satisfechas y de forma conjunta (como indica el AND de la línea correspondiente a la parte *Where*).

Criteria Id	Select/Where/Group	Criteria	Formula	Referred Criteria Id	Argument	Qualified Name	Where/Select condition type	Where condition
1	Select	simulation						
2	Select	cycle						
3	Auxiliary	step						
4	Auxiliary	formula	CONVERT		3 INT 4	51 round		
5	Select	formula	/					
6	Select	environmentID						
7	Auxiliary	object						
8	Auxiliary	formula	LEFT		7	3		
9	Select	formula	SUBSTR		8	3 age		
10	Auxiliary	multiplicity						
11	Select	formula	SUM		10	mult		
12	WhereAux	object						
13	Auxiliary	step						
14	WhereAux	formula	MOD		13	51		
15	Where	formula	AND		12	14		
16	Group	simulation						
17	Group	cycle						
18	Group	round						
19	Group	environmentID						
20	Group	age						

Figura 6.15: Configuración MeCoSim app - Resultado auxiliar

La salida anterior mostraría los resultados de cada simulación individual pero, teniendo presente que estamos interesados en mostrar medias y desviaciones sobre lo anterior, se necesita configurar el resultado final a mostrar (ver figura 6.16) como un resultado dependiente del anterior, obteniendo media (*AVG*) y desviación poblacional (*STDDEV_POP*) de la suma de multiplicidades anteriormente calculada, excluyendo ahora el dato *simulation* de los datos de salida (*Select*) y su agrupación (*Group*), para obtener el dato de multiplicidad media y desviación sobre el conjunto de simulaciones.

Criteria Id	Select/Where/Group	Criteria	Formula	Referred Criteria Id	Argument	Qualified Name
1	Select	cycle				
2	Select	round				
3	Select	environmentID				
4	Select	age				
5	Auxiliary	mult				
6	Auxiliary	formula	AVG	5		
7	Select	formula	ROUND	6	0 media	
8	Auxiliary	formula	STDDEV_POP	5		
9	Select	formula	ROUND	8	5 deviation	
10	Group	cycle				
11	Group	round				
12	Group	environmentID				
13	Group	age				

Figura 6.16: Configuración MeCoSim app - Resultado a mostrar

Empleo del entorno de simulación

Una vez configurada la aplicación a medida, el diseñador de sistemas P puede inicialmente cargar el modelo para el escenario bajo estudio, depurar el modelo, comprobando que se carga correctamente y chequeando que la evolución del sistema PDP asociado es acorde con lo esperado, y analizar paso a paso la evolución de los objetos y reglas implicados.

Así, para un escenario dado y el modelo del mejillón cebra cargado, el primer paso para la depuración es la inicialización. Ésta conlleva la generación de los parámetros para el modelo a partir de los datos del escenario, el reconocimiento léxico-sintáctico del modelo P-Lingua junto con los valores de los parámetros, y la construcción de la estructura necesaria para representar el sistema PDP correspondiente y establecer su configuración inicial. En este punto, el proceso computacionalmente más costoso es el despliegue de las reglas del sistema a partir de los esquemas de reglas escritos en P-Lingua, generando las reglas finales de cada membrana de cada entorno a partir de los esquemas de reglas con iteradores. El resultado de este proceso de inicialización se puede ver en la pestaña de *Parsing Info* dentro de la pestaña de depuración (*Debug*). En ella aparecerá cada regla con un identificador y su probabilidad asociada, como se muestra en la figura 6.17. Al final de los mensajes de esta pestaña aparecerán además datos de resumen del reconocimiento, como el tamaño (número de símbolos) del alfabeto o el número total de membranas (contabilizando cada membrana de cada entorno de forma separada, aunque éstas establezcan la misma estructura esqueleto para cada entorno).

Es posible que puedan aparecer errores durante la generación de parámetros o bien durante el proceso llevado a cabo por el reconocedor léxico-sintáctico, bien sea por errores en el lenguaje o debido, por ejemplo, al intento de acceder a parámetros no definidos. En esos casos se mostrarán en la pestaña *Errors*. Además, se mostrarán en la pestaña *Warnings* avisos que no constituyan un problema para la carga y simulación del sistema pero que puedan alertar al diseñador de alguna circunstancia especial.

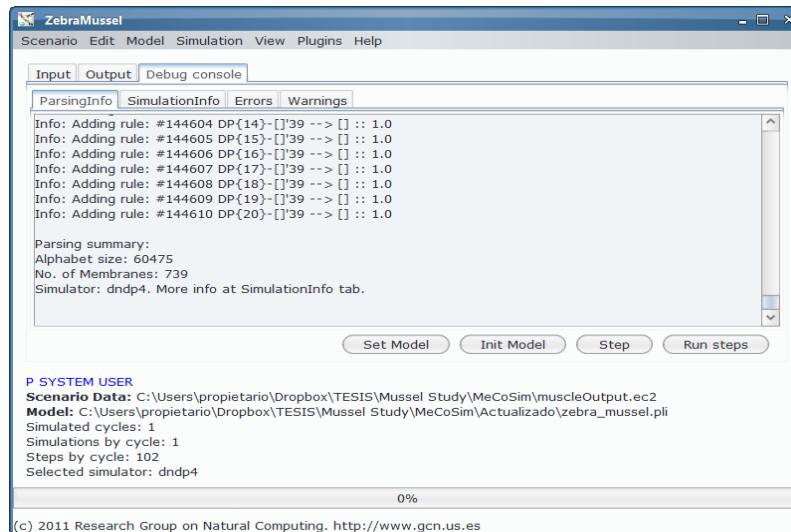


Figura 6.17: MeCoSim app - Depuración - Parsing

Por ejemplo, es frecuente que, en función de los datos del escenario concreto introducido, muchas de las reglas cuya probabilidad dependan de parámetros pueden tener probabilidad 0, como se muestra en la figura 6.18. Estas reglas no son añadidas al sistema instanciado, y esta circunstancia sería advertida de forma que se pueda estudiar si es correcto a partir de los datos del escenario introducido, o bien si existe algún error en el archivo del modelo.

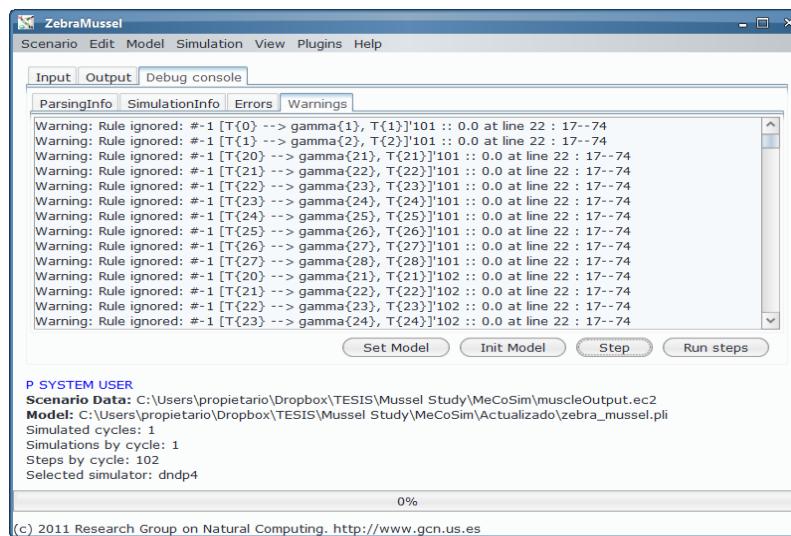


Figura 6.18: MeCoSim app - Depuración - Avisos

Si el modelo inicializado a partir del archivo P-Lingua y el escenario introducido carece de errores, entonces se habilitan de forma automática los botones *Step* y *Run steps* de la ventana de depuración, de tal manera que se pueda simular el sistema paso a paso o bien un número de pasos, e ir observando en la pestaña *Simulation Info* los datos detallados en modo texto de cada configuración, así como de la aplicación de cada regla sobre cada entorno, como se muestra en la figura 6.19.

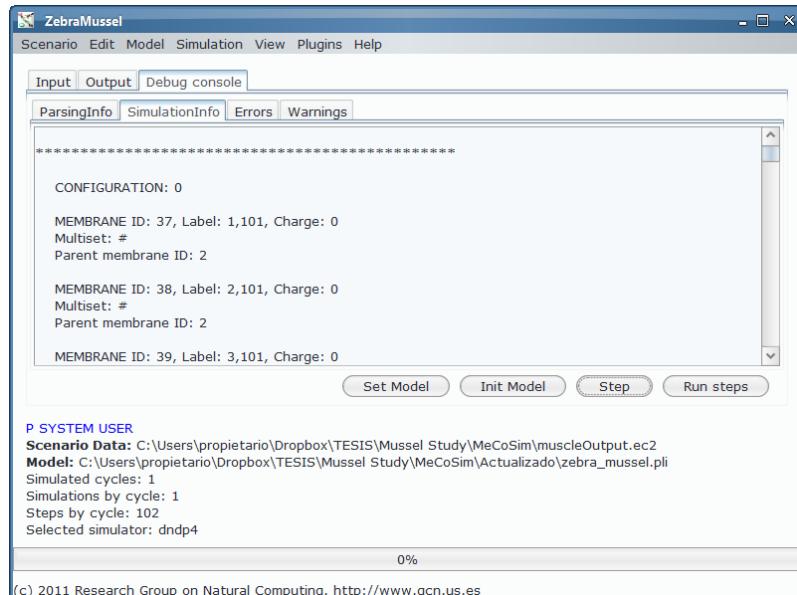


Figura 6.19: MeCoSim app - Simulación paso a paso

Además, en el menú *View* se dispone de varios visores de elementos de la configuración actual del sistema. Por ejemplo, como se puede apreciar en la figura 6.20, el visor del alfabeto muestra los objetos del alfabeto del sistema.

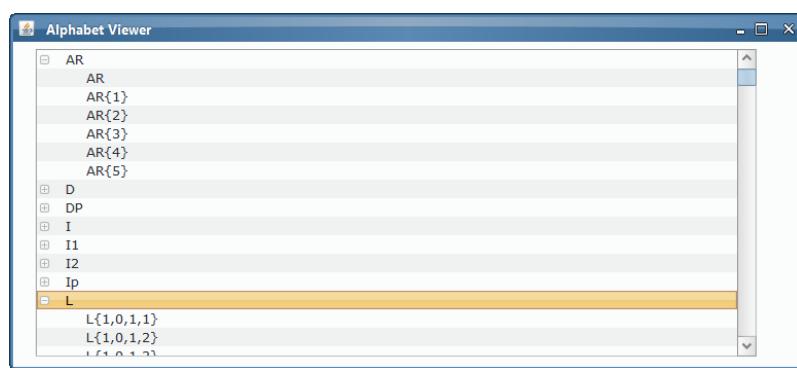


Figura 6.20: MeCoSim app - Ver alfabeto

Como vemos, en este visor se organiza los objetos en forma de árbol, ordenado sintácticamente e incluyendo como hijos de cada elemento a todos aquellos objetos que presenten distintos subíndices para un mismo tipo de objeto (como se muestra en la figura, al desplegar AR vemos que existen objetos AR , AR_1 , AR_2 , AR_3 , AR_4 y AR_5). El alfabeto mostrado no se corresponde exactamente con los alfabetos definidos en los diversos tipos de sistemas P ni diferencia objetos de los propios sistemas P y del entorno, sino que trata de recopilar todos aquellos símbolos que aparecen en los multiconjuntos que intervienen en las reglas del sistema (bien del conjunto \mathcal{R} o del conjunto \mathcal{R}_E), junto con aquellos que aparecen en los multiconjuntos iniciales.

Por su parte, el visor de estructura de membranas permite visualizar los diversos entornos y las distintas membranas que existen dentro de cada entorno, incluyendo sus etiquetas y la de su entorno, estructurados de forma jerárquica, como se muestra en la figura 6.21.

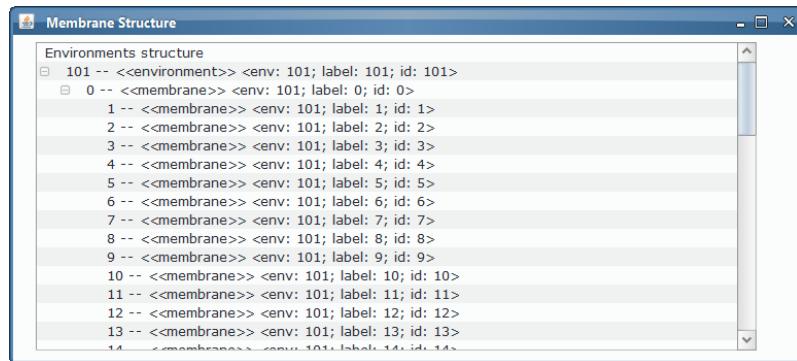


Figura 6.21: MeCoSim app - Estructura de membranas

El último visor es el de los multiconjuntos y, como puede apreciar en la figura 6.22, permite visualizar tanto la estructura como los multiconjuntos presentes en la configuración actual del sistema PDP en cada membrana de cada entorno.

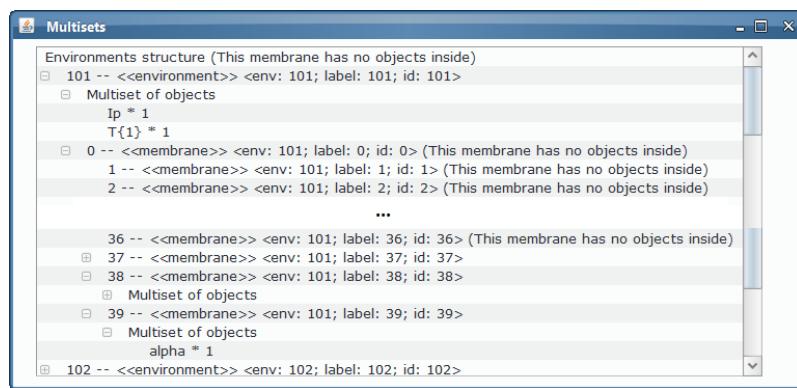


Figura 6.22: MeCoSim app - Multiconjuntos

Sobre el modelo ya depurado, se puede proceder a la validación experimental del mismo, de acuerdo con el criterio de los expertos o contrastando las salidas de las simulaciones con datos reales sobre los mismos escenarios de interés.

Así, para una situación concreta que se desee analizar y de la que se disponga de datos o bien sea susceptible de ser contrastada con los expertos, habría que proporcionar al sistema todos los datos del escenario que se necesiten con el fin de poder estudiar la dinámica del modelo y contrastar si la salida coincide con lo esperado.

Entre los datos de entrada habrá que proporcionar, naturalmente, la población de mejillones presentes en el embalse en el momento inicial a estudiar, lo que dará lugar a unos ciertos multiconjuntos iniciales de objetos X_s y Q_d presentes en cada área del embalse y, por ende, en cada entorno del sistema PDP, generándose con diferentes edades a partir de la información de la población total de individuos por metro cuadrado (ind/m^2) y por área, como se muestra en la figura 6.23.

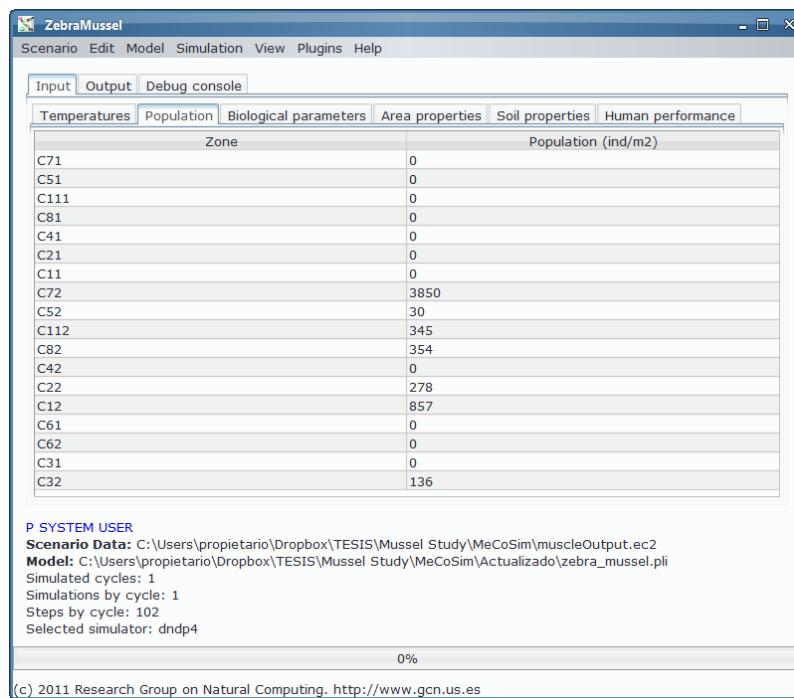


Figura 6.23: MeCoSim app - Población inicial

Además, como se puede visualizar en la figura 6.24, será necesario establecer las condiciones propias del entorno, como son las propiedades de los tipos de suelo o las propiedades del área.

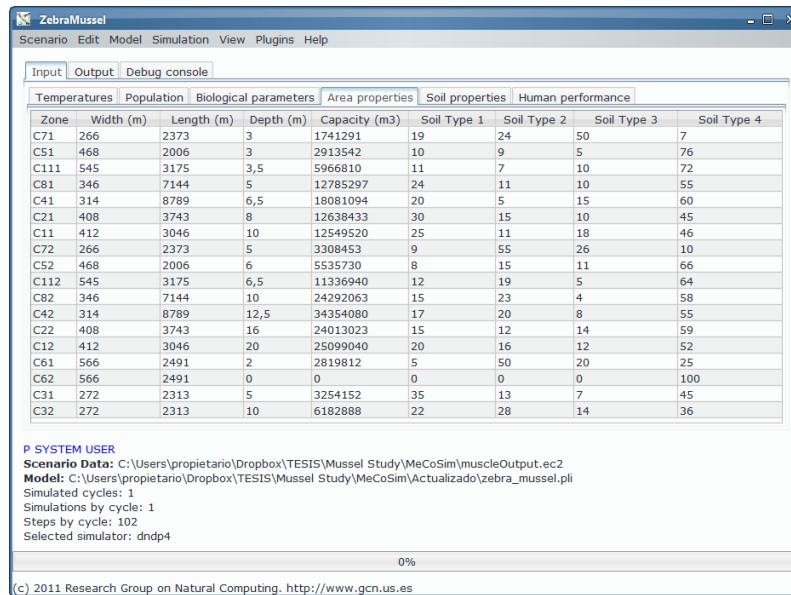


Figura 6.24: MeCoSim app - Propiedades del área

Si el modelo ya ha sido validado por los expertos en la gestión del ecosistema objeto de estudio, el usuario ya podrá utilizar la aplicación para la realización de experimentos virtuales a través de las simulaciones del modelo descrito. Para ello, deberá proporcionar datos de los escenarios de su interés a través de las entrada en forma de tablas, lanzará simulaciones y se visualizarán los resultados de acuerdo con la configuración de los mismos.

Por ejemplo, para los gestores del embalse podría ser interesante analizar lo que sucedería en el caso de una inoculación externa de larvas, una entrada de mejillones por el movimiento de embarcaciones, una renovación del agua más constante debido a un periodo prolongado de abundantes lluvias, o bien por cualquier otra situación pasada (para formular ciertas hipótesis sobre la aparición de mejillones cebra en el embalse) o futura (para plantear posibles escenarios o actuaciones potenciales que se puedan dar en algún momento). A partir de los datos de entrada introducidos, se proporcionan a los gestores los resultados de las simulaciones en función de la configuración establecida en la definición de la aplicación adaptada para el mejillón cebra como, por ejemplo, la salida en forma tabulada que recoge los ejemplares medios por área en una determinada unidad determinada por el usuario, como aparece en la figura 6.25.

En ocasiones será más sencillo analizar la evolución cualitativa de la especie, en lugar del número exacto. Para esos casos, resultará más informativo algún tipo de gráfica de evolución, igualmente configurable a través de la aplicación como, por ejemplo, la que se muestra en la figura 6.26 y que permite estudiar la evolución de la cantidad de larvas en una determinada zona del embalse.

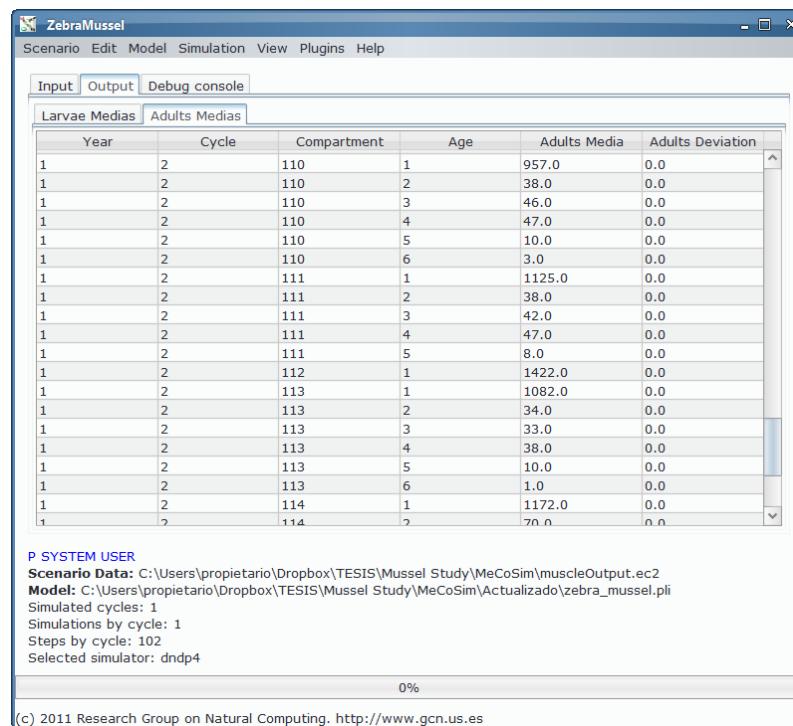


Figura 6.25: MeCoSim app - Salida Adultos

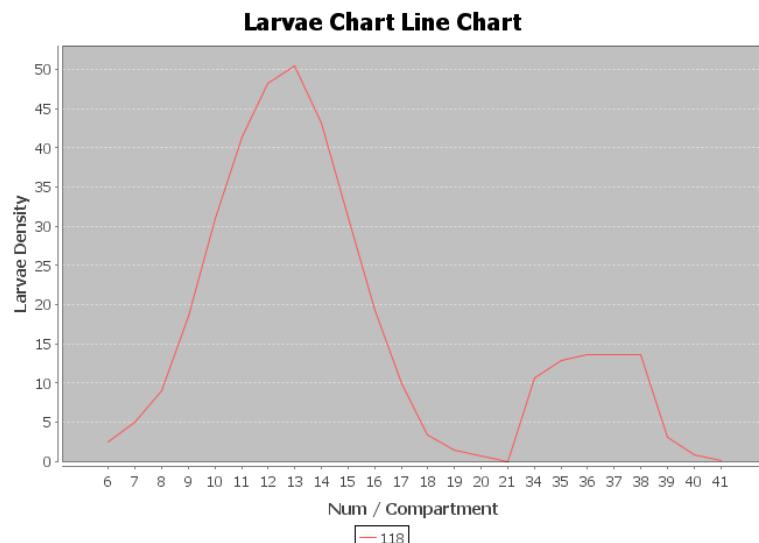


Figura 6.26: MeCoSim app - Salida Gráfico Larvas

7

Caso de estudio 2: Resolución de problemas NP-completos

A lo largo del capítulo 1 se han descrito diversas variantes de sistemas P que proporcionan modelos de computación *eficientes*, en el sentido de ser capaces de resolver problemas computacionalmente duros en tiempo polinomial, acorde con la semántica de los respectivos modelos. En particular, se han presentado soluciones eficientes de 3-COL y SAT que son dos problemas NP-completos bien conocidos.

El entorno de simulación suministrado por MeCoSim tuvo su origen en la necesidad de proporcionar herramientas de propósito general que pudieran ser adaptadas para llevar a cabo procesos de validación experimental de modelos computacionales de ecosistemas reales, desarrollados en el marco de la computación celular con membranas, así como de la implementación de experimentos virtuales en los modelos diseñados.

Sin embargo, se trató de dotar a los mecanismos desarrollados de un carácter más generalista con el fin de permitir la entrada de datos tabulados por parte de los usuarios, la generación de parámetros de los modelos a partir de dichos datos, la depuración, análisis y visualización de la estructura de los sistemas P, así como la simulación y generación de resultados visuales tabulados o gráficos para el usuario.

El enfoque anterior, junto con la disponibilidad del marco **P-Lingua**, con el amplio espectro de variantes de sistemas P cubiertos en cuanto a los procesos de reconocimiento léxico-sintáctico (*parsing*) y simulación, motivó la extensión del objetivo inicial de **MeCoSim** con el fin de que sirviera de entorno visual para trabajar, no solamente con sistemas P de tipo probabilístico (que pasarían a denominarse PDP, *Population Dynamics P systems*), sino con todo tipo de variantes.

De esta manera, **MeCoSim** proporcionaría, de forma general, un IDE (*Integrated Development Environment*) basado en las capacidades como motor de parsing y simulación de **P-Lingua** y las capacidades de diseño de aplicaciones de simulación a medida del núcleo de **MeCoSim**. Así se permitiría, de forma conjunta, la introducción de datos, carga de modelos, generación de parámetros para instanciar los sistemas P, depuración, parsing, análisis visual de alfabetos, estructura subyacente y multiconjuntos contenidos en cada componente básica (membrana/célula/neurona/entorno), almacenamiento de resultados de simulación, diseño de salidas tabuladas y gráficas, extracción de invariantes y verificación.

El objetivo de este capítulo consiste en estudiar las diferentes funcionalidades que aporta la plataforma **MeCoSim**, en conjunción con **P-Lingua**, para la depuración y el análisis de propiedades de los diseños de soluciones de problemas de decisión en el marco de los sistemas P. Específicamente, nos centraremos en los problemas **3-COL** y **SAT**, así como en soluciones de estos problemas proporcionadas por familias de variantes específicas de sistemas P que, como se ha comentado, han sido descritas en el capítulo 1.

El capítulo está estructurado en dos secciones, cada una de las cuales analiza uno de los problemas antes citado.

La primera de ellas está dedicada al problema **3-COL** y se estudian aplicaciones de **MeCoSim** centradas en dos aspectos fundamentales: (a) la generación (automática) de invariantes que justifican la confluencia de los sistemas de una familia de sistemas P que resuelve el problema específico de generación de todas las posibles coloraciones con tres colores en un grafo no dirigido; y (b) la depuración y el análisis de sendas soluciones del problema **3-COL** usando, por una parte, una familia de sistemas P de tejidos con reglas de división celular y, por otra, una familia de *simple kernel P systems*.

En la segunda sección de este capítulo, se ilustran aplicaciones basadas en **MeCoSim** para el análisis de una solución del problema **SAT** mediante una familia de sistemas P con membranas activas.

7.1. El problema 3-COL en MeCoSim

En la Sección 1.4.3 del capítulo 1 se ha descrito el problema 3-COL, así como una solución eficiente del mismo mediante una familia

$$\Pi = \{\Pi(\langle n, m \rangle) : n, m \in \mathbb{N}\}$$

de sistemas P de tejidos con reglas de división celular. Recuérdese que dado un grafo no dirigido concreto \mathcal{G} con n nodos y m aristas, el sistema de la familia que lo procesa será $\Pi(\langle n, m \rangle)$ con multiconjunto de entrada $\text{cod}(\mathcal{G})$ (que representará el conjunto de las aristas del grafo).

La solución antes mencionada resuelve un problema abstracto. Con el fin de manejar esta solución de forma práctica, para instancias concretas del problema, sería interesante definir una aplicación, en nuestro caso basada en **MeCoSim**, que pudiera ser usada tanto por el diseñador de sistemas P que desee analizar a fondo su modelo, asistido por herramientas automáticas, como por el usuario final que desee saber, por ejemplo, si existe una coloración válida con tres colores de un determinado grafo no dirigido.

En el siguiente apartado 7.1.1 nos vamos a centrar en un aspecto parcial del problema 3-COL, específicamente en la fase de generación de todas las posibles coloraciones de un grafo con tres colores, y la aportación de **MeCoSim** y la metodología propuesta para el diseñador de sistemas P.

7.1.1. El problema de la generación de coloraciones

En la sección 5.3.1 se ha descrito una metodología para la modelización, simulación y verificación (mediante extracción de invariantes y técnicas de model checking), ilustrada por el gráfico de la figura 5.1. En dicha metodología, **MeCoSim** juega un papel crucial a través de la integración de las diversas herramientas involucradas en el proceso. Así, en la sección 4.2.3 se describe un plugin para la integración de **MeCoSim** con el software de extracción de invariantes Daikon, generándose ficheros de traza a partir de la simulación en **MeCoSim** y la configuración de las extracciones a medida correspondientes, y llamándose al plugin de Daikon para **MeCoSim** sobre el fichero generado que determine el usuario.

Además, la sección 4.3 incluye un apartado de *verificación de propiedades* que integra el entorno con Spin Model Checker a través de dos posibles mecanismos: la simulación directa en **MeCoSim** mediante un simulador que delega en el proceso externo llevado a cabo por Spin, o bien mediante un plugin que presenta una pantalla para ejecutar paso a paso cada etapa del proceso (en primer lugar, la generación de un fichero Promela a partir del sistema cargado en **MeCoSim** y, a continuación,

la simulación delegando en Spin desde **MeCoSim**, mostrando los resultados en otro panel de la ventana). Lógicamente, el fichero generado automáticamente en **MeCoSim** podría ser tratado manualmente para incorporar las reglas de chequeo deseadas para su verificación durante la simulación con Spin.

En el trabajo [87] se presentó, por primera vez, esta metodología así como la integración de las herramientas implicadas. Además, se analizó la problemática relativa a la fase de generación de todas las coloraciones con tres colores en un grafo no dirigido. La correspondiente solución a ese problema parcial sería una parte de la solución del problema 3-COL descrito en la sección 1.4.3. La traducción directa a **P-Lingua** es la siguiente:

```
@model<tissue_psystems>
def main()
{
    call init_cells();
    call init_multisets(n);
    call init_rules(n);
}
def init_cells()
{
    @mu = [[]'2]'0;
}
def init_multisets(n)
{
    @ms(2) += A{i} : 1<=i<=n;
}
def init_rules(n)
{
    /* r1 */ [A{i}]'2 --> [R{i}]'2 [T{i}]'2 : 1<=i<=n;
    /* r2 */ [T{i}]'2 --> [B{i}]'2 [G{i}]'2 : 1<=i<=n;
}
```

Al igual que sucedía en el caso general para el problema 3-COL, tanto la solución del problema parcial como su traducción a **P-Lingua** se encuentran parametrizados, dependiendo del valor concreto del tamaño del grafo (expresado a través del número de nodos n) para generar el sistema P de tejido correspondiente. Podemos definir una aplicación mínima basada en **MeCoSim** introduciendo datos generales, así como nuestra información específica, de forma que se permita al usuario la introducción del valor n para el que simular de forma visual. Se definirá la estructura visual de la aplicación (pestañas a mostrar), tabla de entrada y parámetro a generar (en este caso n , a partir de la tabla 1, fila 1, columna 1, $\langle 1, 1, 1 \rangle$). En la figura 7.1 vemos la configuración y la aplicación resultante.

Esta aplicación ya estaría lista para ser usada por el diseñador del sistema P, que podrá cargar el modelo, proporcionar valores al parámetro n , visualizar las estructuras creadas y depurar, paso a paso, visualizando la respuesta textual de **P-Lingua**, o viendo en forma de árbol cómo evolucionan tanto la estructura de membranas como los multiconjuntos presentes en cada una, como se muestra en la figura 7.2.



Figura 7.1: Coloraciones - Definición y visualización de entrada

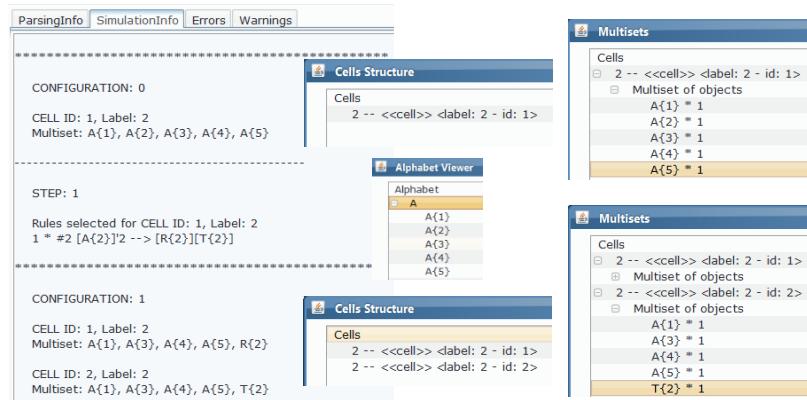


Figura 7.2: Coloraciones - Depuración

Para este problema de generación de coloraciones es interesante ver la evolución del número de células presentes en cada instante de la computación, pero quizás no nos interese tratar de obtener esa información revisando todo el texto o analizando paso a paso los visores mostrados. Así, por ejemplo, podríamos configurar una salida que mostrara la información deseada. Ahora bien, a diferencia de la entrada, donde indicamos qué hacer con ella para generar los parámetros necesarios, con las salidas debemos especificar qué información extraer de toda la computación a fin de generar la salida adecuada.

La definición que realizamos emplea un lenguaje soportado sobre SQL que recopila y agrupa datos de la computación, ya que debe ser lo más general posible con el fin de poder adaptarse a las más diversas necesidades de información. La figura 7.3 muestra la definición del resultado para la salida.

Result Table Id	Result Table Name	Table Id	Referred Table Id					
1	CellsByStepAux	0	0					
2	CellsByStep	2	1					
Criteria Id	Select/Where/Group	Criteria	Formula	ReferredCriteria Id	Argument	Qualified Name	Where condition type	Where condition
1	Select	step						
2	Select	membraneID						
3	WhereAux	membraneID					Integer	0
4	Where	formula	NOT	3				
5	Group	step						
6	Group	membraneID						
Criteria Id	Select/Where/Group	Criteria	Formula	ReferredCriteria Id	Argument	Qualified Name	Where/Select condition type	Where condition
1	Select	step						
2	Auxiliary	parameter	n				Integer	
3	Select	formula	CONVERT	2	INT	valn	DirectString	
4	Auxiliary	membraneID						
5	Select	formula	COUNT	4		ncells		
6	Group	step						

Figura 7.3: Coloraciones - Configuración de la salida

Como se puede apreciar, se están configurando dos salidas. En muchas ocasiones en que se desea obtener determinada información de la simulación en base a criterios de agrupación, según el caso será necesario definir resultados basados en resultados anteriores, a partir de la base de datos subyacente descrita en la figura 4.3. En este caso, la subconsulta obtiene (Select) los pares (step, membraneID) cuyo ID de membrana no sea 0, agrupados por (step, membraneID) para evitar que se devuelvan repetidos. Por la estructura de la base de datos, cada registro representa la presencia de un objeto con una multiplicidad en una membrana en un determinado paso. Por tanto, si no agrupáramos por (step, membraneID) devolveríamos una ocurrencia/aparición del par por cada objeto presente en ese paso en esa célula.

En el problema de generación de coloraciones se dispone de una membrana exterior con etiqueta 0 (e ID 0), y el resto de membranas serán células de etiqueta 2 generadas mediante división, conteniendo diferentes coloraciones parciales (sólo algunos nodos tienen asociado un color) hasta obtener, en el último paso, coloraciones

del grafo completo. La consulta principal, cuyo resultado se mostrará en la tabla 2, devolverá junto a cada paso la cuenta (COUNT) del número de membranas. El “parameter n” que aparece en la configuración va a mostrar una columna adicional con el número n en cada fila, con vista al proceso que se comenta a continuación, si solamente deseamos visualizar paso y membrana, entonces no sería necesario. La figura 7.4 muestra la salida.

Step	n	mult
1	5	2
2	5	4
3	5	8
4	5	16
5	5	32
6	5	63
7	5	115
8	5	179
9	5	227
10	5	243

Figura 7.4: Coloraciones - Salida

Un paso adicional interesante, siguiendo algunos de los objetivos de la metodología presentada, podría ser la extracción de invariantes (fórmulas que son verdaderas en cualquier instante de la computación) no triviales (distintos de la fórmula TRUE). Esos invariantes permitirían obtener propiedades acerca de las soluciones proporcionadas basadas en sistemas P. Ahora bien, conviene hacer notar que, en este caso, la integración de MeCoSim con Daikon permite analizar propiedades como, por ejemplo, la relación entre el número de células en cada paso o al final, y el número total n de nodos del grafo. Para ello, se configuran las extracciones a Daikon conforme al proceso descrito en la sección 4.2.3, en este caso mediante la configuración de la figura 7.5, en la que también se muestra el resultado obtenido.

Haciendo uso de la extracción automatizada de invariantes desde el plugin de Daikon de MeCoSim, junto con la posibilidad de unir varios de los ficheros de extracción de distintas instancias del problema para diversos valores de n antes de llamar al plugin de Daikon, se obtuvieron resultados interesantes acerca de algunas propiedades invariantes que están asociadas a la solución presentada. Algunas de las más relevantes se encuentran en la tabla 7.1. Esta extracción de invariantes que resul-

tan de las trazas del modelo nos pueden dar pistas interesantes acerca del modelo. Además, son muy útiles para esta primera detección, si bien deben ser refrendadas por un model checker como Spin. En la columna derecha de la tabla se indica si la propiedad detectada es finalmente cierta o no.

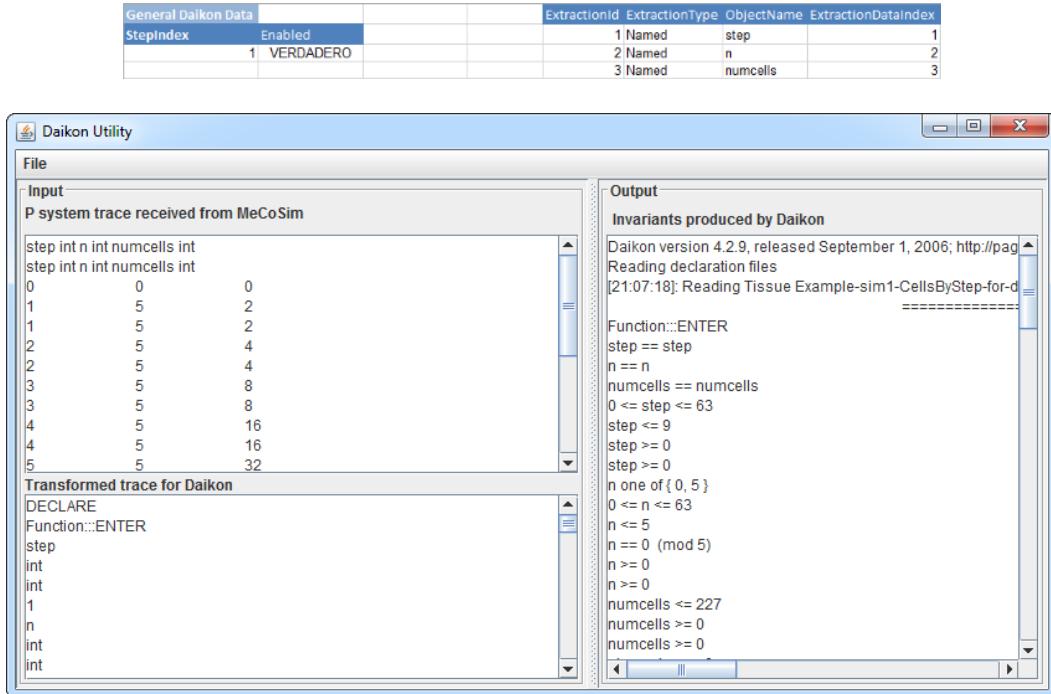


Figura 7.5: Coloraciones - Configuración de la extracción

Extracción	Resultado	Válido
Nº celdas por paso $0 \dots n$	$numcells = 2^{step}$	cierto
Nº celdas por paso $(n + 1) \dots 2n$	$numcells = 3(mod 4)$	falso
Nº celdas por paso $(n + 1) \dots (n + (n/2) + 1)$	$numcells = 3(mod 12)$	falso
Nº celdas último paso, para distintos valores de n conjuntamente	$numcells = 3^n$	cierto

Tabla 7.1: Algunos invariantes detectados

Como se puede observar, se demostró la veracidad de las propiedades detectadas: (a) para todo paso de computación entre 0 y n , se verifica que $\text{numcells} = 2^{\text{step}}$; y (b) en el último paso de computación se verifica que para cualquier entrada de tamaño n , se tiene que $\text{numcells} = 3^n$. Es importante observar que al haber realizado el proceso de verificación mediante el model checker Spin, esta última relación se ha establecido a partir del análisis de todos los posibles caminos del árbol de computación. Por tanto, también nos proporciona información acerca de la confluencia del sistema que se está analizando, ya que la citada propiedad se está cumpliendo para todo camino.

7.1.2. La aplicación 3-COL en MeCoSim

En el apartado anterior, a partir del estudio del problema de generación coloraciones, se ha ilustrado algunas de las funcionalidades que aporta el entorno MeCoSim y su integración con otras herramientas para la depuración y el análisis de las propiedades de los diseños basados en sistemas P situándose, fundamentalmente, en el ámbito del **diseñador**.

En esta sección se va a analizar la aportación de este entorno en el ámbito del **usuario final**. Para ello se parte del caso de estudio del problema 3-COL y de una solución del mismo mediante una familia de sistemas P de tejidos con reglas de división celular que ha sido presentada en la sección 1.3.3 del capítulo 1. Específicamente nos centraremos en la aportación de MeCoSim de cara a proporcionar una aplicación visual al usuario final para trabajar con el problema 3-COL, suministrando los grafos de entrada deseados y visualizando los resultados. Además, trataremos de ilustrar la ventaja de este tipo de herramientas configurables en relación con la inclusión monolítica de datos de instancias concretas en el propio archivo de modelo, mediante la separación entre modelo y escenario. Así mismo, estudiaremos la posibilidad que aparece de emplear la misma aplicación y su configuración de entradas y salidas para analizar soluciones basadas en distintas variantes de sistemas P gracias a ese desacoplamiento.

Veamos la traducción a P-Lingua de la solución completa:

```
@model<tissue_psystems>
def main()
{
    call tricolor_tissue(n,m);
}
def tricolor_tissue(n,m)
{
    call init_cells();
    call init_multisets(n);
    call init_environment(n,m);
    call init_rules(n,m);
}
```

```

def init_cells()
{
    @mu = [[],1,[],2]`0;
}
def init_multisets(n)
{
    @ms(1) = a{1},b,c{1},yes,no;
    @ms(2) = D;
    @ms(2) += A{i} : 1<=i<=n;
    @ms(2) += A{e{i,1},e{i,2}} : 1<=i<=ne;
}
def init_environment(n,m)
{
    @ms(0) = A{i},R{i},G{i},B{i},T{i},noR{i},noB{i} : 1<=i<=n;
    @ms(0) += a{i} : 1<=i<=2*n+m+@ceil(@log(m))+11;
    @ms(0) += c{i} : 1<=i<=2*n+1;
    @ms(0) += d{i} : 1<=i<=@ceil(@log(m))+1;
    @ms(0) += z{i} : 2<=i<=m+@ceil(@log(m))+6;
    @ms(0) += A{i,j},P{i,j},noP{i,j},R{i,j},G{i,j},B{i,j} : i<j<=n,1<=i<n;
    @ms(0) += b,D,noD,e,T,S,N,bb;
}
def init_rules(n,m)
{
    /* r1,i */ [A{i}]`2 --> [R{i}]`2 [T{i}]`2 : 1<=i<=n;
    /* r2,i */ [T{i}]`2 --> [G{i}]`2 [B{i}]`2 : 1<=i<=n;
    /* r3,i */ [a{i}]`1 <--> [a{i+1}]`0 : 1<=i<=2*n+m+@ceil(@log(m))+10;
    /* r4,i */ [c{i}]`1 <--> [c{i+1}*2]`0 : 1<=i<=2*n;
    /* r5 */ [c{2*n+1}]`1 <--> [D]`2;
    /* r6 */ [c{2*n+1}]`2 <--> [d{1},noD]`0;
    /* r7,i */ [d{i}]`2 <--> [d{i+1}*2]`0 : 1<=i<=@ceil(@log(m));
    /* r8 */ [noD]`2 <--> [e,z{2}]`0;
    /* r9,i */ [z{i}]`2 <--> [z{i+1}]`0 : 2<=i<=m+@ceil(@log(m))+5;
    /* r10,i,j */ [d{@ceil(@log(m))+1},A{i,j}]`2 <--> [P{i,j}]`0 : i<j<=n, 1<=i<n;
    /* r11,i,j */ [P{i,j}]`2 <--> [R{i,j},noP{i,j}]`0 : i<j<=n, 1<=i<n;
    /* r12,i,j */ [noP{i,j}]`2 <--> [B{i,j},G{i,j}]`0 : i<j<=n, 1<=i<n;
    /* r13,i,j */ [R{i},R{i,j}]`2 <--> [R{i},noR{j}]`0 : i<j<=n, 1<=i<n;
    /* r14,i,j */ [B{i},B{i,j}]`2 <--> [B{i},noB{j}]`0 : i<j<=n, 1<=i<n;
    /* r15,i,j */ [G{i},G{i,j}]`2 <--> [G{i},noG{j}]`0 : i<j<=n, 1<=i<n;
    /* r16,j */ [noR{j},R{j}]`2 <--> [bb]`0 : 1<=j<=n;
    /* r17,j */ [noB{j},B{j}]`2 <--> [bb]`0 : 1<=j<=n;
    /* r18,j */ [noG{j},G{j}]`2 <--> [bb]`0 : 1<=j<=n;
    /* r19 */ [e,bb]`2 <--> []`0;
    /* r20 */ [e,z{m+@ceil(@log(m))+6}]`2 <--> [T]`0;
    /* r21 */ [T]`2 <--> []`1;
    /* r22 */ [b,T]`1 <--> [S]`0;
    /* r23 */ [S,yes]`1 <--> []`0;
    /* r24 */ [b,a{2*n+m+@ceil(@log(m))+11}]`1 <--> [N]`0;
    /* r25 */ [N,no]`1 <--> []`0;
}

```

Al igual que en el problema de generación de coloraciones, será conveniente contar con una aplicación visual que nos permita introducir el grafo de entrada a fin de analizar su 3-coloreabilidad. Mediante la configuración de la aplicación basada en **MeCoSim** se obtiene una nueva pestaña conteniendo las aristas del grafo, ya que naturalmente habrá muchos posibles grafos de entrada para un sistema P de tejido con un determinado número de nodos n . La entrada podría ser similar a la tabla de la izquierda mostrada en la figura 7.6.

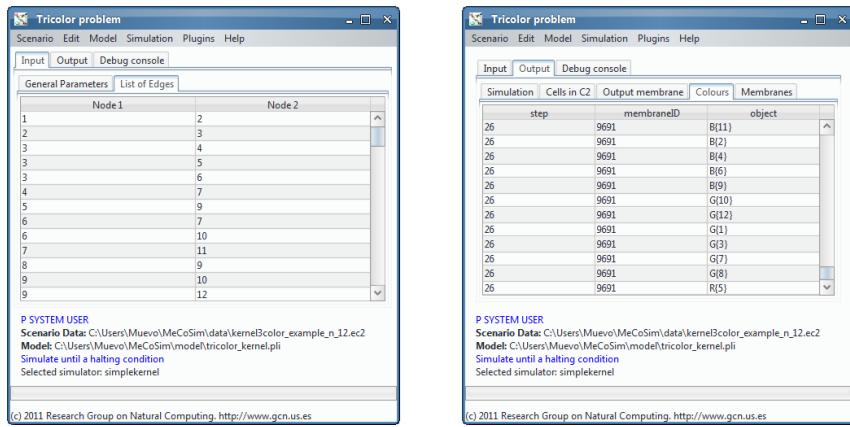


Figura 7.6: 3-COL - MeCoSim app

Gracias al plugin de grafos descrito en la sección 4.2.3, además de las opciones de depuración vistas en el ejemplo anterior, se puede visualizar el grafo correspondiente cargado a partir de las tablas de entrada para datos generales (número de nodos n y de aristas m) y de la lista de aristas (tabla con dos columnas indicando en cada fila o bien el nodo de origen y el de destino para cada arco, en el caso de un grafo dirigido, o bien la arista con el par de nodos interconectados, en el caso de un grafo no dirigido). La visualización presentada en la aplicación 3-COL basada en MeCoSim se muestra en la figura 7.7.

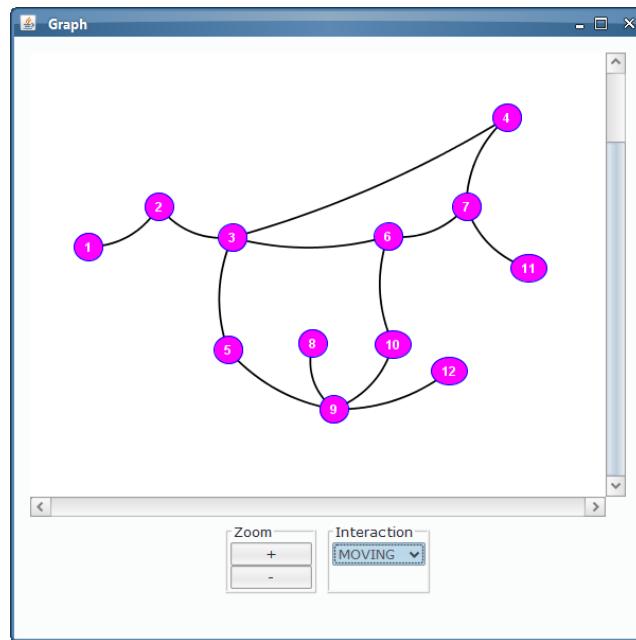


Figura 7.7: 3-COL - Grafo inicial

Estos grafos se muestran a partir de los parámetros generados según la configuración de la aplicación, en este caso produciendo parámetros $e_{k,1}$ y $e_{k,2}$ para cada par de nodos recibido como entrada. Estos pares de objetos generarán los objetos $A_{i,j}$ ($1 \leq i < j \leq n$) correspondientes que forman parte del multiconjunto de entrada $\text{cod}(\varphi)$ y, además, servirán como información que quedará a disposición para ser utilizada por cualquier plugin, como en este caso *GraphsPlugin*.

El grafo generado no contiene información acerca de las posiciones, por lo cual será posible realizar movimientos con los nodos a conveniencia del usuario, según la forma que quiera visualizar. Además, la configuración de la aplicación permite generar parámetros adicionales con el fin de almacenar un árbol de grafos para mostrar, por ejemplo, los objetos que se encuentran en cada paso, membrana a membrana, o bien los que se encuentran en cada membrana, paso a paso. Se trata, por tanto, de un árbol con dos niveles de profundidad en el que cada nodo hoja es un grafo con el contenido de una membrana en un paso determinado, y cada hoja del primer nivel muestra la secuencia de grafos del nivel inferior que dependen de dicho nodo. Se puede ver un elemento del árbol de grafos en el ejemplo mostrado en la figura 7.8 que, como se puede observar, ya se encuentra coloreado.

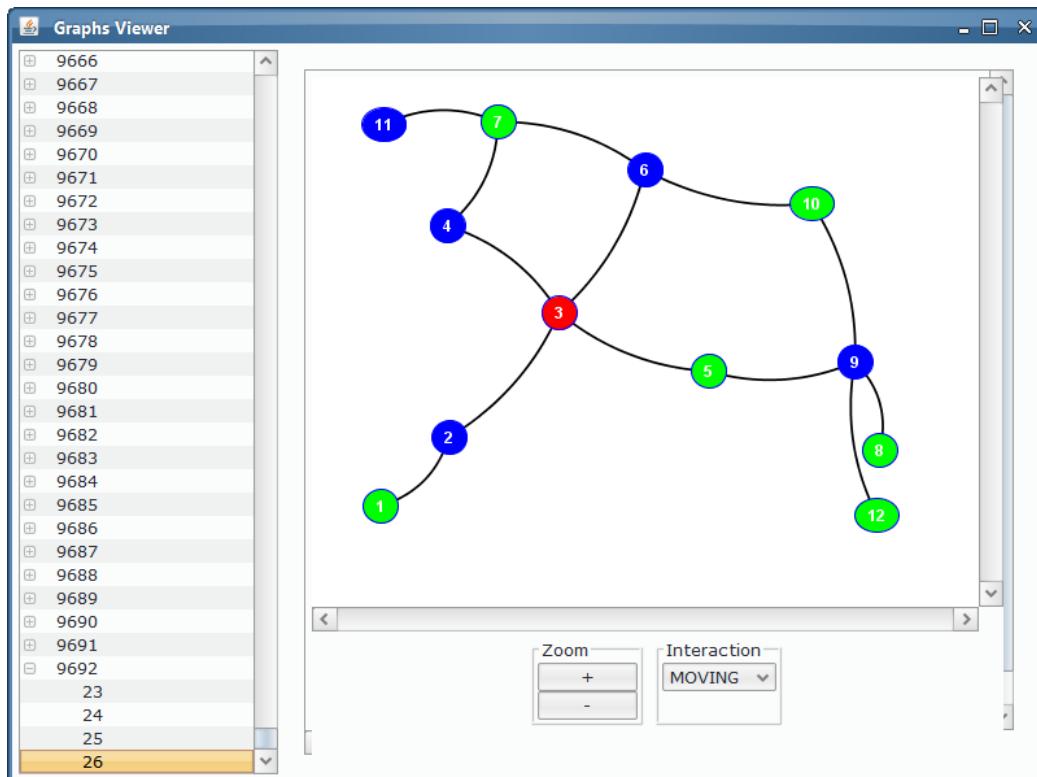


Figura 7.8: 3-COL - Árbol de grafos

Los nodos serán coloreados de acuerdo con la codificación habitual dada por el criterio RGB. Específicamente, si un objeto R_i aparece en una membrana de etiqueta 2, entonces el nodo i será coloreado con el color rojo (Red); si un objeto B_j aparece en una membrana de etiqueta 2, entonces el nodo j será coloreado con el color azul (Blue); y, finalmente, si un objeto G_k aparece en una membrana de etiqueta 2, entonces el nodo k será coloreado con el color verde (Green).

Tanto la coloración como la meta-information presente en el nodo puede ser configurada a través de los métodos de personalización proporcionados por los parámetros que se pueden generar en la configuración de la aplicación a medida basada en MeCoSim. Así, por ejemplo, la figura 7.9 muestra, además de la información inicial, no solamente los colores sino también el texto asociado al color (R , G o B).

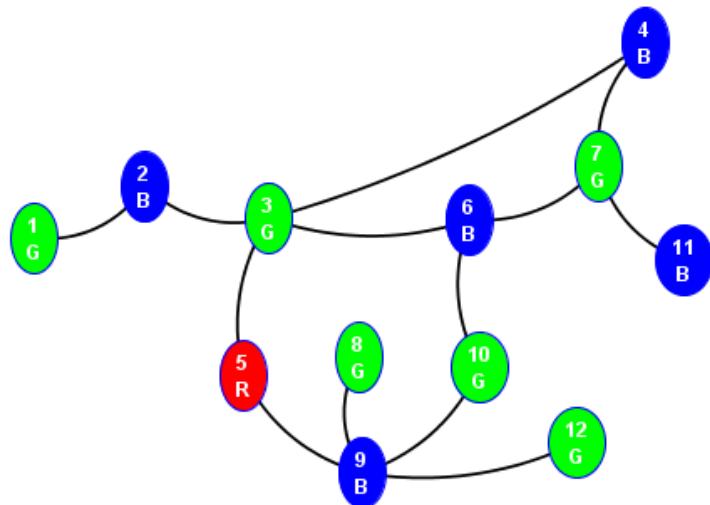


Figura 7.9: 3-COL - Un grafo solución personalizado

Como se puede observar, si nos detenemos en la solución proporcionada, la salida inicial respondía a la cuestión planteada en el problema de decisión: “*¿admite el grafo una coloración válida con tres colores?*”. Sin embargo, el entorno propuesto pone a disposición de los diseñadores herramientas que les permitan dotar a los usuarios de la aplicación de un valor añadido.

7.1.3. Una solución de 3-COL basada en simple kernel P systems usando P-Lingua y MeCoSim

En las secciones anteriores se ha trabajado sobre una solución al problema 3-COL a través de una familia de sistemas P de tejido con división celular.

En la sección 5.2.1 del capítulo 5 se introdujo el marco de los *simple kernel P systems* y se detalló sus peculiaridades y restricciones en cuanto a los tipos de reglas y estrategias de ejecución permitidas. Además, se proporcionó una solución alternativa al problema 3-COL empleando *simple kernel P systems*, tal como se publicó en [62].

En la sección 5.3 se detallaron las modificaciones efectuadas en P-Lingua para incorporar un nuevo marco, el de los simple kernel P systems, tanto a nivel del lenguaje y sus herramientas de reconocimiento léxico-sintáctico (*parsing*), como a nivel de simulación.

Esta sección tiene como objetivo ilustrar la adaptación de la aplicación para 3-COL del apartado anterior a dicha solución basada en simple kernel P systems.

A continuación se aporta el código P-Lingua correspondiente a la solución alternativa del problema 3-COL antes referida.

```
@model<simple_kernel_psystes>
def main()
{
    call three_colouring(n);
}
def three_colouring(n)
{
    call init_cells();
    call init_multisets(n);
    call init_rules(n);
}
def init_cells()
{
    @mu = [[],1[],2]’0;
}
def init_multisets(n)
{
    {
        @ms(1) = a, X{1};
        @ms(2) = A{1}, s;
    };
    @ms(2) += A{e{i,1},e{i,2}} : 1<=i<=n;
}
def init_rules(n)
{
    /*r1,i*/ [X{i}]’1 --> [X{i+1}]’1 : 1<=i<=(2*n+2);
    /*r1,2*n+3*/ [a,T]’1 --> [yes]’0;
    /*r1,2*n+4*/ @guard {>=T} ? [a,X{2*n+3}]’1 --> [no]’0;
    /*r2,2*i-1*/ @guard {=+s} ? [A{i}]’2 |--> [R{i},A{i+1}]’2 [T{i}]’2 : 1<=i<=(n-1);
    /*r2,2*i*/ @guard {=+s} ?
        [T{i}]’2 |--> [B{i},A{i+1}]’2 [G{i},A{i+1}]’2 : 1<=i<=(n-1);
    /*r2,2*n-1*/ @guard {=+s} ? [A{n}]’2 |--> [R{n},X]’2 [T{n}]’2;
    /*r2,2*n*/ @guard {=+s} ? [T{n}]’2 |--> [B{n},X]’2 [G{n},X]’2;
```

```

/*r2,2*n+1*/ @guard
|{={=+A{11,m1}}&&{=+B{11}}&&{=+B{m1}}&&{=-A{11,11}}}: {1<=m1<=(n),1<=11<=(n-1)} ||
|{={=+A{12,m2}}&&{=+G{12}}&&{=+G{m2}}&&{=-A{12,12}}}: {1<=m2<=(n),1<=12<=(n-1)} ||
|{={=+A{13,m3}}&&{=+R{13}}&&{=+R{m3}}&&{=-A{13,13}}}: {1<=m3<=(n),1<=13<=(n-1)} ?
[s]'2 --> []'2;
/*r2,2*n+2*/ [X]'2 --> [Y]'2;
/*r2,2*n+3*/ [Y,s]'2 --> [T]'1;
}

```

Naturalmente, esta solución del problema 3-COL usando *simple kernel P systems* es muy diferente al diseño basado en sistemas P de tejido con división celular. El archivo P-Lingua asociado sigue, por tanto, esa misma premisa. Sin embargo, ¿hay alguna diferencia en el problema en sí? La respuesta es clara: no.

El problema de decisión asociado (3-COL) consiste en responder sí o no (es decir, devolver *yes* o *no*) a la posibilidad de colorear un grafo no dirigido mediante tres colores, de tal modo que no existan dos nodos adyacentes que estén coloreados con el mismo color. Ahora bien, independientemente de la solución por la que se opte será necesario aportar el número de nodos del grafo y la lista de aristas del mismo.

Dada la existencia de nuevas soluciones para un mismo problema abstracto, la misma aplicación, como en este caso la definida en la sección 7.1.2, se puede emplear para cualesquiera de ellas partiendo de los mismos datos de entrada y que pretenda proporcionar las mismas salidas Teniendo presente que la aplicación reunía los requisitos necesarios para introducir el grafo de entrada, es posible reutilizarla para esta solución en el marco de los *simple kernel P systems*. Se trata de una ventaja derivada de la clara separación entre la visión del diseñador de sistemas P, para el que el interés está en la existencia de soluciones sustancialmente distintas empleando diversas variantes de sistemas P, y la visión del usuario final, para el que el problema es el mismo y pretende dar una solución.

Además, en este caso los parámetros generados a partir de los datos de entrada (el número de vértices n y los pares $(e\{i,1\}, e\{i,2\})$ necesarios para la representación de las aristas $A\{e\{i,1\}, e\{i,2\}\}$) son los mismos para ambas soluciones. No obstante, si la representación del problema fuera distinta en ambos casos e implicara diferentes multiconjuntos de entrada $cod(\mathcal{G})$, toda la configuración de entradas y salidas podría mantenerse igual, variando únicamente dicha codificación, a través de la pestaña (*SimulationParams*) de la configuración de la aplicación para generar los parámetros que fueran necesarios a partir de la misma entrada.

Asimismo, si además del problema de decisión se desea resolver el problema de la coloración efectiva del grafo y la computación de diversas soluciones conllevará distinta representación de los nodos en una y otra variante, el resto de la aplicación permanecería igual, centrando el cambio en la diferencia: en este caso, la configuración de la salida correspondiente, indicando cuáles son los objetos de interés según la nueva codificación.

La filosofía vuelve a ser la separación de los roles y evitar, siempre que sea posible, la necesidad de desarrollar software *ad-hoc* para cada problema o para cada solución a un mismo problema, dedicando cada rol el esfuerzo necesario en función del foco de su trabajo.

Finalmente, conviene resaltar que similares aplicaciones basadas en **MeCoSim** y **P-Lingua** se han considerado para estudiar soluciones mediante *simple kernel P systems* de otros problemas **NP**-completos, tales como **Subset Sum** o **Partition** que han sido desarrolladas en diferentes artículos ya publicados[76, 61].

7.2. Estudio y simulación del problema SAT

El problema **SAT** de la satisfactibilidad de la Lógica Proposicional es considerado paradigmático en el marco de la Teoría de la Complejidad Computacional. Se trata del primer problema que se demostró ser **NP**-completo. La prueba fue dada en 1971 por un joven canadiense (32 años) que respondía al nombre de Stephen Cook. En la sección 1.4.2 del capítulo 1 se presentó una solución eficiente del problema **SAT** mediante una familia de sistemas P con membranas activas y otra mediante una familia de sistemas P de tejidos con reglas de separación celular.

Diferentes soluciones del problema **SAT** usando distintos tipos de familia de sistemas P han sido publicadas. Entre ellas, destacan las proporcionadas en el marco de los sistemas celulares que trabajan a modo de tejidos (usando reglas de división celular[117]) y en el de sistemas celulares que trabajan a modo de neuronas, *Spiking neural P systems*[113].

La especificación en **P-Lingua** trata de ser lo más cercana posible a cada solución proporcionada por los diseñadores, si bien será específica para cada caso, dadas las distintas representaciones, sintaxis y semántica de los distintos marcos considerados. Sin embargo, en todos los casos la entrada será una fórmula proposicional en forma normal conjuntiva.

A continuación, se presenta la especificación **P-Lingua** de la solución detallada en 1.4.2 mediante una familia de sistemas P con membranas activas y, posteriormente, se ilustrará la aplicación basada en **MeCoSim** para el estudio y simulación de **SAT** independientemente de la solución escogida.

7.2.1. Especificación de una solución de SAT en P-Lingua

Esta especificación se presentó en [131], donde se detalla el lenguaje y el framework **P-Lingua**.

```

@model<membrane_division>
def main()
{
    call Sat(4,6);
    @ms(2) += x{1,1}, nx{1,2}, nx{2,2}, x{2,3},nx{2,4}, x{3,5}, nx{4,6};
}
def Sat(m,n)
{
    @mu = [[],2]'1;
    @ms(2) = d{1};

    [d{k}]'2 --> +[d{k}]-[d{k}] : 1 <= k <= n;
    {
        +[x{i,1} --> r{i,1}]'2;
        -[nx{i,1} --> r{i,1}]'2;
        -[x{i,1} --> #]'2;
        +[nx{i,1} --> #]'2;
    } : 1 <= i <= m;
    {
        +[x{i,j} --> x{i,j-1}]'2;
        -[x{i,j} --> x{i,j-1}]'2;
        +[nx{i,j} --> nx{i,j-1}]'2;
        -[nx{i,j} --> nx{i,j-1}]'2;
    } : 1<=i<=m, 2<=j<=n;
    {
        +[d{k}]'2 --> []d{k};
        -[d{k}]'2 --> []d{k};
    } : 1<=k<=n;
    d{k}[]'2 --> [d{k+1}] : 1<=k<=n-1;
    [r{i,k} --> r{i,k+1}]'2 : 1<=i<=m, 1<=k<=2*n-1;
    [d{k} --> d{k+1}]'1 : n <= k <= 3*n-3;
    [d{3*n-2} --> d{3*n-1},]1;
    e[]'2 --> +[c{1}];
    [d{3*n-1} --> d{3*n}]'1;
    [d{k} --> d{k+1}]'1 : 3*n <= k <= 3*n+2*m+2;
    +[r{i,2*n}]'2 --> -[]r{i,2*n};
    -[r{i,2*n} --> r{i-1,2*n}]'2 : 1<= i <= m;
    r{i,2*n}-[]'2 --> +[r{0,2*n}];
    -[c{k} --> c{k+1}]'2 : 1<=k<=m;
    +[c{m+1}]'2 --> +[]c{m+1};
    [c{m+1} --> c{m+2},t]'1;
    [t]'1 --> +[]t;
    +[c{m+2}]'1 --> -[]Yes;
    [d{3*n+2*m+3}]'1 --> +[]No;
}

```

La aportación de P-Lingua fue fundamental para permitir al diseñador de sistemas P trabajar con un lenguaje muy cercano a su ámbito de trabajo, el de los sistemas P, con una especificación muy similar a la de la propia descripción sintáctica del diseño realizado, disponiendo además de herramientas para el chequeo de su corrección, depuración y posterior simulación.

Como marco de trabajo para el diseñador de sistemas P, a la hora de diseñar un sistema se introducía la entrada del sistema y sus parámetros asociados en el propio fichero de diseño P-Lingua, como ilustran las líneas siguientes:

```

call Sat(4,6);
@ms(2) += x{1,1}, nx{1,2}, nx{2,2}, x{2,3},nx{2,4}, x{3,5},nx{4,6};

```

Es decir, se definía de forma conjunta la familia dada por el código presente en el interior de la función $\text{Sat}(m,n)$ y el multiconjunto de entrada $\text{cod}(\varphi)$ para la instancia φ concreta del problema sobre la que llevar a cabo la computación.

Como se ha reiterado en anteriores apartados, el objetivo de MeCoSim consiste en proporcionar una clara separación entre las labores de usuario y diseñador. Cabe resaltar que esta separación ya se dio (ver [131]) en aplicaciones *ad-hoc* de la familia *EcoSim*[131] en el ámbito de dos ecosistemas específicos, para presentar a los usuarios ecológicos una visión distinta de la del diseñador de sistemas P, más enfocada al problema que a las entrañas de los sistemas empleados para su modelización.

La idea de MeCoSim fue la de permitir que esta facilidad concreta, desarrollada para dos problemas específicos, pudiera convertirse en una propuesta metodológica, así como en una aplicación de propósito general para una amplia gama de variantes de sistemas P, dotando necesariamente de todo tipo de mecanismos de propósito general para el diseño, lo más sencillo posible, de aplicaciones adaptadas a cada problema particular, conservando esa separación entre el diseñador y el usuario final, y permitiendo la configuración de entradas, parametrización, salidas y todo tipo de herramientas que pudieran enriquecer, de forma general, la labor de diseñadores y usuarios dentro del amplio marco cubierto por el abanico de variantes de sistemas P.

En el caso específico del problema SAT, quedará adaptado el código P-Lingua con el fin de responder a la solución de la familia de sistemas P $\text{Sat}(m,n)$, incorporando la parametrización necesaria que excluya la instancia específica (en este caso, la fórmula proposicional antes incluida en `@ms(2)`), pasando ésta a depender de los parámetros que, para cada simulación, provengan de la entrada específica proporcionada por el usuario.

La modificación del código P-Lingua ilustrará el cambio de enfoque, ya que en lugar de una llamada con $m = 4$ y $n = 6$, tendremos:

```
def main()
{
    call Sat(m,n);
    call define_input();
}
```

Como vemos, la llamada ahora dependerá de los datos de entrada introducidos por el usuario, que generará el sistema P que corresponda de la familia $\text{Sat}(m,n)$ en función de los parámetros m y n . Además, en lugar de la fórmula concreta (*hard-coded* en P-Lingua) ahora tendremos la instrucción parametrizada recibiendo el multiconjunto $\text{cod}(\varphi)$ para cada instancia (fórmula proposicional) sobre la que se lance la computación a partir de la entrada del usuario:

```
@ms(2) += nx{clause{i},variable{i}}*valn{i},x{clause{i},variable{i}}*val{i} : 1<=i<=nvals;
```

Como se puede ver, esta instrucción se ejecutará $nvals$ ocasiones y, en cada una de ellas, se incorporará a los multiconjuntos de la membrana 2 un objeto $nx_{c,v}$ (el término $\neg x_v$ aparece en la cláusula c) si el parámetro $valn$ asociado es 1, o un objeto $x_{c,v}$ (el término x_v aparece en la cláusula c) si es el parámetro val el que vale 1. Estos parámetros se generan según la configuración definida en MeCoSim a partir de la entrada correspondiente, mostrada en la figura 7.10.

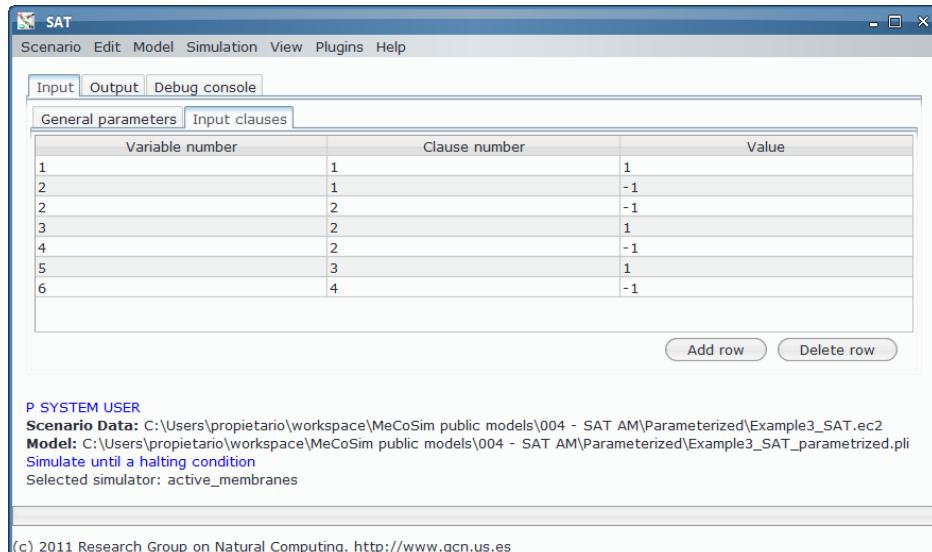


Figura 7.10: SAT - MeCoSim app input

Como se ve en la figura, cada fila de la tabla de entrada incluye un par (*variable*, *cláusula*), al que se asigna un valor 1 si la variable debe aparecer como término en la cláusula, y un valor -1 si es la variable negada la que debe aparecer.

El mecanismo de definición de parámetros de MeCoSim debe establecer en *SimulationParams* cómo traducir esa entrada en los parámetros del modelo. Otra entrada proporciona los valores de m y n , pero su transformación es directa, trivial. La configuración de los parámetros se muestra en la figura 7.11.

Param Name	Param Value	Index 1
n	<1,1,1>	
m	<1,1,2>	
nvals	<@r,2>	
clause	<2,\$1\$,2>	[1..nvals]
variable	<2,\$1\$,1>	[1..nvals]
value	<2,\$1\$,3>	[1..nvals]
val	<@max,value{\$1\$},0>	[1..nvals]
valn	<@func,min,value{\$1\$},0>^2	[1..nvals]

Figura 7.11: SAT - Definición de parámetros

Los parámetros m y n se limitan a tomar tal cuál los valores de la primera tabla, fila 1, columnas 1 y 2 respectivamente. El valor del parámetro $nvals$ vendrá dado por el número de filas de la tabla 2 (la mostrada en la figura 7.10), asignando por cada fila i del rango $[1..nvals]$ los valores de los correspondientes parámetros de cláusula ($clause_i$) y variable ($variable_i$), además de los valores complementarios val_i y $valn_i$, valiendo siempre uno de ellos 1 y su complementario 0.

Esta aplicación devolverá *yes* o *no* en función de la fórmula proposicional introducida por el usuario. Si se desea, es posible configurar salidas a medida del usuario para mostrar gráficos que, de forma llamativa, proporcionen esa misma información, o contentar al diseñador que pueda estar interesado en disponer de una tabla más detallada con el contenido en cada membrana en cada paso de computación, a modo de traza completa de forma tabulada. Esas necesidades específicas también podrán ser fácilmente cubiertas en MeCoSim sin necesidad de desarrollos a medida. Para ello, basta simplemente añadir las salidas a la configuración, obteniendo salidas del tipo de las figuras 7.12 y 7.13.

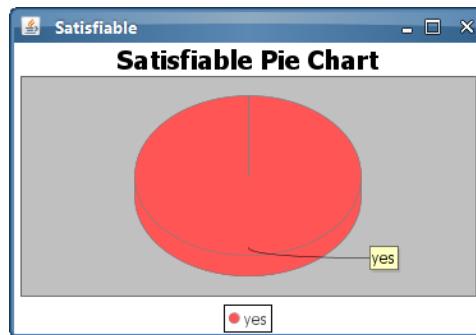


Figura 7.12: SAT - Gráfico de salida

Satisfiable Group	Satisfiability	Simulation				
Step	Environment	Label	Membrane	object	multiplicity	
26	Environment	0	0	x{2,2}	1	
26	Environment	0	0	x{2,3}	1	
26	Environment	0	0	x{2,4}	1	
26	Environment	0	0	x{3,1}	1	
26	Environment	0	0	x{3,4}	1	
26	Environment	0	0	x{3,5}	1	
26	Environment	0	0	x{4,2}	1	
26	Environment	0	0	x{4,4}	1	
26	Environment	0	0	yes	1	
26		1	1	B{5}	1	
26		1	1	T{1}	object = yes	
26		1	1	T{2}	1	
26		1	1	T{4}	1	
26		1	1	alpha{23}	1	
26		1	1	beta{11}	1	

Figura 7.13: SAT - Tabla de simulación

Además de esta solución detallada, se presentó en [131] una solución basada en sistemas P de tejido con reglas de división celular. La siguiente sección muestra una solución desarrollada recientemente empleando sistemas P de tejido con reglas de separación.

7.2.2. Una solución basada en sistemas P de tejido con reglas de separación celular

Recordemos que en el marco de los sistemas P de tejidos, las reglas de separación celular permiten la creación de dos nuevas células a partir de la célula en la que se ejecuta la regla. De tal manera que ese tipo de reglas son disparadas por un objeto del sistema que se consume al ser aplicada la regla, mientras que los restantes objetos de la célula en la que se ejecuta la regla son distribuidos en las dos nuevas células de acuerdo con una partición prefijada del alfabeto de trabajo Γ formada por dos subalfabetos Γ_1 y Γ_2 .

En [114] se estableció la eficiencia computacional del marco antes citado, proporcionando una solución eficiente del problema SAT mediante una familia de sistemas P de tejidos con reglas de división celular. No obstante, las longitudes de las reglas de comunicación del sistema podría ser, a lo sumo, 8 ($TSC(8)$). Posteriormente, en [125] se dió una solución eficiente a través de una familia de sistemas P de tejidos con reglas de división celular que usaban reglas de comunicación de longitud, a lo sumo, 3 ($TSC(3)$).

Recientemente [121], se ha desarrollado una ampliación del lenguaje, su parser correspondiente y simulador dentro del marco de P-Lingua para este tipo de sistemas. Además, en el artículo citado se proporcionaba también la especificación en P-Lingua de la solución para SAT en $TSC(3)$ de [125]. El código P-Lingua correspondiente es el que aparece a continuación.

```
@model<TSCS>

def main()
{
    call init_membrane_structure();
    call init_first_alphabet(m,n);
    call init_second_alphabet(m,n);
    call init_environment(m,n);
    call init_multisets();
    call init_rules(m,n);
    call define_input();
}

def define_input()
{
    @ms(3) +=   nx{variable{i},clause{i}}*valn{i},
                x{variable{i},clause{i}}*val{i} : 1<=i<=nvals;
}
```

```

def init_membrane_structure()
{
    @mu = [ [ ]'1 [ ]'2 [ ]'3 ]'0;
}

def init_first_alphabet(m,n)
{
    @ms1 += A{i},B{i} : 1<=i<=n+1;
    @ms1 += a{i},b{i},T{i},F{i},y{i},v{i},w{i} : 1<=i<=n;
    @ms1 += c{i},t{i},f{i},s{i},z{i} : 1<= i <= n-1;
    @ms1 += E{j} : 1<= j <= m+1;
    @ms1 += alpha{i} : 0<= i <= 3*n+2*m+1;
    @ms1 += beta{i} : 0<= i <= 3*n + 2*m +2;
    @ms1 += q{i,j},r{i,j},u{i,j} : 1<=i<=n-1,1<=j<=n-1;
    @ms1 += x{i,j},nx{i,j},e{i,j},ne{i,j} : 1<=i<=n,1<=j<=m;
    @ms1 += d{i,j,k},nd{i,j,k} : 1 <=i<=n,1<=j<=m,1<=k<=n;
    @ms1 += q{0},S,yes,no;
}

def init_second_alphabet(m,n)
{
    @ms2 += Ap{i},Bp{i} : 1<=i<=n+1;
    @ms2 += ap{i},bp{i},Tp{i},Fp{i} : 1<=i<=n;
}

def init_environment(m,n)
{
    @ms(0) += S;
    @ms(0) += A{i},B{i},Ap{i},Bp{i} : 2<=i<=n+1;
    @ms(0) += T{i},F{i},Tp{i},y{i},w{i} : 1<=i<=n;
    @ms(0) += a{i},ap{i},b{i},bp{i},v{i} : 2<=i<=n;
    @ms(0) += Tp{i},c{i},t{i},f{i},s{i},z{i} : 1<=i<=n-1;
    @ms(0) += E{j} : 1<=j<=m+1;
    @ms(0) += alpha{i} : 1<= i <= 3*n+2*m+1;
    @ms(0) += beta{i} : 1<=i<=3*n+2*m+2;
    @ms(0) += q{i,j},r{i,j},u{i,j} : 1<=i<=n-1,2<=j<=n-1;
    @ms(0) += r{1,1};
    @ms(0) += u{1,1};
    @ms(0) += e{i,j},ne{i,j} : 1<=i<=n,1<=j<=m;
    @ms(0) += d{i,j,k},nd{i,j,k} : 1<=i<=n,1<=j<=m,1<=k<=n;
}

def init_multisets()
{
    @ms(1) = A{1},B{1};
    @ms(2) = a{1},ap{1},b{1},bp{1},v{1},q{1,1},alpha{0},yes,no;
    @ms(3) = beta{0};
}

def init_rules(m,n)
{
    /*1*/ [A{i}]'1 <-> [a{i},ap{i}]'2 : 1<=i<=n;
    [A{n+1}]'1 <-> [E{1}]'2;
    /*2*/ [Ap{i}]'1 <-> [a{i},ap{i}]'2 : 1<=i<=n;
    [Ap{n+1}]'1 <-> [E{1}]'2;
    /*3*/ [B{i}]'1 <-> [b{i},bp{i}]'2 : 1<=i<=n;
    /*4*/ [Bp{i}]'1 <-> [b{i},bp{i}]'2 : 1<=i<=n;
    /*5*/ [T{i}]'1 <-> [t{i}]'2 : 1<=i<=n-1;
    /*6*/ [Tp{i}]'1 <-> [t{i}]'2 : 1<=i<=n-1;
    /*7*/ [F{i}]'1 <-> [f{i}]'2 : 1<=i<=n-1;
    /*8*/ [Fp{i}]'1 <-> [f{i}]'2 : 1<=i<=n-1;
}

```

```

/*9*/ [t{i}]'1    <-> [T{i},Tp{i}]'0 : 1<=i<=n-1;
/*10*/ [f{i}]'1    <-> [F{i},Fp{i}]'0 : 1<=i<=n-1;
/*11*/ [b{i}]'1    <-> [B{i+1},S]'0 : 1<=i<=n;
/*12*/ [B{n+1}]'1 <-> [#]'0;/*
/*12*/ [bp{i}]'1   <-> [Bp{i+1}]'0 : 1<=i<=n;
/*[Bp{n+1}]'1 <-> [#]'0;/*
/*13*/ [a{i}]'1    <-> [T{i},A{i+1}]'0 : 1<=i<=n;
/*14*/ [ap{i}]'1   <-> [Fp{i},Ap{i+1}]'0 : 1<=i<=n;
/*15*/ [A{i}]'2    <-> [c{i}]'0      : 1<=i<=n-1;
[A{i}]'2    <-> [#]'0      : n<=i<=n+1;
/*16*/ [Ap{i}]'2   <-> [c{i}]'0      : 1<=i<=n-1;
[Ap{i}]'2   <-> [#]'0      : n<=i<=n+1;
/*17*/ [B{i}]'2    <-> [c{i}]'0      : 1<=i<=n-1;
[B{n}]'2    <-> [#]'0;
/*18*/ [Bp{i}]'2   <-> [c{i}]'0      : 1<=i<=n-1;
[Bp{n}]'2   <-> [#]'0;
/*19*/ [c{i}]'2    <-> [b{i+1},bp{i+1}]'0 : 1<=i<=n-1;
/*20*/ [v{i}]'2    <-> [y{i}*2]'0      : 1<=i<=n;
/*21*/ [y{i}]'2    <-> [z{i},w{i}]'0      : 1<=i<=n-1;
[y{n}]'2    <-> [w{n}]'0;
/*22*/ [z{i}]'2    <-> [v{i+1}]'0      : 1<=i<=n-1;
/*23*/ [w{i}]'2    <-> [a{i+1},ap{i+1}]'0 : 1<=i<=n-1;
[w{n}]'2    <-> [E{1}]'0;
/*24*/ [q{1,1}]'2   <-> [r{1,1}]'0;
/*25*/ [q{i,j}]'2   <-> [r{i,j}*2]'0      : 1<=i<=n-1,2<=j<=n-1;
/*26*/ [r{i,j}]'2   <-> [s{i},u{i,j}]'0      : 1<=i<=n-1,1<=j<=n-1;
/*27*/ [s{i}]'2    <-> [t{i},f{i}]'0      : 1<=i<=n-1;
/*28*/ [u{1,j}]'2   <-> [q{1,j+1},q{2,j+1}]'0 : 1<=j<=n-2;
/*29*/ [u{i,j}]'2   <-> [q{i+1,j+1}]'0      : 2<=i<=n-2,2<=j<=n-2;
/*30*/ [u{i,n-1}]'2 <-> [#]'0      : 1<=i<=n-1;
/*31*/ [T{i}]'2    <-> [#]'0      : 1<=i<=n-1;
/*32*/ [Tp{i}]'2   <-> [#]'0      : 1<=i<=n-1;
/*33*/ [F{i}]'2    <-> [#]'0      : 1<=i<=n-1;
/*34*/ [Fp{i}]'2   <-> [#]'0      : 1<=i<=n-1;
/*35*/ [S]'1 --> [ ]'1 [ ]'1;
/*36*/ [alpha{i}]'2 <-> [alpha{i+1}]'0 : 0<=i<=3*n+2*m;
/*37*/ [beta{i}]'3  <-> [beta{i+1}]'0 : 0<=i<=3*n+2*m+1;
/*38*/ [x{i,j}]'3   <-> [d{i,j,1}*2]'0 : 1<=i<=n,1<=j<=m;
[nx{i,j}]'3   <-> [nd{i,j,1}*2]'0 : 1<=i<=n,1<=j<=m;
/*39*/ [d{i,j,k}]'3 <-> [d{i,j,k+1}*2]'0 : 1<=i<=n,1<=j<=m,1<=k<=n-1;
[nd{i,j,k}]'3 <-> [nd{i,j,k+1}*2]'0 : 1<=i<=n,1<=j<=m,1<=k<=n-1;
/*40*/ [d{i,j,n}]'3 <-> [e{i,j}]'0 : 1<=i<=n,1<=j<=m;
[nd{i,j,n}]'3 <-> [ne{i,j}]'0 : 1<=i<=n, 1<=j<=m;
/*41*/ [T{i},E{j}]'1 <-> [e{i,j}]'3 : 1<=i<=n,1<=j<=m;
[F{i},E{j}]'1 <-> [ne{i,j}]'3 : 1<=i<=n,1<=j<=m;
[Tp{i},E{j}]'1 <-> [e{i,j}]'3 : 1<=i<=n,1<=j<=m;
[Fp{i},E{j}]'1 <-> [ne{i,j}]'3 : 1<=i<=n,1<=j<=m;
/*42*/ [e{i,j}]'1 <-> [T{i},E{j+1}]'0 : 1<=i<=n,1<=j<=m-1;
[ne{i,j}]'1 <-> [F{i},E{j+1}]'0 : 1<=i<=n,1<=j<=m-1;
/*43*/ [e{i,m}]'1 <-> [E{m+1}]'0 : 1<=i<=n;
[ne{i,m}]'1 <-> [E{m+1}]'0 : 1<=i<=n;
/*44*/ [T{i}]'3 <-> [#]'0 : 1<=i<=n;
[F{i}]'3 <-> [#]'0 : 1<=i<=n;
[Tp{i}]'3 <-> [#]'0 : 1<=i<=n;
[Fp{i}]'3 <-> [#]'0 : 1<=i<=n;
/*45*/ [E{j}]'3 <-> [#]'0 : 1<=j<=m;
/*46*/ [E{m+1}]'1 <-> [yes,alpha{3*n+1+2*m}]'2;
/*47*/ [yes]'1 <-> [beta{3*n+1+2*m+1}]'3;
/*48*/ [alpha{3*n+1+2*m}]'2 <-> [beta{3*n+1+2*m+1}]'3;
/*49*/ [no,beta{3*n+1+2*m+1}]'2 <-> [#]'0;
/*50*/ [yes]'3 <-> [#]'0;
}

```

De acuerdo con la filosofía de **MeCoSim** reiterada en el ejemplo anterior, esta solución se encuentra parametrizada. Es decir, recibe la información específica de la fórmula de entrada a través del multiconjunto $\text{cod}(\varphi)$ y genera los objetos correspondientes en el sistema, en este caso en la célula 3.

A nivel del diseñador de sistemas P, las soluciones comentadas anteriormente y ésta con la que se trabaja ahora, son muy diferentes y, por tanto, se corresponden con distintos archivos P-Lingua. Sin embargo, al igual que sucedía en el ejemplo de 3-COL, las necesidades son las mismas en todos los casos. Esto nos lleva a poder emplear la misma aplicación basada en **MeCoSim** para cada uno de las soluciones.

Como también se comentara en el caso del problema 3-COL con el grafo de entrada, es posible que, en este caso, existan distintas representaciones de la fórmula en diferentes soluciones. Todo ello podría alterar la configuración ya que el proceso de codificación de la entrada cambiaría. No obstante, el resto de la definición de entradas, salidas, etc. de la aplicación sería de nuevo la misma.

Toda la infraestructura de **MeCoSim** utilizada para los casos estudiados hasta ahora de los problemas 3-COL y SAT, han empleado, en todo momento, los mecanismos de propósito general diseñados con el fin de que puedan adaptarse de forma flexible a cada problema concreto. No obstante, siempre podrán surgir necesidades o requerimientos específicos que el usuario desee incorporar al entorno de simulación general. Por ejemplo, para el caso del problema SAT se nos podría ocurrir que sería más fácil y práctico disponer de una ventana a medida en la que se pudiera escribir la fórmula proposicional en forma normal conjuntiva (FNC) que corresponda, y que sea el propio sistema quien genere la tabla de entrada para **MeCoSim**.

Para poder incorporar este tipo de elementos tan específicos a un sistema que pretende ser lo más general posible, se debe dotar de mecanismos de extensibilidad que faciliten la integración. Para ello, se han diseñado distintos mecanismos en **MeCoSim**, uno de los cuáles es la arquitectura de plugins. Se trata de programas implementados en Java y liberados como .jar que permiten realizar llamadas directas a las funcionalidades que se desee, o bien son programas completamente externos que un wrapper de **MeCoSim** llama a través de los diversos sistemas operativos, permitiendo la configuración flexible de las opciones de menú y los parámetros específicos que pasar a los plugins.

En este caso se desarrolló un plugin para la introducción de fórmulas de forma visual, lo más cercano posible al usuario final, que pretende introducir una fórmula proposicional en FNC.

Así, por ejemplo, la fórmula: $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$ se podrá introducir de la siguiente manera $(x_1+nx_2+x_3)*(x_1+x_2+nx_3)*x_2$, sustituyendo los operadores lógicos \wedge y \vee por $*$ y $+$ respectivamente, y anteponiendo n al nombre de la variable para indicar *not*. Así, *SATPlugin* se lanza desde *MeCoSim*, permitiendo introducir la fórmula deseada y generando la tabla de entrada de cada par (variable,cláusula) para *MeCoSim*, como se muestra en la figura 7.14.

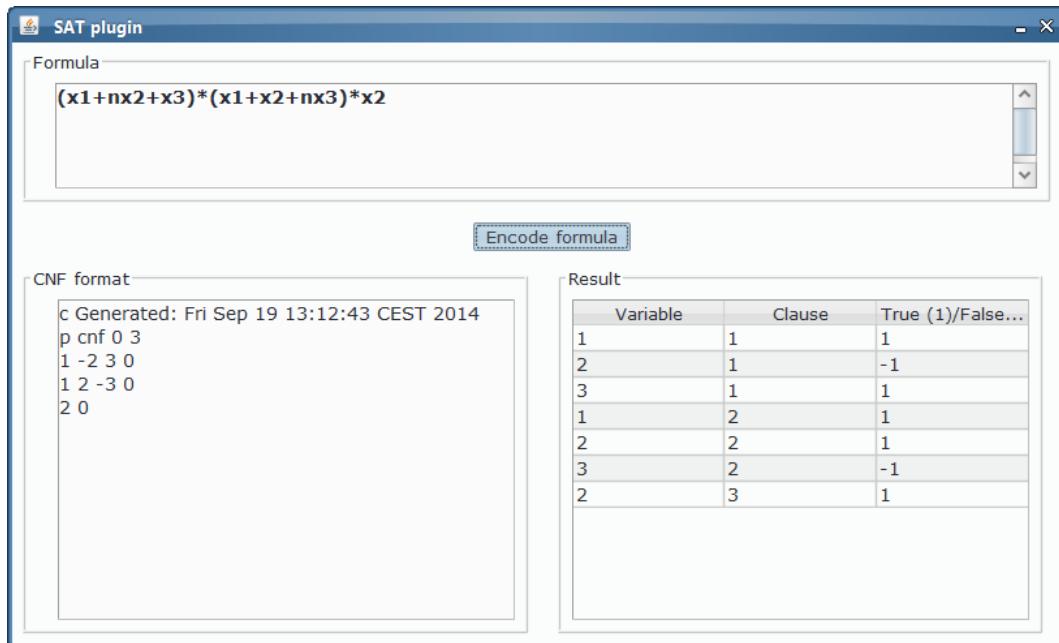


Figura 7.14: MeCoSim - Ventana de SATPlugin

Parte III

Conclusiones y trabajo futuro

8

Conclusiones y trabajo futuro

El presente capítulo tiene como objetivo describir algunas de las principales conclusiones extraídas de la labor desarrollada a lo largo de este trabajo, poniendo especial énfasis en las contribuciones más importantes realizadas y trazando algunos de los posibles caminos a seguir, a partir de la senda que se ha empezado a trazar.

8.1. Conclusiones

La imperiosa necesidad del Hombre de resolver los problemas que afectan a su vida desde el principio de los tiempos, le lleva a una incansable búsqueda de herramientas que le ayuden a enfrentar de la manera más ventajosa posible dichos problemas, en toda la variedad de situaciones actuales o que potencialmente puedan aparecer.

Es fundamental el rol que el conocimiento de los fenómenos por parte del Hombre puede desempeñar ante la tesisura de afrontar los acontecimientos en posiciones más ventajosas o favorables. Ello viene a ser una especie de primer paso antes de poder plantear escenarios y poder valorar las posibles consecuencias de sus actuaciones de forma anticipada, en base a esas herramientas capaces de asistirlo en búsqueda de respuestas o de posibles soluciones a los problemas.

Gran parte de los problemas que se abordan pueden ser resueltos mediante procedimientos mecánicos y, desde el boom de los ordenadores electrónicos, con los que trabajamos desde el siglo pasado, se dispone de una herramienta muy potente que nos ayuda a atacar la resolución práctica de ese tipo de problemas. No obstante, a pesar de las limitaciones ya conocidas que tienen esos dispositivos, continúa esa búsqueda tenaz y persistente de nuevos métodos que nos permitan dar solución a problemas, superando las barreras.

Bajo esa premisa, se mira a la sabia Naturaleza, que ofrece al Hombre una visión del modo en que resuelve los problemas de forma sistemática, entre otros, a través de los procedimientos que rigen el nivel de organización y de funcionamiento de las células vivas y los tejidos que forman, como esencia de la vida, las componentes básicas en las que transcurren de forma natural un gran número de procesos en cada compartimento de su estructura, a nivel bioquímico.

Con esta inspiración nace el paradigma de la Computación Celular con Membranas y, tras unos años de estudio acerca de su capacidad para resolver problemas, comienza a aplicarse esa visión para la resolución práctica de los mismos, tanto a nivel teórico como a nivel práctico con problemas de la vida real que afectan a distintos niveles de organización social, económico o biológico, tanto a escala *micro* (molecular y celular, principalmente) como a escala que podríamos denominar *macro* (dinámica de poblaciones y de sistemas complejos, en general). En este contexto se pone a disposición de la comunidad científica, una herramienta novedosa capaz de resolver problemas y de aumentar el conocimiento de los fenómenos, a través del diseño de *modelos* que capturen la parcela de la realidad objeto de estudio, incorporando los elementos e interrelaciones que se consideren más relevantes durante el proceso de abstracción.

Una vez diseñados y validados esos modelos, se podrá estudiar la posible respuesta de los sistemas complejos subyacentes ante determinadas situaciones pasadas (con el fin de proporcionar alguna explicación a fenómenos ya ocurridos y de los que se desconoce su causa), presentes (para solucionar los problemas inmediatos o a corto plazo, ante las situaciones reales actuales) o futuras (a efectos de anticipar las respuestas de los sistemas estudiados ante determinados escenarios potenciales, o las consecuencias de las actuaciones que el ser humano pueda realizar en la gestión de dichos sistemas).

En este contexto de resolución práctica de problemas, el marco P-Lingua dota a los desarrolladores de software y diseñadores de sistemas P, de potentes herramientas para la especificación, depuración y simulación de soluciones a problemas obtenidas mediante el uso de sistemas P.

El trabajo que se ha desarrollado en la presente memoria trata de dar un paso más hacia la disposición real, para el común de los seres humanos, de herramientas prácticas con las que resolver problemas en el ámbito citado. Para ello, se ha diseñado una metodología que va desde la observación del problema a resolver, hasta la provisión de una aplicación software para asistir en la profundización del conocimiento acerca de los fenómenos objetos de estudio, a través de la realización de experimentos virtuales que permita anticipar la respuesta del sistema estudiado antes diferentes posibles escenarios que pudieran ser de interés.

Junto con la metodología referida, se ha desarrollado un entorno virtual de simulación, **MeCoSim** (*Membrane Computing Simulator*), que articula la propuesta metodológica a través de herramientas lo suficientemente generales y flexibles para abarcar el mayor número de problemas posible y, a la vez, lo suficientemente personalizables con el fin de ajustarse a las necesidades de cada aplicación particular de la metodología. Este entorno permite la liberación de aplicaciones adaptadas para los usuarios finales de cada problema particular, dotándole internamente de todas las herramientas proporcionadas por el marco de **P-Lingua**, pero de forma transparente, de tal manera que los sistemas P no sean para él el fin sino el medio.

Por ello, para los diseñadores de sistemas P el entorno actuará como un IDE (*Integrated Development Environment*) suponiendo una capa adicional de introspección y tratamiento visual de los modelos basados en sistemas P, mientras que para los usuarios finales se dispondrá de una aplicación visual para la gestión del fenómeno de su interés, adaptado al dominio del problema. En esa aplicación se podrán introducir los datos específicos del escenario a estudiar, realizar los experimentos virtuales simulando el comportamiento del sistema real y observar las salidas derivadas del modelo, mediante datos tabulados o gráficas que ilustren la evolución de las componentes básicas, en el ámbito de estudio del usuario final, ajeno a las interioridades de los sistemas P subyacentes y la complejidad del marco sobre el que se construye.

Los objetivos de este entorno requieren que sea, además, lo suficientemente extensible para permitir la ampliación de la cobertura de problemas y funcionalidades actuales y previstas. Para ello, se ha desarrollado, junto con el entorno de simulación **MeCoSim**, una arquitectura de plugins que permita la integración con otras herramientas software implementadas en la misma o en tecnologías distintas a la empleada por **MeCoSim**.

El entorno se ha integrado con algunas herramientas externas a través del desarrollo de un primer conjunto de plugins, como es el caso del detector de invariantes **Daikon** o el Model Checker **Spin**. Además, se han desarrollado otros plugins para la visualización de estructuras basadas en grafos, con el fin de incorporar desde **MeCoSim** funciones en otros lenguajes de programación, como **Haskell**, o bien para la introducción de fórmulas proposicionales en un lenguaje muy similar al empleado por los lógicos.

Conviene destacar la colaboración realizada en el desarrollo y adaptación de distintos simuladores en el ámbito de los sistemas P de Dinámica de Poblaciones, de los sistemas P de tejido, o de los sistemas P que funcionan a modo de neuronas. Ante la diversidad de estos sistemas y variantes, para determinados fines, como es el caso de la verificación automática de modelos, se ha participado en el diseño y aplicación práctica de un nuevo formalismo con la finalidad de unificar, homogeneizar en cierto sentido, la variedad existente de sistemas P. Dentro de éstos, se ha desarrollado en el marco de **P-Lingua** tanto el lenguaje como un reconocedor léxico-sintáctico, así como un primer simulador de un subconjunto de los mismos: los *simple kernel P systems*.

El diseño de una metodología y el desarrollo de una serie de herramientas vertebradoras no pueden, por sí solo, demostrar la utilidad de una propuesta para la resolución práctica de problemas reales que afecten al ser humano, tanto en el ámbito de aplicaciones sobre la vida real como en la resolución de problemas **NP**-completos bien conocidos que puedan repercutir, en última instancia, sobre otros problemas más cercanos a las necesidades del día a día.

Afortunadamente, desde sus inicios, este desarrollo ha venido acompañado de múltiples aplicaciones prácticas de la metodología y de las herramientas, a un gran número de problemas tanto de la vida real (estudio de la dinámica de ecosistemas reales como el quebrantahuesos en la zona pirenaico catalana o en el pirineo navarro, análisis de los efectos de los pestivirus en la población del rebeco en el pirineo catalán, reintroducción de una especie de pájaros en una zona de Cataluña, alianzas de plantas, tritones, desarrollo y crecimiento de anfibios en estanques naturales, producción animal porcina, plan de control de la especie invasora del mejillón cebra, estudio de pandemias o redes de genes, etc.) como del ámbito de la teoría de la complejidad computacional, aplicándose a la resolución de problemas **NP**-completos bien conocidos como **SAT**, **3-COL**, **Subset Sum**, **Vertex Cover** o **Partition**. En ellas se han hecho uso de diversas variantes como los sistemas P con membranas activas, sistemas P de tejidos con reglas de división o de separación celular, sistemas P que trabajan a modo de neuronas, *simple kernel P systems* o sistemas P de Dinámica de Poblaciones (*PDP systems*) en el ámbito de la modelización de sistemas complejos.

Las herramientas desarrolladas están siendo utilizadas en la actualidad por grupos de investigación de diversas universidades en Lleida, Sheffield (U.K.), Bucarest y Pitesti (Rumanía), Wuhan y Chengdu (China). Además, tenemos conocimiento de la aparición de nuevos trabajos que comienzan a emplearlas (el último de ellos, que tengamos constancia, desde Malasia [142]).

8.2. Trabajo futuro

A continuación se relacionan algunas de las principales ideas que se plantean como posibles líneas de investigación de trabajo futuro y que se derivan de la presente memoria.

- Extender la metodología propuesta con el fin de incorporar aspectos adicionales como: (a) el tratamiento de la incertidumbre, para diseños basados en sistemas P dependiendo de parámetros de valores desconocidos o bien que sean conocidos pero con un cierto grado de precisión; y (b) el análisis de los datos resultantes de las simulaciones a nivel estadístico y, en general, en términos de *Data Science*.
- Ampliar y estandarizar el entorno de simulación del proceso de lanzamiento de simuladores externos, de tal modo que se pueda emplear la potencial flexibilidad de MeCoSim, como **front end** conectable con una gran variedad de tecnologías externas, delimitando con claridad los protocolos y formatos de comunicación, mediante el diseño sólido de uno específico con reconocedores, intérpretes y simuladores externos, locales o remotos, de manera general y configurable, acorde con la filosofía de MeCoSim.
- Modificar los simuladores incorporados en P-Lingua o desarrollar de nuevos simuladores dentro de ese marco, pero no centrados en la aportación de mayores niveles de paralelismo (vía que ya está siendo explorada por otras iniciativas muy interesantes en la comunidad de *Membrane Computing*), sino materializados a través de la sustitución de reglas individuales con las que se trabaja aisladamente, por reglas que trabajen a nivel simbólico, de manera que un esquema de reglas, dado por la posible presencia de numerosos índices cubriendo un rango conjunto de cientos de miles o millones de reglas individuales, sea tratada por una única regla a nivel simbólico incorporando, de algún modo, los rangos de cada índice y la intersección de rangos en distintas reglas simbólicas con el fin de determinar posibles bloques en competencia. Esto implicaría un ahorro enorme en términos de espacio ocupado por la definición de las reglas y, posiblemente, en lo que respecta a eficiencia en términos de tiempo real de simulación ya que, generalmente, sólo un pequeño subconjunto de reglas de un esquema son aplicables y, a día de hoy, hay que recorrer los posibles millones de reglas individuales para determinar si cada una de ellas es o no aplicable.
- Profundizar en la idea iniciada recientemente de acelerar la simulación con técnicas basadas en el algoritmo RETE, empleado en otro tipo de paradigmas computacionales que, al igual que los sistemas P, incorporan diseños basados en reglas. Sería interesante explorar la combinación de la definición simbólica con algún tipo de adaptación del algoritmo RETE.

- Definir sistemas P a través de herramientas visuales como complemento a la posibilidad de definirlos con P-Lingua, a modo de MDE (*Model Driven Engineering*). Podrían emplearse herramientas bien conocidas representando estructuras celulares como, por ejemplo, **Cell Designer** e integrar la salida, proporcionada generalmente a través de ficheros estándar como SBML o CellML, como entrada de MeCoSim, o bien proporcionar, dentro de MeCoSim, mecanismos de diseño visual de manera similar a las interesantes interfaces desarrolladas en MetaPLab[30]. En cualquiera de los casos, se podría generar a partir de los diseños visuales los ficheros P-Lingua correspondientes.
- Diversificar los formatos de salida de las computaciones y salidas a medida de las simulaciones de tal modo que, al margen de las salidas tabuladas y gráficas, se pueda dotar de formatos estándares como los aceptados por gnuplot o json, opción esta última en la que ya se ha empezado a trabajar de forma experimental, proporcionando salidas de computación completa, tablas y gráficos mediante formato json, y posterior apertura automática vía web mediante bibliotecas basadas en jQuery, Highcharts o Datatables.
- Mejorar la documentación de MeCoSim, con guías más completas a nivel de usuario, diseñador de sistemas P y desarrollador, y referenciar con detalle los diversos lenguajes y herramientas proporcionados. Se ha empezado a trabajar en ello en la web de MeCoSim.
- Proporcionar un mecanismo general para la extensión de las funciones disponibles en MeCoSim para la generación de parámetros, la simulación en P-Lingua u otros mecanismos que se puedan diseñar a futuro. Se podría desarrollar una interfaz común en la que definir funciones a través de un gran número de lenguajes de programación actuales o de futura creación, haciendo uso de algún mecanismo estándar como el propuesto dentro de tecnología Java en el JCP por la petición JSR 223 (*Scripting for the Java Platform*). Un referente interesante en este sentido es la plataforma SAGE, que pone a disposición muchísimas funcionalidades a nivel simbólico y numérico bajo una interfaz común, muchas de ellas proporcionadas a través de numerosos paquetes software externos pero de forma transparente al desarrollador que emplea SAGE.
- Dotar de más visibilidad a la inclusión de nuevos repositorios de MeCoSim. El mecanismo actual de repositorios permite a cualquier usuario, poner a disposición de la comunidad los plugins, apps, models o scenarios basados en MeCoSim que diseñe o desarrolle y, además, la configuración de repositorios es sencilla. No obstante, se hace necesario articular un buen mecanismo de consolidación de estos repositorios y de difusión de los mismos. De tal manera que sean bien conocidos por la comunidad y cada usuario sepa cómo acceder fácilmente a todo repositorio publicado, desde MeCoSim o desde su web.

Un referente en este terreno sería el sistema de plugins de Eclipse [57], tomado como inspiración.

A nivel de difusión y compartición de los repositorios de modelos, también sería interesante explorar otros sistemas que ponen a disposición una librería de modelos como NetLogo [158].

A nivel de desarrollo, se podría tomar como referente la comunidad de desarrolladores de R [135], una comunidad activa desarrollando plugins y compartiéndolos para la comunidad de usuarios y desarrolladores, que presenta cada día un gran número de nuevos paquetes de software de distintos miembros de la comunidad en su web.

En definitiva, sería muy interesante tratar de diseñar un buen mecanismo para la gestión de los repositorios, incluyendo el envío y recepción de solicitudes de nuevos plugins, apps, modelos, soluciones o escenarios disponibles, validación, estandarización de formato e información para su publicación, difusión, etc.

- Ampliar los modelos de computación cubiertos y variantes de los mismos. Los mismos mecanismos de entradas, salidas o plugins podrían ser útiles para lanzar computaciones, tanto para sistemas P no contemplados por el momento, ni dentro de P-Lingua ni en simuladores externos, como para modelos ajenos al paradigma de *Membrane Computing*, con aproximaciones completamente distintas pero proporcionando soluciones al mismo tipo de problemas, de tal modo que se pudieran efectuar así comparaciones dentro del mismo marco o en distintos marcos, fundamentalmente en otros campos de la Inteligencia Artificial.
- Explorar la posibilidad de incorporar en MeCoSim la resolución de problemas de optimización en el marco de los *Evolutionary Membrane Algorithms*, recientemente introducidos [162].

A través de esos algoritmos se exploran las interacciones entre una disciplina, *Membrane Computing*, bien asentada y con fundamentos teóricos muy fuertes, y otra, *Evolutionary Algorithms*, con propiedades tan interesantes como la flexibilidad, la facilidad de comprensión y su utilidad para dar respuestas satisfactorias a problemas interesantes de la vida real de un amplio espectro.

- Incorporar en MeCoSim la posibilidad de actuar como sistema de simulación desconectado, permitiendo que se le pueda realizar peticiones y que éstas se lancen mediante simuladores internos o externos, sin que el usuario necesite quedar a la espera del resultado.

Habría que determinar el mecanismo para enviar las salidas de la simulación, o las diversas posibilidades de visualización de las respuestas recibidas en cualquier momento en MeCoSim.

- Integrar controles visuales que faciliten la introducción de escenarios de interés, a modo de los botones y controles empleados por NetLogo [158], de tal manera que no sea necesario introducir toda la información de entrada mediante tablas. Esto podría facilitar al usuario final la introducción de parámetros determinados en algunos modelos. Para ello, se debería implementar el correspondiente mecanismo de diseño asociado, con el fin de definir qué datos van a provenir de los controles visuales y cómo comunicarse con el mecanismo de entrada actual.

A

La Historia Interminable del software

Esta tesis, como todo en la Ciencia o en el software, no parte de cero. Se nutre de los esfuerzos de tantos que desde los anales de la Historia fueron haciendo este camino al andar, al construir sus propias historias, al desarrollar sus propios pensamientos, al explorar el mundo con sus propios ojos y tratar de dar sus propias soluciones a los problemas. Día a día leemos lo que otros escribieron, aprendemos de las historias que otros vivieron, de los pensamientos que otros desarrollaron o de las teorías que otros pudieron demostrar. Vivimos en una historia interminable en la que en este momento alguien está leyendo lo que yo he escrito, y mientras escribo este texto pienso en lo que leí de otros, muchas historias interminables surgirán de la bifurcación de este camino, el mío, que partió de tan solo uno de los múltiples caminos interminables que se abrieron antes que el mío...

Permítaseme utilizar este anexo para escribir estas palabras en una reflexión que por seguro muchos otros hicieron antes que yo y que otros muchos continuarán haciendo conforme la Historia se siga escribiendo. Por favor, no consideren pretenciosas unas palabras que más bien siento infantiles mientras escribo y evoco ciertas lecturas y visualizaciones de mi niñez, pero que tan presentes encuentro cada día en la Ciencia, en el Software, o en la labor docente en la que transmitimos a los que vienen nuestra propia historia, fruto de nuestra propia visión de la parcela de conocimiento que tratamos de transmitir.

A lo largo de la evolución del Hombre, éste ha ido acumulando experiencias, testimonios, contribuyendo al aumento del conocimiento acerca del mundo que nos rodea y las leyes que lo rigen. El valor del conocimiento y la sabiduría desarrollada por tantos seres humanos a lo largo de la Historia es de un valor incalculable, y cada esfuerzo que ponemos en ello trata de sumar para que entre todos lleguemos a un nuevo granito. No puedo por menos que sumarme a la visión del eminente Grigori Perelman al considerar que su contribución a la resolución de la conjetura de Pointcaré no era mayor que la de otros predecesores. No por la afirmación en sí, con la que quizás no coincida, al entender que la aportación de cada uno de nosotros es genuina y tiene un valor único porque no hay dos mentes iguales capaces de concebir el mundo y la resolución de los problemas exactamente desde el mismo prisma. No por esa afirmación, sino por la visión subyacente de que nuestra aportación se sustenta sobre las que hicieron otros, que a su vez se apoyaron tanto en los caminos abiertos como descartados por otros, no negando en modo alguno la valía de cada aportación pero sí dando siempre el valor que merecen sus predecesores.

Este pequeño anexo pretende por tanto, en primer lugar, hacer justicia con aquellas aplicaciones y bibliotecas software que han hecho posible, facilitándolo enormemente, el desarrollo de MeCoSim. Para un modesto proyecto software como éste, se ha hecho uso de numerosos recursos desarrollados por otros, desde entornos de desarrollo hasta bibliotecas de software para Java o para la visualización de contenidos Web. Mi infinito agradecimiento a los esfuerzos de los que invirtieron su tiempo y su energía en el desarrollo de herramientas que me permitieron resolver los problemas a los que me enfrentaba. Los textos científicos a los que alude esta memoria de tesis quedaban debidamente referenciados en el apartado de *Bibliografía*, pero no podía concluir esta memoria sin referenciar aquí las herramientas software que tanto han facilitado el desarrollo de mi trabajo.

En segundo lugar, y para continuar con esta historia interminable, la última sección del anexo pretende describir las funcionalidades principales del sitio web de MeCoSim, como punto de partida para todos aquellos a los que puedan ser de utilidad la información y las herramientas desarrolladas, como lo fueron para mí aquellas desarrolladas por otros. Se trata únicamente de un resumen de la funcionalidad, con el mero ánimo de ilustrar sobre los contenidos que se pueden encontrar en la citada referencia, pero pretende ser un lugar de encuentro en constante actualización, en toda la medida posible, siempre que se considere que puede ser una herramienta útil para desarrolladores de software, diseñadores de sistemas P o usuarios finales de cualesquiera modelos basados en sistemas P para los que el entorno desarrollado pueda aportar un valor.

El software empleado por MeCoSim

- Plataforma y lenguaje Java. <http://www.java.com/>
- Eclipse IDE. <http://www.eclipse.org/>
- Netbeans IDE. <https://netbeans.org/>
- Framework P-Lingua (pLinguaCore). <http://www.p-lingua.org/>
- Biblioteca Colt. <http://acs.lbl.gov/hoschek/colt/>
- Biblioteca Jdom. <http://www.jdom.org/>
- Jcommon. <http://www.jfree.org/jcommon/>
- Biblioteca Jfreechart. <http://www.jfree.org/jfreechart/>
- Java Swing (componentes visuales).
<http://docs.oracle.com/javase/tutorial/uiswing/>
- Visual Swing for Eclipse. <https://code.google.com/p/visualswing4eclipse/>
- Apache POI. <http://poi.apache.org/>
- Substance Look & Feel.
<http://grepcode.com/snapshot/repo1.maven.org/maven2/com.github.insubstantial/substance/7.1>
- SQLite. <http://www.sqlite.org/>
- HSQLDB. <http://hsqldb.org/>
- EclipseLink. <http://www.eclipse.org/eclipselink/>
- Apache Commons. <http://commons.apache.org/>
- Log4j. <http://logging.apache.org/log4j/2.x/>
- JavaCC. <https://javacc.java.net/>
- ANTLR. <http://www.antlr.org/>
- Daikon invariant detector. <http://plse.cs.washington.edu/daikon/>
- Spin. <http://spinroot.com/spin/whatispin.html>
- GHC. <http://www.haskell.org/ghc/>
- JUNG Java universal Network/Graph framework.
<http://jung.sourceforge.net/>
- Gson. <https://code.google.com/p/google-gson/>
- jQuery. <http://jquery.com/>
- jQuery datatables. <http://www.datatables.net/>
- Highcharts. <http://www.highcharts.com/>
- Twitter Bootstrap. <http://getbootstrap.com/>

- Bootbox. <http://bootboxjs.com/>
- Bootstrap slider. <http://www.eyecon.ro/bootstrap-slider/>
- FileReader.js. <http://bgrins.github.io/filereader.js/>
- FileSaver.js. <https://github.com/eligrey/FileSaver.js/>
- JsonEditor. <http://jsoneditoronline.org/>
- IzPack. <http://izpack.org/>
- DCP Setup Maker. <http://sourceforge.net/projects/devcompack/>
- Getdown. <https://github.com/threerings/getdown/>
- Sphinx web doc generation. <http://sphinx-doc.org/>

El sitio web de MeCoSim

Durante el transcurso de esta tesis, como objetivos complementarios al diseño de la metodología propuesta, el desarrollo del entorno visual de simulación y la aplicación de ambos a casos de estudio de la vida real y problemas NP-completos, se ha tratado de sentar las bases para la adecuada difusión del software desarrollado y los casos de aplicación. Para ello, se ha puesto a disposición el sitio web de MeCoSim, que pretende erigirse en el lugar de encuentro para la comunidad de usuarios actuales y potenciales del entorno de simulación.

Los aspectos fundamentales que pretende recopilar este sitio web son los siguientes:

- Descarga e instalación del software, basado en Java 1.7, disponible para plataformas basadas en Windows, Unix y Mac OS X. La instalación se lanza automáticamente desde el botón **Install** de la web (ver figura A.1), muestra un asistente para la instalación que instala la aplicación de forma rápida y sencilla pulsando sucesivamente *Siguiente* en cada pantalla y crea en el escritorio del usuario un acceso directo que queda disponible para el lanzamiento de MeCoSim.

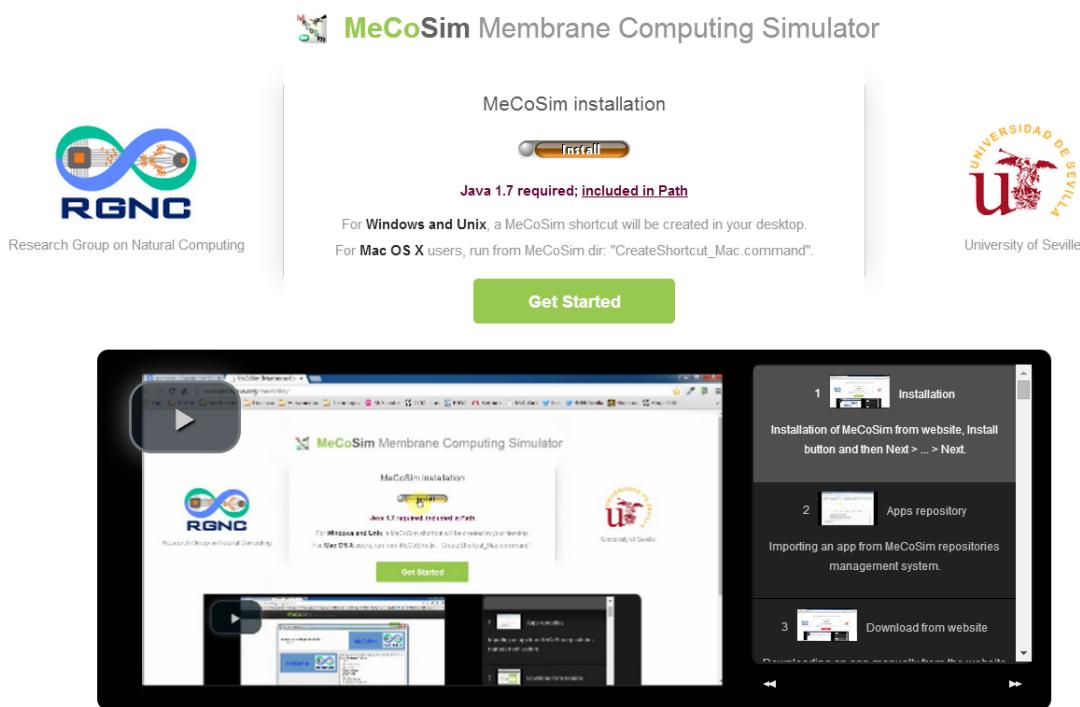


Figura A.1: Sitio web de MeCoSim - Instalación

- Descripción general del entorno de simulación, principales características y datos más relevantes.

Overview

MeCoSim (Membrane Computing Simulator) is a software that offers the users a *General Purpose Application* to *model, design, simulate, analyze and verify* different types of models based on **P systems**. Some of the main features of MeCoSim are the following:

- **Simulation** of models of P systems under different initial conditions. It enables the load of P-Lingua based models, parsing, edition, **debugging**, and different simulation types.
- **Visualization** capabilities for **analyzing** P systems: *alphabet, membrane structure, multisets and graphs* viewers.
- **Highly customizable** platform for defining **inputs, outputs**, parameters and graphs for each model of a family of P systems.
- **Repositories** system for the visual management of available online resources, including **plugins**, custom applications, **models** and scenarios.
- **Export** option for releasing **end-user applications** for solving practical problems, abstracting P system functionalities.
- **Plugins architecture**, permitting the extension of the functionality with Java jars or external non-Java programs. MeCoSim is written in Java.
- **Auto-update** capability, using the latest release of the program whenever it runs.

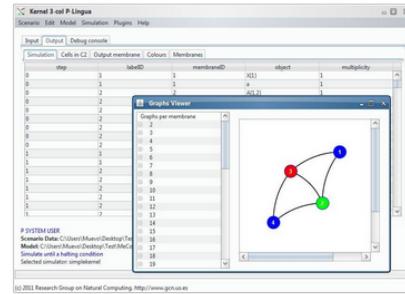


Figura A.2: Sitio web de MeCoSim - Características

- Documentación para empezar a usar **MeCoSim**, incluyendo manual de usuario, guía de configuración de aplicaciones a medida, artículo inicial sobre el software y un buen número de vídeos ilustrando las principales funcionalidades de las herramientas disponibles en el programa.

Getting started

The best resources to start using MeCoSim are included in this user guide, including a number of useful **videos** for illustrating the main features and functionalities:

- User guide
 - Quick start
 - Installation and first use
 - Custom apps
 - Models
 - Scenarios
 - Working with models and simulations
 - Debug
 - Simulation
 - Repositories management
 - Plugins
 - Help - About
 - Playlist
 - User guide videos playlist
- Foundational paper, to illustrate the initial goal and philosophy of MeCoSim.
- User manual, showing the basic features and operation in MeCoSim.
- MeCoSim Quick MeCoSim Apps Development Guide, contribution from Luis Felipe Macías providing a quick guide to develop new MeCoSim-based applications by means of customization.



Figura A.3: Sitio web de MeCoSim - Documentación

- Casos de estudio, categorizados por tipos de sistemas P, incluyendo sistemas P que trabajan a modo de células (con ejemplos de transición y membranas activas), a modo de tejidos (con ejemplos con reglas de división y con reglas de separación celular), a modo de neuronas, simple kernel P systems y un buen número de casos de estudio en el ámbito de los sistemas P de Dinámica de Poblaciones.

Para cada ejemplo, además de su descripción general, se proporciona una serie de ficheros para trabajar con el ejemplo en MeCoSim. Estos ficheros incluyen archivos de aplicación (.xls, con la definición de la aplicación basada en MeCoSim, adaptada para el caso de estudio), de modelo/solución (.pli, con el código P-Lingua correspondiente al diseño realizado), y archivos de escenario (.ec2, con los datos de entrada y/o salida de cada escenario a estudiar por el usuario).

Case studies

MeCoSim development started in 2010, and a number of case studies have been analyzed since then, covering many variants of P systems, different areas of interest and application domains. This section tries to collect a non-exhaustive but as complete as possible of those case studies, grouped by areas. The examples contain, at least, some snippets of code, along with the minimal needed files to run some scenario in MeCoSim. The number of scenarios and the level of detail may vary among the different case studies, possibly including detailed descriptions, references to related publications, charts and videos.

- Case studies
 - Cell like P systems
 - Tissue like P systems
 - Spiking neural P systems
 - Population Dynamics P systems
 - Simple Kernel P systems
 - Others

If you have developed some case study and you have information please let us know (valencia@us.es) and we will be happy to include your work in our repository.

Figura A.4: Sitio web de MeCoSim - Casos de estudio

Bibliografía

- [1] Bearded vulture c++ ad-hoc simulator. <http://www.gcn.us.es/?q=node/338>.
- [2] Gnu gpl license. <http://www.gnu.org/licenses/gpl.html>.
- [3] HsqlDb sql syntax web page. <http://www.hsqldb.org/doc/guide/ch09.html>.
- [4] The mecosim quick application developing reference guide. http://www.p-lingua.org/mecosim/doc/_downloads/MeCoSimQuickGuide.pdf.
- [5] Nvidia cuda c programming guide 4.2. [http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA\\$\\$_C\\$_\\$Programming\\$_\\$Guide.pdf](http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA$$_C$_$Programming$_$Guide.pdf).
- [6] Opencl standard webpage. <http://www.khronos.org/opencl>.
- [7] The openmp api specification for parallel programming, official website. <http://www.openmp.org>.
- [8] P-lingua supported models variants. http://www.p-lingua.org/wiki/index.php/Supported_models.
- [9] Spin web site. <http://www.spinroot.com/>.
- [10] The stochastic pi-machine. <http://www.doc.ic.ac.uk/~{}anp/spim/>.
- [11] Gypaetus barbatus (Bearded Vulture) european commission web site. http://ec.europa.eu/environment/nature/conservation/wildbirds/threatened/g/gypaetus_barbatus_en.htm, Feb. 2014.
- [12] H. M. A. Arkin, J. Ross. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells. *Genetics*, 149(4):1633–1648, 1998.

- [13] J. Ackerman and R. Claudi. The early life history of zebra mussels: Overwintering juveniles and post-settlement movements. *Journal of Shellfish Research*, 1(11):217, 1991.
- [14] J. D. Ackerman, B. Sim, S. J. Nichols, and R. Claudi. A review of the early life history of zebra mussels (*dreissena polymorpha*): comparisons with marine bivalves. *Canadian journal of zoology (Canada)*, Canadian journal of zoology (Canada). Jul 1994. v. 72(7) p. 1169-1179; 1994-07-01.
- [15] L. Adleman. Molecular computations of solutions to combinatorial problems. *Science*, 226:1021–1024, 1994.
- [16] A. Alhazov and D. Sburlan. Static Sorting P Systems. In G. Ciobanu, G. Păun, and M. Pérez-Jiménez, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 215–252. Springer Berlin Heidelberg, 2006.
- [17] M. Ben-Ari. *Principles of the Spin Model Checker*. Springer-Verlag, London, 2008.
- [18] D. Besozzi and G. Ciobanu. A P systems description of the sodium-potassium pump. In *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, Italy, June 2004*, pages 154–160, Milano, Italy, June 2004.
- [19] U. S. Bhalla and R. Iyengar. Emergent properties of networks of biological signaling pathways. *Science*, 283(5400):381–387, 1999.
- [20] R. Blossey, L. Cardelli, and A. Phillips. A compositional approach to the stochastic dynamics of gene networks. In C. Priami, L. Cardelli, and S. Emmott, editors, *Transactions on Computational Systems Biology IV*, volume 3939 of *Lecture Notes in Computer Science*, pages 99–122. Springer Berlin Heidelberg, 2006.
- [21] J. Borcherding. The annual reproductive cycle of the freshwater mussel *dreissena polymorpha pallas* in lakes. *Oecologia*, 87(2):208–218, 1991.
- [22] J. Borcherding. Morphometric changes in relation to the annual reproductive cycle in *dreissena polymorpha* - a prerequisite for biomonitoring studies with zebra mussels. In D. N. . H. Jenner, editor, *The Zebra Mussel *Dreissena polymorpha*: Ecology, Biological Monitoring and First Applications in the Water Quality Management*, pages 89–99. Gustav Fischer Verlag. Stuttgart, Jena, New York., 1992.
- [23] G. Bravo, L. Fernández, and M. A. Pena. Hierarchical master-slave architecture for membrane systems implementation. In M. Sugisaka and H. Tanaka, editors, *13th International Symposium on Artificial Life and Robotics*, pages 485–490, 2008.

- [24] F. Cabarle, H. Adorna, and M. A. M. del Amor. A spiking neural p system simulator based on cuda. *Lecture Notes in Computer Science*, 7184:87–103, 2012.
- [25] F. C. Cabarle, H. N. Adorna, M. A. M. del Amor, and M. J. Pérez-Jiménez. Improving gpu simulations of spiking neural p systems. *Romanian Journal of Information Science and Technology*, 15:5–20, 06/2012 2012.
- [26] M. Cardona, M. A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and D. Sanuy. A computational modeling for real ecosystems based on p systems. *Natural Computing*, 2010.
- [27] M. Cardona, M. A. Colomer, A. Margalida, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and D. Sanuy. A p system based model of an ecosystem of some scavenger birds. In *Membrane Computing*, volume 5957 of *Lecture Notes in Computer Science*, pages 182–195. Springer Berlin / Heidelberg, 01/2010 2010. 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009, Revised Selected and Invited Papers.
- [28] M. Cardona, M. A. Colomer, M. J. Pérez-Jiménez, D. Sanuy, and A. Margalida. A p system modeling an ecosystem related to the bearded vulture. *6th Brainstorming Week on Membrane Computing*, 6:51–66, 2008.
- [29] M. Cardona, M. A. Colomer, M. J. Pérez-Jiménez, D. Sanuy, and A. Margalida. Modeling ecosystems using p systems: the bearded vulture, a case study. In D. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 137–156. Springer Berlin / Heidelberg, Berlín, 01/2009 2009.
- [30] A. Castellini and V. Manca. MetaPlab: A computational framework for metabolic P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing: 9th International Workshop*, volume 5391 of *Lecture Notes in Computer Science*, pages 157–168, 2009.
- [31] R. Ceterchi and C. Martín-Vide. P systems with Communication for Static Sorting. In M. Cavaliere, C. Martín-Vide, and G. Păun, editors, *Pre-Proceedings of Brainstorming Week on Membrane Computing, Tarragona, February 2003*, pages 101–117, Tarragona, February 5-11 2003.
- [32] D. D. Chamberlin and R. F. Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, SIGFIDET '74, pages 249–264, New York, NY, USA, 1974. ACM.

- [33] M. E. Chase and R. C. Bailey. The ecology of the zebra mussel (*dreissena polymorpha*) in the lower great lakes of north america: I. population dynamics and growth. *Journal of Great Lakes Research*, 25(1):107–121, 1999.
- [34] S. Cheruku, A. Paun, F. J. Romero-Campero, M. J. Pérez-Jiménez, and O. H. Ibarra. Simulating fas-induced apoptosis by using p systems. *Progress in Natural Science*, 17:424–431, 2007.
- [35] G. Ciobanu and G. Wenyuan. P systems running on a cluster of computers. In C. Martín-Vide, G. Mauri, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 2933 of *Lecture Notes in Computer Science*, pages 123–139. Springer Berlin Heidelberg, 2004.
- [36] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187 – 243, 2002. Rewriting Logic and its Applications.
- [37] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.
- [38] M. A. Colomer, A. Margalida, and M. J. Pérez-Jiménez. Population dynamics p system (pdp) models: A standardized protocol for describing and applying novel bio-inspired computing tools. *PLoS ONE*, 4:e60698, 04 2013.
- [39] N. A. Connelly, C. R. J. O'Neill, B. A. Knuth, and T. Brown. Economic impacts of zebra mussels on drinking water treatment and electric power generation facilities. *Environmental Management*, 40(1):105–112, 2007.
- [40] J. A. Daraio, L. J. Weber, T. J. Newton, and J. M. Nestler. A methodological framework for integrating computational fluid dynamics and ecological models applied to juvenile freshwater mussel dispersal in the upper mississippi river. *Ecological Modelling*, 221(2):201–214, 1/24 2010.
- [41] M. A. Davis. *Invasion Biology*. OUP Oxford, Oxford, GBR, 200901 2009. ID: 10346532.
- [42] M. A. M. del Amor, I. Pérez-Hurtado, M. García-Quismondo, L. F. Macías-Ramos, L. Valencia-Cabrera, Á. R. Jiménez, C. G. Díaz, A. Riscos-Núñez, M. A. Colomer, and M. J. Pérez-Jiménez. Dcba: Simulating population dynamics p systems with proportional object distribution. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, and G. Vaszil, editors, *Membrane Computing*, volume 7762 of *Lecture Notes in Computer Science*, pages 257–276. Springer Berlin Heidelberg, Amsterdam, The Netherlands, 2013.

- [43] M. A. M. del Amor, I. Pérez-Hurtado, M. García-Quismondo, L. F. Macías-Ramos, L. Valencia-Cabrera, Á. Romero-Jiménez, C. Graciani-Díaz, A. Riscos-Núñez, M. A. Colomer, and M. J. Pérez-Jiménez. Dcba: Simulating population dynamics p systems with proportional object distribution. In M. García-Quismondo, L. F. Macías-Ramos, I. Pérez-Hurtado, G. Păun, and L. Valencia-Cabrera, editors, *Tenth Brainstorming Week on Membrane Computing*, volume II, pages 27–56, Seville, Spain, 02/2012 2012. Fénix Editora.
- [44] M. A. M. del Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez, and M. A. Colomer. A new simulation algorithm for multienvironment probabilistic p systems. In R. L. A. K. N. R. T. K. Li, Z. Tang, editor, *IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010)*, volume 1, pages 59–68, Changsha, China, 09/2010 2010. IEEE Press.
- [45] M. A. M. del Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez, and F. Sancho-Caparrini. A simulation algorithm for multienvironment probabilistic p systems: A formal verification. *International Journal of Foundations of Computer Science*, 22(1):107–118, 2011.
- [46] D. Díaz-Pernil, C. Graciani-Díaz, M. A. Gutiérrez-Naranjo, I. Pérez-Hurtado, and M. J. Pérez-Jiménez. Software for p systems. *The Oxford Handbook of Membrane Computing*, pages 437–454, 2010.
- [47] D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A linear-time tissue p system based solution for the 3-coloring problem. *Electronic Notes in Theoretical Computer Science*, 171:81–93, 2007.
- [48] D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A uniform family of tissue p systems with cell division solving 3-col in a linear time. *Theoretical Computer Science*, 404:76–87, 2008.
- [49] D. Díaz-Pernil, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A P-Lingua programming environment for membrane computing. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers*, volume 5391 of *Lecture Notes in Computer Science*, pages 187–203. Springer, 2009.
- [50] L. Dib, Z. Guessoum, N. Bonnet, and M. Laskri. Multi-agent system simulating tumoral cells migration. In S. Zhang and R. Jarvis, editors, *AI 2005: Advances in Artificial Intelligence*, volume 3809 of *Lecture Notes in Computer Science*, pages 624–632. Springer Berlin Heidelberg, 2005.

- [51] L. Diez Dolinski, R. Núñez Hervás, M. Cruz Echeandía, and A. Ortega. Distributed simulation of p systems by means of map-reduce: First steps with hadoop and p-lingua. In J. Cabestany, I. Rojas, and G. Joya, editors, *Advances in Computational Intelligence*, volume 6691 of *Lecture Notes in Computer Science*, pages 457–464. Springer Berlin Heidelberg, 2011.
- [52] J. Dolz and J. Armengol. Estudio de la dinámica sedimentaria y batimetría de precisión del embalse de riba-roja d’ebre. Informe técnico de Flumen (Dinámica Fluvial y Enginyeria Hidrológica) para la Confederación Hidrográfica del Ebro, 2009.
- [53] C. Durán, M. Lanao, A. Anadón, and V. Touyá. Management strategies for the zebra mussel invasion in the ebro river basin. *Aquatic Invasions*, 5(3):309, 2010.
- [54] C. Durán, M. Lanao, L. P. y Pérez, C. Chica, A. Anadón, and V. Touya. Estimación de los costes de la invasión del mejillón cebra en la cuenca del ebro (periodo 2005-2009). *Limnetica*, 31(2):213–230, 2012.
- [55] 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.
- [56] L. Fernandez, V. Martinez, F. Arroyo, and L. Mingo. A hardware circuit for selecting active rules in transition p systems. In *in G. Ciobanu, Gh. Paun, Pre-Proc. of First International Workshop on Theory and Application of P Systems, Timisoara, Romania, September 26-27*, pages 45–48, 2005.
- [57] T. E. Foundation. Eclipse IDE. <http://www.eclipse.org/>, 2004.
- [58] 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.
- [59] M. García-Quismondo, L. F. Macías-Ramos, and M. J. Pérez-Jiménez. Implementing enzymatic numerical p systems for ai applications by means of graphic processing units. *Beyond Artificial Intelligence: Contemplations, Expectations, Applications*, pages 137–157, 10/2012 2013.
- [60] R. Gershoni, E. Keinan, G. Păun, R. Piran, T. Ratner, and S. Shoshani. Research topics arising from the (planned) P systems implementation experiment in Technion. In D. Díaz-Pernil, C. Graciani, M. A. Gutiérrez-Naranjo, G. Păun, I. Pérez-Hurtado, and A. Riscos-Núñez, editors, *Sixth Brainstorming Week on Membrane Computing*, pages 183–192, 2008.

- [61] M. Gheorghe, F. Istrate, C. Dragomir, L. Mierla, L. Valencia-Cabrera, M. García-Quismondo, and M. J. Pérez-Jiménez. Kernel p systems - version i. *Eleventh Brainstorming Week on Membrane Computing (11BWMC)*, pages 97–124, 08/2013 2013.
- [62] M. Gheorghe, F. Istrate, R. Lepticaru, 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.
- [63] M. Gheorghe, C. Martín-Vide, V. Mitrana, and M. J. Pérez Jiménez. An agent based approach of collective foraging. In J. Mira and J. Ávarez, editors, *Computational Methods in Neural Modeling*, volume 2686 of *Lecture Notes in Computer Science*, pages 638–645. Springer Berlin Heidelberg, 2003.
- [64] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.
- [65] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- [66] P. J. E. Goss and J. Peccoud. Quantitative modelling of stochastic systems in molecular biology by using Petri Nets. *Proceedings of the National Academy of Sciences of the United States of America*, 95:6750–6754, 1998.
- [67] M. P. A. Group. Daikon web page.
- [68] M. P. A. Group. The daikon invariant detector user manual, 2010.
- [69] A. Gutiérrez and S. Alonso. P systems: from theory to implementation. In Z. Zhao, editor, *Sequence and Genome Analytics. Methods and Applications*, pages 205–226. Concept Press Ltd, 2010.
- [70] A. Gutiérrez, L. Fernández, F. Arroyo, and S. Alonso. Hardware and software architecture for implementing membrane systems: A case of study to transition p systems. In M. Garzon and H. Yan, editors, *DNA Computing*, volume 4848 of *Lecture Notes in Computer Science*, pages 211–220. Springer Berlin Heidelberg, 2008.
- [71] S. Hallstan, U. Grandin, and W. Goedkoop. Current and modeled potential distribution of the zebra mussel (*dreissena polymorpha*) in sweden. *Biological Invasions*, 12(1):285, In: Biological Invasions. (Biological Invasions, January 2009, 12(1):285-296); 2009-01-01.

- [72] S. N. Higglins and M. J. V. Zanden. What a difference a species makes: A meta-analysis of dreissenid mussel impacts on freshwater ecosystems. *Ecological Monographs*, 80(2):179, In: Ecological Monographs. (Ecological Monographs, May 2010, 80(2):179-196); 2010-05-01.
- [73] M. Holcombe, L. Holcombe, M. Gheorghe, and N. Talbot. A hybrid machine model of rice blast fungus, magnaporthe grisea. *Biosystems*, 68(2–3):223 – 228, 2003. {IPCAT} 01 SI.
- [74] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.
- [75] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 1 edition, Sept. 2003.
- [76] F. Istrate, R. Lefticaru, L. Mierla, L. V. Cabrera, H. Han, G. Zhang, C. Dragomir, and M. J. P. Jiménez. Kernel p systems: Applications and implementations. *Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, 2013, 212:1081–1089, 07/2013 2013.
- [77] F. Istrate, R. Lefticaru, and C. Tudose. Formal verification of P systems using Spin. *International Journal of Foundations of Computer Science*, 22(1):133–142, 2011.
- [78] D. Jackson, M. Holcombe, and F. Ratnieks. Coupled computational simulation and empirical research into the foraging system of pharaoh's ant (monomorium pharaonis). *Biosystems*, 76(1–3):101 – 112, 2004. Papers presented at the Fifth International Workshop on Information Processing in Cells and Tissues.
- [79] D. E. Jackson, M. Holcombe, and F. L. W. Ratnieks. Trail geometry gives polarity to ant foraging networks. *Nature*, 432:907–909, 2004.
- [80] H. A. Jenner, J. W. Whitehouse, C. J. Taylor, and M. Khalanski. Cooling water management in European power stations Biology and control of fouling. *Hydroécologie Appliquée*, 10:I–225, 1998.
- [81] C. J. JOHNSON and M. P. GILLINGHAM. An evaluation of mapped species distribution models used for conservation planning. *Environmental Conservation*, null:117–128, 6 2005.
- [82] R. A. Juayong, F. G. Cabarle, H. N. Adorna, and M. A. M. del Amor. On the simulations of evolution-communication p systems with energy without antiport rules for gpus. *Tenth Brainstorming Week on Membrane Computing*, I:267–290, 02/2012 2012.

- [83] A. Karatayev, L. Burlakova, and D. Padilla. Impacts of zebra mussels on aquatic communities and their role as ecosystem engineers. In E. Leppäkoski, S. Gollasch, and S. Olenin, editors, *Invasive Aquatic Species of Europe. Distribution, Impacts and Management*, pages 433–446. Springer Netherlands, 2002.
- [84] A. Y. Karatayev, L. E. Burlakova, and D. K. Padilla. Growth rate and longevity of dreissena polymorpha (pallas): a review and recommendations for future study. *Journal of Shellfish Research*, 25(1):23–32, 04/01; 2014/07 2006. doi: 10.2983/0730-8000(2006)2523:GRALOD]2.0.CO;2; 26.
- [85] D. Kirk and W. mei Hwu. *Programming massively parallel processors :a hands-on approach*. Morgan Kaufmann Publishers, Burlington Massachusetts, 2010. David B. Kirk and Wen-mei W. Hwu; :il. ;24 cm; MSC 68Nxx, 68N19; Bibliografia. Índice; 1. Introduction 2. History of GPU computing 3. Introduction to CUDA 4. CUDA threads 5. CUDA memories 6. Performance considerations 7. Floating point considerations 8. Application case study: advanced MRI reconstruction 9. Application case study: molecular visualization and analysis 10. Parallel programming and computational thinking 11. A brief introduction to OpenCL 12. Conclusion and future outlook. A. Matrix multiplication host-only version source code. B. GPU compute capabilities. Index.
- [86] B. R. Labajos and K. Dittmer. Socio-environmental impacts of the invasive species dreissena polymorpha (zebra mussel) in the ebro river. In *8th International Conference of the European Society for Ecological Economics*, Transformation, Innovation and Adaptation for Sustainability, 2009.
- [87] R. Lefticaru, F. Ipaté, L. Valencia Cabrera, A. Turcanu, C. Tudose, M. Gheorghe, M. J. Pérez-Jiménez, I. M. Niculescu, and C. Dragomir. Towards an integrated approach for model simulation, property extraction and verification of P systems. In *Proceedings of 10th Brainstorming Week on Membrane Computing*, pages 291–318, 2012.
- [88] R. Lefticaru, C. Tudose, and F. Ipaté. Towards automated verification of P systems using Spin. *International Journal of Natural Computing Research*, 2(3):1–12, 2011.
- [89] J. B. Lingrel and T. Kuntzweiler. Na⁺,k(+)-atpase. *Journal of Biological Chemistry*, 269(31):19659–62, 1994.
- [90] M. L. Ludyanskiy, D. McDonald, and D. MacNeill. Impact of the zebra mussel, a bivalve invader: Dreissena polymorpha is rapidly colonizing hard surfaces throughout waterways of the united states and canada. *BioScience*, 43(8):533–544, 1993.

- [91] G. Mackie, W. Gibbons, B. Muncaster, and I. Gray. The zebra mussel, dreissena polymorpha, a synthesis of european experiences and a preview for north america. Technical report, Queen's Printer for Ontario, 1989.
- [92] R. Margalef. *Ecología*. Omega, Barcelona, 1974. Ramón Margalef, 10 reimpresiones hasta 2005.
- [93] C. Maxfield. *The design warrior's guide to FPGAs :devices, tools and flows*. Newnes etc, Boston, MA etc, 2004. Clive "Max" Maxfield.; :il. ;24 cm. +CD-ROM.
- [94] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [95] R. McMahon. Impact of european zebra mussel infestation to the electric power industry. *Proceedings of the American Power*, page 988, 1990.
- [96] T. C. Meng, S. Somani, and P. Dhar. Modeling and simulation of biological systems with stochasticity. *In Silico Biology*, 4:293–309, 2004.
- [97] R. Milner. *Communicating and Mobile Systems: The pi-calculus*. Cambridge University Press, New York, NY, USA, 1999.
- [98] D. Minchin, F. Lucy, and M. Sullivan. Ireland: a new frontier for the zebra mussel dreissena polymorpha(pallas). *Oceanological and Hydrobiological Studies*, 34(1):19–30, 2005.
- [99] Y. Morales, L. J. Weber, A. E. Mynett, and T. J. Newton. Mussel dynamics model: A hydroinformatics tool for analyzing the effects of different stressors on the dynamics of freshwater mussel communities. *Ecological Modelling*, 197(3-4):448–460, / 08 / 25 / 2006. ID: edselc.2-52.0-33745845268; M2: 448; Accession Number: edselc.2-52.0-33745845268; (Ecological Modelling, 25 August 2006, 197(3-4):448-460) Publication Type: Academic Journal; Rights: Copyright 2008 Elsevier B.V., All rights reserved.
- [100] A. Munshi, B. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, 1st edition, 2011.
- [101] T. M. H. nad Jon M. Grennan and T. K. McCarthy. Effects of recent zebra mussel invasion on water chemistry and phytoplankton production in a small irish lake. *Aquatic Invasions*, 3(1), 2008.
- [102] E. Navarro, M. Bacardit, L. Caputo, T. Palau, and J. Armengol. Limnological characterization and flow patterns of a three-coupled reservoir system and their influence on dreissena polymorpha populations and settlement during the stratification period. *Lake and Reservoir Management*, 22(4):293–302, 2006.

- [103] E. H. V. Nes and M. Scheffer. A strategy to improve the contribution of complex simulation models to ecological theory. *Ecological Modelling*, 185(24):153–164, 7/10 2005.
- [104] M. Àngels Colomer, A. Margalida, L. Valencia, and A. Palau. Application of a computational model for complex fluvial ecosystems: The population dynamics of zebra mussel *dreissena polymorpha* as a case study. *Ecological Complexity*, 20(0):116–126, 12 2014.
- [105] V. Nguyen, D. Kearney, and G. Gioiosa. Balancing performance, flexibility, and scalability in a parallel computing platform for membrane computing applications. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 385–413. Springer Berlin Heidelberg, 2007.
- [106] V. Nguyen, D. Kearney, and G. Gioiosa. An extensible, maintainable and elegant approach to hardware source code generation in Reconfig-P. *The Journal of Logic and Algebraic Programming*, 79(6):383–396, Aug. 2010.
- [107] V. Nguyen, D. Kearney, and G. Gioiosa. A region-oriented hardware implementation for membrane computing applications. In G. Păun, M. Pérez Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5957 of *Lecture Notes in Computer Science*, pages 385–409. Springer Berlin Heidelberg, 2010.
- [108] R. Niculescu, M. J. Dinneen, and Y.-B. Kim. Structured modelling with hyperdag p systems: Part a. In *Membrane Computing, Seventh Brainstorming Week, BWMC 2009, Sevilla, Spain, February 2009*, pages 85–107, 2009.
- [109] C. R. O'Neill. *The Zebra Mussel. Impacts and Control*, volume Cornell University, State University of New York, Cornell Cooperative Extension. Information Bulletin. 1996.
- [110] M. Oreska and D. Aldridge. Estimating the financial costs of freshwater invasive species in great britain: a standardized approach to invasive species costing. *Biological Invasions*, 13(2):305–319, 2011.
- [111] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.
- [112] A. Palau, I. Cía, D. Fargas, M. Bardina, and S. Massuti. *Resultados preliminares sobre ecología básica y distribución del Mejillón cebra en el embalse de Riba-roja (Río Ebro)*. Monografía de ENDESA, 2003.

- [113] L. Pan, G. Paun, and M. de J. Pérez-Jiménez. Spiking neural p systems with neuron division and budding. *Science China. Information Sciences*, 8:1596–1607, 08/2011 2011.
- [114] L. Pan and M. J. Pérez-Jiménez. Efficiency of tissue p systems with cell separation. volume II, pages 169–196, Sevilla, España, 02/02/2009 2009. Fénix Editora.
- [115] D. A. Pathy. 1994. the life history and demography of zebra mussel, dreissena polymorpha, populations in lake st. clair, lake erie, and lake ontario. Master’s thesis, University of Guelph, Guelph, Ontario., 1994.
- [116] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. and *Turku Center for Computer Science-TUCS Report No 208*.
- [117] G. Paun, M. J. Pérez-Jiménez, and A. Riscos-Núñez. Tissue p systems with cell division. *International Journal of Computers, Communications & Control*, III:295–303, 2008.
- [118] G. Păun, G. Rozenberg, and A. Salomaa. Membrane computing with external output. *Fundamenta Informaticae*, 41(3):313–340, February 2000.
- [119] F. Peña-Cantillana, D. Díaz-Pernil, H. A. Christinal, and M. A. Gutiérrez-Naranjo. Implementation on cuda of the smoothing problem with tissue-like p systems. *International Journal of Natural Computing Research*, 2:25–34, 2011.
- [120] I. Pérez-Hurtado. P-Lingua webpage. <http://www.p-lingua.org>, June 2009.
- [121] I. Pérez-Hurtado, L. Valencia-Cabrera, J. M. Chacón, A. Riscos-Núñez, and M. J. Pérez-Jiménez. A p-lingua based simulator for tissue p systems with cell separation. *Romanian Journal of Information Science and Technology*, 17(1):89–102, 2014.
- [122] 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-inspired Computing: Theories and Applications (BIC-TA 2010)*, I:637–643, 2010.
- [123] M. J. Pérez-Jiménez and F. J. Romero-Campero. A study of the robustness of the egfr signalling cascade using continuous membrane systems. *Lecture Notes in Computer Science*, 3561:268–278, 2005.

- [124] M. J. Pérez-Jiménez, Á. Romero-Jiménez, and F. Sancho-Caparrini. A polynomial complexity class in p systems using membrane division. *Fifth International Workshop on Descriptive Complexity of Formal Systems*, pages 284–294, 12-14/07/2003 2003.
- [125] M. J. Pérez-Jiménez and P. Sosik. Improving the efficiency of tissue p systems with cell separation. *Tenth Brainstorming Week on Membrane Computing*, II:105–140, 02/2012 2012.
- [126] B. Petreska and C. Teuscher. A reconfigurable hardware membrane system. In C. Martín-Vide, G. Mauri, G. Păun, GrzegorzRozenberg, and A. Salomaa, editors, *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July, 17-22, 2003, Revised Papers*, volume 2933 of *Lecture Notes in Computer Science*, pages 269–285. Springer, July 2003.
- [127] A. Phillips and L. Cardelli. A correct abstract machine for the stochastic pi-calculus. In *In Bioconcur'04. ENTCS*, 2004.
- [128] D. Pimentel, R. Zuniga, and D. Morrison. Update on the environmental and economic costs associated with alien-invasive species in the United States. *Ecological Economics*, 52(3):273–288, Feb. 2005.
- [129] M. Pogson, R. Smallwood, E. Qwarnstrom, and M. Holcombe. Formal agent-based modelling of intracellular chemical interactions. *Biosystems*, 85(1):37 – 45, 2006. Dedicated to the Memory of Ray Paton.
- [130] C. Prabhu. *Grid and Cluster Computing*. Prentice-Hall of India Pvt.Ltd, 2008.
- [131] I. Pérez-Hurtado. *Desarrollo y aplicaciones de un entorno de programación para Computación Celular: P-Lingua*. PhD thesis, Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, 2010.
- [132] C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25 – 31, 2001. Process Algebra.
- [133] A. Păun, M. Pérez-Jiménez, and F. J. Romero-Campero. Modeling signal transduction using p systems. In H. Hoogeboom, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 2006, Revised, Selected, and Invited Papers.*, volume 4361 of *Lecture Notes in Computer Science*, pages 100–122. Springer Berlin Heidelberg, 2006.
- [134] C. R. and M. G.L. *Practical manual for zebra mussel monitoring and control*. 01/01/1993; 1993-01-01.

- [135] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.
- [136] V. N. Reddy, M. N. Liebman, and M. L. Mavrovouniotis. Qualitative analysis of biochemical reaction systems. *Computers in Biology and Medicine*, 26(1):9 – 24, 1996.
- [137] A. Regev and E. Shapiro. Cellular abstractions: Cells as computation. *Nature*, 419(6905):343, Sept. 2002.
- [138] A. Regev and E. Shapiro. The π -calculus as an abstraction for biomolecular systems. In G. Ciobanu and G. Rozenberg, editors, *Modelling in Molecular Biology*, Natural Computing Series, pages 219–266. Springer Berlin Heidelberg, 2004.
- [139] F. J. Romero-Campero and M. J. Pérez-Jiménez. A model of the quorum sensing system in vibrio fischeri using p systems. *Artificial Life*, 14:95–109, 2008.
- [140] F. Sanz-Ronda, S. López-Sáenz, R. San-Martín, and A. Palau-Ibars. Physical habitat of zebra mussel (*dreissena polymorpha*) in the lower ebro river (northeastern spain): influence of hydraulic parameters in their distribution. *Hydrobiologia*, 735(1):137–147, 2014.
- [141] D. W. Schneider, C. D. Ellis, and K. S. Cummings. A transportation model assessment of the risk to native mussel communities from zebra mussel spread. *Conservation Biology*, 12(4):788–800, 1998.
- [142] B. Shaalan and R. Muniyandi. Implementing mitogen activated protein kinases cascade on membrane computing using p-lingua. 11(1):178–187, 2015.
- [143] I.-I. Sivtrr. Shell growth of the zebra mussel (*dreissena polymorpha* (pallas)) in relation to selected physico-chemical parameters in the lower rhine and some associated lakes. 1992.
- [144] M. Sprung. Observations on shell growth and mortality of *dreissena polymorpha* in lakes. In *The Zebra Mussel Dreissena polymorpha: Ecology, Biological Monitoring and First Applications in the Water Quality Management*, volume 4 of *Limnologie Aktuell*, pages 19–28. Gustav Fischer Verlag. Stuttgart, Jena, New York., 1992.
- [145] M. Sprung. The other life: an account of present knowledge of the larval phase of *dreissena polymorpha*. In T. Nalepa and D. Schloesser, editors, *Zebra Mussels: Biology, Impacts and Control*, pages 39–53. Lewis Publishers, Chelsea, London, 1993.

- [146] D. Strayer and L. Smith. Relationships between zebra mussels (*dreissena polymorpha*) and unionid clams during the early stages of the zebra mussel invasion the hudson river. *Freshwater Biology*, 36(3):771–779, 1996.
- [147] D. L. Strayer. Twenty years of zebra mussels: lessons from the mollusk that made headlines. *Frontiers in Ecology*, 7(3):135–141, 2009.
- [148] D. L. Strayer, N. Cid, and H. M. Malcom. Long-term changes in a population of an invasive bivalve and its effects. *Oecologia*, 165(4):1063, In: *Oecologia*. (*Oecologia*, April 2011, 165(4):1063-1072); 2011-04-01.
- [149] D. L. Strayer and H. M. Malcom. Long-term demography of a zebra mussel (*dreissena polymorpha*) population. *Freshwater Biology*, 51(1):117, In: *Freshwater Biology*. (*Freshwater Biology*, January 2006, 51(1):117-130); 2006-01-01.
- [150] A. Syropoulos, E. G. Mamatas, and P. C. A. and Konstantinos T. Sotiriades. A distributed simulation of transition P systems. In C. Martín-Vide, G. Mauri, G. Gheorghe Păun, and A. Salomaa, editors, *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers*, volume 2933 of *Lecture Notes In Computer Science*, pages 357–368. Springer-Verlag Heidelberg, 2004.
- [151] A. Tejedor, L. Fernandez, F. Arroyo, and G. Bravo. An architecture for attacking the bottleneck communication in P systems. In M. Sugisaka and H. Tanaka, editors, *Proceedings of the 12th Int. Symposium on Artificial Life and Robotics*, pages 500–505, Beppu, Oita, Japan, Jan 25-27 2007.
- [152] L. Timar and D. J. Phaneuf. Modeling the human-induced spread of an aquatic invasive: The case of the zebra mussel. *Ecological Economics*, 68(12):3060–3071, 10/15 2009.
- [153] L. Valencia-Cabrera. MeCoSim web site. <http://www.p-lingua.org/mecosim>, July 2010.
- [154] J. V. Vilaplana and L. J. Hushak. *Recreation and the zebra mussel in Lake Erie, Ohio*. Ohio State University, Ohio Sea Grant College Program, Columbus, Ohio, 1994. ID: 33927776.
- [155] D. Walker, J. Southgate, G. Hill, M. Holcombe, D. Hose, S. Wood, S. M. Neil, and R. Smallwood. The epitheliome: agent-based modelling of the social behaviour of cells. *Biosystems*, 76(1–3):89 – 100, 2004. Papers presented at the Fifth International Workshop on Information Processing in Cells and Tissues.
- [156] N. Walz. The energy balance of the freshwater mussel *dreissena polymorpha pallas* in laboratory experiments and in lake constance. ii. reproduction. In *Archives of Hydrobiology*, volume 55. 1978.

- [157] J. M. Ward and A. Ricciardi. Impacts of dreissena invasions on benthic macroinvertebrate communities: a meta-analysis. *Diversity and Distributions*, 13(2):155–165, 2007.
- [158] U. Wilensky. Netlogo. <http://ccl.northwestern.edu/netlogo/>. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL., 1999.
- [159] Y. Wu, S. Bartell, J. Orr, J. Ragland, and D. Anderson. A risk-based decision model and risk assessment of invasive mussels. *ECOLOGICAL COMPLEXITY*, 7(2):243–255, ECOLOGICAL COMPLEXITY; JUN, 2010, 7 2, p243-p255, 13p. Special Issue: SI; 2010-06-01.
- [160] C. Zandron, C. Ferretti, and G. Mauri. Solving NP-complete problems using P systems with active membranes. In I. Antoniou, C. Calude, and M. Dinneen, editors, *Unconventional Models of Computation*, pages 289–301, London, February 2000. Springer-Verlag. Contributed paper.
- [161] X. Zeng, H. Adorna, M. A. M. del Amor, L. Pan, and M. J. Pérez-Jiménez. Matrix representation of spiking neural p systems. *Lecture Notes in Computer Science*, 6501:377–392, 2011.
- [162] G. Zhang, M. Gheorghe, L. Pan, and M. J. Pérez-Jiménez. Evolutionary membrane computing: A comprehensive survey and new results. *Information Sciences*, 279(0):528–551, 9/20 2014.