UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño e implementación de un simulador de sistemas P con plásmidos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Gil Adrover, Mario

Tutor/a: Sempere Luna, José María

CURSO ACADÉMICO: 2022/2023

Resum

Els punts exposats en la memòria d'aquest Treball de Fi de Grau detallen el disseny i implementació d'una eina informàtica que permet realitzar una simulació de sistemes P amb plàsmids. Els sistemes P són un dels models de computació natural existents en l'actualitat, els quals estan inspirats en l'estructura i comportament d'una cèl·lula, així com d'estructures cel·lulars. Aquesta estructura es reprodueix en forma de membranes, creant així una agrupació jeràrquica d'elles, raó per la qual aquest tipus de models de computació també reben el nom de "models de computació amb membranes". Així doncs, el model presentat en aquest treball és una variant dels sistemes P, introduint una molècula coneguda en la biologia: el plàsmid.

En aquest Treball de Fi de Grau s'implementa, per tant, un simulador capaç d'executar els processos d'aquest model proposat. Aquest simulador es desenvolupa en el llenguatge de programació Java. A més, es realitzaran proves basades en operacions aritmètiques bàsiques que validarán la seva funcionalitat.

Paraules clau: Models de computació, Sistemes P, Plasmids, Simulació, Operacions aritmètiques

---

Resumen

Los puntos presentados en la memoria de este TFG muestran el diseño e implementación de una herramienta informática que permite realizar una simulación de Sistemas P con plásmidos. Los Sistemas P son uno de los modelos de computación natural actualmente existentes, el cual está inspirado en la estructura y comportamiento de una célula, así como de estructuras celulares. Esta estructura queda recreada en forma de membranas, formando así una agrupación jerárquica de ellas, razón por la que este tipo de modelos de computación también recibe el nombre de "modelos de computación con membranas". Así pues, el modelo presentado en este trabajo es una variante de los Sistemas P, introduciendo una molécula presente en la biología conocida: el plásmido.

En este TFG se implementa, por tanto, un simulador capaz de ejecutar los procesos de este modelo propuesto. Este simulador quedará desarrollado en el lenguaje de programación Java. Además, se realizarán pruebas basadas en operaciones aritméticas básicas que validarán su funcionalidad.

Palabras clave: Modelos de computación, Sistemas P, Plásmidos, Simulación, Operaciones aritméticas

---

Abstract

The points outlined in the report of this Final Year Project detail the design and implementation of a software tool that enables the simulation of P systems with plasmids. P systems are one of the existing models of natural computation, which are inspired by the structure and behavior of a cell, as well as cellular structures. This structure is replicated in the form of membranes, thus creating a hierarchical grouping of them. For this reason, this type of computational models is also referred to as "membrane computing models". The model introduced in this work is a variant of the P systems, incorporating a well-known molecule in biology: the plasmid.

In this Final Year Project has been impemented a simulator capable of executing the processes of this proposed model. This simulator has been developed in the Java programming language. Additionally, tests based on basic arithmetic operations will be executed to validate its functionality.

Key words: Computation models, P Systems, Plasmids, Simulation, Arithmetic Operations

Índice general

Índice general v
Índice de figuras vii

1 Introducción 1
1.1 Motivación . . . 2
1.2 Objetivos . . . 3
1.3 Estructura de la memoria . . . 3
2 Estado del arte 5
2.1 Marco biológico . . . 5
2.2 Sistemas P . . . 7
2.2.1 Funcionamiento . . . 9
2.2.2 Formalización . . . 9
2.2.3 Otras variantes de Sistemas P . . . 10
2.2.4 Aplicaciones de los Sistemas P . . . 11
3 Sistemas P con plásmidos 13
3.1 El plásmido . . . 13
3.2 El modelo . . . 14
3.2.1 Definición formal . . . 16
3.2.2 Funcionamiento . . . 18
4 Implementación del simulador 21
4.1 Tecnologías utilizadas . . . 21
4.1.1 Java como lenguaje de programación . . . 21
4.1.2 Visual Studio Code como IDE . . . 22
4.1.3 Github para el almacenamiento del proyecto . . . 22
4.2 UML . . . 23
4.3 Implementación de código . . . 27
4.3.1 Las entidades . . . 27
4.3.2 Utilidades y servicios . . . 32
4.3.3 La clase Simulator y flujo del simulador . . . 36
5 Casos de prueba 39
5.1 Resta aritmética . . . 39
5.2 Producto matemático . . . 41
6 Conclusiones 45
6.1 Trabajos futuros . . . 46
Bibliografía 49

Apéndices
A Archivos de entrada y salida de los casos de prueba 51
A.1 Archivo de entrada para el caso de prueba 1: La resta aritmética. En formato JSON. . . . 51
A.2 Archivo de salida para el caso de prueba 1: La resta aritmética. En formato JSON. . . . 53

A.3 Archivo de entrada para el caso de prueba 2: Producto de números enteros. En formato JSON. 56
A.4 Archivo de salida para el caso de prueba 2: Producto de números enteros. En formato JSON. 58

B Objetivos de Desarrollo Sostenible (ODS) 63

Índice de figuras

2.1 Diferencia entre célula eucariota y procariota. [7] 6
2.2 Estructura de un modelo de computación con membranas. [11] 7
2.3 Reglas de evolución de objetos dentro de un sistema P. 8

3.1 Estructura de un plásmido. En una bacteria *e coli*. [2] 14
3.2 Tipos de transporte membranal. [1] 18

4.1 Diagrama UML sin plásmidos. 23
4.2 Diagrama UML con plásmidos. 24
4.3 Representación de la estructura de membranas del problema propuesto. . 33

5.1 Diagrama de Venn del ejemplo del caso de prueba de resta aritmética. . . 41
5.2 Diagrama de Venn del ejemplo del caso de prueba de producto de enteros. 43

CAPÍTULO 1
Introducción

Para introducir el tema y comprender los objetivos de este TFG y su memoria que aquí se presenta, es preciso comenzar por la década de los años 30, momento en el que nacieron la teoría de la computabilidad y la teoría de la complejidad computacional, a manos de grandes matemáticos y científicos como Kurt Gödel (1908-1978), Alonzo Church (1908-1978), Stephen Kleene (1909-1994) o el mismo Alan Turing (1912-1954), cuyo nombre se repetirá a lo largo de esta memoria dada la relación de su trabajo con este.

En este campo de estudio aparecen los modelos de computación como descripción de máquinas o sistemas teóricos, que cuentan con un número de operaciones que permiten llevar a cabo algoritmos, entendiendo por "algoritmo" como el conjunto de operaciones ordenadas que permiten resolver algún cálculo. En 1936 Turing introduce al mundo un modelo matemático denominado Máquina de Turing, que trataba de resolver la pregunta de si a todo problema se le podía diseñar un algoritmo que lo resolviera. Así mismo, gracias a este modelo, los problemas se pueden dividir en decidibles y no decidibles, siendo estos últimos los problemas para los cuales no se puede tener una máquina que lo pueda resolver, aun disponiendo de las capacidades físicas necesarias. La Máquina de Turing es un modelo capaz de resolver cualquier problema decidible, lo que aporta una condición que se usaría en este gran marco de estudio a partir de ese momento: un problema o función es Turing-computable si existe una máquina de Turing que la pueda resolver en un tiempo finito.

A partir de este momento aparecen numerosos modelos de computación, que han llevado la computación al punto que se encuentra hoy en día. Entre ellos se encuentran todo un abanico de modelos de computación natural, funciones recursivas, cálculo lambda, o incluso los modelos de computación cuántica. Estos modelos de computación tienen como objetivo ser equivalentes a la máquina de Turing, es decir, tener la capacidad de resolver cualquier problema computable. Por esta razón, se verá como en las conclusiones de este trabajo se trata de demostrar que el modelo presentado en esta memoria es equivalente a la Máquina de Turing, o que es Turing-completo.

Como se acaba de afirmar, en este TFG se va a exponer un modelo de computación, que no es más que una variante de los Sistemas P. Se cree que los Sistemas P, o modelo de computación con membranas, probablemente reciban su nombre del apellido de la persona que en 1998 los introdujo, Gheorghe Păun. Los Sistemas P [12] son parte de los modelos bioinspirados, [1] o modelos de computación natural, y tienen dedicado todo un apartado de esta memoria, ya que es necesario exponerlos para lograr entender la

[1] Un sistema bioinspirado está basado en la naturaleza, ya sea en estructuras, comportamientos, procesos o entornos, entre muchos de los aspectos que aparecen en ella.

variante que se propone en el trabajo: introducir una molécula estudiada en el campo de la biología, el plásmido. Por esta razón, se realiza además una explicación teórica de los componentes biológicos de los Sistemas P, así como de esta molécula mencionada, antes de exponer formalmente el diseño del modelo.

Por otro lado, para este modelo variante de los Sistemas P, se ha desarrollado una aplicación, usando el lenguaje de programación Java, que permite realizar una simulación del funcionamiento de este. Además, se presentan algunos casos de prueba que se han ejecutado en el simulador para validar su correcto funcionamiento, y para mostrar algunas de las múltiples capacidades de este modelo.

Finalmente, se recopilarán los resultados y se presentarán las conclusiones pertinentes, y para aquellos interesados en las partes más técnicas de este trabajo podrán encontrar parte del código desarrollado y algunos otros datos de interés en los apéndices.

1.1 Motivación

Desde su aparición en 1998, los Sistemas P forman un área de gran interés científico en distintos ámbitos, como el de la informática, las matemáticas o la biología. Su base reside en el estudio de algunos organismos vivos, como la célula, los tejidos celulares o incluso órganos, abstrayendo su estructura y funcionamiento en un modelo matemático. Actualmente existen distintas variantes al sistema originalmente propuesto, que se acercan cada vez más a un escenario biológico real. Esto quiere decir, que con las herramientas necesarias para lograr llevar este modelo a la realidad, es decir, configurando cadenas de ADN, o programando células o algunos componentes de estas, para que implementen las reglas diseñadas en un sistema de este tipo de modelos, se puede tener una máquina de cómputo en un organismo vivo, con procesamiento distribuido de la información.

Por esta razón, en este TFG se propone una variante más a estos Sistemas P, incluyendo una molécula real que se puede encontrar en la biología, el plásmido, y que supone un acercamiento más hacia un ecosistema real, aportando al modelo nuevas ventajas, tales como las que aporta esta molécula en el mundo en el que vivimos.

Personalmente, mi motivación por el desarrollo de este trabajo surgió al cursar una asignatura del tercer año del grado de Ingeniería Informática: Computabilidad y Complejidad, de la rama de Computación. En esta asignatura aprendí los conceptos de “modelo de computación”, “Máquina de Turing”, y muchos más. En las prácticas de la asignatura, impartidas por el tutor de este trabajo, se presentaron algunos modelos de computación natural, como el modelo de computación con membranas, que causó interés en mí desde el primer momento. Opino que es un área de estudio fascinante y muy prematura, pues actualmente se está tratando de llevar el diseño de estos modelos a un organismo real, como he comentado anteriormente. Creo que la idea de este TFG puede dar lugar a grandes proyectos usando los Sistemas P, y me enorgullece participar en ella.

Por otro lado, la implementación del simulador es algo de mi agrado, ya que el arte de la programación captó mi atención antes incluso de entrar en este grado. Realizar el diseño y la implementación de la aplicación, usando tecnologías punteras que se comentarán más adelante, me parece una buena forma de terminar mi cuarto año, con este Trabajo de Fin de Grado.

1.2 Objetivos

Los objetivos de este trabajo son:

1. Objetivo 1: Proponer una variante de los Sistemas P, a los que se introducirá un nuevo tipo de objeto: el plásmido.
2. Objetivo 2: Desarrollar una herramienta informática capaz de simular el sistema propuesto.
3. Objetivo 3: Demostrar que el nuevo modelo es Turing-computable.
4. Objetivo 4: Dar pie a nuevos trabajos que exploten este u otra de las variantes de los Sistemas P con los que, utilizando un organismo biológico real, se pueda realizar el tipo de operaciones que se van a ver en esta memoria.

1.3 Estructura de la memoria

Esta memoria se compone de cinco capítulos, de los cuales el primero es la introducción ya presentada.

En primer lugar se va a exponer el marco teórico actual en el capítulo 2, en el cual se hace una revisión del estado actual de los Sistemas P, explicando este modelo de computación con membranas y algunas de sus variantes existentes. Además, aparecen también algunas definiciones tomadas del campo de la biología, ya que se trata de un modelo bioinspirado.

Seguidamente, en el capítulo 3, se define detalladamente el modelo de Sistemas P con plásmidos, usando su definición formal y exponiendo sus diferencias frente a la formalización de otras variantes de los Sistemas P. Además, en este capítulo se introducen las ventajas de usar este modelo. Este capítulo es, por lo tanto, en el que se expone la idea principal de este trabajo, lo que lo convierte en el núcleo de la memoria.

Por otro lado, en el capítulo 4 se expone el diseño y la implementación del simulador desarrollado para el modelo comentado. En este capítulo se hará una revisión de las tecnologías utilizadas en el desarrollo, así como de algunos diagramas que sirven de esquema general de la aplicación. Más adelante se explica detalladamente la implementación en código de cada una de las partes del diagrama presentado, así como de otras partes auxiliares que han sido necesarias para el funcionamiento correcto del sistema.

Acercándose al final de la memoria, en el capítulo 5, se llevará a cabo la exposición de algunos casos de prueba, con los que se han creado algunos casos de operaciones aritméticas utilizando el modelo de Sistemas P con plásmidos, que se han podido ejecutar en el simulador, mostrando y validando su funcionamiento. En este capítulo se puede observar el formato de las entradas y salidas manejadas por la herramienta.

Finalmente, la memoria termina con una recopilación de las conclusiones obtenidas del diseño del modelo y su simulador, además del estudio de los objetivos iniciales, una vez terminado el proyecto. En último lugar, se pueden encontrar la bibliografía y los apéndices.

CAPÍTULO 2

Estado del arte

En este capítulo se va a poner en contexto al lector del marco actual de los Sistemas P. Para ello, en primer lugar se realizará una breve explicación de los conceptos biológicos necesarios. En concreto, se entrará en breve detalle de la estructura y el funcionamiento de la célula, ya que esta será la base que servirá de inspiración para el siguiente apartado.

Por otro lado, se hace una presentación de los Sistemas P, profundizando en su origen y descripción. Más adelante se mostrará la relación que tienen los Sistemas P con el marco biológico explicado en su apartado anterior, y se explicarán las formas de representación más comunes para este tipo de sistemas. Por último, se hará una breve revisión de las aplicaciones y de algunas de las variantes actuales de los Sistemas P. Para los más interesados, se puede encontrar información actualizada sobre los Sistemas P, artículos de distinta naturaleza que los comentan, así como trabajos creados a partir de ellos, en [4].

2.1 Marco biológico

Se podría decir que este mundo está dividido en dos partes: lo vivo y lo no vivo. Pero, ¿cuál es la principal característica que permite dividirlos? La célula. Todo ser vivo está compuesto por una menor o mayor cantidad de células, siendo este el componente básico de todos ellos. Es de interés para este trabajo tanto el funcionamiento como la estructura de estos pequeños elementos, por lo que se enfocará la mayor parte de la explicación en estas dos características [6].

La células se pueden dividir en dos tipos: eucariotas y procariotas, teniendo como diferencia principal que el material genético de las células eucariotas se encuentra dentro de un núcleo, mientras que en las células procariotas este material genético se encuentra "flotando" por la célula. Debe de haber, por lo tanto, una pared que divida lo que se encuentra dentro de la célula de lo que no. A esta pared se le llama "membrana plasmática", y es de especial interés para este trabajo. Esta membrana celular se encarga de delimitar a la célula con su entorno, aunque también tiene una función de suma importancia, que es controlar el paso de moléculas tanto de dentro a fuera de la célula como en el sentido contrario, lo cual permite a la célula obtener sustancias del exterior y expulsar otras.

Otro elemento principal de las células, que ya ha sido mencionado, es el material genético, almacenado en moléculas de ADN y ARN, que en el caso de las células eucariotas está almacenado en un núcleo que se encuentra dentro de la célula.

El resto de la célula recibirá el nombre de "citoplasma", y en él se encuentran algunos otros tipos de vesículas que realizan distintas funciones para la célula. Entre ellos se pueden encontrar las mitocondrias, el aparato de Golgi, el retículo endoplasmático o los lisosomas. No se va a hacer una explicación detallada de la función de cada uno de estos elementos, pero se puede decir que entre todos ellos se realizan todas las funcionas vitales de la célula.

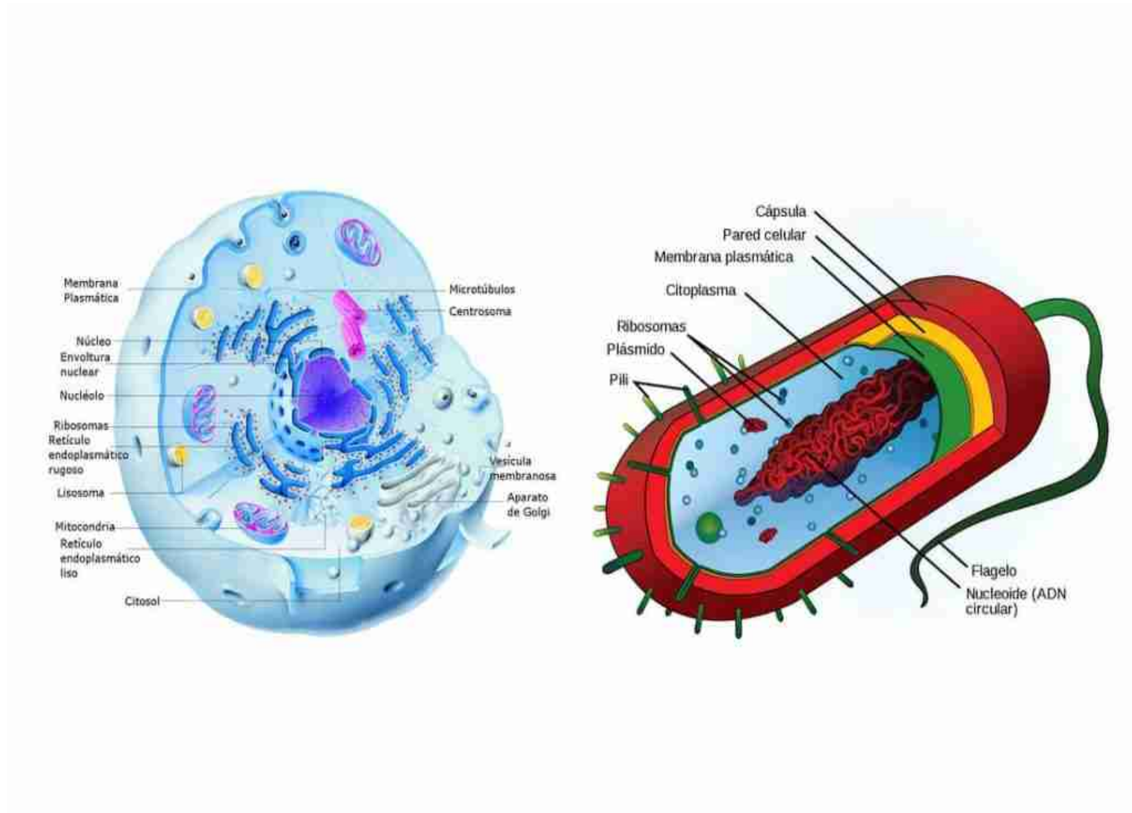

Figura 2.1: Diferencia entre célula eucariota y procariota. [7]

Un aspecto importante de los elementos de la célula es que se encuentran delimitados por distintos tipos de membranas, formadas por moléculas de lípidos, carbohidratos y proteínas, y a través de las cuales se realiza la mayor parte de actividad celular. Parándose a observar los puntos descritos en esta descripción, se puede ver una clara estructura jerárquica de membranas, cada una con una función específica y de suma importancia para las funciones vitales de la célula, generando una gran cantidad de reacciones químicas que se llevan a cabo de forma paralela y sin interferencias entre ellas. Es por esta razón que estos aspectos de la célula son de gran inspiración para la creación de un modelo de computación que se comentará en el siguiente apartado.

Antes de finalizar con este punto, cabe destacar también que estas características comentadas de la célula, pueden en parte extenderse a un nivel más alto, a un nivel supracelular. A este nivel se puede considerar una agrupación de células como una membrana más grande, donde se pueden encontrar células consideradas como membranas más pequeñas, cada una con su funcionamiento propio, y ciertos comportamientos de relación entre ellas. Estas relaciones pueden verse como una comunicación a través de ciertas moléculas, o incluso en la muerte o reproducción de las mismas células.

2.2 Sistemas P

En 1998, un matemático de nombre George Păun saca a la luz su trabajo sobre un nuevo modelo de computación basado en la estructura y comportamiento de una célula [12]. Este modelo recibió el nombre de "computación con membranas", ya que consiste en una estructura jerárquica de membranas, tal y como la que se ha visto en la estructura propia de la célula. Este modelo tuvo tal aceptación que pronto se convertiría en un nuevo campo de investigación dentro de las ciencias de la computación. Tanto es así, que han surgido distintas variantes a este modelo, creando numerosas posibilidades para la bioinformática. Este modelo de computación natural, recibe también el nombre de "Sistemas P", y se piensa que es el apellido de este científico al principio mencionado el que da lugar a ese nombre.

Este modelo matemático se compone de varios ingredientes principales. El primero de ellos, y el que le da su nombre, es la **estructura de membranas**. Como ya se ha dicho, existe una estructura jerárquica de membranas, lo que quiere decir que cada membrana puede contener en su interior a otras membranas. Evidentemente, hay una membrana que no tiene una membrana padre, es decir, que no está contenida en ninguna otra membrana, pues esta es la que separa al exterior de todo el sistema. Esta membrana recibe el nombre de "piel", o *skin*. Además, a las membranas donde se pueden encontrar otras membranas en su interior se les denomina "membranas no elementales", mientras que las "membranas elementales" serán aquellas que no contengan ninguna otra membrana dentro. Se entiende por membrana, por lo tanto, como un espacio tridimensional, cuya función es separar el exterior de ella con lo que se encuentra en su interior, que recibirá el nombre de "región". Esta estructura se puede comprender mejor viendo la figura 2.2.

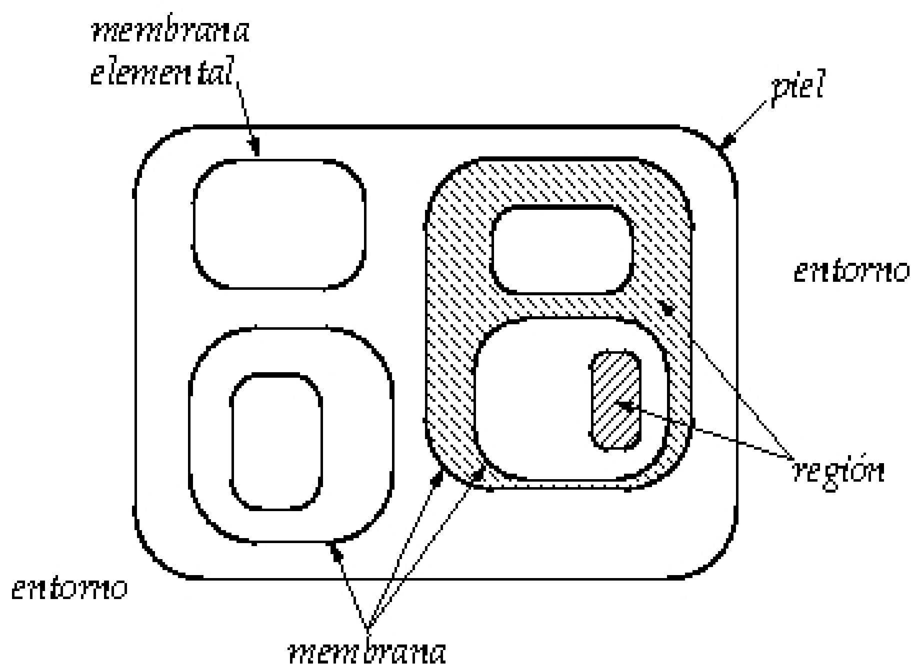

Figura 2.2: Estructura de un modelo de computación con membranas. [11]

Por otro lado, el segundo elemento imprescindible de un modelo de computación con membranas son los **objetos**. Los objetos pueden evolucionar o pueden entrar y salir de una membrana a otra. Desde una vista de modelo computacional, estos objetos son los datos del sistema. El programador puede definir los objetos de entrada del sistema, así como observar los objetos de salida. Por lo tanto, cada membrana, o mejor dicho, cada

región, tiene su propio multiconjunto de objetos. Un multiconjunto no es más que un grupo de conjuntos de objetos. Por ejemplo, en una región puede haber dos objetos de un tipo, y otros tres de otro tipo, siendo todo esto el multiconjunto de objetos de la región.

El útlimo ingrediente principal de los Sistemas P serían las **reglas**. Las reglas de una membrana, o de una región, definen qué se puede hacer con los objetos de dicha región. En el modelo más básico, las reglas pueden ser de evolución o de comunicación. Una regla de evolución indica que un objeto se transforma en otro, dentro de la región que contiene la regla. Una regla de comunicación, por otro lado, indica que un objeto de una región se transforma en otro en una región adyacente. Esto se puede comprender mejor visualizando la figura 2.3. Lo que esta figura representa es la acción que se realiza en cada una de estas posibles reglas. En todas ellas, el objeto "v" desaparece de dentro de la membrana "B", y es cambiado por los objetos indicados en las reglas. Es importante mencionar que no solo puede evolucionar un objeto a otro, sino que tanto la parte izquierda como la parte derecha de la regla son multiconjuntos de objetos.

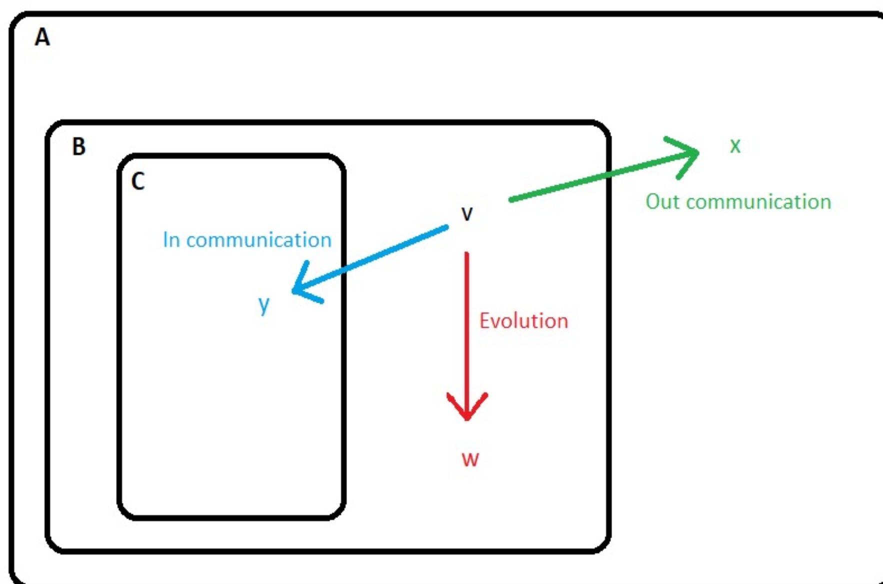

Figura 2.3: Reglas de evolución de objetos dentro de un sistema P.

Una vez definidos los ingredientes de los Sistemas P, se pueden llegar a apreciar las similitudes entre estos y la célula, y por tanto entender cómo esta última fue de inspiración para la creación de estos modelos de computación natural. Por un lado, la estructura de membranas, como la de la figura 2.2, está claramente relacionada con la de la célula, donde la membrana "piel" hace referencia a la membrana plasmática. El resto de vesículas o elementos de la célula vendrían representados como el resto de las regiones del sistema P, siendo cada membrana de este sistema la pequeña membrana que recubre cada uno de estos elementos.

En segundo lugar, los multiconjuntos de objetos presentes en estos modelos, no son más que una abstracción de las moléculas que se pueden encontrar dentro y fuera de cada región. Estas pueden ser moléculas como los polímeros, proteínas, lípidos, etc.

Por último, las reglas de cada región son una representación de las funciones metabólicas de cada uno de los elementos de la célula, así como de ella misma.

2.2.1. Funcionamiento

El modo de funcionamiento es el de "máximo paralelismo". Esto quiere decir que en cada paso de ejecución se ejecutan las reglas de cada membrana de la siguiente forma: de entre las reglas que se pueden ejecutar, se elige una de forma no determinista, es decir, aleatoria, y se ejecuta tantas veces como sea posible. El concepto de paso de ejecución vendría marcado por una especie de reloj global, que indica que cada membrana se ejecuta al mismo tiempo, y de forma paralela, a todas las demás membranas. Por lo tanto, en cuanto dos membranas compiten por un objeto, se elige también de manera no determinista la ganadora.

Estos pasos de ejecución marcan, por lo tanto, la computación del sistema. Esto significa que el sistema parte de una configuración inicial, y en cada paso de ejecución se transiciona a la siguiente configuración, tal y como ocurre en la teoría de autómatas. Una computación define entonces la transición entre una configuración inicial y una final, y es considerada como exitosa si esta configuración final no transiciona a otra distinta en siguientes pasos de ejecución, es decir, si ya no se puede ejecutar ninguna otra regla más.

2.2.2. Formalización

Una forma de representar un sistema de computación con membranas es mediante el uso de diagramas de Venn, como se puede ver en las figuras 2.2 o 2.3. Estos diagramas, usados en la teoría de conjuntos, aportan una representación visual y muy esclarecedora de lo que es la estructura del sistema. Sin embargo, esta forma de representación perdería la claridad al aumentar en gran medida el número de elementos del sistema. Por lo tanto, el modelo puede ser representado mediante su formalización matemática, que se va a presentar a continuación.

Un modelo de computación con membranas viene definido por la tupla:

$$\Pi = (V, H, \mu, w_1, w_2, ..., w_m, R_1, R_2, ..., R_m, i_0)$$

donde:

1. V es el alfabeto de objetos.
2. H es el alfabeto de etiquetas de las membranas.
3. μ define la estructura de membranas, de grado m, tal que $m \geq 1$, con todas las membranas etiquetadas con etiquetas de H. Una membrana de etiqueta "h" vendrá representada como: $[_h]_h$. Así mismo, una estructura jerárquica de membranas se representaría de la misma forma. Por ejemplo, las membranas de la figura 2.3 se representa formalmente como: $[_A\ [_B\ [_C]_C\ ]_B\ ]_A$
4. $w_1, w_2, ..., w_m$ especifican los multiconjuntos iniciales de objetos sobre V para cada una de las membranas de μ.
5. $R_1, R_2, ..., R_m$ son los conjuntos finitos de reglas de cada una de las membranas de μ. Las reglas pueden ser de los siguientes tipos:
   a) $v \rightarrow w_{here}$ con $v, w \in V^*$ (reglas de evolución).
   b) $v \rightarrow w_{out}$ con $v, w \in V^*$ (reglas "out communication").

c) $v \to w_{in_j}$ con $v, w \in V^*$ (reglas "in communication").

Como se puede observar, las partes derechas de las reglas tienen asociadas una dirección que indica el tipo de regla. Esta dirección marca la membrana destino en la que se van a generar los objetos que indica la regla. Por lo tanto, la etiqueta *here* indica que se trata de una regla de evolución y los objetos se generarán en la misma membrana en la que se encuentra la regla ($[_h v]_h \to [_h w]_h$). La etiqueta *out* significa que los objetos se generarán en la membrana padre de la membrana que contiene la regla ($[_h v]_h \to w[_h]_h$). Por último, la etiqueta in_j indica que el destino es la membrana j, siendo $j \in \mu$ ($[_h v[_j]_j]_h \to [_h[_j w]_j]_h$). Cabe mencionar que, para evitar redundancia, el uso de la etiqueta *here* no suele emplearse, y se da por hecho este tipo de dirección en los casos en los que no aparece etiqueta.

6. $i_0 \in \{1, ..., m\} \cup \{\infty\}$ indica la o las regiones de salida del sistema, donde vendrá definido el resultado de la computación (∞ representa el ambiente, el exterior). Los objetos de toda membrana que no sea una membrana de salida no serán considerados como resultado de la computación.

2.2.3. Otras variantes de Sistemas P

Desde su aparición en 1998, este modelo de cómputo ha dado lugar a diversas variantes, las cuales aplican alguna extensión que aporta mayor flexibilidad o eficiencia computacional al modelo básico. Entre ellas se encuentran los *Sistemas P con membranas activas*, los *Sistemas P basados en neuronas de impulsos* o los *Sistemas P con proteínas*, entre otros.

Los **Sistemas P con membranas activas** [13] son una extensión al modelo de Sistemas P en el que se introducen reglas que aportan flexibilidad a las membranas, dado que el modelo básico mantiene un estado estático de la estructura de membranas durante toda la computación. Un tipo de regla que se introduce en esta variante es el de la división de membranas. En esta regla se produce una acción similar a la mitosis celular, en la que una membrana se duplica debido a la regla que se ha disparado por un objeto. Un ejemplo de este tipo de reglas es: $[_h a]_h \to [_h b]_h \; [_h c]_h$.

Otra regla introducida en esta variante es la de la creación de membranas, como puede ser, por ejemplo, la regla $a \to [_h b]_h$, en la que la membrana h es creada debido a la existencia de un objeto a, y éste evoluciona a un objeto b que aparece dentro de la membrana creada. Además de estos, existen otros tipos de reglas que aparecen en esta variante con membranas activas.

Por otro lado, los **Sistemas P basados en neuronas de impulsos** [8] o *spiking neural P Systems* (SNP) en inglés, es otra de las variantes de los Sistemas P. Esta variante pretende simular el comportamiento de una red neuronal que funciona a través de impulsos, es decir, que las neuronas envían y reciben impulsos eléctricos o *spikes*. Las reglas en esta variante se encargan de realizar esas acciones de envío de objetos de tipo "spike".

Por último, los **Sistema P con proteínas** son otra variante, propuesta por el mismo Paun, en la que se incorpora la idea de simular las acciones y reacciones de las proteínas en un contexto biológico real.

2.2.4. Aplicaciones de los Sistemas P

Como ya se ha mencionado anteriormente, el objetivo de este TFG es proponer una variante de los Sistemas P, como extensión a estos. Lo que esto quiere decir es, entre otras cosas, que las aplicaciones prácticas y teóricas de este trabajo podrían ser similares a las de los Sistemas P actuales [10].

Este tipo de modelos tiene numerosas aplicaciones, pero no hay que dejar de tener en cuenta que se trata de un campo en investigación, por lo que, aunque no sean muchas, pueden ser el principio de otras ideas innovadoras. Entre sus aplicaciones se encuentran:

1. Simulación biológica: Cabe recordar que la computación con membranas es un modelo bioinspirado, por lo que puede ser una gran herramienta a la hora de realizar simulaciones de ciertos procesos biológicos. Un caso particular podría ser el modelado del comportamiento de varios elementos biológicos, como por ejemplo, el de una sustancia química con un tipo específico de célula, cosa que sería interesante para la prueba de un fármaco en un paciente particular, o para observar la interacción de una sustancia con un agente patógeno. Es por ello que los Sistemas P son de especial interés en el campo de la medicina y la farmacología.

   Además de estas, existen numerosas posibilidades de simulaciones que se pueden llevar a cabo usando Sistemas P. Anteriormente se han realizado simulaciones poblacionales, en las que se estudia alguna característica específica entre los individuos de una población, como puede ser el caso de un contagio epidemiológico.

2. Computación paralela: Actualmente, tras años de grandes avances, la informática se ve dificultada por ciertas limitaciones físicas, por lo que la computación paralela cobra gran importancia a la hora de llevar a cabo varios procesos al mismo tiempo, de forma paralela. Dado que los Sistemas P son un modelo teórico en los cuales se realiza la computación de manera paralela y distribuida entre distintas membranas, este es un campo que ofrece numerosas posibilidades a la computación paralela.

   Como caso concreto, se puede encontrar la resolución de problemas. En diversos campos de la ciencia, como las matemáticas o la computación, aparecen continuamente ciertos problemas que suponen un reto a resolver. Casos como problemas de búsqueda de una solución óptima, pueden ser resueltos de forma más eficiente si se exploran varias soluciones al mismo tiempo, por lo que un modelo que explote la computación paralela, como es el caso de los Sistemas P, puede ser de gran ayuda para ello.

3. Computación basada en ADN: Como ya se ha comentado anteriormente en la motivación de este trabajo (apartado 1.1), los Sistemas P pueden ser una herramienta que permita llevar la computación a la codificación de una cadena de ADN. El objetivo de esto sería aprovechar las propiedades que ofrece el ADN a la hora de resolver cálculos. Entre las muchas de estas propiedades, es de especial interés, hilando con el punto anterior, el hecho de que el ADN trabaja realizando numerosas operaciones al mismo tiempo. Además, no se puede dejar de lado el hecho de que el ADN puede almacenar información en un tamaño todavía muy alejado del *hardware* de la informática actual, lo que puede suponer un avance en la reducción del espacio de la máquina de computación.

CAPÍTULO 3

Sistemas P con plásmidos

Como se ha descrito en el punto 2.2: *Sistemas P* del capítulo anterior, el modelo de computación que da nombre a ese punto es un modelo computacional basado en una estructura jerárquica de membranas inspirada en la estructura y comportamiento de una célula. Estos sistemas ofrecen diferentes perspectivas y aplicaciones en distintas áreas, como también se detalla en el punto mencionado.

Dentro del gran abanico de elementos que nos ofrece la biología celular, existe uno que aporta unas características únicas, y recibe el nombre de "Plásmido". Estos minúsculos elementos, dadas sus propiedades, que se explicarán a continuación, representan un mecanismo interesante y versátil de transferencia de información.

Por esta razón, en este capítulo se cumple uno de los objetivos principales de este TFG, que es proponer una innovadora variante de los Sistemas P, a los que se les incorpora este elemento presente en algunas células. Este cambio pretende aumentar la flexibilidad de los modelos de computación con membranas actuales, así como ampliar en la medida de lo posible las aplicaciones de estos.

En este capítulo se va a hacer un recorrido que comienza por una descripción del plásmido como elemento biológico, seguido de la descripción de los Sistemas P con plásmidos. En este último punto se hará una justificación de por qué puede resultar interesante introducir los plásmidos a los Sistemas P, además de explicar cuáles son las diferencias entre esta variante con su versión tradicional. Por último, de manera similar a lo que se ha hecho para los Sistemas P en el punto 2.2, se realizará una definición formal del modelo con plásmidos.

3.1 El plásmido

Un plásmido es una pequeña estructura molecular compuesta de ADN que se puede encontrar mayoritariamente en las bacterias y algunos otros microorganismos biológicos [9]. Fue inicialmente introducido por el biólogo Joshua Lederberg en 1952 y presenta una peculiar característica: los plásmidos pueden replicarse de forma independiente, lo que quiere decir que pueden duplicarse por sí mismos sin la necesidad de que se duplique el cromosoma bacteriano de la bacteria en la que se encuentra.

Los plásmidos (ver figura 3.1) a menudo contienen genes beneficiosos para los organismos que los poseen. Cabe destacar, por ejemplo, que algunos plásmidos contienen genes que ofrecen resistencia a los antibióticos, permitiendo que la bacteria sobreviva en

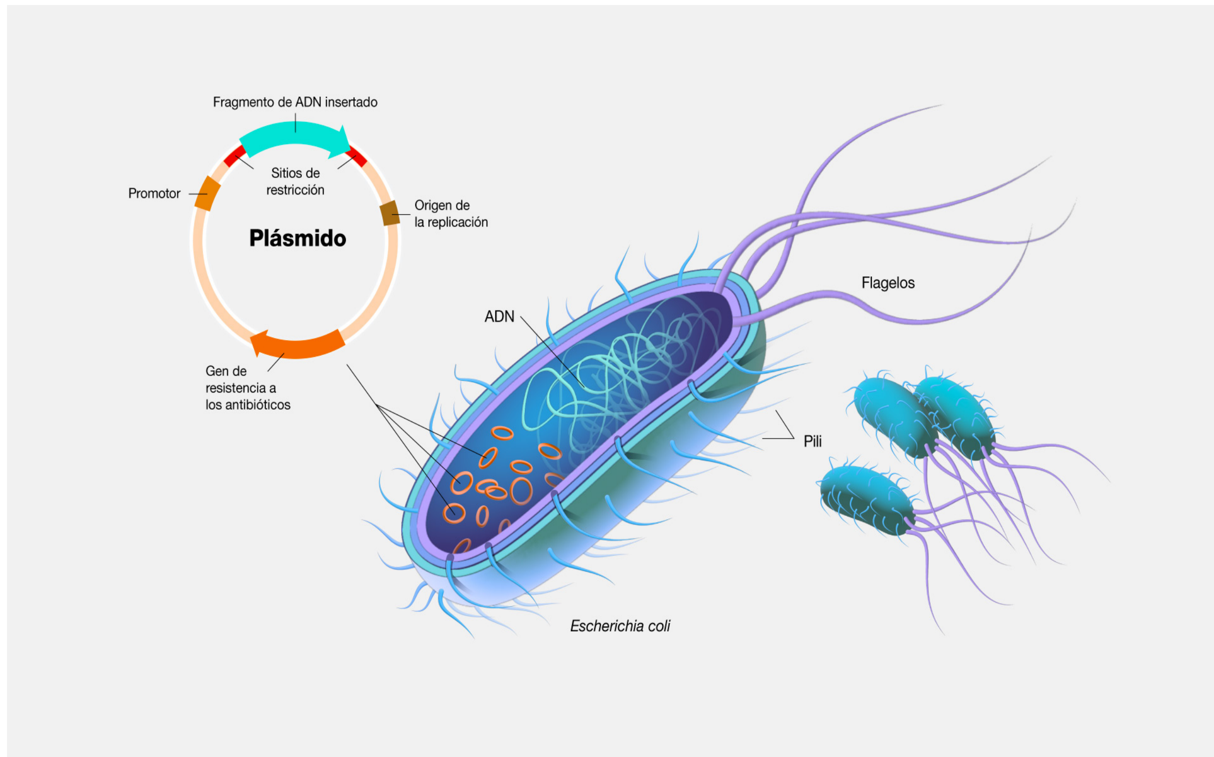

Figura 3.1: Estructura de un plásmido. En una bacteria *e coli*. [2]

presencia de estos medicamentos. Además, existen plásmidos que aportan a las bacterias ciertas habilidades especiales, como la capacidad de metabolizar fuentes de nutrientes poco comunes o generar toxinas.

Los plásmidos tienen diferentes aplicaciones en el mundo actual, dado que permiten realizar procesos de clonación de ADN, aprovechando su especial capacidad de replicación independiente. Este proceso consiste en insertar la secuencia de ADN a clonar en el plásmido ("programando un plásmido"), que posteriormente es introducido en una célula huésped, como una bacteria. Mediante este proceso se consigue que el plásmido se replique junto a la bacteria, permitiendo la producción masiva de la secuencia de interés. Un ejemplo de las múltiples aplicaciones de los plásmidos en campos como la genética y la biotecnología, es la producción de alguna proteína específica, como la insulina o algunos antibióticos.

3.2 El modelo

En la sección anterior se deja claro que los plásmidos representan una gran herramienta en campos de estudio como la medicina, dando lugar a funciones vitales como la producción de ciertos fármacos para uso médico. Las ventajas que ofrecen los plásmidos en un escenario real son fácilmente aplicables a un campo teórico que represente precisamente el funcionamiento de estos escenarios, como los Sistemas P. Ahí reside, por lo tanto, la motivación de introducir los plásmidos en este tipo de modelos computacionales. De las posibilidades que ofrece trabajar con los plásmidos en un escenario real, se deducen las ventajas de introducirlos en un modelo de Sistemas P con plásmidos:

1. Replicación independiente: Al igual que ocurre en las bacterias, un objeto que simule el comportamiento de un plásmido en un modelo de computación con membranas debería poder replicarse de forma autónoma. Esta habilidad aportaría al sistema la posibilidad de replicar y trasladar información e instrucciones dentro del sistema, lo que puede aumentar la flexibilidad y eficiencia de las operaciones.

   Además, esta función puede cambiar de cierta manera la manera de configurar un modelo de Sistemas P, ya que los plásmidos harían del sistema un modelo más dinámico y colaborativo, ya que permite conseguir que se compartan instrucciones entre varias membranas, tal como se hace con los genes compartidos con plásmidos por las bacterias. Como se ha comentado anteriormente, los Sistemas P representan un modelo teórico universal, es decir, que permiten resolver cualquier problema computable por una Máquina de Turing. Por lo tanto, añadir plásmidos al modelo no aporta la habilidad de resolver un problema que no pueda ser resuelto por un modelo sin plásmidos, pero sí puede hacerlo de una manera más eficiente, dada la flexibilidad que otorga al sistema.

2. Modularidad y adaptabilidad: La modularidad representa una característica realmente importante para los sistemas informáticos. Los plásmidos dentro de un sistema computacional, siendo entendidos como un módulo de información, aportan la posibilidad de agregar, sustraer o distribuir estos módulos de forma fácil y rápida, proporcionando al sistema adaptabilidad a escenarios dinámicos.

   De este modo, la adaptabilidad a los escenarios cambiantes aporta al sistema una mayor seguridad frente a ataques maliciosos o errores del propio sistema.

3. Almacenamiento de la información: Siendo los plásmidos un elemento capaz de almacenar información de distinto tipo, su adición en un sistema le otorga la capacidad de replicación de la información de una manera sencilla y eficiente, aportando una característica muy buscada en un sistema informático: la alta disponibilidad y tolerancia a los fallos, obteniendo este último de la capacidad de no perder información frente a un fallo del sistema.

4. Investigación: Además de las ventajas comentadas, enfocadas a su aplicación en un sistema informático, un modelo de Sistemas P con plásmidos otorga a las distintas áreas de investigación la capacidad de comprender mejor el funcionamiento y las posibilidades de tratar con plásmidos en un escenario biológico.

Sin embargo, a pesar de todas estas ventajas, no hay que dejar de tener en cuenta los posibles desafíos o limitaciones que podría otorgar este modelo, por lo que es importante llevar esta variante de modelo con membranas al estudio e investigación.

Una vez comentadas las ventajas de introducir los plásmidos a un modelo de Sistemas P, se puede proceder a describir el modelo. Realmente no son tantas las diferencias de este modelo frente al modelo sin plásmidos. Las diferencias son, a grandes rasgos, la introducción de un nuevo objeto "Plásmido" al sistema, así como las reglas para manejarlos.

En primer lugar, teniendo en cuenta la definición de membrana explicada en la sección 2.2, ésta puede ser considerada como un objeto que representa un límite entre su región y el exterior de esta, manteniendo en su interior un multiconjunto de objetos, un conjunto de reglas que definen su funcionalidad y una subestructura embebida de membranas hijas, que continúan con la jerarquía de membranas que componen al sistema. De este modo, ya que el conjunto de reglas de la membrana son en parte una abstracción del ADN de una célula, no es difícil llegar a la conclusión de que un plásmido tendría

una estructura similar a la de la membrana, pero almacenando únicamente un conjunto propio de reglas. Esta sería la representación de su definición biológica, es decir, de que un plásmido está formado por una molécula de ADN.

Resumiendo las especiales habilidades que aporta un plásmido a una célula, se puede recordar que el material genético del plásmido, al estar éste contenido en la célula, pasa a ser considerado como parte de ella, siendo por lo tanto una extensión del material genético de la misma mientras que el plásmido se encuentre en su interior. Sin embargo, el ADN del plásmido es independiente al de la célula, por lo que funciona de esta misma manera. Esto es, el plásmido puede replicarse sin la necesidad de la replicación de la célula, así como salir de ésta sin que el material genético de la célula se vea afectado, entre otras características.

De manera paralela, por lo tanto, el plásmido dentro del modelo computacional, considerándolo como un objeto que contiene un conjunto de reglas, es capaz de introducirse dentro de una membrana, así como salir de ella, permitiendo que las reglas contenidas en el plásmido sean consideradas como parte de las reglas de la membrana durante su estancia dentro de ella, lo cual puede aportar a la membrana ciertas funcionalidades que originalmente no tiene. Además, el plásmido tendría la posibilidad de replicarse dentro de una membrana, así como de introducirse en el dominio de otra membrana adyacente.

En segundo lugar, aparece otro concepto necesario para el funcionamiento de este modelo, que son las reglas necesarias para el manejo de los plásmidos. Entre estas reglas se encuentran las reglas que permitan trasladar un plásmido de una región a otra vecina, disparadas por la aparición de otros objetos. Por otro lado, también serían necesarias reglas para la replicación del plásmido, de forma similar a como se realiza una mitosis celular. De hecho, se puede observar como las reglas definidas para los plásmidos tienen la misma naturaleza que las reglas de un Sistema P con membranas activas (explicadas en la sección 2.2.3), y esto se debe a la similitud comentada entre el objeto *membrana* y el objeto *plásmido*. Sin embargo, por simplicidad, en este modelo solo se ha tenido en cuenta el primer tipo de reglas, es decir, las de movimiento de plásmidos.

Una vez descrito a grandes rasgos el modelo de Sistemas P con plásmidos, se puede proceder a realizar su definición formal, que va a resultar similar a la definición del modelo sin plásmidos (sección 3.2.1), ya que este último va a ser extendido por el modelo que se va a definir.

3.2.1. Definición formal

En primer lugar, un **plásmido** es un conjunto finito de reglas de tipo de evolución y comunicación de uno de estos tipos:

1. $v \to w_{here}$ con $v, w \in V^*$ (reglas de evolución).
2. $v \to w_{out}$ con $v, w \in V^*$ (reglas "out communication").
3. $v \to w_{in_j}$ con $v, w \in V^*$ (reglas "in communication").

Por otro lado, un **modelo de Sistemas P con plásmidos de grado m y q plásmidos**, tal que $m \geq 1$ y $q \geq 0$, viene definido por la tupla:

$$\Pi = (V, H, \mu, w_1, w_2, ..., w_m, R_1, R_2, ..., R_m, p_1, p_2, ..., p_q, i_0)$$

donde:

1. V es el alfabeto de objetos.
2. H es el alfabeto de etiquetas de las membranas.
3. μ define al estructura de membranas, de grado m, tal que $m \geq 1$, con todas las membranas etiquetadas con etiquetas de H. La estructura viene representada como se puede ver en el punto 2.2.2.

   De ahora en adelante, cuando se hable de "árbol de membranas" se estará haciendo referencia al árbol (entendido como la estructura de datos en programación), que representará a la estructura jerárquica de membranas que viene definida en μ, por lo que los nodos del árbol serán las membranas, la raíz será la membrana "piel" (*skin*), y los nodos hijos de cada nodo serán las membranas que se encuentran dentro de la membrana que representa. De esta forma, los nodos hoja representarán las membranas que no contienen nignuna membrana en su interior, es decir, las membranas elementales (figura 2.2).
4. w_i, tal que $1 \leq i \leq m$, representa el multiconjunto de objetos sobre V de la región i definida en μ.
5. R_i, tal que $1 \leq i \leq m$, representa al conjunto de reglas que rigen la región i definida en las membranas de μ. Las reglas pueden ser de los siguientes tipos:

   a) $v \rightarrow w_{here}$ con $v, w \in V^*$ (reglas de evolución).
   b) $v \rightarrow w_{out}$ con $v, w \in V^*$ (reglas "out communication").
   c) $v \rightarrow w_{in_j}$ con $v, w \in V^*$ (reglas "in communication").

   Al igual que como se ha comentado en el punto 2.2.2, las partes derechas de las reglas tienen asociadas una dirección que indica el tipo de regla: dirección $\in \{here, out, in_j\}$. Se recomienda ver el punto mencionado para entender mejor su funcionamiento.

   En adición a estos tipos de reglas, se añaden al modelo otros tipos de reglas que permitan el manejo de los plásmidos dentro del sistema. Estos tipos son:

   a) $p_i x[_h u]_h \rightarrow y[_h p_i v]_h$ con $x, y, u, v \in V^*$, y con $1 \leq i \leq q$ (reglas "in symport")
   b) $x[_h p_i u]_h \rightarrow p_i y[_h v]_h$ con $x, y, u, v \in V^*$, y con $1 \leq i \leq q$ (reglas "out symport")
   c) $p_i x[_h u p_j]_h \rightarrow p_j y[_h p_i v]_h$ con $x, y, u, v \in V^*$, y con $1 \leq i, j \leq q$ (reglas "antyport")

   Por otro lado, se han añadido al modelo algunas reglas que aporta la variante de *Sistemas P con membranas activas* (vista en la sección 2.2.3), que permiten crear o trasladar membranas:

   a) $a[_h]_h \rightarrow [_h b]_h [_h c]_h$ (reglas de duplicación de membranas)
   b) $a[_h]_h [_g]_g \rightarrow b[_g c[_h]_h]_g$ (reglas de transporte de membranas)
6. $p_1, p_2, ..., p_q$ son los plásmidos del sistema.
7. $i_0 \in \{1, ..., m\} \cup \{\infty\}$, indica la o las regiones de salida del sistema, donde vendrá definido el resultado de la computación (∞ representa el ambiente, el exterior). Los objetos de toda membrana que no sea una membrana de salida no serán considerados como resultado de la computación.

Por si es demasiado densa la definición de las reglas para el manejo de los plásmidos, se procede a explicar cada uno de sus tipos. Cabe destacar que las reglas de tipo "Symport" y "Antyport" ya se han usado anteriormente en otras variantes de Sistemas P [5], ya que representan un tipo de transporte de sustancias que también es aportado por la biología molecular (ver figura 3.2). En ese contexto, a grandes rasgos, el *symport* representa el movimiento de dos moléculas o iones a través de una membrana, en el mismo sentido, mientras que el *antyport* representa un movimiento similar, pero en el que las sustancias se trasladan en la misma dirección pero sentidos opuestos. Tanto los *simportadores* como los *antiportadores* son proteínas transmembranales que componen un grupo denominado *cotransportadores*.

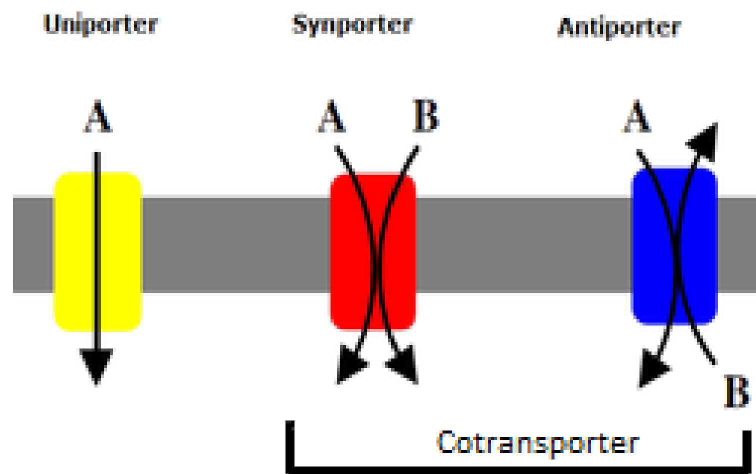

Figura 3.2: Tipos de transporte membranal. [1]

Dicho este contexto, las reglas para trabajar con los plásmidos dentro del sistema son de tipo "symport" o "antyport". Las reglas "symport" representa al movimiento de un plásmido de una región a otra (en las reglas "out symport" la región destino sería la circundante, mientras que en las reglas "in symport" el destino es la región indicada en la regla). Por lo tanto, tras la ejecución de esta regla, el plásmido se traslada de una región a otra, y tanto los objetos de la región origen como de la destino sufren una evolución. Por otro lado, las reglas de tipo "antyport" realizan las mismas acciones, además de añadir el movimiento de otro plásmido en la misma dirección transmembranal, pero en el sentido contrario.

3.2.2. Funcionamiento

El funcionamiento de este modelo es muy similar al del modelo sin plásmidos (ver punto 2.2.1), salvo algunas adiciones necesarias de definir para la permitir la incorporación de los plásmidos. Por esta razón, en este punto tan solo se van a comentar las diferencias respecto al punto 2.2.1. De esta manera, el concepto de "computación" descrito en ese apartado permanece intacto.

Se recuerda que el modo de funcionamiento es de "máximo paralelismo" y no determinista. Esto último quiere decir que a la hora de seleccionar una regla de una región se hace de forma completamente aleatoria. Por lo tanto, la primera diferencia a destacar

del funcionamiento de este modelo es que, a la hora de seleccionar una regla de una región, se tienen en cuenta también, además de las reglas propias de la región, las reglas de los plásmidos que se encuentran en la región en el momento de la selección. En el momento en el que un plásmido abandona una región, las reglas de éste dejarán de ser consideradas como parte de las reglas de la región.

En segundo lugar, también se recuerda un punto importante en el funcionamiento del modelo sin plásmidos: en el momento en el que más de una regla (sean de la misma región o de distintas regiones) compita por un objeto, ya que las reglas de cada región se ejecutan de forma simultánea, y dado que no existe prioridad entre reglas en este modelo, la regla ganadora será elegida de forma totalmente indeterminista (de manera aleatoria). Del mismo modo, este nuevo modelo cuenta con una regla similar, que es la siguiente: en el momento en el que más de una regla (sean de la misma región o de regiones diferentes) compita por un plásmido, la selección de la regla ganadora se hará también de manera indeterminista, siendo esta la regla que se ejecute.

Por último, en este modelo planteado se asume cierta configuración inicial respecto a los plásmidos, y esta es: inicialmente, todos los plásmidos se encuentran en la región "entorno", en el ambiente (región identificada como "∞" en la definición formal del modelo), y solo podrán introducirse en otras regiones mediante las reglas descritas en el apartado anterior. Además, se entiende por plásmido como un elemento único, lo que significa que en el momento en el que un plásmido se introduce en una región, abandona al tiempo la región en la que se encontraba (incluso si ésta era el propio ambiente). Esto quiere decir que solo existe una copia para cada plásmido. Sin embargo, si se quisiera tener una mayor cantidad de plásmidos idénticos, se podría extender la definición del modelo con tal de facilitar la aparición de plásmidos repetidos. En tal caso, el modelo extendido de esta forma vendría definido por la tupla:

$$\Pi = (V, H, \mu, w_1, w_2, ..., w_m, R_1, R_2, ..., R_m, (p_1, n_1), (p_2, n_2), ..., (p_q, n_q), i_0)$$

donde el número de copias del plásmido p_i, siendo $1 \leq i \leq q$, vendría definido por $n_i \geq 1$.

CAPÍTULO 4

Implementación del simulador

A continuación se va a explicar el desarrollo de la implementación del simulador del sistema P con plásmidos, comenzando por comentar las tecnologías utilizadas, donde se verá tanto el lenguaje de programación empleado, como el IDE utilizado o el almacenamiento del repositorio.

En segundo lugar, se procederá a explicar la implementación del propio simulador, viendo el UML que lo especifica, y definiendo el flujo de desarrollo del algoritmo principal. Esto servirá para dar pie al siguiente capítulo, en el que se mostrarán algunos de los ejemplos utilizados como pruebas de este simulador, así como los comentarios de los resultados obtenidos al ponerlos en marcha.

4.1 Tecnologías utilizadas

4.1.1. Java como lenguaje de programación

A pesar de que al principio el lenguaje de programación destinado al desarrollo de esta aplicación iba a ser Python, y el proyecto fue comenzado en este lenguaje, se decidió cambiar el lenguaje a Java, debido a su potente uso de objetos y relaciones entre clases, como se va a pasar a explicar en este apartado.

En lo que a este proyecto respecta, Python tiene algunas ventajas sobre Java:

1. Es un lenguaje de programación a muy alto nivel, por lo que a rasgos generales es un lenguaje mucho más legible y, lo que es más importante para el desarrollo de este trabajo: es más fácil y rápido de escribir.
2. En Python se pueden instanciar diccionarios de una forma muy sencilla, por lo que la lectura de un fichero JSON [1] y conversión a diccionario se puede realizar con una sola asignación. Esto es una ventaja ya que, como se verá más adelante, los ficheros de entrada y salida del simulador vienen en formato JSON.
3. Python es un lenguaje de tipado dinámico, y las listas y tuplas pueden contener elementos de distintos tipos, lo que aportaría mucha flexibilidad a los objetos contenidos en una membrana. En Java se puede conseguir esto indicando que los objetos

[1] JSON, o JavaScript Object Notation, es un formato de texto sencillo diseñado para el intercambio de datos.

de la membrana son de tipo “Object”, haciendo que admita cualquier tipo de clase que herede de esta, es decir, todas en este lenguaje de programación.

Sin embargo, y aún teniendo en cuenta todas estas afirmaciones, se decidió usar Java ya que aporta estas otras ventajas:

1. En contraposición a lo comentado para el lenguaje de Python, Java es un lenguaje en menor alto nivel, por lo que el desarrollo genera una mayor cantidad de código, de más difícil legibilidad, pero con una interpretación[2] y ejecución más rápidas.
2. Existen numerosos frameworks para Java, como Java Springboot, que es el que se ha utilizado en este desarrollo, ya que aporta gran facilidad al desarrollo de aplicaciones. De este framework se han usado anotaciones que facilitan la configuración e instanciación de algunas clases del proyecto, como los servicos principales, que se verán más adelante.
3. La principal razón por la que se ha decidido usar Java es que este es un lenguaje con mayor relación al paradigma de POO,[3] por lo que aporta gran facilidad al programador a la hora de crear relaciones entre clases, necesaria en este proyecto dada su naturaleza.

4.1.2. Visual Studio Code como IDE

El IDE o Entorno de Desarrollo Integrado utilizado ha sido Visual Studio Code, que a partir de ahora se nombrará como VS Code por simplicidad. La principal razón para elegir este entorno es la familiaridad del autor con el mismo, pero también por todas las funcionalidades que trae consigo y que han ayudado a realizar este proyecto.

Por un lado, VS Code tiene disponibilidad de numerosas extensiones que aportan diferentes funcionalidades a necesidad del programador. Las utilizadas para este proyecto han sido las extensiones de Springboot, que permiten crear un proyecto plantilla de forma muy sencilla y rápida, utilizando Maven para el despliegue de aplicaciones y gestión de dependencias (mediante un fichero *pom.xml*). Además, se han usado también extensiones de *debug* y *Languaje Support* para Java.

Por otro lado, VS Code ofrece la posibilidad de poner puntos de parada en el código, que permiten pausar la aplicación mientras está en marcha en el punto indicado, para observar los valores de las clases en ese instante, lo cual permitió, junto a mensajes o *logs* con información relevante, poder probar la aplicación y resolver los posibles errores de programación ocurridos.

4.1.3. Github para el almacenamiento del proyecto

Github es una plataforma de almacenamiento de proyectos basada en *Git*.[4] Esta plataforma permite tener un repositorio almacenado en la nube, lo que posibilita que varios autores puedan cooperar en el desarrollo del proyecto de una forma controlada y con

[2]Java es un lenguaje de programación tanto compilado como interpretado, lo que quiere decir que cuando se genera un archivo de código, este pasa un primer paso de compilación que genera un bytecode ejecutable por cualquier máquina con JVM (Java Virtual Machine), que se encarga de interpretarlo y ejecutarlo.

[3]Un lenguaje POO es un lenguaje de Programación Orientada a Objetos.

[4]Git es un conocido sistema de control de versiones distribuido.

posibilidad de revertir posibles cambios destructivos, cosa que no hizo falta en este proyecto; también permite, al no estar almacenado de forma local en la máquina del usuario, que se pueda trabajar en distintos lugares y distintas máquinas, lo cual sí fue útil durante este desarrollo.

En adición a esto, tener el proyecto almacenado en Github ha permitido al tutor tener acceso al proyecto al mismo tiempo que se avanzaba en él, permitiendo acceder siempre a la última versión guardada.

4.2 UML

El UML, o Lenguaje Unificado de Modelado, es un lenguaje mediante el cual se pueden especificar distintos diagramas, siguiendo un patrón de estandarización. Al principio de este proyecto fue desarrollado un diagrama UML que fue facilitado por el tutor al autor, como guía básica a seguir para el desarrollo del simulador. Este gráfico, que se puede ver en la figura 4.1, recoge las clases necesarias para un proyecto de Sistemas P (sin incluir los plásmidos), y las relaciones entre ellas.

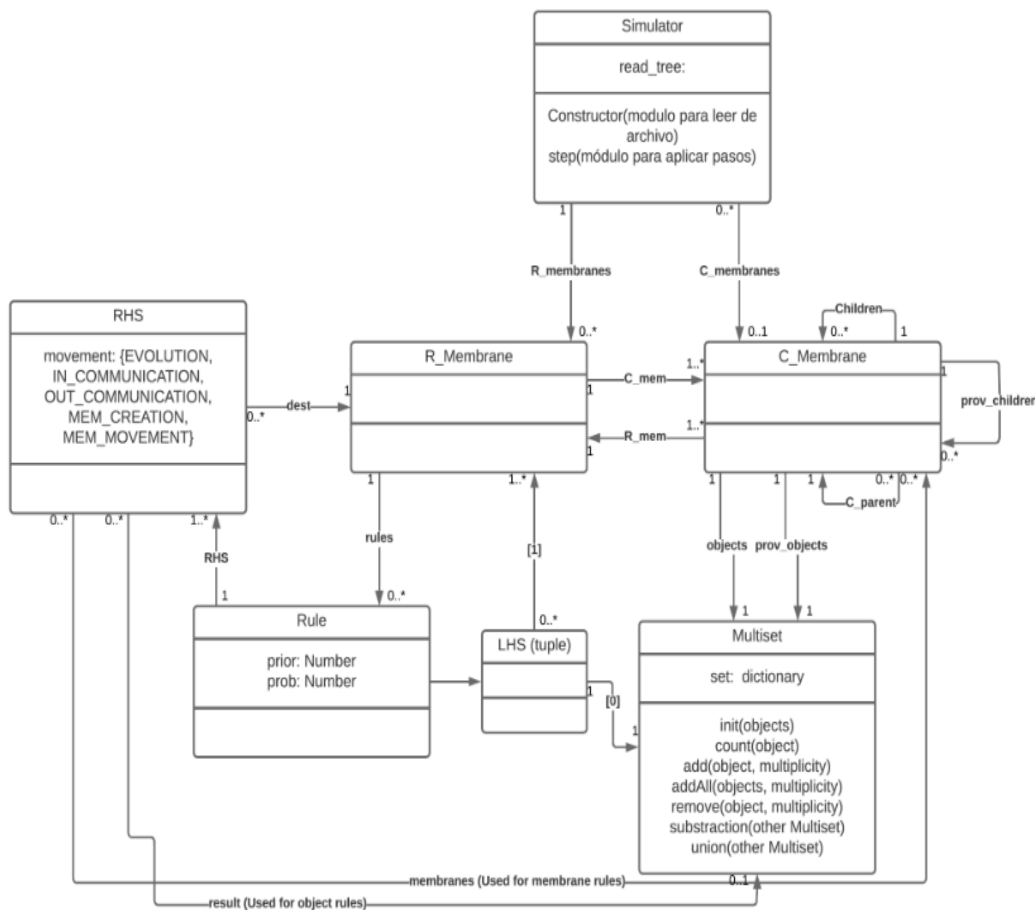

Figura 4.1: Diagrama UML sin plásmidos.

Sin embargo, las clases que se van a proceder a explicar no son las que se ven en este diagrama, sino las que aparecen en el que se ha diseñado específicamente para la creación de este proyecto, en el cual se introduce la clase *Plasmid*, y se elimina la idea

de *R_Membrane*, que pretendía recoger las reglas de una membrana de forma separada al resto de objetos, que se encontrarían en la clase *C_Membrane*. A continuación se va a mostrar este segundo diagrama comentado (figura 4.2) y se va a explicar más en detalle la función de cada clase en el simulador.

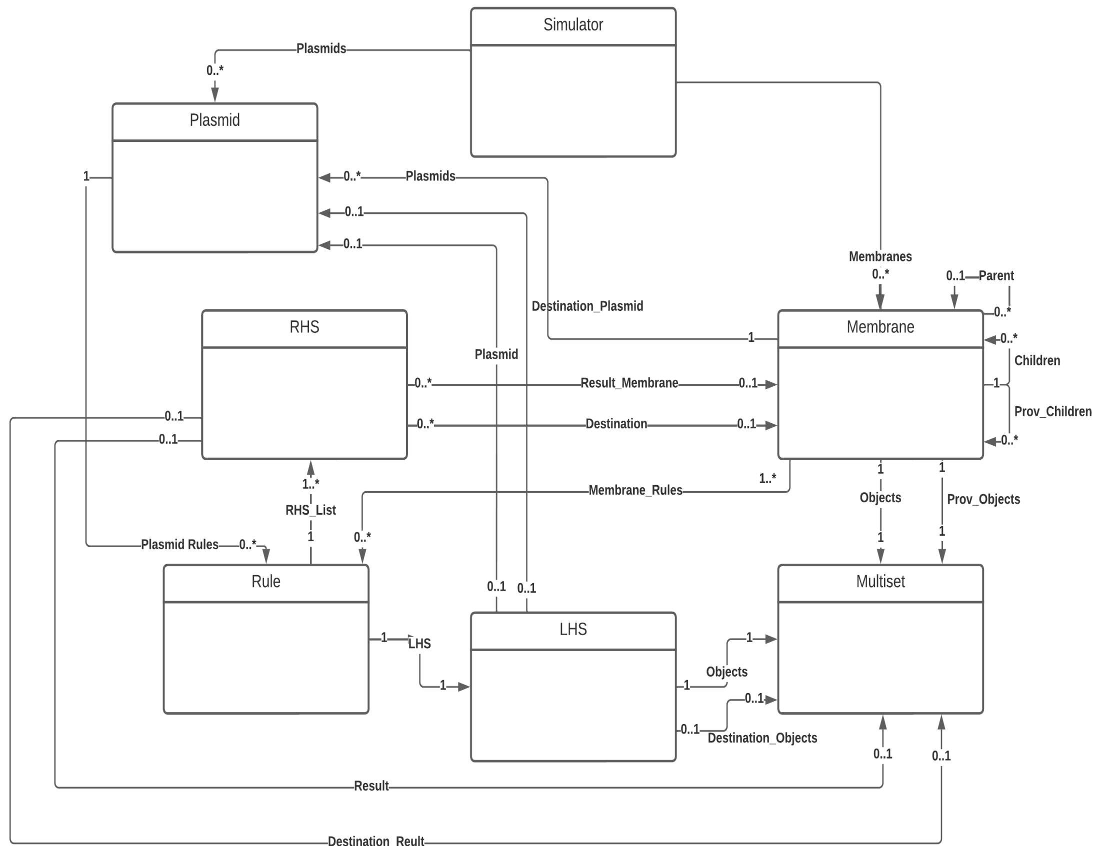

Figura 4.2: Diagrama UML con plásmidos.

1. Membrane: Esta clase representa a la idea de membrana. Cada membrana puede tener sus propias reglas, otras membranas hijas, así como una membrana padre. Además, debido a la finalidad de este TFG, también aparece en la membrana un conjunto de plásmidos. Por lo tanto, los atributos que aparecen en esta clase son:

    a) Parent: Referencia a la membrana padre, si lo tiene.[5]

    b) Children: Referencias a las membranas hijas, si las tiene.

    c) Prov_Children: Referencias a las membranas hijas que se han generado en este paso (paso de ejecución, al ejecutar una regla de esta membrana).

    d) Objects: Conjunto de datos propios de la membrana. Se pueden usar para ejecutar reglas tanto de esta membrana como las de su membrana padre, si lo tuviera.

    e) Prov_Objects: De la misma manera que "Prov_Children", este es el conjunto de objetos generados durante este paso de ejecución. Al finalizar el paso, este conjunto se añade al de "Objects" y se vacía para el siguiente paso.

[5]En esta implementación se representa al entorno en el que se desarrolla toda la acción como una membrana padre que lo contiene todo. Así mismo, las reglas del entorno aparecen como reglas de esta membrana.

f) Plasmids: Conjunto de plásmidos que se encuentran en esta membrana en este momento. [6] Como ya se verá más adelante, cada plásmido contiene un conjunto propio de reglas, por lo que cada plásmido de la membrana añade todas sus reglas a las de la membrana como si fueran de ella, mientras este se encuentre en la membrana.

2. Multiset: Esta clase representa el multiconjunto de datos de las membranas. Esta clase también se usa como los objetos que aparecen en las reglas, como se va a ver a continuación. En la siguiente sección se profundizará en la implementación de cada una de estas entidades.

3. Rule: Instancias de esta clase son las reglas de las membranas o de los plásmidos. Además de esta entidad, aparecen como parte de cada regla su parte izquierda y su parte derecha representadas en otras entidades, lo que da mucha flexibilidad a la hora de crear un sistema con reglas complejas. Los atributos de esta clase son, por tanto:

   a) LHS: Un objeto de tipo LHS, que veremos más adelante.

   b) RHS_List: Lista de partes derechas de una regla. Esto está implementado así ya que en los Sistemas P se pueden diferenciar varios tipos de acciones (esto se explicará en la definición de la clase RHS, mas se puede adelantar que aparecen reglas de objetos, plásmidos y membranas), por lo que cada parte derecha de una regla representa a una acción propia de la misma.

4. LHS: Como ya se ha mencionado, esta entidad representa a la parte izquierda de una regla, es decir, que contiene los objetos que es estrictamente necesario encontrar en la membrana que posea esta regla, para poder ejecutarla. Por lo tanto, como atributos de esta clase aparecen:

   a) Objects: Los objetos necesarios a encontrar en la membrana que contenga esta regla.

   b) Destination_Objects: Los objetos que es necesario encontrar en la membrana destino de la regla, en caso de que lo tenga (por ejemplo, en una regla de movimiento de plásmido).

   c) Plasmid: Necesario para el tipo de reglas de plásmidos (reglas de movimiento de plásmidos). Este objeto representa el plásmido que se debe encontrar en la membrana que contiene esta regla.

   d) Destination_Plasmid: Al igual que el campo anterior, este objeto es el plásmido que se debe encontrar en la membrana destino.

5. RHS: Tal y como ya se ha visto, una regla tiene una lista de partes derechas, ya que cada una de estas realiza una acción. Estas acciones son, en esta implementación: [7]

   a) Tipos de reglas para objetos:

      1) EVOLUTION: Evolución de objetos de una regla. Los objetos de la parte derecha de una regla se transforman en los de la parte derecha.

[6] En esta implementación se ha permitido que una membrana pueda albergar más de un plásmido al mismo tiempo, ya que así ocurre en la biología real. Sin embargo, en el planteamiento inicial una membrana tan solo podría contener un plásmido al mismo tiempo.

[7] Las distintas variantes de Sistemas P ofrecen tipos de reglas de distinta naturaleza. En esta implementación se han cogido los más básicos de ellos, además de los tipos necesarios para incluir los plásmidos al sistema.

2) IN_COMMUNICATION: Movimiento de objetos entre membranas, en el que una membrana padre se comunica con una de sus membranas hijas. Los objetos de la parte izquierda desaparecen en la membrana que contiene la regla, y en la membrana destino se añaden los objetos que aparecen en la parte derecha.
3) OUT_COMMUNICATION: Otro tipo de movimiento de objetos entre membranas, aunque en este caso la membrana que contiene la regla es la membrana hija, y el destino es su membrana padre.

b) Tipos de reglas para membranas:

1) MEM_CREATION: En un sistema P tradicional, este tipo de regla representa una mitosis o división celular en biología. Esto es, si la membrana hija de la membrana que contiene esta regla tiene los objetos especificados en la parte izquierda, se crea una copia exacta de esta membrana como hija de la misma membrana padre. Se podría decir que se crea una membrana gemela dentro de la membrana que tiene esta regla.
2) MEM_MOVEMENT: Para el movimiento de membranas. Dentro de una membrana, una membrana hija puede moverse dentro de otra, y pasar a ser hija de esta última.

c) Tipos de reglas de plásmidos, basados en el comportamiento de los plásmidos tal y como se han estudiado en biología:

1) IN_SYMPORT: Movimiento de un plásmido hacia el dominio de una membrana. La membrana que contiene la regla obtiene el plásmido indicado, ya sea de una membrana padre o una membrana hija, indicada como destino de la regla.
2) OUT_SYMPORT: De manera muy similar al anterior, este tipo de regla realiza el movimiento de un plásmido desde la membrana que contiene la regla hacia una membrana destino, que puede ser su membrana padre o una membrana hija.
3) ANTYPORT: Este tipo de reglas indica un movimiento de tipo intercambio de plásmidos, en la que una membrana contiene un plásmido, y una membrana hija suya contiene otro, y estos intercambian posiciones.

Además, en estos dos tipos de reglas, también aparecen objetos necesarios para ejecutar las reglas, y también se realizan operaciones con estos objetos como en los tipos de reglas de objetos.

Una vez definidos los tipos de partes derechas de reglas que pueden aparecer en este sistema, se puede pasar a enumerar los atributos de esta clase RHS:

a) Result: Los objetos que se generan en la membrana que contiene la regla.
b) Destination_Result: Los objetos generados en la membrana destino de la regla, si lo hay.
c) Destination: Referencia a la membrana destino.
d) Result_Membrane: Membrana que se mueve a la membrana destino, si este existe, en las reglas de tipo plásmido.
e) Rule_Type: El tipo de la regla. Este atributo no se puede encontrar en el UML, ya que en este solo se muestran las entidades y sus relaciones entre ellas.

6. Plasmid: Esta clase representa a la idea de plásmido que se ha introducido en esta nueva variante de sistema P propuesta. Este tan solo contiene un conjunto de reglas propias a él mismo. Cuando el plásmido se encuentra en una membrana, sus reglas

pasan a tomarse como propias de la membrana, mas en el momento en el que el plásmido abandona esta membrana sus reglas se mueven con él.

7. Simulator: Esta clase es el cerebro del simulador. Contiene toda la información del sistema, esto es: las membranas, los plásmidos, las reglas y todo lo anteriormente mencionado. Además, esta clase contiene la lógica de la ejecución del simulador, aunque esto se explicará profundamente en un apartado de la sección posterior.

Tras definir todas las entidades que componen la estructura de clases del simulador, se va a proceder a explicar su desarrollo en código escrito en Java.

4.3 Implementación de código

Llegado a este punto, se va a pasar a explicar de manera detallada el funcionamiento e implementación de código de cada una de las entidades vistas en la sección anterior. Además, se verán también algunas clases auxiliares que aportan funcionalidades al simulador, como son por ejemplo los métodos que convierten un archivo JSON a un objeto usable por el simulador, y viceversa, para la visualización del resultado.

4.3.1. Las entidades

Siguiendo la misma metodología empleada para definir las clases del UML en la sección anterior, en este apartado se va a hacer una enumeración de cada una de las entidades definidas, explicando de manera lo más concisa y relevante posible los atributos de cada una así como su función en el código. Para mayor entendimiento se va a seguir el mismo orden de enumeración que se ha realizado en la anterior sección.

A la hora de enumerar los atributos de una entidad, se hará siguiendo la siguiente metodología: primero se escribirá el nombre que recibe el atributo dentro del código, seguido de el tipo del atributo entre corchetes, y finalizando con una descripción del mismo. Gran parte de los atributos que se van a enumerar ya se han definido en la sección anterior, por lo que no se va a realizar una explicación demasiado detallada de cada uno de ellos.

Por último, es importante mencionar que en algunas de las clases se mencionarán algunos de sus métodos implementados, pero sin mostrar el código de los mismos. Además de los métodos visibles en el código, se ha hecho uso de la anotación "@Data" de Springboot, que entre otras cosas proporciona de manera automática los métodos *Getter* y *Setter* de todos los atributos de la clase, por lo que todos ellos serán *private*.

Membrane

Esta es una de las clases con más información del simulador, ya que contiene todo lo relevante a una membrana en un sistema P, contando además con los plásmidos. Por lo tanto, los atributos de esta clase son:

1. id [String]: Para simplificar parte de la lógica del simulador, se ha hecho uso de atributos de tipo Id, por lo que a la hora de escribir un archivo de entrada, una lista

de membranas podría venir representada como una lista de sus Ids. Cabe destacar que esta Id debe de ser única para cada membrana. En adición, este atributo aporta otras funcionalidades que se verán más adelante.

2. objects [Multiset]: Los objetos propiamente dichos de la membrana. En breve se definirá la composición del tipo Multiset.

3. prov_objects [Multiset]: Los objetos generados en un paso de ejecución. Esto se verá más en detalle en la clase Rule.

4. children [Set<Membrane>]: Referencias a las clases hijas de la membrana. Como se puede observar, el conjunto de datos que las agrupa es de tipo Set y no List, dado que cada membrana es única, y no se podría tener dos referencias a la misma membrana como si fueran diferentes. De esta manera, la clase Set de Java, no permite añadir un objeto a un conjunto de datos en el que ya se encuentra dentro.

   Hay que mencionar, además, que para poder usar el tipo Set, se deben realizar comparaciones entre objetos de esta clase Membrane, y al no ser una clase propia de Java, se debe implementar externamente el método "int hashCode()". Esto permite asignar un entero a un objeto, y una comparación entre objetos no es más que una comparación entre estos números enteros. Es por esta razón, que se ha implementado este método en esta clase, haciendo que el *hash code* de un objeto Membrane sea igual al de su atributo "id", que al ser de tipo String ya viene definido.

5. prov_children [Set<Membrane>]: Referencias a las clases hijas generadas para esta membrana en un paso de ejecución.

6. parent [Membrane]: Referencia a la membrana padre de esta membrana.

7. rules [Set<Rule>]: Conjunto de reglas propias de la membrana.

8. plasmidIds [Set<String>]: Al igual que se ha hecho en esta clase, la clase Plasmid también contiene un atributo de tipo Id, por lo que, por simplicidad de código, en lugar de tener un conjunto de plásmidos se tiene un conjunto de sus Ids, que es lo necesario para realizar comparaciones dentro de las reglas.

Por último, cabe mencionar que esta clase tiene algunos métodos útiles como el método "getAllRules", que devuelve un conjunto de reglas que reúne las reglas propias de la membrana junto a las reglas de cada uno de los plásmidos dentro de ella.

Multiset

Esta clase no contiene ningún atributo, ya que se ha decidido aprovechar la lógica ya aportada por Java, haciendo que esta clase extienda de la clase HashMap de Java, e indicando los tipos que se van a recibir en la clave y el valor. Por lo tanto, la cabecera de esta clase es:

$$public\ class\ Multiset\ extends\ HashMap < String,\ Integer >$$

Como se puede deducir de esta cabecera, los conjuntos de datos de tipo Multiset que aparecen en este simulador son no más que un diccionario en el que la clave es una cadena y el valor es la cantidad de objetos de ese tipo que se encuentran en el Multiset. De esta manera, se puede conocer la cantidad existente de un objeto dentro de un Multiset

con coste constante, ya que este es el coste de hacer un "get(clave)"dentro de un Map de Java. En futuras implementaciones o mejoras de este simulador, se podría hacer que la clave no estuviera restringida a un tipo String, lo que permitiría realizar simulaciones más realistas en las que una membrana pudiera tener objetos de tipos muy variados.

Como es de esperar, esta clase tiene algunos métodos adicionales que aumentan las funcionalidades dadas por la clase HashMap. Algunos de estos métodos son, por ejemplo, la función "boolean containsObjects(Multiset other)", que devuelve verdadero si los objetos del Multiset "other" están contenidos dentro del Multiset que realiza la llamada al método; o el método "void substract(Multiset other)" que sustrae del Multiset que realiza la llamada a este método los objetos del Multiset "other".

Rule

Tal y como se ha indicado en la descripción del UML, esta clase representa a las reglas de los Sistemas P, dividiéndose en parte izquierda y parte derecha. Sus atributos son:

1. lhs [LHS]: La parte izquierda de la regla. Para ejecutar esta regla, se han de cumplir las condiciones dadas en su parte izquierda.
2. rhsList [List<RHS>]: La lista de partes derechas de la regla. Son las acciones a ejecutar si se ejecuta la regla.

Por simplicidad del código se ha asumido que en una sola regla solo va a haber una acción de plásmido, ya que la parte izquierda de la regla solo puede tener en cuenta un plásmido.

Además de los atributos, esta clase contiene dos métodos, clave para la ejecución del simulador. En ambos métodos se envía un Map que relaciona una Id con su Membrane, ya que en las partes derechas de la regla aparecen las Ids de membranas, y se necesita obtener el objeto membrana para poder realizar las acciones pertinentes en ella. Por un lado se encuentra el método:

boolean validateCondition(Membrane membrane, Map<String, Membrane>membraneMap)

Esta función comprueba que la regla que llama al método se pueda ejecutar, y para ello evalúa su parte izquierda. Así mismo, primero comprueba que los objetos de la parte izquierda se encuentren en la membrana que contiene la regla, que se pasa como primer argumento en la función. Además, en caso de que alguna parte derecha sea de tipo de plásmido, comprueba también que las condiciones necesarias se cumplan. Sin embargo, estas comprobaciones se hacen como llamadas a la clase LHS, por lo que se comentarán posteriormente.

El segundo método, pero no menos importante, es la función:

void execute(Membrane membrane, Map<String, Membrane>membraneMap)

Este método es el encargado de ejecutar la regla que lo llama. Por un lado elimina de la membrana que contiene la regla, que se pasa como primer argumento de la función, los objetos que aparecen en la parte izquierda de la regla. En segundo lugar itera sobre la lista de partes derechas de la regla, ejecutando cada una de ellas. En función del tipo de parte derecha que sea, se realizan las acciones necesarias:

1. EVOLUTION: Se añaden en la membrana los objetos resultado.
2. IN_COMMUNICATION: Se añaden en la membrana destino los objetos resultado.
3. OUT_COMMUNICATION: Se añaden en la membrana padre de la membrana que contiene la regla los objetos resultado.
4. MEM_CREATION: Al conjunto de membranas hijas de la membrana se agrega una copia de la membrana resultado.
5. MEM_MOVEMENT: La membrana resultado es extraída del conjunto de membranas hijas de la membrana que contiene la regla, y a su vez se añade a la lista de membranas hijas de la membrana resultado.
6. IN_SYMPORT: A la membrana destino, que ya puede ser padre o hija de la membrana que contiene al regla, se le agregan los objetos resultado de destino y se le extrae el plásmido indicado del conjunto de plásmidos de la membrana. Por otro lado, a la membrana que contiene la regla se le agregan tanto los objetos resultado como el plásmido extraído de la otra membrana. Además, se extraen de la membrana destino los objetos de destino de la parte izquierda de la regla.
7. OUT_SYMPORT: En este tipo de regla las acciones son muy similares a las del anterior, con la diferencia de que el plásmido se extrae de la membrana que contiene la regla y se agrega a la membrana destino.
8. ANTYPORT: En este tipo de regla se realizan exactamente las mismas acciones que en el anterior, pero además se extrae otro plásmido del conjunto de plásmidos de la membrana destino, y se añade al de la membrana que contiene la regla.

LHS

El nombre de esta clase viene de *Left Hand Side of the rule*, o parte izquierda de la regla. La clase contiene toda la información que es necesario encontrar en la membrana en la que se encuentra la regla o en la membrana destino de la regla. Por ello, los atributos necesarios para llevar esto a cabo son:

1. objects [Multiset]: Multiconjunto de objetos que hace falta encontrar en la membrana de la regla para que sea ejecutable.
2. destinationObjects [Multiset]: Multiconjunto de objetos que hace falta encontrar en la membrana destino de la regla, si lo tiene.
3. plasmidId [String]: Id del plásmido que debe encontrarse en la membrana de la regla, si hay una parte derecha de tipo plásmido.
4. destinationPlasmidId [String]: Id del plásmido que ha de existir en la membrana destino de la regla, si lo hay.

Como métodos que ofrece esta clase, se encuentra uno que ya se ha mencionado:

boolean validateCondition(Multiset otherMultiset)

A esta función se le pasa como parámetro el Multiset de objetos de la membrana de la regla, y se encarga de comprobar que este Multiset contenga los objetos del Multiset “objects”.

Por otro lado, y para finalizar esta clase, existe otro método también mencionado:

boolean validatePlasmidRuleCondition(Membrane membrane, Membrane destination)

Esta función se encarga de realizar las comprobaciones pertinentes en las reglas de plásmidos, es decir, de comprobar que en la membrana que se le pasa como primer parámetro se encuentre el plásmido de la LHS, y en la membrana destino que se pasa como segundo parámetro, el plásmido y los objetos necesarios para esta membrana.

RHS

De manera similar a la clase anterior, esta representa a una parte derecha de una regla, y su nombre viene del inglés *Right Hand Side of the rule*. Esta clase contiene los atributos necesarios para realizar una acción de una regla:

1. result [Multiset]: Los objetos que se han de generar en la membrana que contiene esta regla.
2. membraneId [String]: Para las acciones de tipo movimiento de membrana, esta es la Id de la membrana que se ha de mover.
3. destinationResult [Multiset]: Los objetos que han de generarse en la membrana destino de esta parte derecha.
4. destinationMembraneId [String]: Id de la membrana destino de esta parte derecha, sobre la cual se realizan algunas acciones dependiendo del tipo de regla.
5. ruleType [RuleType]: Este atributo indica el tipo de esta parte derecha de regla. El tipo de este atributo es un enumerador que está definido en esta misma clase y que se muestra a continuación.
6. RuleType [enum]: {EVOLUTION, IN_COMMUNICATION, OUT_COMMUNICATION, MEM_CREATION, MEM_MOVEMENT, IN_SYMPORT, OUT_SYMPORT, ANTYPORT}

Esta clase tan solo contiene tres métodos auxiliares que ayudan a simplificar código:

1. isPlasmidRHS [boolean]: Indica si el tipo de la RHS es IN_SYMPORT, OUT_SYMPORT o ANTYPORT.
2. isObjectRHS [boolean]: Si el tipo de la RHS es EVOLUTION, IN_COMMUNICATION u OUT_COMMUNICATION.
3. isMembraneRHS [boolean]: Si el tipo de la RHS es MEM_CREATION o MEM_MOVEMENT.

Plasmid

Esta entidad representa a los plásmidos introducidos en este simulador, y no contiene ningún método definido en ella. Sin embargo, lo que sí que contiene son dos atributos a los que ya se ha hecho mención en las explicaciones de otras clases, que son:

1. id [String]: La Id del plásmido. Permite a otras clases guardar solamente esta Id en lugar de todo el objeto, lo cual es útil para las comprobaciones de plásmidos en las LHS, ya que tan solo la Id es relevante en ellas.

2. rules [Set<Rule>]: Conjunto de reglas del plásmido.

Antes de pasar a explicar la clase Simulator, que reúne la lógica central del simulador, es importante exponer las otras clases que se han añadido al sistema para mejorar el desarrollo, añadir funcionalidades y mejorar la legibilidad del código. Todo esto se va a desarrollar en el siguiente apartado.

4.3.2. Utilidades y servicios

En primer lugar, se ha introducido una clase que engloba todo lo referente al sistema sobre el cual se va a realizar la simulación, esto es, las membranas, las relaciones entre ellas, las reglas, los plásmidos, etc.

En segundo lugar, se han creado clases de tipo DTO [8] para la conversión entre los archivos de entrada y salida, y los objetos propiamente dichos que usa el simulador. Estas clases son *P_System_Dto* y *MembraneDto*.

Por último, se ha creado una clase denominada *ServiceUtils* que aporta funcionalidades extra al simulador, como son la conversión de archivos JSON a objetos de Java y viceversa, o algunas utilidades auxiliares. Ahora se va a proceder a detallar cada una de estas clases mencionadas.

P_System

Como ya se ha dicho, esta clase está diseñada para almacenar la información del simulador. Para ello, se compone de los siguientes atributos:

1. membraneMap [Map<String, Membrane>]: Diccionario que relaciona cada Id de membrana con su membrana objeto.

2. plasmidMap [Map<String, Plasmid>]: De manera similar que con el atributo anterior, es un diccionario que relaciona cada Id de plásmido con su plásmido objeto. Es importante que cada membrana y cada plásmido tengan una Id única, ya que sino en estos diccionarios tan solo se guardaría el último objeto leído con la misma Id. Sin embargo, una membrana y un plásmido pueden tener la misma Id, ya que se almacenan de manera completamente independiente.

3. rootId [String]: Id de la membrana que ocupa el puesto de raíz del árbol. En esta implementación, esta membrana será la que represente al ambiente, que contendrá a la raíz del árbol de membranas propiamente dicha.

4. outputId [String]: Id de la membrana que se considera como salida. La información que contenga esa membrana al final de la simulación, será considerada como resultado de la operación que el sistema esté diseñado a resolver.

[8] Data Transfer Object (DTO), o Objeto de Transferencia de Datos, son objetos creados con la finalidad de transportar datos entre procesos.

Con esta lista de atributos propios de la clase, es suficiente para almacenar toda la información necesaria para representar a un sistema P con plásmidos. Sin embargo, existe un problema a la hora de escribir un archivo de entrada, y es que algunos objetos de este sistema tienen referencias a otros objetos, que si se tuvieran que escribir en un archivo de entrada en formato JSON, se tendrían que escribir completos y de forma repetida tantas veces como referencias existan. Por esa razón, se ha decidido crear una clase DTO para esta misma clase así como de la clase Membrane, en la que las referencias se realizan mediante la Id, y no guardando el objeto completo.

MembraneDto

Esta es una clase creada para representar a los objetos de membranas en un archivo de entrada o de salida. A continuación se va a presentar el problema que existiría si se usara la clase Membrane en lugar de esta.

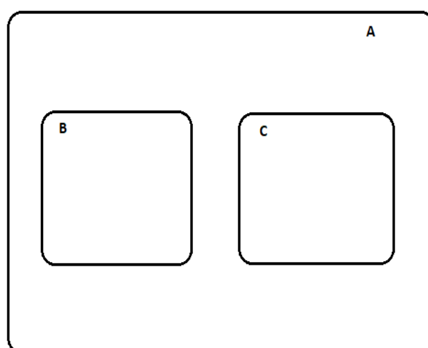

Figura 4.3: Representación de la estructura de membranas del problema propuesto.

Como ya se ha dicho en secciones anteriores, una membrana tiene referencias tanto a su membrana padre como a sus hijas, y en la clase Membrane, el atributo utilizado para almacenar esta dependencia es también de tipo Membrane. Se va a usar como ejemplo un sistema (figura 4.3) en el que existe una membrana A, que tiene a las membranas B y C como hijas. Si se convirtiera el objeto de la membrana B a formato JSON con el fin de representarlo en un archivo de salida que representara el resultado de la simulación, nos encontraríamos con el siguiente problema: se representa la membrana B con todos sus atributos, entre los cuales está su membrana padre A, que a su vez se representaría completamente con todos sus atributos, y entre ellos sus membranas hijas, entre las cuales se encuentra la membrana B. En conclusión, se crea un bucle infinito con cada una de las relaciones bidireccionales entre membranas. Sin embargo, este problema se puede resolver muy fácilmente si las referencias entre objetos se realizan usando las Ids de los objetos referenciados.

Una vez aclarado por qué es importante el uso de la clase MembraneDto, se procede a exponer sus atributos:

1. id [String]: Id única que representa a la membrana.
2. objects [Multiset]: Los objetos de la membrana, almacenados de la misma manera que en la clase Membrane.
3. plasmidIds [Set<String>]: Conjunto de Ids de los plásmidos de la membrana.

4. childrenIds [Set<String>]: Conjunto de Ids de las membranas hijas de esta membrana. En ambos casos se ha usado el conjunto de datos Set de Java, ya que las Ids han de ser únicas.

5. parentId [String]: Id de la membrana padre de esta membrana.

6. rules [Set<Rule>]: Conjunto de reglas propias de la membrana.

Como cabe suponer, si en los archivos de entrada y salida se usa este tipo de objeto, mientras que en el simulador se usa el tipo Membrane, es necesario realizar una conversión entre ambas clases, en las que se ha de realizar una resolución de dependencias en aquellos casos en los que se pase de almacenar una Id a almacenar el objeto completo, como es el caso de las referencias a las otras membranas. No obstante, esta resolución de dependencias se va a mostrar en el siguiente apartado.

P_System_Dto

Esta clase auxiliar tiene la finalidad de ser usada para escribir los archivos de entrada y salida, y es una variante de la clase P_System. Los atributos de ambas son muy similares:

1. membranes [List<MembraneDto>]: Lista de membranas (usando la clase MembraneDto para representarlas).

2. plasmids [Set<Plasmid>]: Conjunto total de plásmidos del sistema.

3. rootId [String]: Id de la membrana raíz del sistema.

4. outputId [String]: Id de la membrana de salida del sistema, donde se expone el resultado.

Ya se ha comentado en el apartado anterior que existía una conversión entre los DTOs y sus entidades, pues es en el constructor de la clase P_System donde se realiza. Este constructor recibe un solo argumento de tipo P_System_Dto y realiza las conversiones necesarias para crear un objeto de tipo P_System.

Código dentro de la clase P_System:

```
1  public P_System(P_System_Dto input) {
2      //Los atributos con tipos primitivos e identicos entre las dos clases
3      // se pueden instanciar de forma normal.
4      this.rootId = input.getRootId();
5      this.outputId = input.getOutputId();
6
7      //Para copiar los plasmidos de una lista en un mapa,
8      // se hace usando la Id como clave y el plasmido como valor.
9      for (Plasmid plasmid : input.getPlasmids()) {
10         this.plasmidMap.put(plasmid.getId(), plasmid);
11     }
12
13     //De manera similiar a los plasmidos, se guardan las membranas en un mapa
14     // y se almacenan tambien en una variable del metodo las MembraneDtos en
15     // otro mapa, que se usara para resolver las dependencias.
16     Map<String, MembraneDto> membraneDtoMap = new HashMap<>();
17     for (MembraneDto dto : input.getMembranes()) {
18         membraneDtoMap.put(dto.getId(), dto);
```

```
        Membrane membrane = new Membrane(dto);
        membraneMap.put(dto.getId(), membrane);
    }

    //Se resuelven las dependencias de cada una de las membranas.
    for (Membrane membrane : this.membraneMap.values()) {
        membrane.solveDependencies(
            membraneDtoMap.get(membrane.getId()),
            this.membraneMap
        );
    }
}
```

Se puede observar como al final de este método se hace una llamada a un método de la clase Membrane. Se expone también este otro método:

```
public void solveDependencies(MembraneDto dto, Map<String, Membrane>
    membraneMap) {
    //Para cada una de las Ids de membrana hija del Dto de esta membrana
    // se agrega el objeto Membrane a la lista de hijas de esta membrana,
    // usando el mapa que relaciona cada Membrane con su Id.
    for (String childId : dto.getChildrenIds())
        this.children.add(membraneMap.get(childId));

    //Se hace lo mismo para la membrana padre.
    this.parent = membraneMap.get(dto.getParentId());
}
```

La clase ServiceUtils

En primer lugar, dentro de esta clase vienen implementados los métodos para convertir un archivo de entrada en un objeto de tipo P_System_Dto, así como para convertir un objeto de esta clase a un archivo de salida. La representación de estos archivos, por lo tanto, no es más que una representación en formato JSON del objeto comentado. Más adelante se mostrarán algunos ejemplos.

Para realizar estas conversiones se ha hecho uso de una librería especializada, más concretamente de la clase ObjectMapper de Jackson[3]. Esta clase permite realizar una conversión de un objeto de una clase a un archivo JSON, y viceversa. Para ello, basta con indicar la clase a la que se quiere convertir o de la que se quiere convertir, así como el archivo JSON del que se debe leer o escribir en él.

Así mismo, en esta clase ServiceUtils se han creado dos métodos "P_System_Dto readInput(String filename)" y "void fromSystemToJson(P_System)" que realizan estas acciones descritas.

Por otro lado, esta clase sirve de utilidad para implementar métodos auxiliares, que ayudan a tener una mayor limpieza en otras clases del proyecto. A pesar de que a lo largo del desarrollo del proyecto esta clase ha albergado varios de estos métodos, finalmente solo ha quedado uno debido a optimización de código. A este único método se le pasan una lista de reglas y una membrana, y se encarga de seleccionar una regla a ejecutar. Para ello, filtra la lista de reglas, seleccionando únicamente aquellas que se pueden ejecutar en la membrana recibida haciendo uso de los métodos de validación de reglas anteriormente descritos, y posteriormente es escogida una de estas reglas de forma aleatoria, haciendo uso de la clase Random de *java.util*. En caso de que ninguna regla sea válida este mé-

todo devolverá *null*, como se va a ver en la clase Simulator en el apartado que viene a continuación.

4.3.3. La clase Simulator y flujo del simulador

En la clase Simulator es donde se desarrolla la lógica principal del simulador. Esto es, realizar las iteraciones de simulación indicadas, seleccionar las reglas, ejecutarlas, etc. Esta clase contiene un solo atributo, que es un objeto de tipo P_System, que como ya se ha visto es suficiente para representar una instancia de un sistema P con plásmidos.

A pesar de que toda la lógica de esta clase se podría haber reunido en un solo método que realizase la simulación, el código se ha separado en un par de métodos más para mayor legibilidad.

En primer lugar, sin contar los constructores, los Getters o los Setters, se encuentra el método "simulate()" sin argumentos, que sirve para realizar una simulación con un número de iteraciones predeterminado (aquí se ha decidido que este número sea de 100 iteraciones) en caso de que no se indicase ninguno. Por lo tanto, este es solo un método lanzadera para el siguiente método.

En segundo lugar está el método principal de esta clase: "simulate(int iterations)" que realiza una simulación dado el número de iteraciones a realizar. Este método realiza en cada iteración un recorrido de todas las membranas del sistema, en orden desde la raíz hasta las hojas del árbol de membranas, y realiza una ejecución propia de cada membrana, llamando al siguiente método.

Por último, el método "executeMembrane(Membrane membrane)" es el encargado de ejecutar la membrana indicada. Para ello, selecciona una regla llamando al método "selectRule" de ServiceUtils pasando todas las reglas de la membrana y sus plásmidos, con la llamada al método "getAllRules" de la clase Membrane. Una vez seleccionada la regla, se ejecuta tantas veces como se pueda con la llamada al método "execute" de la clase Rule.

Como se puede observar, haber separado la lógica entre las distintas clases del proyecto libra a este método de tener una cantidad excesiva y poco legible de código, además de aportar a cada clase las funcionalidades propias de la misma.

En este punto de la memoria ya se han definido todas las clases implementadas y sus métodos, así como explicado su función en el proyecto. Sin embargo, se va a exponer ahora un flujo de los pasos que sigue el simulador en su totalidad que permita comprender de manera clara y concisa el funcionamiento de la aplicación desarrollada.

1. Se ejecuta la aplicación.
2. Entrada de los datos.
   a) Se muestra un menú (por consola) que permite seleccionar las opciones de realizar una simulación o salir y cerrar la aplicación. Si se selecciona la opción de realizar la simulación se pasa al siguiente paso, sino se cierra y se termina la aplicación.
   b) Se piden el nombre del archivo de entrada (archivo en formato JSON) y el número de iteraciones a realizar por el simulador. El archivo de entrada es usado para crear un objeto de tipo P_System, con el que se va a crear la instancia de la clase Simulator.

3. Ejecución del simulador. En cada iteración:

   a) Se obtiene la membrana raíz del sistema y se añade a la lista de membranas a ejecutar.

   b) Se obtiene la primera membrana de la lista de membranas a ejecutar, y se ejecuta el paso 4. Tras ejecutar la membrana se añaden sus membranas hijas a la lista de membranas a ejecutar y se retira esta, y se repite este paso hasta que la lista quede vacía.

4. Ejecución de la membrana seleccionada.

   a) Se obtiene la lista de reglas de la membrana, uniendo las reglas propias de la misma junto a las reglas de los plásmidos que se encuentren en ese momento en ella.

   b) Se selecciona aleatoriamente una de estas reglas de entre las que se puedan ejecutar porque se cumple su parte izquierda.

   c) Se ejecuta la regla seleccionada tantas veces como sea posible, realizando todos los movimientos y transformaciones de objetos, membranas o plásmidos que vengan indicados en ella.

   d) Se repite desde el paso a hasta que ninguna regla pueda ser seleccionada porque no se puede ejecutar.

5. Salida de los datos: se genera un archivo JSON con el objeto P_System que ha resultado tras realizar todas las iteraciones de simulación.

CAPÍTULO 5

Casos de prueba

En este capítulo se van a mostrar varios ejemplos de casos de prueba con los que se ha probado el simulador. En primer lugar, estos casos de prueba se van a definir en lenguaje matemático ya que ambos representan un sistema P con plásmidos. Seguidamente se mostrará el flujo de los pasos que se realizan en la simulación. Además, en los apéndices se pueden encontrar los archivos de entrada, que se han realizado traduciendo las definiciones que se van a presentar al formato JSON, y los archivos de salida, que muestran una representación del sistema una vez terminada la simulación.

Los ejemplos que se van a presentar a continuación son dos, y ambos sirven para realizar operaciones aritméticas sencillas, ya que la definición de estos casos de prueba es muy compleja y densa, y en este trabajo tan solo se quiere demostrar el uso del simulador implementado. Así pues, el primero de ellos realiza una resta aritmética de dos enteros, mientras que el segundo realiza una multiplicación.

5.1 Resta aritmética

A continuación se va a ver que para este caso de prueba se va a hacer uso de tan solo tres membranas. La membrana raíz, denominada s, contiene dos membranas hijas: 1 y 2. En la membrana 1 habrá inicialmente un número entero n de objetos a, mientras que en la membrana 2 habrá un número entero m de objetos b. El objetivo de este sistema es, por lo tanto, que al final de la ejecución el resultado de la resta aritmética en valor absoluto $|n - m|$ quede recopilado en la membrana 2. Si n es mayor que m, en la membrana 2 se encontrarían $n - m$ objetos a, mientras que si m fuera mayor, ese resultado se vería reflejado en el número de objetos b.

En este ejemplo, tanto la membrana 1 como la 2 no poseen ninguna regla, por lo que las operaciones que se realicen dentro de las mismas vendrán dadas por las reglas de los plásmidos que se van a incorporar en este sistema. Por un lado, el plásmido p_1 envía los objetos a que encuentra en su membrana a la membrana padre de esta. Por otro lado, el plásmido p_2 transforma un objeto a y otro b, en un objeto c que envía fuera de la membrana.

Así pues, el plásmido p_1 está destinado a introducirse en la membrana 1, mientras que el plásmido p_2 se introduce en la membrana 2.

Definición

Se va a poner a la derecha de cada regla un identificador entre paréntesis para hacer referencia a ella en el siguiente apartado.

$p_1 : \{a \rightarrow a_{out}\}$ (1)

$p_2 : \{ab \rightarrow c_{out}\}$ (2)

$\mu = [_s[_1]_1[_2]_2]_s$

$w_s = pq$

$w_1 = a^n$

$w_2 = b^m$

$i_0 = 2$

Reglas de las regiones:

$[_s\, a[_2]_2]_s \rightarrow [_s[_2\, a\,]_2]_s$ (3)

Reglas para los plásmidos:

$p_1[_s\, p]_s \rightarrow p[_s\, p_1]_s$ (4)

$p_2[_s\, q]_s \rightarrow q[_s\, p_2]_s$ (5)

$p_1[_1]_1 \rightarrow [_1\, p_1]_1$ (6)

$p_2[_2]_2 \rightarrow [_2\, p_2]_2$ (7)

Flujo de la simulación

Iteración 1:

1. Se ejecutan las reglas 4 y 5, introduciendo los plásmidos p_1 y p_2 en la membrana s.
2. Se ejecutan las reglas 6 y 7 en la membrana s. Esta acción introduce el plásmido p_1 en la membrana 1 y el plásmido p_2 en la membrana 2.
3. Para cada a que se encuentra en la membrana 1 (hay n a's) se ejecuta la regla 1 del plásmido p_1, lo que mueve n a's a la membrana s.

Iteración 2:

1. En la membrana s se ejecuta la regla 3, moviendo las n a's a la membrana 2.
2. En la membrana 2, se ejecuta la regla 2, del plásmido p_2, que transforma cada cadena ab en un objeto c que se mueve a la membrana s.

En conclusión, en tan solo dos iteraciones se ha podido realizar una resta aritmética, cuyo resultado se encuentra en la membrana 2.

Ejemplo

Para este sistema diseñado se ha creado un ejemplo de prueba, tal que $n = 4$ y $b = 8$. Esto puede encontrarse en el campo "objects" de las membranas 1 y 2 respectivamente del archivo en formato JSON que se encuentra en el apéndice A.1. A su vez, el resultado de este ejemplo se encuentra en el mismo campo "objects" de la membrana de salida, la membrana 2, al final de la computación. Este resultado se puede encontrar en el apéndice A.2. Como se puede observar, en esta membrana se encuentran 4 b's, ya que este es el resultado de $|4 - 8|$.

En la siguiente figura (figura 5.1), se muestra la configuración inicial (izquierda) y final (derecha) del ejemplo creado. Los plásmidos están representados en color rojo y los objetos en color verde.

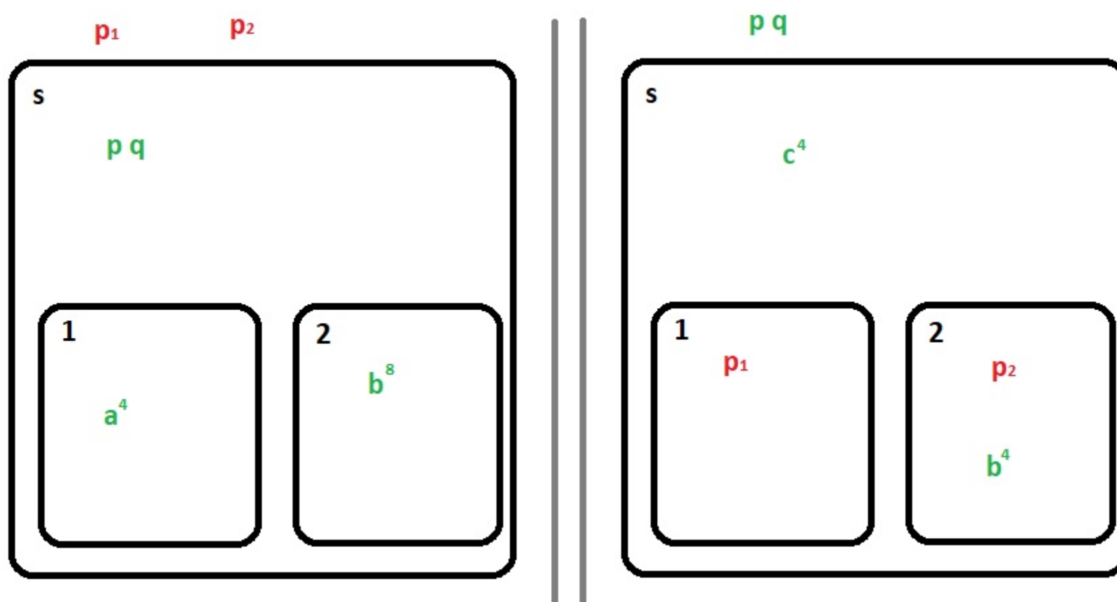

Figura 5.1: Diagrama de Venn del ejemplo del caso de prueba de resta aritmética.

5.2 Producto matemático

Aumentando ligeramente la complejidad, se ha diseñado también un segundo caso de prueba capaz de realizar productos de números enteros. La jerarquía de membranas es la misma que en el caso anterior (membranas s, 1 y 2), y el número y nombre de los plásmidos son también los mismos (p_1 y p_2).

En este caso, dentro de la membrana 1 se encontrará inicialmente un objeto b y n objetos a, mientras que en la membrana 2 se encontrarán 1 objeto b y m a's. Además, la membrana s tendrá un objeto p en su interior.

En este sistema el plásmido p_1 se introduce en la membrana 1 cada vez que encuentra un símbolo a en ella, habiendo pasado primero por la membrana s. En ese momento, el plásmido p_1 sustrae un objeto a a la membrana s y el plásmido sale. A su vez, cada vez que este plásmido realiza esa acción, el plásmido p_2 se introduce en la membrana 2, genera un objeto b en la membrana s por cada objeto a que encuentra y sale. Por lo tanto, al final de la ejecución se podrán encontrar en la membrana s una cantidad $m * n$ de símbolos b.

Finalmente, se puede deducir que la complejidad de este sistema es mayor al anterior, ya que en cada iteración uno de los plásmidos se introduce en su membrana y realiza las acciones pertinentes, y el número de iteraciones depende de n.

Definición

$p_1 : \{ba \rightarrow b\}$

$p_2 : \{a \rightarrow a\ b_{out}\}$

$\mu = [_s[_1]_1[_2]_2]_s$

$w_s = p$

$w_1 = ba^n$

$w_2 = ba^n$

$i_0 = s$

Reglas para los plásmidos:

$p_1[_s\ p]_s \rightarrow [_s\ p_1 p']_s$

$p_2[_s\ p']_s \rightarrow [_s\ p_2 q]_s$

$p_1 q[_1\ a]_1 \rightarrow r[_1\ p_1 a]_1$

$r[_1 p_1]_1 \rightarrow p_1 s[_1]_1$

$p_2 s[_2]_2 \rightarrow t[_2\ p_2]_2$

$t[_2\ p_2]_2 \rightarrow p_2 q[_2]_2$

Ejemplo

En el apéndice A.3 se puede encontrar un ejemplo implementado, en formato JSON, en el que $n = 4$ y $m = 5$. Esto se puede ver en el campo "objects" de las membranas 1 y 2. Así mismo, el resultado puede verse en el apéndice A.4, en el que se muestra la salida, también en formato JSON, de la entrada anterior. En este archivo, puede encontrarse el resultado del producto en la membrana s, concretamente en el número de b's dentro del campo "objects" de esta membrana.

De forma análoga a como se ha hecho para el caso anterior, en la figura 5.2 se muestra la configuración inicial (izquierda) y final (derecha) del ejemplo propuesto para el caso de prueba del producto matemático. Los plásmidos se representan en color rojo, mientras que los objetos vienen en color verde.

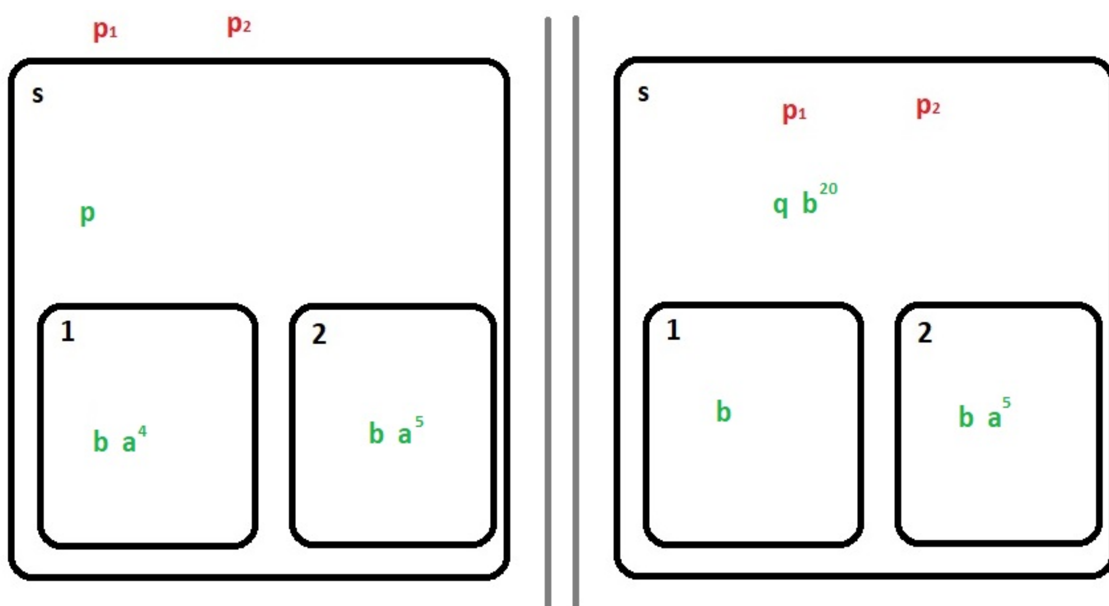

Figura 5.2: Diagrama de Venn del ejemplo del caso de prueba de producto de enteros.

CAPÍTULO 6
Conclusiones

Esta memoria está llegando a su fin, pues en este último capítulo se van a presentar las conclusiones extraídas respecto a los objetivos inicialmente propuestos en la sección 1.2 de la memoria.

En primer lugar se ha descrito el estado actual de los Sistemas P, sus aplicaciones y la definición formal del modelo que se presentó al mundo en 1998. Para ello se ha hecho también una breve definición de los aspectos biológicos relacionados con el tema para poder comprender mejor este modelo de computación. Esto se puede encontrar en el capítulo 2: *Estado del arte*.

Seguidamente se abre paso el capítulo 3: *Sistemas P con plásmidos*, en el que inicialmente se define el concepto de *plásmido* tal y como aparece en la biología molecular, así como de sus aplicaciones reales actuales, como son la genética y la biotecnología. En este capítulo se exponen las ventajas que podría suponer introducir este elemento biológico a los Sistemas P, entre las cuales aparecen un aumento de la flexibilidad y eficiencia de la computación con membranas.

Por lo tanto, a lo largo de este capítulo se cumple uno de los objetivos planteados para este TFG, que es el diseñar y definir formalmente un modelo de Sistemas P con plásmidos. Además, este se plantea como una extensión del modelo sin plásmidos definido en el capítulo anterior, lo cual permite que a continuación se cumpla otro de los objetivos: demostrar que el modelo de Sistemas P con plásmidos es universal. Resumiendo la definición de este último aspecto comentado, explicado en el capítulo 1: *Introducción*, se puede decir que un sistema es universal, o Turing-completo, si es equivalente a la Máquina de Turing, es decir, que puede resolver cualquier problema Turing-computable en un tiempo finito (para una mayor definición, ver capítulo 1).

Demostrar que el modelo propuesto en este TFG es Turing-completo resulta bastante sencillo. Se puede deducir, sin lugar a dudas, que el modelo de *Sistemas P con plásmidos* es completamente equivalente al modelo de *Sistemas P*, ya que el primero puede simular al segundo con la simple acción de optar por no agregar ningún plásmido al sistema. En ese caso, el modelo ha conseguido ser equivalente al modelo tradicional de Sistemas P y, de hecho, si no se introduce ninguna regla de manejo de plásmidos, resultaría imposible identificar si realmente se trata de un modelo con plásmidos. Además, dada la naturaleza de este modelo, que está descrito en la sección 3.2.1, se puede afirmar que en el modelo no se pierde ninguna funcionalidad del modelo sin plásmidos, por lo que sería tan solo una extensión del mismo.

Es de suma importancia recordar que el modelo de Sistemas P introducido por George Păun es un modelo que ha demostrado ser Turing-completo. De esta manera, dado que el modelo que da nombre a este trabajo y a esta memoria no es más que una extensión de los Sistemas P, y por lo tanto un sistema con plásmidos es capaz de simular a un modelo de comptación con membranas sin plásmidos, se puede afirmar y queda demostrado que la variante propuesta es universal, o Turing-completa.

Por otra parte, se ha realizado una herramienta informática en forma de aplicación desarrollada en el lenguaje Java capaz de simular ese modelo descrito. La descripción detallada de este programa se encuentra en el capítulo 4: *Implementación del simulador*. Además, esta aplicación queda validada gracias a varios casos de prueba realizados que se pueden encontrar en el capítulo 5: *Casos de prueba*. Esta herramienta desarrollada, u otra similar, puede servir para realizar simulaciones del sistema de computación con membranas con plásmidos, y llevar su uso a distintas áreas de investigación. Como se ha mencionado anteriormente, y como se puede ver en los casos de prueba realizados, el uso de los plásmidos, entendidos como la idea de plásmido introducida en la variante de Sistemas P expuesta, proporciona al sistema mayor flexibilidad y mayores posibilidades, así como un aumento de la eficiencia en algunos casos, dado que ofrece formas de resolver un mismo problema con más herramientas.

Finalmente, el último objetivo de este TFG era el de dar pie a otros trabajos futuros que puedan desarrollarse a partir de este, por lo que ese va a ser el objetivo de la sección que viene a continuación.

6.1 Trabajos futuros

Este trabajo y su memoria pueden dar lugar a un gran abanico de posibles trabajos, que utilicen como base este mismo. Uno de ellos puede ser el desarrollo del simulador en otro lenguaje de programación. En el apartado de *Tecnologías utilizadas* del capítulo 4 de esta memoria, se exponen las razones por las que la aplicación desarrollada para este TFG ha sido escrita en Java, pero esta es solo una de las múltiples opciones que ofrece el mercado de los lenguajes de programación. Realizando un estudio apropiado, se puede encontrar un lenguaje con las propiedades necesarias para mejorar la eficiencia o el funcionamiento de la aplicación. Otros aspectos que podrían llegar a mejorarse de este modo son los relacionados con la *Ingeniería del software*, como es el mantenimiento del código, por ejemplo.

También haciendo referencia al simulador, un posible trabajo futuro es la implementación de una IGU (Interfaz Gráfica de Usario), que permita trabajar con la herramienta desarrollada de una forma mucho más fácil y completa. Este era inicialmente uno de los objetivos de este trabajo, pero finalmente no se pudo llevar a cabo.

Por otro lado, respecto al modelo de *Sistemas P con plásmidos*, puede ser de verdadero interés la integración de éste con otras variantes de los Sistemas P, dando lugar a opciones de modelos muy completas, con mayor cantidad de herramientas disponibles, y permitiendo que esta área de estudio siga creciendo. No obstante, cabe destacar que algo similar se ha hecho en el simulador desarrollado, ya que se puede observar en los tipos de reglas que permite realizar el simulador (ver capítulo 4) que éste engloba reglas de creación y movimiento de membranas, a pesar de que estos tipos de reglas no son pertenecientes al modelo más básico de Sistemas P. De esta misma forma, la unión

de diversas variantes existentes, así como de otras que puedan llegar a crearse, es una magnífica opción a la hora de crear sistemas más flexibles y eficientes.

Por último, tal y como se comenta en varios capítulos de esta memoria, una de las aplicaciones, tanto de los Sistemas P como del modelo con plásmidos, es el uso de éstos en campos de investigación, donde se pueda aprender más sobre el funcionamiento de los organismos celulares o, en este caso, de los plásmidos. Por esta razón, puede llegar a plantearse la creación de simulaciones de distinta clase haciendo uso del modelo de computación con plásmidos, o incluso del simulador aquí creado.

Bibliografía

[1] Cotransporte: Symport y Antyport, en algunas implicaciones médicas Consultado en https://es.wikibooks.org/wiki/Cotransporte:_Symport_y_Antyport,_en_algunas_implicaciones_mÃľdicas.

[2] Definición de plásmido. Consultado en https://www.genome.gov/es/genetics-glossary/Plasmido.

[3] Introducción al ObjectMapper de Jackson, Baeldung. Consultado en https://developer.adobe.com/experience-manager/reference-materials/cloud-service/javadoc/com/fasterxml/jackson/databind/ObjectMapper.html.

[4] P Systems Oficial Web Site. The P Systems Webpage. http://ppage.psystems.eu. 2015.

[5] Artiom Alhazov, Rudolf Freund, and Yurii Rogozhiz. *Computational power of symport/antyport: history, advances, and open problems.* Springer, 2006.

[6] Alberts Bruce, Johnson Alexander, Julian Lewis, Morgan David, Raff Martin, Roberts Keith, and Peter Walter. *Molecular Biology of the Cell, Sixth Edition.* Garland Science, sexta edición, 2014.

[7] Laura Cobano. Diferencia entre Célula Eucariota y Procariota. https://diferenciaentre.es/diferencia-entre-celula-eucariota-y-procariota/

[8] Chen, H., Ionescu, M., Ishdorj, T.O., Păun, A., Păun, Gh., Pérez-Jiménez, M. *Spiking neural P systems with extended rules: universality and languages*. Natural Computing 7, 147–166 (2008)

[9] Marcelo E. Tolmasky, Juan C. Alonso. *Plasmids: Biology and Impact in Biotechnology and Discovery*. John Wiley & Sons, 24 jul 2020 - 512 páginas.

[10] G. Zhang, J. Cheng, T. Wang, X. Wang, J. Zhu. *Membrane Computing: Theory and Applications.* Science Press, Beijing, China, 2015.

[11] Mario de Jesús Pérez Jiménez, Fernando Sancho Caparrini. *Máquinas moleculares basadas en ADN*. Universidad de Sevilla, 2003.

[12] Gheorge Păun. *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.

[13] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc.198 Madison Ave. New York, NYUnited States. Febrero, 2010.

APÉNDICE A

Archivos de entrada y salida de los casos de prueba

A.1 Archivo de entrada para el caso de prueba 1: La resta aritmética. En formato JSON.


```
{
    "membranes": [
        {
            "id" : "ENVIRONMENT",
            "objects": {},
            "plasmidIds": [ "p1", "p2" ],
            "childrenIds": [ "s" ],
            "parentId": null,
            "rules": [
                {
                    "lhs": {
                        "destinationObjects": { "p":1 },
                        "plasmidId": "p1"
                    },
                    "rhsList": [
                        {
                            "result": { "p":1 },
                            "destinationMembraneId": "s",
                            "ruleType": "OUT_SYMPORT"
                        }
                    ]
                },
                {
                    "lhs": {
                        "destinationObjects": { "q":1 },
                        "plasmidId": "p2"
                    },
                    "rhsList": [
                        {
                            "result": { "q":1 },
                            "destinationMembraneId": "s",
                            "ruleType": "OUT_SYMPORT"
                        }
                    ]
                }
            ]
        },
        {
            "id" : "s",
```

```
40             "objects": { "p":1, "q":1 },
41             "plasmidIds": [],
42             "childrenIds": [ "1", "2" ],
43             "parentId": "ENVIRONMENT",
44             "rules": [
45                 {
46                     "lhs": {
47                         "plasmidId": "p1"
48                     },
49                     "rhsList": [
50                         {
51                             "destinationMembraneId": "1",
52                             "ruleType": "OUT_SYMPORT"
53                         }
54                     ]
55                 },
56                 {
57                     "lhs": {
58                         "plasmidId": "p2"
59                     },
60                     "rhsList": [
61                         {
62                             "destinationMembraneId": "2",
63                             "ruleType": "OUT_SYMPORT"
64                         }
65                     ]
66                 },
67                 {
68                     "lhs": {
69                         "objects": { "a":1 }
70                     },
71                     "rhsList": [
72                         {
73                             "destinationResult": { "a":1 },
74                             "destinationMembraneId": "2",
75                             "ruleType": "IN_COMMUNICATION"
76                         }
77                     ]
78                 }
79             ]
80         },
81         {
82             "id" : "1",
83             "objects": { "a":4 },
84             "plasmidIds": [],
85             "childrenIds": [],
86             "parentId": "s",
87             "rules": []
88         },
89         {
90             "id" : "2",
91             "objects": { "b":8 },
92             "plasmidIds": [],
93             "childrenIds": [],
94             "parentId": "s",
95             "rules": []
96         }
97     ],
98     "plasmids": [
99         {
100            "id": "p1",
101            "rules": [
102                {
103                    "lhs": { "objects": { "a":1 } },
```

```
                    "rhsList": [
                        {
                            "result": { "a":1 },
                            "ruleType": "OUT_COMMUNICATION"
                        }
                    ]
                }
            ]
        },
        {
            "id": "p2",
            "rules": [
                {
                    "lhs": { "objects": { "a":1, "b":1 } },
                    "rhsList": [
                        {
                            "result": { "c":1 },
                            "ruleType": "OUT_COMMUNICATION"
                        }
                    ]
                }
            ]
        }
    ],
    "rootId": "ENVIRONMENT",
    "outputId": "2"
}
```


A.2 Archivo de salida para el caso de prueba 1: La resta aritmética. En formato JSON.


```
{
    "membranes": [
        {
            "id": "1",
            "plasmidIds": ["p1"],
            "parentId": "s",
            "rules": []
        },
        {
            "id": "2",
            "objects": {
                "b": 4
            },
            "plasmidIds": ["p2"],
            "parentId": "s",
            "rules": []
        },
        {
            "id": "s",
            "objects": {
                "c": 4
            },
            "childrenIds": ["1", "2"],
            "parentId": "ENVIRONMENT",
            "rules": [
                {
                    "lhs": {
                        "objects": {
                            "a": 1
```

```
30                          }
31                      },
32                      "rhsList": [
33                          {
34                              "destinationResult": {
35                                  "a": 1
36                              },
37                              "destinationMembraneId": "2",
38                              "ruleType": "IN_COMMUNICATION",
39                              "objectRHS": true
40                          }
41                      ]
42                  },
43                  {
44                      "lhs": {
45                          "plasmidId": "p1"
46                      },
47                      "rhsList": [
48                          {
49                              "destinationMembraneId": "1",
50                              "ruleType": "OUT_SYMPORT",
51                              "plasmidRHS": true
52                          }
53                      ]
54                  },
55                  {
56                      "lhs": {
57                          "plasmidId": "p2"
58                      },
59                      "rhsList": [
60                          {
61                              "destinationMembraneId": "2",
62                              "ruleType": "OUT_SYMPORT",
63                              "plasmidRHS": true
64                          }
65                      ]
66                  }
67              ]
68          },
69          {
70              "id": "ENVIRONMENT",
71              "objects": {
72                  "p": 1,
73                  "q": 1
74              },
75              "childrenIds": ["s"],
76              "rules": [
77                  {
78                      "lhs": {
79                          "plasmidId": "p2",
80                          "destinationObjects": {
81                              "q": 1
82                          }
83                      },
84                      "rhsList": [
85                          {
86                              "result": {
87                                  "q": 1
88                              },
89                              "destinationMembraneId": "s",
90                              "ruleType": "OUT_SYMPORT",
91                              "plasmidRHS": true
92                          }
93                      ]
```

```
94                  },
95                  {
96                      "lhs": {
97                          "plasmidId": "p1",
98                          "destinationObjects": {
99                              "p": 1
100                         }
101                     },
102                     "rhsList": [
103                         {
104                             "result": {
105                                 "p": 1
106                             },
107                             "destinationMembraneId": "s",
108                             "ruleType": "OUT_SYMPORT",
109                             "plasmidRHS": true
110                         }
111                     ]
112                 }
113             ]
114         }
115     ],
116     "plasmids": [
117         {
118             "id": "p1",
119             "rules": [
120                 {
121                     "lhs": {
122                         "objects": {
123                             "a": 1
124                         }
125                     },
126                     "rhsList": [
127                         {
128                             "result": {
129                                 "a": 1
130                             },
131                             "ruleType": "OUT_COMMUNICATION",
132                             "objectRHS": true
133                         }
134                     ]
135                 }
136             ]
137         },
138         {
139             "id": "p2",
140             "rules": [
141                 {
142                     "lhs": {
143                         "objects": {
144                             "a": 1,
145                             "b": 1
146                         }
147                     },
148                     "rhsList": [
149                         {
150                             "result": {
151                                 "c": 1
152                             },
153                             "ruleType": "OUT_COMMUNICATION",
154                             "objectRHS": true
155                         }
156                     ]
157                 }
```

```
                ]
            }
    ],
    "rootId": "ENVIRONMENT",
    "outputId": "2"
}
```

A.3 Archivo de entrada para el caso de prueba 2: Producto de números enteros. En formato JSON.

```
{
    "membranes": [
        {
            "id" : "ENVIRONMENT",
            "objects": {},
            "plasmidIds": [ "p1", "p2" ],
            "childrenIds": [ "s" ],
            "parentId": null,
            "rules": [
                {
                    "lhs": {
                        "destinationObjects": { "p":1 },
                        "plasmidId": "p1"
                    },
                    "rhsList": [
                        {
                            "destinationResult": { "p′":1 },
                            "destinationMembraneId": "s",
                            "ruleType": "OUT_SYMPORT"
                        }
                    ]
                },
                {
                    "lhs": {
                        "destinationObjects": { "p′":1 },
                        "plasmidId": "p2"
                    },
                    "rhsList": [
                        {
                            "destinationResult": { "q":1 },
                            "destinationMembraneId": "s",
                            "ruleType": "OUT_SYMPORT"
                        }
                    ]
                }
            ]
        },
        {
            "id" : "s",
            "objects": { "p":1 },
            "plasmidIds": [],
            "childrenIds": [ "1", "2" ],
            "parentId": "ENVIRONMENT",
            "rules": [
                {
                    "lhs": {
                        "objects": { "q":1 },
                        "destinationObjects": { "a":1 },
                        "plasmidId": "p1"
                    },
```

```
51                     "rhsList": [
52                         {
53                             "result": { "r":1 },
54                             "destinationResult": { "a":1 },
55                             "destinationMembraneId": "1",
56                             "ruleType": "OUT_SYMPORT"
57                         }
58                     ]
59                 },
60                 {
61                     "lhs": {
62                         "objects": { "s":1 },
63                         "plasmidId": "p2"
64                     },
65                     "rhsList": [
66                         {
67                             "result": { "t":1 },
68                             "destinationMembraneId": "2",
69                             "ruleType": "OUT_SYMPORT"
70                         }
71                     ]
72                 },
73                 {
74                     "lhs": {
75                         "objects": { "r":1 },
76                         "destinationPlasmidId": "p1"
77                     },
78                     "rhsList": [
79                         {
80                             "result": { "s":1 },
81                             "destinationMembraneId": "1",
82                             "ruleType": "IN_SYMPORT"
83                         }
84                     ]
85                 },
86                 {
87                     "lhs": {
88                         "objects": { "t":1 },
89                         "destinationPlasmidId": "p2"
90                     },
91                     "rhsList": [
92                         {
93                             "result": { "q":1 },
94                             "destinationMembraneId": "2",
95                             "ruleType": "IN_SYMPORT"
96                         }
97                     ]
98                 }
99             ]
100        },
101        {
102            "id" : "1",
103            "objects": { "b":1, "a":4 },
104            "plasmidIds": [],
105            "childrenIds": [],
106            "parentId": "s",
107            "rules": []
108        },
109        {
110            "id" : "2",
111            "objects": { "b":1, "a":5 },
112            "plasmidIds": [],
113            "childrenIds": [],
114            "parentId": "s",
```

```
115             "rules": []
116         }
117     ],
118     "plasmids": [
119         {
120             "id": "p1",
121             "rules": [
122                 {
123                     "lhs": { "objects": { "a":1, "b":1 } },
124                     "rhsList": [
125                         {
126                             "result": { "b":1 },
127                             "ruleType": "EVOLUTION"
128                         }
129                     ]
130                 }
131             ]
132         },
133         {
134             "id": "p2",
135             "rules": [
136                 {
137                     "lhs": { "objects": { "a":1 } },
138                     "rhsList": [
139                         {
140                             "result": { "a":1 },
141                             "ruleType": "EVOLUTION"
142                         },
143                         {
144                             "result": { "b":1 },
145                             "ruleType": "OUT_COMMUNICATION"
146                         }
147                     ]
148                 }
149             ]
150         }
151     ],
152     "rootId": "ENVIRONMENT",
153     "outputId": "s"
154 }
```

A.4 Archivo de salida para el caso de prueba 2: Producto de números enteros. En formato JSON.

```
1  {
2      "membranes": [
3          {
4              "id": "1",
5              "objects": {
6                  "b": 1
7              },
8              "parentId": "s"
9          },
10         {
11             "id": "2",
12             "objects": {
13                 "a": 5,
14                 "b": 1
15             },
16             "parentId": "s"
```

```
17          },
18          {
19              "id": "s",
20              "objects": {
21                  "q": 1,
22                  "b": 20
23              },
24              "plasmidIds": ["p1", "p2"],
25              "childrenIds": ["1", "2"],
26              "parentId": "ENVIRONMENT",
27              "rules": [
28                  {
29                      "lhs": {
30                          "objects": {
31                              "t": 1
32                          },
33                          "destinationPlasmidId": "p2"
34                      },
35                      "rhsList": [
36                          {
37                              "result": {
38                                  "q": 1
39                              },
40                              "destinationMembraneId": "2",
41                              "ruleType": "IN_SYMPORT",
42                              "plasmidRHS": true
43                          }
44                      ]
45                  },
46                  {
47                      "lhs": {
48                          "objects": {
49                              "s": 1
50                          },
51                          "plasmidId": "p2"
52                      },
53                      "rhsList": [
54                          {
55                              "result": {
56                                  "t": 1
57                              },
58                              "destinationMembraneId": "2",
59                              "ruleType": "OUT_SYMPORT",
60                              "plasmidRHS": true
61                          }
62                      ]
63                  },
64                  {
65                      "lhs": {
66                          "objects": {
67                              "q": 1
68                          },
69                          "plasmidId": "p1",
70                          "destinationObjects": {
71                              "a": 1
72                          }
73                      },
74                      "rhsList": [
75                          {
76                              "result": {
77                                  "r": 1
78                              },
79                              "destinationResult": {
80                                  "a": 1
```

```
81                              },
82                              "destinationMembraneId": "1",
83                              "ruleType": "OUT_SYMPORT",
84                              "plasmidRHS": true
85                          }
86                      ]
87                  },
88                  {
89                      "lhs": {
90                          "objects": {
91                              "r": 1
92                          },
93                          "destinationPlasmidId": "p1"
94                      },
95                      "rhsList": [
96                          {
97                              "result": {
98                                  "s": 1
99                              },
100                             "destinationMembraneId": "1",
101                             "ruleType": "IN_SYMPORT",
102                             "plasmidRHS": true
103                         }
104                     ]
105                 }
106             ]
107         },
108         {
109             "id": "ENVIRONMENT",
110             "childrenIds": ["s"],
111             "rules": [
112                 {
113                     "lhs": {
114                         "plasmidId": "p2",
115                         "destinationObjects": {
116                             "p'": 1
117                         }
118                     },
119                     "rhsList": [
120                         {
121                             "destinationResult": {
122                                 "q": 1
123                             },
124                             "destinationMembraneId": "s",
125                             "ruleType": "OUT_SYMPORT",
126                             "plasmidRHS": true
127                         }
128                     ]
129                 },
130                 {
131                     "lhs": {
132                         "plasmidId": "p1",
133                         "destinationObjects": {
134                             "p": 1
135                         }
136                     },
137                     "rhsList": [
138                         {
139                             "destinationResult": {
140                                 "p'": 1
141                             },
142                             "destinationMembraneId": "s",
143                             "ruleType": "OUT_SYMPORT",
144                             "plasmidRHS": true
```

```
145                         }
146                     ]
147                 }
148             ]
149         }
150     ],
151     "plasmids": [
152         {
153             "id": "p1",
154             "rules": [
155                 {
156                     "lhs": {
157                         "objects": {
158                             "a": 1,
159                             "b": 1
160                         }
161                     },
162                     "rhsList": [
163                         {
164                             "result": {
165                                 "b": 1
166                             },
167                             "ruleType": "EVOLUTION",
168                             "objectRHS": true
169                         }
170                     ]
171                 }
172             ]
173         },
174         {
175             "id": "p2",
176             "rules": [
177                 {
178                     "lhs": {
179                         "objects": {
180                             "a": 1
181                         }
182                     },
183                     "rhsList": [
184                         {
185                             "result": {
186                                 "a": 1
187                             },
188                             "ruleType": "EVOLUTION",
189                             "objectRHS": true
190                         },
191                         {
192                             "result": {
193                                 "b": 1
194                             },
195                             "ruleType": "OUT_COMMUNICATION",
196                             "objectRHS": true
197                         }
198                     ]
199                 }
200             ]
201         }
202     ],
203     "rootId": "ENVIRONMENT",
204     "outputId": "s"
205 }
```

APÉNDICE B

Objetivos de Desarrollo Sostenible (ODS)

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**			X	
ODS 7. **Energía asequible y no contaminante.**	X			
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**			X	
ODS 11. **Ciudades y comunidades sostenibles.**			X	
ODS 12. **Producción y consumo responsables.**			X	
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**			X	
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En este momento se da por hecho que el lector de este anexo tiene el conocimiento suficiente respecto al TFG, de los Sistemas P con plásmidos y de sus posibles aplicaciones, comentadas en la memoria. Se recuerda que en este trabajo se está desarrollando un modelo de computación basado en la biología, y aporta ventajas y herramientas a campos de investigación como la genética, la biotecnología y otras ciencias. Por esta razón, los ODS más relacionados con el trabajo son:

1. **Salud y bienestar**. Como se comenta en varios puntos de la memoria, el modelo de “Sistemas P con plásmidos” consiste en un modelo bioinspirado, variante al modelo de computación con membranas conocido por “Sistemas P”. Este último es un modelo que ya está siendo usado actualmente por varios campos de las ciencias de la salud. Aplicaciones basadas en este modelo, como es la de este TFG, permiten realizar simulaciones de procesos biológicos o poblacionales, entre otros. Un posible ejemplo puede ser el de una simulación de la evolución y propagación de una pandemia en una población, siendo este un trabajo realizado anteriormente por miembros de la UPV. Este tipo de simuladores permite estudiar, entre otras cosas, aspectos importantes en la medicina, como es el ejemplo de un contagio, u otros procesos biológicos capaces de ser definidos en un modelo de Sistemas P. Así mismo, la variante con plásmidos que se introduce en este trabajo no es más que una herramienta extendida de los Sistemas P tradicionales, por lo que sus aplicaciones también son similares. Además, este modelo aporta la capacidad de estudiar y comprender mejor el funcionamiento de los plásmidos en la biología molecular.

2. **Educación de calidad**. Como se ha comentado al final del punto anterior, el modelo presentado aporta herramientas que pueden ser usadas en distintos campos de investigación, considerando estos como otro tipo más de educación.

3. **Energía asequible y no contaminante**. Una de las características de los modelos de computación con membranas es la del particular uso que puede aportar a campos como la genética y la biotecnología. Si se parte del hecho de que estos campos nos permitan modificar o “programar” cadenas de ADN de una célula, o de un plásmido, estos modelos computacionales son una herramienta que permite utilizar elementos vivos reales como células o tejidos para que se comporten de la manera con la que se le ha programado, convirtiéndose en una máquina de cómputo, con procesamiento paralelo y distribuido de la información, y con un coste energético considerablemente inferior al de las máquinas computacionales actuales.

4. **Industria, innovación e infraestructuras**. Reuniendo lo que se ha comentado en todos los puntos anteriores, realmente no sería necesario añadir nada más. Sin embargo, se puede destacar el hecho de que las tecnologías de la información se ven actualmente limitadas por condiciones físicas, como puede ser el tamaño mínimo que un circuito electrónico debe tener, o la obtención de materia prima necesaria para la fabricación de los dispositivos. Los modelos de computación natural como el que se propone en este TFG son un punto de partida para explotar otro tipo de materias, otro tipo de máquinas y, desde luego, otro tipo de formas de cómputo.