2 P Systems Implementation on P-Lingua Framework

2.1 Introduction

As previously mentioned, from the early days of the discipline, different approaches have been followed to provide software tools assisting the P systems designers in their design and verification tasks for a number of membrane system types and variants. However, at the beginning, the most common case was the development of specific-purpose tools devoted to the solution of a particular model based on a P system or P systems family. While these early initiatives constituted relevant achievements for membrane computing, their usefulness for the general community was significant mostly in the context of the specific paper or scientific result they were developed around. Surveys on the first generation of software tools related to membrane computing can be found in [5, 49].

In order to move a step forward in this sense, aiming to provide some solution for the P systems community in the form of a set of general tools for the software implementation of P systems, P-Lingua framework emerged more than a decade ago. As a first crucial element in the framework, a specification language, the so-called P-Lingua language, was defined, aiming to be a standard for the community to speak the same language when defining P systems. One of the advantages of having such a standard is to avoid ambiguities, and moreover to foster collaboration, facilitating that researchers share their designs, even if they use different simulation software—similarly as the Systems Biology Markup Language (SBML) format works for the systems biology community.

The language started with some very general elements common to most P system types, such as the membrane structure, objects, membrane labels, or rewriting rules. Along with such general elements, each P system type or variant would admit specific rules, and the framework would provide parsing tools to detect syntactic or semantic errors. Along with the specification language, P-Lingua framework provided from the beginning a number of built-in simulators, capturing the semantic and dynamic aspects of each P system type. Such simulators were included for the

G. Zhang et al., *Membrane Computing Models: Implementations*,
https://doi.org/10.1007/978-981-16-1566-5_2

sake of completeness of the tool, but they were not intended to compete against existing software. Actually, the framework included the functionality to compile P-Lingua code into something else so that one could provide such compiled result as an input for an external simulator. Software implementation of P systems is further explored in Chap. 3.

The chapter elaborates on some of the main capabilities of the framework and is structured as follows. The main elements involved in P-Lingua language will be introduced in Sect. 2.2, along with a classification of the main types and variants of P systems supported by the framework, including the main references related with them. Then, several simulation algorithms will be presented in Sect. 2.3, capturing the dynamics of some especially relevant types of P systems used in the solution of real-life problems. Finally, in Sect. 2.4, a higher-level tool will be presented, MeCoSim, as a step forward to provide a visual virtual research environment.

2.2 P-Lingua Language

P-Lingua is a domain-specific language started in 2008 [4] that has been continuously evolving since then (technical details on the foundations can be found in [36] and a survey together with some recent developments in [40]). The approach is to keep the definitions as simple as possible, being a sort of "LaTeX-like" pseudocode, in such a way that P systems designers can use similar notation to the one used in the literature. A P system can thus be defined in a (plain text) `.pli` file, where the designer indicates the *model*, *structure*, *initial multisets*, *variables* (if any), etc. The elements of the definition will be further explained in what follows.

2.2.1 P System Models

When designing a membrane system in P-Lingua, the instruction `@model` must be present at the beginning of the `.pli` file, followed by a keyword identifying the model used. Through the development of P-Lingua, several classes of P systems have been included within the framework, while others have been discarded due to the lack of their use. The latest stable version, pLinguaCore 4.0, was released in 2013, covering only 10 model types. The P-Lingua framework has been continuously expanding since then although the development efforts have been focusing in the core distributed within MeCoSim. The following tables illustrate the diversity of variants considered in the current version, with the corresponding keywords and a reference introducing the model. For more details about the exact pLinguaCore release where some models were included or discarded, we refer the reader to [39].

In the case of neural-like P systems, the model keyword `spiking_psystems` is slightly overloaded since it covers multiple subclasses. Each time the model has been extended, special symbols and tokens were used so that the parser and the simulator are capable to identify which type of rules are being used and how they should be interpreted.

2.2.2 Membrane Structure

The topology of the membrane system to be simulated will depend on the model selected in the file. If an invalid structure with respect to the model is defined, the *parser* will show a message notifying it. The instruction `@mu`[1] is used to define the architecture of the system. The syntax is similar to the one used in the literature. For example, for cell-like membrane systems, the definition

$$\texttt{@mu} = [[[]'\texttt{4}]'\texttt{2}[]'\texttt{3}]'\texttt{1}$$

would lead to a P system with a skin membrane labelled by 1 and 2 internal membranes: an elementary membrane labelled by 3, and a membrane labelled by 2 which contains an elementary membrane labelled by 4 inside.

In the case of tissue P systems, membranes (called *cells*) are not hierarchically arranged, but they can be connected by means of an arbitrary graph, which is not required to be explicitly given in the definition. Typically, the set of directed arcs connecting cells can be reconstructed from the implicit information provided by the set of rules. However, in P-Lingua format, it is necessary to indicate the initial cells in the system, formally considering them as elementary membranes located within an external compartment labelled by 0, that will act as the environment of the system.

Spiking neural P systems need both the initial neurons and the *synapses* defined in order to work. The former is defined as previously with the `@mu` instruction and the later with `@marcs` indicating with pairs of labels which arcs will be present in the underlying graph of the SNP system.

Note that not all the definitions of `@mu` must be in the same line, but instead of the = symbol, it is possible to add more compartments to a specific region. For instance, in cell-like membrane systems, it is possible to use

$$\texttt{@mu} = [[]'\texttt{2}]'\texttt{1};\ \ \texttt{@mu(1)}+ = []'\texttt{3};$$

in order to generate a structure identical to the one defined above. Note that a semicolon indicates the end of an instruction.

[1] The usual notation for the structure of P systems is the Greek letter μ.

2.2.3 Initial Multisets

In order to describe the initial multisets of the different compartments, the command `@ms` is used in a similar way to the literature, with braces {} as the delimiters of the multisets. Like before, the $+=$ symbols can be used to add new objects to a predefined multiset. The multiplicity of a symbol is indicated by the `*` symbol as in a multiplication (e.g., `c*5` indicates 5 copies of object `c`).

2.2.4 P System Rules

Rules, like the structure of the P system, depend on the model of membrane system being simulated. The P-Lingua parser was defined in such a way that P systems researchers can use a very close language to the one used in literature, putting special emphasis in the definition of rules. Therefore, brackets [] are used in P-Lingua files as in the definition of rules in research papers. An evolution rule is defined in the following way: `+[ a1 - -> b,c ]'1`. Note the differences between the P-Lingua and the formal definitions: The subscripts are between braces, the arrow is replaced by an ASCII version of an arrow, the label is preceded by a `'` symbol instead of being a subscript, and the polarization precedes the rule instead of being a superscript. For tissue P systems, instead of using parentheses, a similar brackets notation is used with a double arrow as follows: `[a]'1 <- -> [b]'2` to denote the rule $(1, a/b, 2)$.

Usually, several rules with the same structure but with different subindexes are defined in P systems, and it can be translated into a P-Lingua file with the colon`:` symbol, followed by the corresponding limits. Let $r \equiv [a_i \rightarrow a_{i+1}]_1$ for $0 \leq i \leq n$ be a set of rules of a P system with active membranes that can be defined in a P-Lingua file as follows: `[ a{i} - -> a{i+1} ]'1 : 0 <= i <= n`. If two or more variables have to be defined, they will be declared from the right to the left; that is, if a variable j is limited by i, then the range of i must be written "before" (to the right), for example, `[ a{i,j} - -> a{i,j+1}]'1 : 0 <= j < i, 0 <= i <= n.`

The user must define all these parameters (except the model type) in a `main` function. A `function` in the P-Lingua language is defined with the keyword `def` followed by the name of the function. A function can have parameters whose names will be indicated between parentheses and separated by commas. More than one function can be defined in a single P-Lingua file, and they are widely used, for instance, to construct the membrane system in a modular way. An example of this would be the following code:

```
@model<membrane_division>

n = 3 /* A parameter n is defined to be used later */
m = 1000 /* A parameter m is defined to be used later */
```

```
def main() {
  define_structure();
  define_initial_multisets(m);
  define_rules(n);
}
def define_structure() {
  @mu = []'1;
}
define_initial_multisets(number_objects) {
  @ms(1) += a{0}*{number_objects};
}
define_rules(number_steps) {
  [a{i} - -> a{i+1}]'1 : 0 <= i < {number_steps};
}
```

Note that it is allowed to insert comments in P-Lingua files, surrounded by the symbols /* and */. As indicated above, several examples of the different types of P systems implemented in P-Lingua can be found in the websites of the P-Lingua project [43] and MeCoSim [24].

2.3 Simulation Algorithms

P systems are bioinspired devices that work in a massively parallel and nondeterministic way. While there are preliminary studies analyzing the problems related to implementations in biological means, there is still a long way to reach this ultimate goal. That is why developing hardware/software implementations of P systems becomes a vital necessity for the advancement of scientific activities in membrane computing.

The P-Lingua framework includes a Java library called *pLinguaCore* that provides at least one simulation algorithm for each P system variant. A simulation algorithm for membrane computing can be described as an algorithm which is able to reproduce P system computations on conventional software/hardware architectures. Usually, only one branch of computation is considered, and it is expected to display the sequence of configurations, including information on the executed rules for each step of computation. Concerning the hardware used, the simulation algorithms can be designed to run on sequential machines (single-thread CPU) or parallel architectures (multi-thread CPU, GPU, FPGA, etc.). The simulation algorithms in pLinguaCore are designed for single-thread CPU, but it is possible to parse a P-Lingua file and compile it into an appropriate input for an external simulator.

All simulation algorithms in pLinguaCore share the same underlying implementation of a computation step as a loop divided into two stages: selection stage and execution stage. The selection stage consists in searching for applicable rules and selecting which ones will actually be executed in each membrane of a given configuration, taking into account the restrictions dictated by the system semantics.

Then, the execution stage actually implements the changes on the configuration caused by the execution of the selected rules, and this completes the simulation of the computation step. The input data for the selection stage contains the description of the membranes with their multisets (strings over the working alphabet of objects, labels associated with the membrane, etc.) and the set of defined rules. The output data of this stage are the multisets of selected rules. Only the execution stage changes the information of the configuration. It is the reason why execution stage needs synchronization when accessing to the membrane structure and the multisets. At the end of the execution stage, the simulation process restarts the selection stage in an iterative way until a halting configuration is reached (i.e., none of the rules is applicable). Alternatively, a maximum number of iterations can be set at the beginning of the simulation to avoid getting stuck on too long (or even infinite) computations.

With the general design explained above, the pLinguaCore library includes simulation algorithms for the cell-like, tissue-like, and neural-like P systems enumerated in Sect. 2.2. For more information, see the corresponding references in Tables 2.1, 2.2, and 2.3, respectively. There exist in the literature other P system variants whose computations are not synchronized by a global clock in a step-by-step fashion (e.g., asynchronous, time-free, or stochastic models). Such variants are not currently supported under the P-Lingua framework, but there exist fully functional alternative implementations available (see Chaps. 4 and 5).

Other variants are also contemplated [2, 8, 10, 11]. A special mention should be given to the simulation algorithms for population dynamics P systems (PDP systems) which is a variant widely used for simulation of ecosystem dynamics (see Chap. 6) in which each rule has a probability associated. The first description of probabilistic semantics was quite ambiguous: "Rules should be applied in a maximally parallel way, according to their probabilities." There are many ways of interpreting this sentence, and each one could lead to different behaviors. While all of them might be "correct" from a formal point of view, not all simulation algorithms are acceptable when the goal is to reproduce the behavior of a complex system. Since P-Lingua is a general-purpose framework, indicating which is the appropriate choice should be a decision of the model designer (Table 2.4).

AQ1

Three simulation algorithms have been designed for PDP systems and implemented in pLinguaCore [19]:

- DNDP algorithm [21].
- BBB algorithm [19].
- DCBA algorithm [22].

In the algorithm DNDP, the rules are selected individually according to its probabilities. On the other hand, algorithms BBB and DCBA work by grouping rules in blocks by analyzing the left-hand side, each block has the same left-hand side, and all the rule probabilities must sum 1. DCBA uses a refined definition of block in which the charges of the right-hand side must be consistent. More about simulation algorithms for PDP systems will be explained in Chap. 6 since this kind

Table 2.1 Cell-like membrane systems implemented in P-Lingua

Variant of membrane systems	Model specification keyword	Ref.	t5.1
P systems with active membranes and membrane creation	`membrane_creation`	[25]	t5.2
P systems with active membranes and membrane division	`membrane_division` / `dam`	[33]	t5.3
P systems with symport/antiport rules	`symport_antiport` / `infEnv_symport_antiport`	[18]	t5.4
Polarizationless P systems with active membranes with minimal cooperation and membrane division	`dam_wp`	[48]	t5.5
Polarizationless P systems with active membranes with minimal cooperation, membrane division, and without dissolution	`dam_wp_wd`	[48]	t5.6
Polarizationless P systems with active membranes with minimal cooperation and membrane division only for elementary membranes and without dissolution	`dam_wp_wd_wn`	[48]	t5.7
P systems with active membranes with minimal cooperation and membrane separation	`sam`	[47]	t5.8
Polarizationless P systems with active membranes with minimal cooperation and membrane separation	`sam_wp`	[47]	t5.9
Polarizationless P systems with active membranes with minimal cooperation and membrane separation and without dissolution	`sam_wp_wd`	[47]	t5.10
Polarizationless P systems with active membranes with minimal cooperation and membrane separation only for elementary membranes and without dissolution	`sam_wp_wd_wn`	[47]	t5.11
Transition P systems	`transition` / `rewriting`	[32]	t5.12

Table 2.2 Tissue-like membrane systems implemented in P-Lingua

Variant of membrane systems	Model specification keyword	Ref.	t8.1
Tissue P systems with cell division	`tissue_psystems` / `tpdc`	[20, 34]	t8.2
Tissue P systems with cell division and antiport rules	`tpda`	[34]	t8.3
Tissue P systems with cell division and symport rules	`tpds`	[34]	t8.4
Tissue P systems with cell separation	`TSCS`	[27, 38]	t8.5
Tissue P systems with evolutional communication rules with cell division	`evolution_communication` / `ev_symport_antiport`	[31]	t8.6
Tissue P systems with evolutional communication rules with cell separation	`tsec`	[31]	t8.7
Tissue P systems with promoters	`tpdc` / `tpda` / `tpds`	[45]	t8.8

Table 2.3 Neural-like membrane systems implemented in P-Lingua

Variant of membrane systems	Model specification keyword	Ref.
Asynchronous SN P systems	`spiking_psystems`	[3]
Asynchronous SN P systems with local synchronization	`spiking_psystems`	[41]
Cell-like spiking neural P systems	`cell_like_snp`	[46, 50]
Dendrite P systems	`dendrite`	[26]
Fuzzy reasoning spiking neural P systems	`fuzzy_psystems`	[13, 17]
Limited asynchronous SN P systems	`spiking_psystems`	[29]
Spiking neural P systems	`spiking_psystems`	[13, 15, 16]
Spiking neural P systems with anti-spikes	`spiking_psystems`	[28]
Spiking neural P systems with hybrid astrocytes	`spiking_psystems`	[30]
Spiking neural P systems with structural plasticity	`spiking_psystems`	[1]

Table 2.4 Other variants of membrane systems implemented in P-Lingua

Variant of membrane systems	Model specification keyword	Ref.
Enzymatic numerical P systems	`enps`	[35]
Population dynamics P systems	`probabilistic`	[2]
Probabilistic guarded P systems	`probabilistic_guarded_ _psystems`	[8, 10]
Regenerative P systems	`regenerative_psystems`	[11]
Simple kernel P systems	`simple_kernel_psystems`	[12, 14]
Simple regenerative P systems	`simple_regenerative_ _psystems`	[11]
Stochastic P systems* *(discontinued)	`stochastic`	[42]

of algorithms requires a large amount of computational power being suitable for high-performance computing platforms such as CUDA.

As it was mentioned before, there are various approaches in the literature where the standard semantics of P systems (namely, nondeterministic behavior and maximally parallel application of the rules) is modified by adding different regulation elements, which need to be carefully described in order to explain how the system evolves. In particular, it is worth highlighting that the concept of "simulation algorithm" is used in this section in a theoretical sense, that is, a formalization that translates the specification of the semantics into a pseudocode capturing precisely the routine that the system follows when deciding what rules to apply. It should not be confused with an implementation of such algorithm in a programming language. Some attempts trying to bring semantic elements explicitly into the description of a P system in P-Lingua language have been already initiated, and it is being considered for upcoming release of P-Lingua 5.0.

2.4 Membrane Computing Simulator (MeCoSim)

The previous sections have presented the essential elements of P-Lingua framework: the standard specification language and the simulation engines to run the computations of the given P systems. These elements constitute the core features, the chassis of our car. However, in order for this vehicle to move smoothly, several additional pieces (as the external body but also others as the steering wheel, the pedals, or the dashboard icons, among others) are needed to provide the users with the desired driving experience, in order for them to sit down and enjoy while conducting their virtual experiments with models based on P systems.

With the metaphor introduced, we aim to present the main idea behind MeCoSim (membrane computing simulator) [24, 37], conceived in a search for the generalization of certain high-level visual applications to manage population dynamics models, known as EcoSim product family [36, 44]. Built on top of P-Lingua core, this visual environment provides a higher level of abstraction, transforming the solid set of tools of our internal chassis and engine given by P-Lingua core into a whole car, complementing the previous elements with an external layer allowing the drivers conduct their experiments through the proper sensors and actuators.

Thus, MeCoSim was devised with a manifold purpose, assisting in the design of the heart of the cars (P system-based models), delivering the final cars (custom apps based on the models), and helping users drive their vehicles (through the tools coming with the apps). Firstly, the visual interface provides the expert users (P system designers) with an interface where they can specify, debug, and run (*step-by-step* or entire) computations of their P systems in a smoother way; this is made easier with the tools provided by a friendly environment, aiding in the task of designing and verifying the core part of our cars: the P systems modelling certain case studies. Secondly, the environment provides certain tools to make the technical pieces constituting the model become a final product, *that is*, bridging the gaps to convert the engine, car axles, controls, or wheels into the final car. To this purpose, MeCoSim provides some tools to define, through configuration files, a final visual application using the core elements of the framework, plus the P system (or P system family) specified, and the inputs and outputs to control and monitor, respectively, each trip made with the car (i.e., each computation of the system, each virtual experiment conducted). Finally, end users receive their car: the customized application satisfying their needs. Probably, they will have no idea about the internal specifications of the car, they are not car mechanics/technicians, but they will be able to drive their specific car. In such car, the custom app, they will be able to sit, introduce the details about each particular trip (experiment) they want to make (run), decide about the speed, and control the steering wheel and pedals, enjoying the drive and finally getting to their destiny (the end of the computation) while obtaining all the desired additional information through the monitoring system provided by the dashboard.

In this context, everything starts with the identification of a certain need, such as solving a certain NP-complete problem or modelling a real-life system in economy, ecology, medicine, or any other field. In this context, a P system (or P system family) must be defined to satisfy such need. Thus, the definition of the system requires the translation of the P system into a file using P-Lingua standard specification language. Then, an iterative process of design, debug, and verification starts, progressing with the problem until the model has been properly validated according to the experts in the problem domain.

Then, the central problem has been solved, but only the technicians could use the tools to run computations of the system. Then, a new effort can be made by such P system designers to set in a spreadsheet configuration file the specific elements of the final car and the application where the end users (ecologists, economists, etc.) will be able to conduct their visual experiments. The custom elements will include the hierarchical structure arranging all the visual blocks of the final app, the tables allowing the introduction of specific input data by the end user, and the outputs to monitor the activity and the final results of the trip. Now, everything is ready for the end user to drive, to analyze each particular scenario of interest (each trip), introducing in the tables the specific parameter values and input data for each scenario of interest and run each virtual experiments, getting the desired results in their dashboard given by the custom output tables and charts defined in the configuration file set by the P system designers during the building of the car (the custom app).

The description above has been probably illustrative at a general level, but people not familiarized with MeCoSim might find it difficult to figure out how this approach look like at a deeper level. The following subsections will try to clarify those aspects only outlined before, detailing the main goals achieved (Sect. 2.4.1) and the software components involved (Sect. 2.4.2). In Ref. [44], a methodology is proposed based for the solution of a problem through membrane-based systems making use of P-Lingua framework and MeCoSim, where the corresponding tools described are employed in a systematic way.

2.4.1 Primary goals

As it has just been depicted, MeCoSim's main intent is the provision of a high-level visual interface to handle P system-based models. This is right, but what should we exactly expect from this environment? Let us try to clarify this by analyzing the origin and initial view of the tools involved.

To start with this overview, it is worth recalling that we are studying a paradigm, membrane computing, where many computing models have been defined along the years. As specifically addressed by this book, the implementation of these computational devices is crucial in order to take advantage of all the theoretical properties, the strengths, of such machines. However, our biologically inspired models present certain features that are not easy to implement in certain biological or artificial substrate, and even if it can be done, it implies major efforts to apply these machines to each particular problem. Nevertheless, there is a faster convenient approach that can be applied in order to make this process more manageable in

(a) the study of theoretical aspects of the different types of devices (such as their computational power, efficiency, etc.) or the practical use of models based on these devices (its tasks such as the design, verification, or validation of properties and virtual experimentation) and (b) the simulation of the computations for the given models.

This volume is devoted to implementations of membrane computing models. Consequently, we actually expect real devices that can capture all the features of the theoretical machines. Some of the solutions provided by later chapters will succeed, addressing aspects such as capturing the inherent parallelism of such ideal machines. However, those real machines will need to take many things into account at a very technical level, it will be a very tough process, and this will make it very challenging to validate the proper functioning of these devices according to all the properties of the types of P systems used, along with all the properties of the models built for a particular problem, based on such types of P systems.

In order to avoid attacking all the problems at once, a different approach can be followed: first facing the “soft” implementation of the theoretical devices, the intended type of P systems, through sequential software simulators, not addressing all the technical aspects required by the actual implementations but properly simulating the computations of the theoretical systems, conducting to the very same results. This allows us to validate a simpler implementation at a functional level, in terms of the results of the computations, permitting a first step toward more complex high-performance hardware, hybrid, or biological implementations. These initial simulators could handle any solution or model solving a certain problem (from SAT, 3-COL, or HAM-CYCLE to the population dynamics of an ecosystem or an economy system, among others) by means of the types of P systems implemented in the corresponding simulators. Such handling will involve helping in the design, debug, and verification tasks, but of course also the computation of the given model or solution according to the semantic and dynamic rules of the theoretical devices. Naturally, for any problem of certain size, analyzing a manual trace of the computation in a paper would be too tedious or practically unfeasible for significantly big instances. Therefore, even if a real implementation with the desirable parallelism is not available, it would be necessary to have at disposal a machine where one could simulate computations to validate the model or analyze the evolution of the system under certain scenarios. That would be the approach followed by P-Lingua framework and MeCoSim so that we can focus on aspects such as reliability or feasibility (to preserve the same evolution and results of the theoretical systems), along with user-friendliness, over the efficiency of other later implementations.

We have clarified the first goal of MeCoSim approach: providing an environment for the design, verification, and simulation of the models based on P systems in a reliable and user-friendly way, albeit not prioritizing efficiency. However, there are more aspects to analyze in our approach. A major one is the search for the generalization, *that is*, a definite purpose of providing general-purpose tools to be applied to each particular membrane system type and each particular solution for a problem based on them. Thus, the development of *ad hoc* simulators (for a specific solution for a problem, a single instance/scenario, or different instances of that problem) is definitely different from the development of a general-purpose simulator for certain types of P systems, capturing the ingredients of the theoretical model of computation so that this machine allows the provision of any P system and scenario and performs its corresponding intended computations.

Many software developments in scientific research are focused on the first approach, providing simulators for certain problems or even specific instances of the problem only. Other studies address the development of tools for the analysis or verification of a specific type or variant of membrane system, such that they can handle any P system of the type. However, the approach of P-Lingua framework and MeCoSim has been more ambitious from its origin: providing tools being as general as possible, for as many types of P systems as possible (including many variants of cell-like, tissue-like, and neuron-like P systems, among others), while preserving the strict deep analysis of the syntactic, semantic, and dynamic aspects of each P system variant, in order to control that the corresponding constraints are met. The wide range of variants covered include computing models with different global structures (hierarchical, plain graph, or graph with nodes containing trees—as in multienvironment, PDP systems), a variety of ingredients in terms of rules or other elements (dissolution, division, charges, stopping objects, etc.), and different handling of semantic aspects (related with sequentiality, nondeterminism, priorities, maximality, probabilistic or stochastic behaviors, among others).

As described above and in Sects. 2.2 and 2.3, many types and variants of P systems have been covered by the tools developed within the framework. Moreover, as detailed in Sect. 2.2, along with the specification language and its corresponding parsers for each computing models, many simulators were developed inside `P-Lingua` project. This infrastructure provided from the beginning [36] a complete programming environment for *membrane computing* and has kept incorporating new elements along the years, staying as an alive project, including a living version inside MeCoSim.

In Sect. 3.4, the basic steps of the approach followed with these tools are depicted, illustrating their use for real applications.

2.4.2 Main Functional Components

As previously mentioned, a clear separation of the roles involved in modeling and simulation process is stated (apart from the software developers in charge of P-Lingua and MeCoSim development): (a) P systems designer and (b) end users of a simulation app. What does `MeCoSim` provide within this scope?

As shown in Fig. 2.1, the software developer releases different versions of `MeCoSim` (and certain plug-ins) available for any potential users. In contrast, P systems designer, possibly unrelated with software development, defines a simulation app based on `MeCoSim`, customized for its particular problem. Then, he can debug its solution and analyze the underlying P system. Finally, the end user

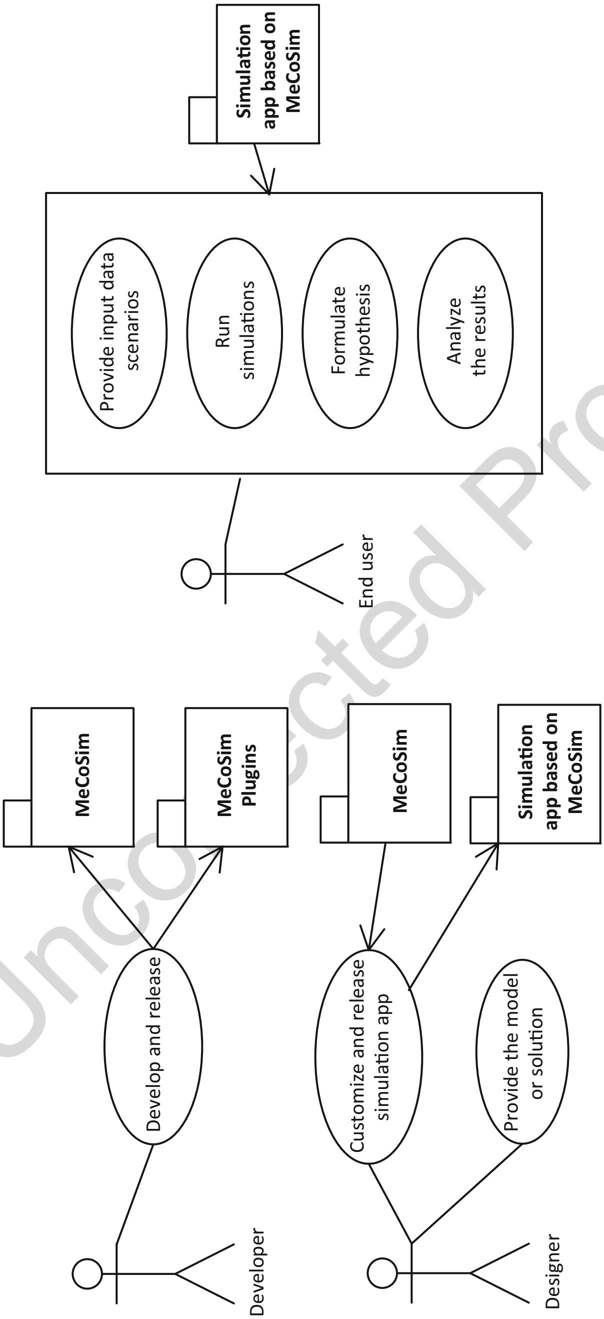

Fig. 2.1 Roles and uses of `MeCoSim`

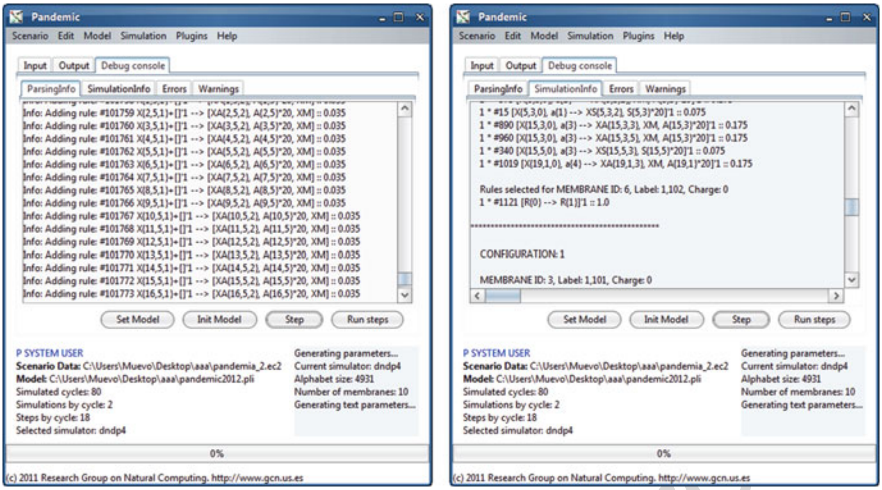

Fig. 2.2 Model debugging

employs the custom simulation app to study different scenarios of interest involving specific instances of the problem.

In summary, the main functionalities of `MeCoSim` are the following:

- *General environment to simulate computations of P systems*
  With the default custom application, any P-Lingua file not requiring additional inputs can be edited, while detecting aspects to modify; parsed and debugged (see Fig. 2.2), to find possible warnings or errors, both at a syntactic and a semantic level, alerting the P systems designer if some rules of the intended model type are violated; and simulated (through the algorithm selected in the interface, as shown in Fig. 2.3), generating the initial structure and multisets of the system and then running the computation either step-by-step or until its end (after a fixed number of steps or when a halting configuration is reached, where no rules can be applied). Besides, the default output is given in the form of a flat table (with a row for each object symbol present in each computation step inside each region, with a certain multiplicity), and also some of the main internal elements of the P system can be visualized at any moment (membrane structure, multisets, and alphabet).
- *Mechanism for the definition of custom simulation apps*
  Any custom app consists of:
  – A hierarchical structure for the visual arrangement of the information (inputs and outputs) in the app for the end user, according to the setting introduced by the designer in an *.xls* spreadsheet file, as illustrated by the first table in Fig. 2.4.
  – A definition of input tables, and output tables and charts, to respectively introduce data and visualize results. More details about the definition of such

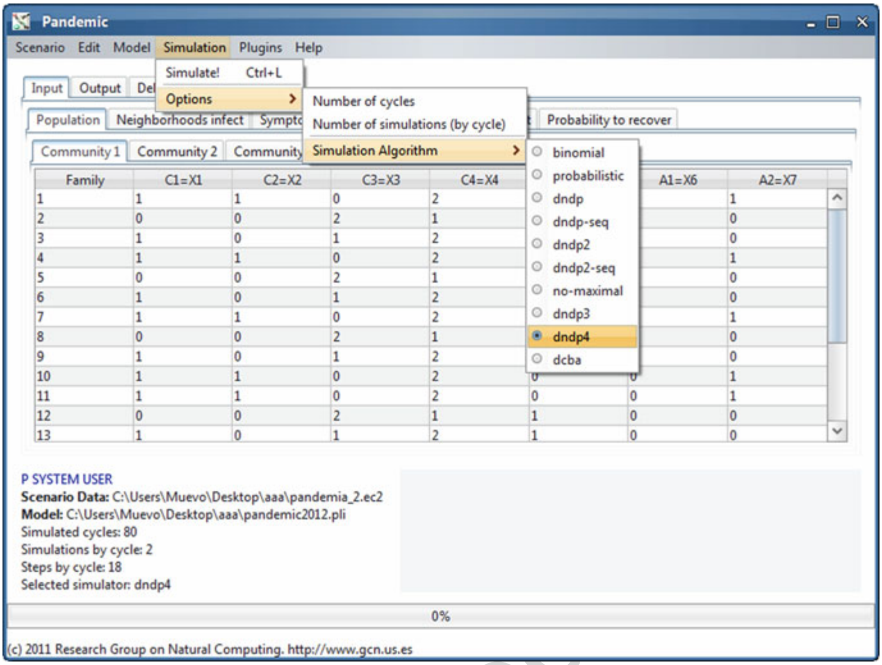

Fig. 2.3 Available simulators for a loaded P systems variant

Tab Id	Tab Name	Tab Parent Id
1	Tissue Example	0
2	Input	1
3	Size (n)	2

Table Id	Table Name	Tab Id	Columns	Init Rows	Save To File	Input / Output
1	Size (n)	3	1	1	TRUE	Input

Column Id	Column Name	Default Value	Editable	Tooltip	GraphicRole
1	Size	3	TRUE	n	

Param Name	Param Value
n	<5,1,1>

Fig. 2.4 Custom app definition—input

components can be found at Refs. [24, 37, 44], and a basic example of tables (header and columns) configuration is given by the second and third tables present in Fig. 2.4.

– The configuration of parameters, establishing which parameters and input data for the model, should be generated from the input tables, either directly taking the value from the table at the beginning of the simulation or applying some processing from the input data to generate calculated derived values. To this purpose, a specific parameters generation language was defined, as described in detail in [37, 44]. The generation of a basic parameter n from the first row and column of a table with $id = 5$ is illustrated in the last table of Fig. 2.4. Much more complex parameters can be generated, as described in [37, 44].

– Results: In order for the output tables and charts to show some information about the simulation, the custom configuration must define which elements from the computation should be taken into account when extracting information from all the computation trace data. An additional language is used in the *.xls* spreadsheet file to provide a flexible mechanism to express the retrieval of information from the computation. Internally, for every simulation performed, from the previous definition, a database query is generated, being executed against the given *on-memory* database containing the flat structure with the computation.

Apart from this core functionalities, additional features and abilities are provided in the form of MeCoSim plug-ins following a certain architecture proposed [24, 37, 44]. These *add-ons* can be given either as Java-based packages (as a graph viewer [see Fig. 2.5], a window for the introduction and encoding of logical formulas or a tool to define and detect invariants in the models based on Daikon [7]) or as external programs being called from `MeCoSim` and properly connected (using the so-called *processes plug-in*). A detailed description of the underlying mechanisms and the plug-ins developed is given in [44].

Additionally to the features of MeCoSim software and its plug-ins, a system of repositories was made available, manageable from MeCoSim environment, having access to repositories of four types: apps (*.xls*), models (*.pli*), scenarios (*.ec2*), and plug-ins (*.jar*). Besides the official repositories, any user can provide additional ones (through the definition of the corresponding *.xml* file for the desired type of repository), providing the corresponding URL to the resource.

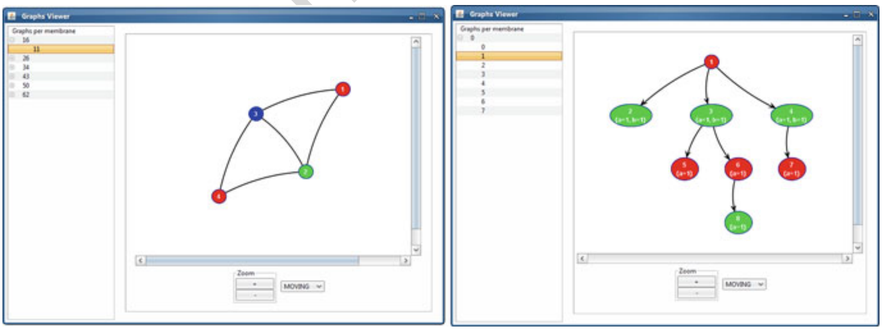

Fig. 2.5 GraphsPlugin—trees of graphs

2.5 Conclusion

This chapter presented the very representative and widely used P-Lingua language for a variety of P systems such as cell-like P systems, tissue-like P systems, spiking neural P systems, and kernel P systems. The description of P-Lingua language with pLinguaCore consists of P systems models, membrane structure, initial multisets, and P system rules. The simulation algorithms in P-Lingua and MeCoSim on top of pLinguaCore with primary goals and main functional components were also discussed.

References

1. F.G.C. Cabarle, H.N. Adorna, N. Ibo, Spiking neural P systems with structural plasticity, in *Pre-proceedings of 2nd Asian Conference on Membrane Computing, Chengdu, China* (2013), pp. 13–26
2. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, A computational modeling for real ecosystems based on P systems. Nat. Comput. **10**(1), 39–53 (2011). https://doi.org/10.1007/s11047-010-9191-3
3. M. Cavaliere, O. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth, Asynchronous spiking neural P systems: decidability and undecidability, in *DNA Computing. Lecture Notes in Computer Science*, ed. by M. Garzon, H. Yan, vol. 4848 (2008), 246–255. https://doi.org/10.1007/978-3-540-77962-9_26
4. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, P-Lingua: a programming language for Membrane Computing, in *Proceedings of the Sixth Brainstorming Week on Membrane Computing*, Fénix Editora, D. Díaz-Pernil, C. Graciani, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez (2008), pp. 135–155
5. D. Díaz-Pernil, C. Graciani, M.A. Gutiérrez-Naranjo, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Software for P systems, in *The Oxford Handbook of Membrane Computing*, ed. by Gh. Păun, G. Rozenberg, A. Salomaa (Oxford University, Oxford, 2009), pp. 437–454. Chapter 17
6. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P-Lingua programming environment for Membrane Computing, in *Membrane Computing (WMC 2008)*, ed. by D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5391 (2009), pp. 187–203. https://doi.org/10.1007/978-3-540-95885-7_14
7. M.D. Ernst, J.H. Perkins, P.J. Guo, S. McCamant, C. Pacheco, M.S. Tschantz, C. Xiao, The Daikon system for dynamic detection of likely invariants. Sci. Comput. Program. **69**(1–3), 35–45 (2007). https://doi.org/10.1016/j.scico.2007.01.015
8. M. García-Quismondo, *Modelling and Simulation of Real-life Phenomena in Membrane Computing*. Ph.D. Thesis (Universidad de Sevilla, Sevilla, 2014). http://hdl.handle.net/11441/66147
9. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, An overview of P-Lingua 2.0, in *Membrane Computing. WMC 2009. Lecture Notes in Computer Science*, vol. 5957, ed. by Gh. Păun, M.J. Pérez-Jiménez, A. Riscos, G. Rozenberg, A. Salomaa (2010), pp. 264–288. https://doi.org/10.1007/978-3-642-11467-0_20
10. M. García-Quismondo, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, Probabilistic guarded P systems: a formal definition, in *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, Fénix Editora, ed. by L.F. Macías-Ramos, M.A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (2014), pp. 183–206
11. M. García-Quismondo, M. Levin, D. Lobo, Modeling regenerative processes with membrane computing. Inf. Sci. **381**, 229–249 (2017). https://doi.org/10.1016/j.ins.2016.11.017

12. M. Gheorghe, F. Ipate, R. Lefticaru, M.J. Pérez-Jiménez, A. Turcanu, L. Valencia, M. García-Quismondo, F. Mierla, 3-COL problem modelling using simple kernel P systems. Int. J. Comput. Math. **90**(4), 816–830 (2013). https://doi.org/10.1080/00207160.2012.743712
13. M. Ionescu, Gh. Păun, T. Yokomori, Spiking Neural P systems. Fundam. Inform., **71**(2–3), 279–308 (2006)
14. F. Ipate, R. Lefticaru, L. Mierla, L. Valencia, H. Hang, G. Zhang, C. Dragomir, M.J. Pérez-Jiménez, M. Gheorghe, Kernel P systems: applications and implementations. Adv. Intell. Syst. Comput. **212**, 1081–1089 (2013). https://doi.org/10.1007/978-3-642-37502-6_126
15. L.F. Macías-Ramos, *Developing Efficient Simulators for Cell Machines*. Ph.D. Thesis (Universidad de Sevilla, Seville, 2016). http://hdl.handle.net/11441/36828
16. L.F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P-Lingua based simulator for Spiking Neural P systems, in *Membrane Computing (CMC 2011)*, ed. by M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, S. Verlan. Lecture Notes in Computer Science, vol. 7184 (2012), pp. 257–281. https://doi.org/10.1007/978-3-642-28024-5_18

17. L.F. Macías-Ramos, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, Simulating FRSN P systems with real numbers in P-Lingua on sequential and CUDA platforms, in *Membrane Computing (CMC 2015)*, ed. by G. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron. Lecture Notes in Computer Science, vol. 9504, pp. 262–276 (2015). https://doi.org/10.1007/978-3-319-28475-0_18
18. L.F. Macías-Ramos, L. Valencia-Cabrera, B. Song, T. Song, L. Pan, M.J. Pérez-Jiménez, A P-lingua based simulator for P systems with symport/antiport rules. Fundam. Inform. **139**(2), 211–227 (2015). https://doi.org/10.3233/FI-2015-1232
19. M.A. Martínez-del-Amor, *Accelerating Membrane Systems Simulators using High Performance Computing with GPU*. Ph.D. Thesis (Universidad de Sevilla, Sevilla, 2013). http://hdl.handle.net/11441/15644
20. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P-Lingua based simulator for Tissue P systems. J. Logic Algebraic Program. **79**(6), 374–382 (2010). https://doi.org/10.1016/j.jlap.2010.03.009
21. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, M.A. Colomer, A new simulation algorithm for multienvironment probabilistic P systems, in *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, Changsha, 2010, vol. 1 (2010), pp. 59–68. https://doi.org/10.1109/BICTA.2010.5645352
22. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Romero-Jiménez, C. Graciani-Díaz, A. Riscos-Núñez., M.A. Colomer, M.J. Pérez-Jiménez, DCBA: simulating Population Dynamics P Systems with proportional object distribution, in *Membrane Computing. CMC 2012*, ed. by E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, G. Vaszil. Lecture Notes in Computer Science, vol. 7762 (2012), pp. 291–310. https://doi.org/10.1007/978-3-642-36751-9_18
23. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Romero-Jiménez, C. Graciani, A. Riscos-Núñez, M.A. Colomer, M.J. Pérez-Jiménez, DCBA: simulating population dynamics P systems with proportional objects distribution, in *Membrane Computing (CMC 2012)*, ed. by E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, G. Vaszil. Lecture Notes in Computer Science, vol. 7762 (2013), pp. 257–276. https://doi.org/10.1007/978-3-642-36751-9_18
24. MeCoSim website. http://www.p-lingua.org/mecosim
25. M. Mutyam, K. Krithivasan, P systems with membrane creation: universality and efficiency, in *Machines, Computations, and Universality (MCU 2001)*, ed. by M. Margenstern, Y. Rogozhin. Lecture Notes in Computer Science, vol. 2055 (2001), pp. 276–287. https://doi.org/10.1007/3-540-45132-3_19

26. D. Orellana-Martín, M.A. Martínez-del-Amor, L. Valencia-Cabrera, I. Pérez-Hurtado, Agustín Riscos-Núñez, M.J. Pérez-Jiménez, Dendrite P Systems toolbox: representation, algorithms and simulators. Int. J. Neural Syst.. Available online 30 September 2020. https://doi.org/10.1142/S0129065720500719
27. L. Pan, T.-O. Ishdorj, P systems with active membranes and separation rules. J. Universal Comput. Sci. **10**(5), 630–64 (2004). https://doi.org/10.3217/jucs-010-05-0630
28. L. Pan, Gh. Păun, Spiking neural P systems with anti-spikes. Int. J. Comput. Commun. Control **4**(3), 273–282 (2009). https://doi.org/10.15837/ijccc.2009.3.2435
29. L. Pan, J. Wang, H.J. Hoogeboom, Limited asynchronous spiking neural P systems. Fundam. Inform. **110**(1–4), 271–293 (2011). https://doi.org/10.3233/FI-2011-543
30. L. Pan, J. Wang, H.J. Hoogeboom, Asynchronous extended spiking neural Psystems with astrocytes, in *Membrane Computing (CMC 2011)*, ed. by M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, S. Verlan. Lecture Notes in Computer Science, vol. 7184 (2012), pp. 243–256. https://doi.org/10.1007/978-3-642-28024-5_17
31. L. Pan, B. Song, L. Valencia-Cabrera, M.J. Pérez-Jiménez, The computational complexity of tissue P systems with evolutional symport/antiport rules. Complexity **2018**, Article ID 3745210, 21 (2018). https://doi.org/10.1155/2018/3745210
32. Gh. Păun, Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693. First circulated at TUCS Research Report No. 208, November 1998. http://www.tucs.fi
33. Gh. Păun, P systems with active membranes: attacking NP complete problems. J. Autom. Lang. Comb. **6**(1), 75–90 (2000). Auckland University, CDMTCS Report No 102 (1999)
34. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, Tissue P systems with cell division. Int. J. Comput. Commun. Control **3**(3), 295–303 (2008). https://doi.org/10.15837/ijccc.2008.3.2397
35. A.B. Pavel, O. Arsene, C. Buiu, Enzymatic numerical P systems: a new class of Membrane Computing systems, in *Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010), Changsha, China, September 23–26* (2010), pp. 1331–1336. https://doi.org/10.1109/BICTA.2010.5645071
36. I. Pérez-Hurtado, *Desarrollo y Aplicaciones de un Entorno de Programación para Computación Celular: P-Lingua*. Ph.D. Thesis (Universidad de Sevilla, Sevilla, 2010, in Spanish). http://hdl.handle.net/11441/66241
37. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez, MeCoSim: a general purpose software tool for simulating biological phenomena by means of P systems, in *Proceedings of the IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010)*, vol. I, ed. by K. Li, Z. Tang, R. Li, A.K. Nagar, R. Thamburaj (2010), pp. 637–643. https://doi.org/10.1109/BICTA.2010.5645199
38. I. Pérez-Hurtado, L. Valencia-Cabrera, J.M. Chacón, A. Riscos-Núñez, M.J. Pérez-Jiménez, A P-lingua based simulator for tissue P systems with cell separation. Rom. J. Inf. Sci. Technol. **17**(1), 89–102 (2014)
39. I. Pérez-Hurtado, D. Orellana-Martín, M.A. Martínez-del-Amor, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez, 11 years of P-Lingua: a backward glance, in *Proceedings of the 20th International Conference on Membrane Computing (CMC20)*, ed. by Gh. Păun (2019), pp. 451–462
40. I. Pérez-Hurtado, D. Orellana-Martín, G. Zhang, M.J. Pérez-Jiménez, P-Lingua in two steps: flexibility and efficiency. J. Membr. Comput. **1**(2), 93–102 (2019). https://doi.org/10.1007/s41965-019-00014-1
41. T. Song, L. Pan, Gh. Păun. Asynchronous spiking neural P systems with local synchronization. Inf. Sci. **219**, 197–207 (2013). https://doi.org/10.1016/j.ins.2012.07.023
42. A. Spicher, O. Michel, M. Cieslak, J.-L. Giavitto, P. Prusinkiewicz, Stochastic P systems and the simulation of biochemical processes with dynamic compartments. Biosystems, **91**(3), 458–472 (2008). https://doi.org/10.1016/j.biosystems.2006.12.009
43. The P-Lingua website. http://www.p-lingua.org

44. L. Valencia-Cabrera, *An Environment for Virtual Experimentation with Computational Models Based on P Systems*. Ph.D. Thesis (Universidad de Sevilla, Sevilla, 2015). http://hdl.handle.net/11441/45362
45. L. Valencia-Cabrera, B. Song, Tissue P systems with promoter simulation with MeCoSim and P-Lingua framework. J. Membr. Comput. **2**(2), 95–107 (2020). https://doi.org/10.1007/s41965-020-00037-z
46. L. Valencia-Cabrera, T. Wu, Z. Zhang, L. Pan, M.J. Pérez-Jiménez, A simulation software tool for cell-like spiking neural P systems. Rom. J. Inf. Sci. Technol. **20**(1), 71–84 (2017)
47. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez, Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. Fundam. Inform. **153**(1–2), 147–172 (2017). https://doi.org/10.3233/FI-2017-1535
48. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez, Reaching efficiency through collaboration in membrane systems: dissolution, polarization and cooperation. Theor. Comput. Sci. **701**, 226–234 (2017). https://doi.org/10.1016/j.tcs.2017.04.015
49. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, An interactive timeline of simulators in Membrane Computing. J. Membr. Comput. **1**(3), 209–222 (2019). https://doi.org/10.1007/s41965-019-00016-z
50. T. Wu, Z. Zhang, Gh. Păun, L. Pan, Cell-like spiking neural P systems. Theor. Comput. Sci. **623**, 180–189 (2016). https://doi.org/10.1016/j.tcs.2015.12.038

AQ3