Journal of Membrane Computing (2020) 2:392–402
https://doi.org/10.1007/s41965-020-00056-w

SURVEY PAPER

Simulation challenges in membrane computing

Luis Valencia-Cabrera[1] · **Ignacio Pérez-Hurtado**[1] · **Miguel Á. Martínez-del-Amor**[1]

Received: 22 July 2020 / Accepted: 6 October 2020 / Published online: 31 October 2020
© Springer Nature Singapore Pte Ltd. 2020

Abstract
P system simulators are critical tools to enable them as formal modeling framework for real-life applications. Such simulators abstract the concept of P systems in various ways, depending on the needs of the users and the requirements of the specific application. We identify three main levels of abstraction: graphical user interfaces, simulation engines and parallel implementations. In this paper, we survey the state of the art at these levels and discuss the main challenges under consideration for future developments.

Keywords Membrane computing · Simulation · P-Lingua · MeCoSim · PMCGPU · Parallelism

1 Introduction

P systems have been used in a wide variety of real-life applications serving as formal modeling framework [32]. In this context, simulation tools are essential for the debugging, analysis and refinement of such models and solutions based on P systems [31]. Nobody questions nowadays the need for such software assistants, complementing the tedious process of manually following computation traces of the designed P systems. Besides, for problem instances beyond a certain size, this handmade work is impractical or just unfeasible [6].

In the literature, we can identify different orientations when developing P system simulators, which can be identified as levels of abstractions. That is, how much the syntax and semantics of the P system being simulated is abstracted to the end user. In this paper, we will focuse in the following ones:

- Graphical user interface (GUI): this level provides the highest level of abstraction, allowing users to handle certain scenarios without even realizing the details about the underlying P system design. GUIs can be used to ease the process of simulation and modeling with P systems, by giving the pertinent interfaces for inputs and outputs.
- Simulation engine: this level is the one that handles all the P system information, both syntactical and semantic. The behavior of the theoretical model to be simulated must be reproduced accordingly. Sometimes, this must be restricted for certain models, but there is a current trend on providing a flexible framework for simulation.
- Parallel simulation: at this level, not only the semantics of the P system must be handled, but also the inherent parallelism of these devices must be used efficiently to provide accelerated simulators. This can be seen as a low-level abstraction, since one needs to take into account both the formal details on how the system performs a computation step as well as the technical details on how P system rules are going to be simulated in parallel on the available hardware.

For example, an end-to-end simulation tool like MeCoSim [30] includes the GUI and simulation levels of abstraction, P-Lingua [11] involves the simulation level, and PMCGPU [1] works on the parallel simulation level. Thus, in this paper, we will focus on the state of the art at these levels of abstraction and specific tools. We will survey the main milestones and the current developments, and use this as a base for further discussion on the challenges that have been

✉ Luis Valencia-Cabrera
lvalencia@us.es

Ignacio Pérez-Hurtado
perezh@us.es

Miguel Á. Martínez-del-Amor
mdelamor@us.es

[1] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, E.T.S. Ingeniería Informática, Avda. Reina Mercedes S/N, 41012 Sevilla, Spain

encountered. These challenges will drive future developments in this concern.

The paper is structured as follows: Sect. 2 introduces the state of the art at the three different layers for P system simulation; Sect. 3 discusses the main challenges in each of these layers; and Sect. 4 ends the paper with some conclusions.

2 Layers of abstraction in simulation

In this section, we will discuss the three layers of abstractions when simulating P systems, and current solutions in each one.

2.1 Graphical user interfaces

When a new model is being created by a P system designer, certain processes can be considerably time-consuming. For instance, generating different instances for scenarios of interest and simulating them, while debugging the model. Additionally, extracting results, generating charts and analyzing the outputs involves a significant amount of work. These tasks are common in most of the models, and imply a huge effort if each scenario of interest requires the manual encoding of every element by the P system designer. Consequently, it is crucial to have not only tools to parse and simulate different types of P systems but also tools making it easier for the modelers to design, debug or create new scenarios, as well as to automate certain processes to encode the information of the system.

The aspects just pointed out might be solved with different approaches, and the provision of a graphical user interface (GUI) for P system experts to debug, visualize and run experiments seems a good option to increase their productivity and potentially reduce the risk of introducing errors or undesired effects in the solutions and models based on P systems. Additionally, these graphical interfaces present a benefit derived from the higher level of abstraction they provide, in relation to programming environments or command-line tools. This advantage is the fact that they can hide many internal aspects of the underlying models, thus opening the possibility for end users, not familiar with P systems, to have a virtual environment where they simply introduce their data, load the model, run experiments and receive the results of their scenarios of interest. Instead of models, configurations, membrane structures, objects or rules, they will simply observe the behavior of the system under study (in Biology, Economy, Logics or any other field they are experts in), focusing on elements of their problem domain.

All in all, the difference between these high-level interfaces and the alternative of command-line tools or directly programming each specific problem and scenario is not in the problems they solve, but the language they offer to the user, the point of view they offer. While the latter approach focuses on the control of the internal details of the systems, the former one focuses on the problem itself being solved by the internal system. For example, consider the differences between R programming through RStudio environment, that provides the capability to control every detail of the studies with great flexibility, versus using interfaces like SPSS or SAS, enabling a number of pre-built functionalities performing the operations of interest, by simply introducing data and running the desired studies through the corresponding buttons or menu options.

In the particular case of P system software tools, the most widely used framework is P-Lingua [11], covering a number of P system types and variants with a general-purpose approach. It provides a standard language for P system specification, along with pLinguaCore library, including parsing, debugging and simulation tools. With respect to graphical user interfaces, within this project, several ad hoc tools were created for several ecosystem models, and the common patterns in terms of input needs, output requirements, parameters handling and communication with P-Lingua were extracted to plan a new goal: having a customizable visual interface to be adapted to any possible ecosystem model based on P systems. This was the origin of the so-called membrane computing simulator (MeCoSim [30]), providing high-level features as the ones highlighted above, both for P system designers (focusing on the internal details of P systems), and for the end users (taking advantage of the ready-to-use models and apps as black boxes for their virtual experiments).

MeCoSim is built on top of P-Lingua framework and provides general-purpose tools for all the types of P systems covered by such framework, mostly using its parser and simulation engines (along with some additional external simulators and tools). On top of those functionalities, it provides a mechanism to prepare custom apps through a spreadsheet configuration (an alternative JSON format is also accepted). In Fig. 1 the main interface of MeCoSim is given in the background, where the custom apps can be imported. In the foreground, one of such apps is shown, along with some charts and graphs obtained from the computation.

This high-level interface abstracts the users from the simulation engines used. Thus, whenever a P system-based model is introduced in MeCoSim, using P-Lingua format, and the input data are provided through input tables, the environment reads the type of model the solution is based on, and chooses a default simulator for such model. With that simulator, the end users run their virtual experiments, making the system evolve for the number of cycles requested by the user (or until a halting configuration is reached). This process is transparent for such end users, simply introducing data and analyzing the results obtained after the simulation has finished.

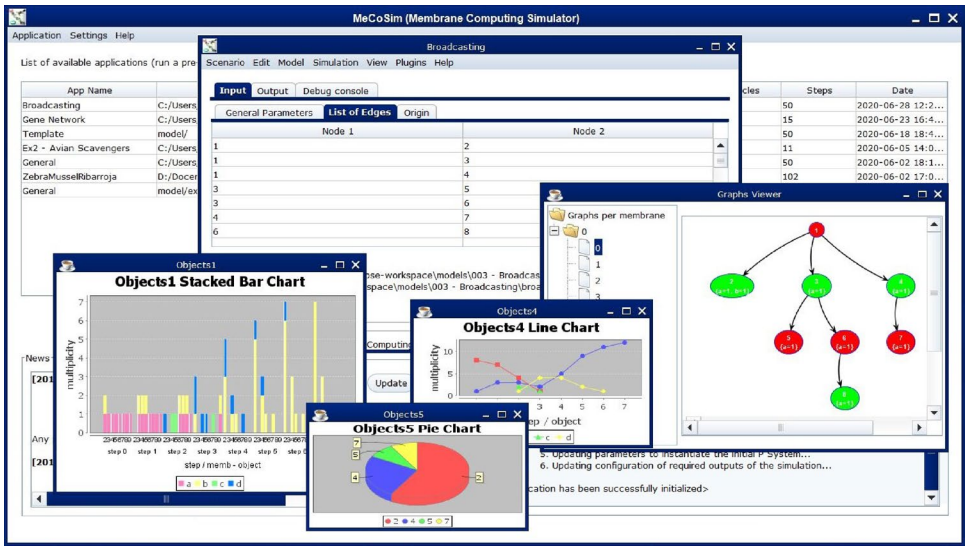

Fig. 1 MeCoSim main panel and custom app

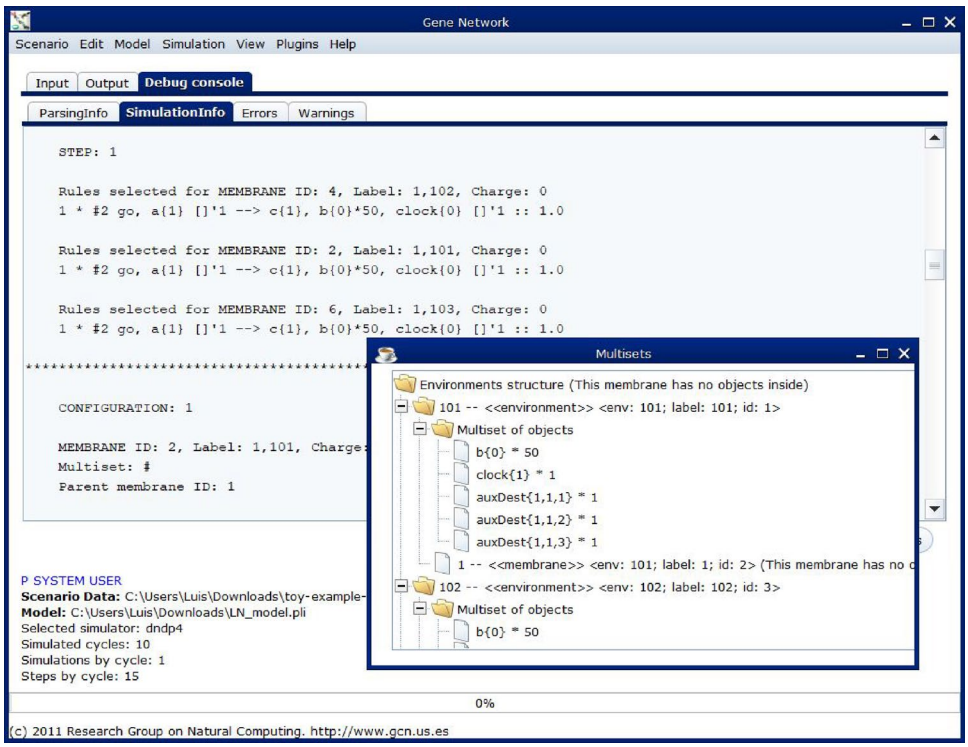

Fig. 2 MeCoSim debugging facilities: parsing, step-by-step simulation and visualization

For more technical users, designing the underlying models based on P systems, MeCoSim provides debugging capabilities inherited from P-Lingua, plus some visual aids making it easier to see how the objects are present inside each region of the system, as shown in Fig. 2.

Some of the mechanisms enabling these features, as parsing the models in P-Lingua language or simulating the behavior of the system (step by step or entirely) are using pLinguaCore as its engine. As this framework covers a broad range of P system types and variants, the proper parsers and simulation engines are automatically selected depending on the type of model loaded. Then the visualization of structures and multisets, built as part of MeCoSim, shows the information based on the type of P system structure (cell like, tissue like, neuron like or multi-environment).

Along with these more general mechanisms, other aspects as the particular encoding of the input of the user as objects of the P system or the decoding of the objects of the system into the elements of the problem domain that are meaningful for the end user, are in the essence of MeCoSim. These aspects are the crucial elements that a designer user defines

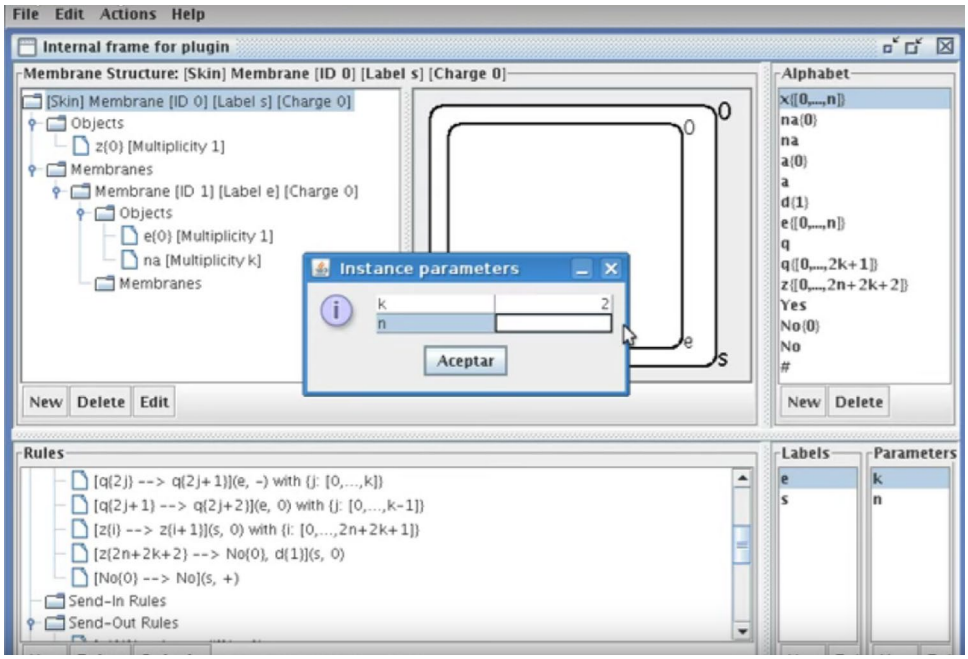

Fig. 3 P-Lab: a prototype to define P systems using a GUI

in the spreadsheet configuration file for a custom app for an end user: tabs hierarchy to visually arrange the inputs and outputs, input tables to introduce the data, simulation parameters to be generated from the input tables, simulation results to extract from the results of the computation, and output tables and charts to visualize such results. With such custom app configuration file (.xls) and the P-Lingua model file (.pli) loaded in MeCoSim, any end user not familiar with P system can simply introduce the desired inputs in the tables, run the system and observe the outputs. This environment could be used for the managers in charge of certain problems our P system-based models are trying to represent, in such a way that they analyze potential scenarios of interest and get valuable information for their decision-making processes.

2.2 Simulation engines

As mentioned above, to develop simulators for membrane computing is absolutely needed to validate the designed models and provide virtual experiments. We can define a simulator as a software or hardware tool which is able to reproduce computations of a given P system, following the corresponding derivation mode. Usually, a simulator provides only one branch of computation each time it is executed and different simulation modes should be included: step-by-step simulation, where the simulator provides a trace of the computation showing the selected rules for each step of computation, as well as the intermediate configurations. Another typical simulation mode is to simulate until a halting configuration.

Regardless of the software/hardware architecture of a simulator, one common problem to solve is how to define the P system to be simulated, i.e, the initial membrane structure, the initial multisets and the initial set of rules. There are several approaches, one could use a graphical user interface, as P-Lab (Fig. 3), a prototype designed in 2007 by members of the RGNC that was abandoned before being published. There are two main problems when defining P systems using GUIs: one is the dependency on the current technology; another one is the lack of flexibility.

For example, P-Lab was designed in Java 6, today the last version of Java is 14 and version 6 is obsolete. A big effort should be done to update versions and, sadly, researchers do not have the needed resources and technical people.

Another approach is to use definition languages, in this sense, the failure of P-Lab was the origin of P-Lingua.

P-Lingua is a definition language for P systems which was created in 2008 and it has become a standard in membrane computing. P-Lingua is able to define P systems in a modular and parametric way. From version 1.0 to 4.0, the language comes with a library called pLinguaCore which is responsible to parse P-Lingua files, detect syntax and semantic errors and produce output files in other formats such as XML and JSON, the output files codify the same P system given as input, but after the parsing process, i.e., free of errors. Thus, third-party simulators can use such outputs as inputs without bothering to check errors. PLinguaCore also includes a battery of simulators, at least one for each type of supported P system. The semantics or derivation mode, i.e., the way in which the rules of a particular P system are

executed, is hard-coded in the corresponding simulator within the pLinguaCore library. The next example is a transition P system codified in P-Lingua:

```
@model<transition>
def main()
{
  @mu = [[[]'3 []'4]'2]'1;
  @ms(3) = a,f;

  [a --> a,bp]'3;
  [a --> bp,@d]'3;
  [f --> f*2]'3;

  [bp --> b]'2;
  [b []'4 --> b [c]'4]'2;
  (1) [f*2 --> f ]'2;
  (2) [f --> a,@d]'2;
}
```

The first line provides a unique identifier for the derivation mode to be used, only a predefined list of derivation modes can be used. The next line is the definition of the main module, in this example, only one module is used, but a P-Lingua file can include several modules. The `@mu` instruction defines the initial membrane structure and the `@ms(3)` instruction defines the initial multiset for membrane labeled `3`. The rest of the module are rules. The syntax of rules in P-Lingua tries to be close to the standard scientific notation.

2.3 Parallel implementations

Implementing P system parallelism is a good option to increase the performance of the simulators. However, the development of parallel P-system simulators is a challenge itself. Translating the semantics of these systems into a simulator is already a non-trivial task. Implementing the natural parallelism of P systems in more classical parallel computing platforms, while maintaining the correctness of the simulated semantics, is a complex but interesting task, which has received important attention from the community, and even led to a number of doctoral theses [10, 17, 21].

The increasing need of efficient simulators for P systems, motivated by the requirements of both real-life applications and theoretical research, has led to a new research line within membrane computing focusing on implementing real parallelism on high-performance computing platforms [33]: FPGAs[1], GPUs, computer networks, big data environments, etc. In this section, we will focus on the implementation of P system parallelism on graphics processing units (GPUs), which turns out to be one of the most explored technologies for this purpose [22]. The main reason for the popularity of GPUs on the simulation of P systems is that they offer a cheap platform, with a high parallel degree, easy to program, flexible and with a shared-memory environment (required to efficiently implement the synchronization of the global clock).

Indeed, after the introduction of CUDA in 2007 [16], several parallel simulators for P systems have been developed and investigated. The main open-source project that gather these developments is called `PMCGPU` (parallel simulators for membrane computing on the GPU) [1, 21]. Some of the models simulated on the GPU within PMCGPU are the following:

- PCUDA: P systems with active membranes [6, 22].
- ABCDGPU: population dynamics P systems[2] [22, 23, 26, 27].
- PCUDASAT: a family of P systems with active membranes solving `SAT` problem [5, 7, 22].
- TSPCUDASAT: a family of tissue-like P systems with cell division solving `SAT` problem [22, 24].
- ENPS-GPU: enzymatic numerical P systems [12, 22].
- RRT-ENPS-GPU: a family of enzymatic numerical P systems implementing the `RRT/RRT*` algorithms [28].
- CuSNP: spiking neural P systems [3, 4].

Many other developments involving the usage of GPUs to speedup the simulation of P systems can be found in the literature: extended simulation of P systems with active membranes [19], kernel P systems [8, 14], membrane algorithms [34], evolution–communication P systems with energy [15], fuzzy reasoning spiking neural P systems [18], etc.

We can build a taxonomy of these simulators by classifying them by the level of flexibility [20]:

- Generic simulators: they are designed for a P system variant, accepting any model, or a wide range of them, of that variant. It receives the description of a P system model and then simulates one or several computations.
- Specific simulators: they are usually designed for specific problem solved by a P system model, or sometimes for a family of them, within a variant. It receives the parameters specific of the problem they are designed for, and compute the result by simulating the model.
- Adaptative simulators: they are slightly enriched generic simulators, that receive the description of a P system model together with some extra information (e.g.,

[1] Field-programmable gate arrays, energy-efficient devices with application in HPC.

[2] a.k.a. PDP systems.

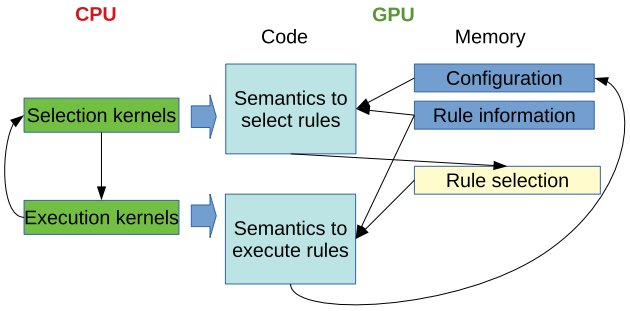

Fig. 4 Scheme of a parallel generic simulator

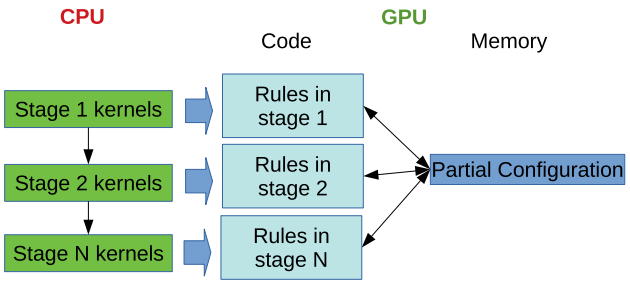

Fig. 5 Scheme of a parallel specific simulator

modules of rules, computation stages, etc.) intended to improve the simulation.

Let us discuss first the usual design of a parallel generic simulator, as shown in Fig. 4. Here, the simulation of a P system is controlled by a main loop, with an iteration per transition step. In every iteration, there are two phases: selection and execution phases. The former involves visiting all the defined rules in the P system and annotating how many times each rule is going to be executed (i.e., the number of selections) in that transition step. The latter actually performs the execution of the selected rules, by updating the configuration according to the left-hand (LHS) and the right-hand (RHS) sides of the executed rules. We need to store the information of the rules (LHS, RHS, charge, probability, priority, etc.), and the configuration of the P system (multisets within each region, charges in membranes, etc.). The codification of a configuration of the P system is typically made in an extended way [20]; that is, with an array using a position per object symbol, saying the number of copies of each object on each region. Such arrays are usually very sparse, but they are required to efficiently pinpointing the objects in the process of rules selection. The GPU kernels implement the semantics of the P system in parallel, by launching many threads over either the rules or the objects. Usually, the selection phase is the most complex, and consists of several sub-phases (this is how DCBA algorithm[3] [25] for PDP systems [26] works, for instance). Examples of generic simulators are PCUDA, ABCDGPU, CuSNP and ENPS-GPU.

In Fig. 5, the scheme of a specific parallel simulator is shown. Contrary to a generic simulator, where the rules of the P system to be simulated are not known at development time, in the specific case information about rules can be encoded in its majority inside the source code. For this, the computation of the specific P system family is usually divided into stages, where specific rules are known to be executed. Hence, the kernels are written to reproduce the behavior of those rules by effectively selecting and executing them without separated phases. It is very important to remark that a "fair" specific simulator must reproduce the execution of the P system rules, so that it should be possible to infer efficiently the configuration of the P system at any given step. Otherwise, we will say that the code is not simulating the P system solving a certain problem, but it is just solving the problem. Moreover, because of this, it would be required to just store some objects of the alphabet in memory, usually in a dense representation; *i.e.,* with a map associating each object with its multiplicity (only when it is greater than 0).

Finally, in Ref. [27], a novel idea was introduced: adaptative simulators. They are essentially generic simulators that receive extra, high-level information from the model designer, along with the P system description. This information can be either dismissed (hence, becoming a generic simulator), or employed (usually to improve the simulation). Specifically, the first adaptative simulator was based on ABCDGPU, and the PDP system was given with additional pieces of information called features. These were used to say to which module, defined by the algorithmic scheme of the ecosystem model, each rule belongs to. This is very useful because one can safely skip visiting many rules that are known not to be applicable at certain transition steps. In other words, the model designer is telling the simulator which range of rules should be visited at every step. This helped to achieve an extra 2.5× of speedup with a K40 GPU.

Finally, let us stand out that parallel generic simulators usually assume that the P systems to simulate are confluent[4]. This eases drastically the GPU code, because the aim of the selection of rules is to find any valid output. However, for systems not required to be confluent (e.g., spiking neural P systems), the non-determinism has been employed as a new level parallelism [4].

[3] DCBA is a simulation algorithm for PDP systems, aiming to "fairly" distribute the consumption of objects among competing rules.

[4] Confluent systems may present different computations for a given input, but all of them leading to the same output.

3 Challenges in the simulation of P systems

In this section, we discuss challenges that will drive future developments of simulation tools for P systems.

3.1 Challenges in graphical user interfaces

The availability of high-level visual interfaces has proved essential in order to make it easier for P system experts to debug their models or solutions to certain problems. Additionally, as clarified in Sect. 2.1, end users interested in the problems themselves, also take advantage of interfaces where they can focus on their scenarios of interest and perform virtual experiments, abstracting them from the complexities happening behind the scene.

However, it is not easy to fill the gap between the modeling of complex systems by P systems and the entities of the real world that the end user view requires. To this purpose, some mechanisms are needed to convert real-world inputs into model parameters instantiating a particular P system inside a family of P systems solving a problem or providing a model for a case study, and providing the multisets of objects for a particular input of such P system. Similarly, the evolution of the P system and its potential results might be far from the real-world entities understood by the end user, so a new mechanism is required to translate the details about configurations and transition steps into meaningful outputs. Not surprisingly, the input and outputs required for completely different underlying systems are inherently different, just as the corresponding problems under study are. Examples range from SAT problem (where a Boolean answer is expected for each input formula) to zebra mussel ecosystem modeling (where one expects graphics showing the population dynamics over time). Consequently, it is very challenging to abstract those mechanisms with a general-purpose philosophy, preserving the usability for the end user but being flexible enough to cover any possible problem solvable by P systems.

For instance, the approach followed by MeCoSim is to organize as tabs with tables all the inputs and outputs, organizing the way they are displayed and arranged according to a configuration file of the particular app. Along with the definition of their names and fields, two more complex mechanisms are defined: first, a language to extract the information from the input tables into the parameters to instantiate the specific P system depending on the scenario introduced by the end user; and second, a language to produce specific views, as tables or charts, for the desired objects, membrane, computation steps, etc. for each particular view of interest for the end user.

While the first three mechanisms (definition of input and output tables and generation of simulation parameters) are relatively easy to implement, the last one is not as user-friendly as it should, as the variety of possible outputs a user might be expecting from the system is huge. For this reason, the approach followed was storing a flattened view of all the computation steps for each object inside each region, and defining in the configuration file some simplified version of a query on a database. While this approach has proved to be very flexible, there is still much room for improvement to make it more intuitive for the designers of the P systems defining the interfaces to be used by the end users.

3.2 Challenges in simulators

There are several challenges related to the development of simulators. As mentioned in Sect. 2.2, one of the main modules for all simulators is the one in charge to define the P system to be simulated, the definition of a P system can be divided into two parts: On the one hand, the definition of the syntactical elements, such as initial structures, initial multisets of objects and initial set of rules; on the other hand, the definition of the derivation mode or semantics, i.e, the way in which computations should be generated for the defined P system. Defining a semantics is a hard task and most of the current simulators include the semantics in the simulation algorithm, i.e., the simulator can only simulate a fixed set of types of P systems, a.k.a. variants of P systems. In the case of P-Lingua for versions 1.0–4.0, the first line of the P-Lingua file should include an identifier of the variant to be used. If the variant is not implemented in the source code of the simulator, the P system definition cannot be used. This is a lack of flexibility, especially when designers are interested in playing with experimental variants. There are several research works introducing how to define the semantics of a P system [9], but until now, there does not exist an efficient way to implement them. It is an important challenge currently being addressed by the team of P-Lingua 5, which is under development. A preliminary version of such ideas can be found in [29].

Moreover, another important issue in software/hardware simulators is how to simulate the non-determinism, since computers based on the Von Neumann architecture are inherently deterministic. Pseudo-random numbers are usually used, being a well-accepted solution. One particular case is related to probabilistic models in general, and PDP systems in particular, where the non-determinism is conducted by multinomial probability distributions. Pseudo-random numbers are not actual random numbers, they depend on a seed which is usually based on the CPU time. Using a true source of randomness such as a specific hardware device is an interesting approach to be studied, at least until true non-deterministic hardware could be designed.

Furthermore, there are many simulators for P systems that can be found in the literature [31], each one defined for different variants and models. However, we point out the lack of uniformity in these developments. The membrane computing community should make the efforts to simplify this process by constructing a common environment for P system simulators. This way, it would decrease the learning curve to both new developers, designers and end users. The challenge in this concern would be to find a common interface to define P systems by creating a standard. This involves creating a working committee to select a candidate (e.g., P-Lingua) and evolving it for the community. Moreover, a communication protocol between input interfaces and simulation engines should be also created and adopted, which should be common for all P system variants. This protocol will help to integrate all simulators into just one platform using this standard language. Moreover, it would be interesting to integrate this idea into a common universal package for P systems in scientific languages such as Python, R, C++ or Haskell, so that users can define, simulate, play, debug and collect results in a programmatic way.

3.3 Challenges in parallel simulators

In this section, we will discuss the main challenges to solve in future developments concerning the parallel simulation of P systems. We will not discuss the simulation of new P system models and applications, given that their wide range cannot be summarized in this paper. Thus, we will discuss how to improve the design of parallel simulators, requiring efforts in both sides: from the design of P system models to the improvement of the simulator source codes.

Simulating P systems is a memory-demanding task, given that the execution of rules requires several accesses to memory for just one conditional operation:

- Computation: check how many times the LHS can be consumed, if it can be done more than once, and check whether there are or not conflicts with other rules according to the semantics.
- Memory accesses: memory reads to know the multiplicities of the objects in the LHS, and memory writes to consume the LHS and produce the RHS are a non-computationally intensive task for threads.

Because of this, the GPU simulators are memory-bandwidth bounded [21]. That is, they spend more time accessing and updating data (multisets) than executing computation. In this sense, one challenge is to design P system variants where the model contains a higher computational intensity; in other words, the rules perform more computation other than rewriting the multisets [13]. Moreover, memory accesses can be partially reduced by improving data structures using a compacted, dense and well-ordered memory representation of P systems. In [20], it was shown that specific simulators already take advantage of knowing which objects can appear at certain steps, so that large arrays are not required to represent all objects defined in the alphabet. The latter can lead to very sparse arrays that all full of zeroes, which will lead to useless threads accessing to empty positions. This is usually worsened when the models use objects as counters, where the counter is a subindex of the object type (e.g., O_i, for $i = 1.1000$); however, they are all different objects to be represented in the arrays.

Another bottleneck in the simulators are the selection phases. Here, rule competitions for the objects have to be sorted out. A P system model with cooperation in the LHS usually leads to this issue, making it more difficult when the cooperation is larger. While many assumptions are taken in theory when designing P systems, considering that the selection of rules can be made non deterministically, in practice rules have to make an agreement within the simulators. This sometimes requires extra phases to control maximality, what can be ensured only by a sequential method (a loop over the remaining rules). Hence, a challenge is to find efficient ways to lightweight the selection phase when rule competition takes place. This can be done from two points of view: from the simulator, applying extra effort to pre-compute the real competitions to reduce the selection time; and from the model, to what extend models with no cooperation or even with minimal cooperation can have enough power for certain problems such as ecological modelng.

Specific simulators are much more efficient than their generic counterparts [22] given that most of the information of the P system to be simulated is known at development time. Hence, the programmer writes the simulator source code bearing in mind the P system model/family. On the other side, generic simulators know the P system to simulate at run time, hence wasting many resources to tackle worst-case scenarios. However, a challenge is to move the knowledge of the P system from development to compilation time. That is, using meta-programming [2], to generate the source code of a simulator from a specific P system model/family. New achievements in P-Lingua will open that way [29], but there is a long way to go.

Finally, another challenge is to find ways to extend the idea of adaptative simulators [27]. They can be used also to reduce the complexity of simulators when selecting rules in solutions to, for example, the `SAT` problem, if the stages of the computation can be encoded along with the P system description. Moreover, objects can be annotated somehow to declare that they are "counters"[5], and hence, only one of

[5] We say that a collection of objects a_i is being used as a counter if the role of the index i is just counting steps (typically the rules associated are of the type $a_i \to a_{i+1}$.

them will appear at each instant, so there is no need to provide a position for each one in the array for multisets (i.e., merging sparse with dense representations).

4 Conclusions for future work

In this paper, we have surveyed the simulation of P systems from a new point of view, which is by abstraction levels. These are: graphical user interfaces for specific problems, simulation engines for a variety of semantics with similar syntax, parallelism implementation using GPUs for specific and different variants. Current developments on each level are presented, surveying in this way the state of the art. This has served as the baseline for further discussions concerning challenges for future research. In this way, we have shown the main topics that will drive the next generation of simulation tools for P systems and its applications, such as: translation of real-world inputs into model parameters, definition of P system semantics, and meta-programming for instantiating parallel simulators.

Acknowledgements This work was supported by the research project TIN2017-89842-P (MABICAP), co-financed by *Ministerio de Economía, Industria y Competitividad (MINECO)* of Spain, through the *Agencia Estatal de Investigación (AEI)*, and by *Fondo Europeo de Desarrollo Regional (FEDER)* of the European Union.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. The PMCGPU (parallel simulators for membrane computing on the GPU) project website. http://sourceforge.net/p/pmcgpu. Accessed Feb 2020.
2. Abrahams, D., & Gurtovoy, A. (2005). *C++ template metaprogramming*. Boston: Addison-Wesley.
3. Carandang, J., Villaflores, J., Cabarle, F.G.C., Adorna, H.N., & Martínez-del-Amor, M.A. (2017). CuSNP: Spiking neural P systems simulators in CUDA. *Romanian Journal of Information Science and Technology 20*(1), 57–70. https://www.imt.ro/romjist/Volum20/Number20_1/cuprins20_1.htm.
4. Carandang, J. P., Cabarle, F. G., Adorna, H. N., Hernandez, Hope S. N., & Martínez-del-Amor, M. A. (2019). Handling non-determinism in spiking neural P systems: algorithms and simulations. *Fundamenta Informaticae*, *164*, 139–155. https://doi.org/10.3233/FI-2019-1759.
5. Cecilia, J. M., García, J. M., Guerrero, G. D., Martínez-del-Amor, M. A., Pérez-Hurtado, I., & Pérez-Jiménez, M. J. (2010). Simulating a P system based efficient solution to SAT by using GPUs. *Journal of Logic and Algebraic Programming*, *79*(6), 317–325. https://doi.org/10.1016/j.jlap.2010.03.008.
6. Cecilia, J. M., García, J. M., Guerrero, G. D., Martínez-del-Amor, M. A., Pérez-Hurtado, I., & Pérez-Jiménez, M. J. (2010). Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics*, *11*(3), 313–322. https://doi.org/10.1093/bib/bbp064.
7. Cecilia, J. M., García, J. M., Guerrero, G. D., Martínez-del-Amor, M. A., Pérez-Jiménez, M. J., & Ujaldón, M. (2012). The GPU on the simulation of cellular computing models. *Soft Computing*, *16*(2), 231–246. https://doi.org/10.1007/s00500-011-0716-1.
8. Elkhani, N., Muniyandi, R. C., & Zhang, G. (2018). Multi-objective binary PSO with kernel P system on GPU. *International Journal of Computers Communications and Control*, *13*, 323–336. https://doi.org/10.15837/ijccc.2018.3.3282.
9. Freund, R., Pérez-Hurtado, I., Riscos-Núñez, A., & Verlan, S. (2013). A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics*, *90*(4), 801–815. https://doi.org/10.1080/00207160.2012.748899.
10. García-Quismondo, M. (2014). Modelling and simulation of real-life phenomena in membrane computing. PhD Thesis. Universidad de Sevilla. 2014. https://idus.us.es/handle/11441/66147.
11. García-Quismondo, M., Gutiérrez-Escudero, R., Martínez-del-Amor, M., Orejuela-Pinedo, E., & Pérez-Hurtado, I. (2009). P-lingua 2.0: a software framework for cell-like P systems. *International Journal of Computers, Communications and Control*, *4*(3), 234–243. https://doi.org/10.15837/ijccc.2009.3.2431.
12. García-Quismondo, M., Macías-Ramos, L. F., & Pérez-Jiménez, M. J. (2013). *Implementing enzymatic numerical P systems for AI applications by means of graphic processing units* (pp. 137–159). Berlin: Springer. https://doi.org/10.1007/978-3-642-34422-0_10.
13. Henderson, A., & Nicolescu, R. (2019). Actor-like cP systems. In T. Hinze, G. Rozenberg, A. Salomaa, & C. Zandron (Eds.), *Membrane computing* (pp. 160–187). Lecture notes in computer science. Cham: Springer International Publishing.
14. Ipate, F., Lefticaru, R., Mierlă, L., Valencia-Cabrera, L., Han, H., Zhang, G., Dragomir, C., Pérez-Jiménez, M., & Gheorghe, M. (2013). Kernel P systems: applications and implementations. In Proc. 8th int. conf. on bio-inspired computing: theories and applications, *Advances in intelligent systems and computing* (vol. 2012, pp. 1081–1089).
15. Juayong, R., Cabarle, F. G., Adorna, H. N., Martínez-del-Amor, M. A.. (2012). On the simulations of Evolution-Communication P systems with Energy without antiport rules for GPUs. In *10th Brainstorming Week on Membrane Computing, BWMC12*, Seville, Spain, February 2012, Proceedings (vol. I, pp. 267–290).
16. Kirk, D.B., & Hwu, W.W. (2016). *Programming massively parallel processors: a hands-on approach*, 3rd edn. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. https://www.sciencedirect.com/science/book/9780128119860.
17. Macías-Ramos, L. (2016). Developing efficient simulators for cell machines. PhD Thesis. Universidad de Sevilla. 2016. https://idus.us.es/handle/11441/36828.
18. Macías-Ramos, L. F., Martínez-del-Amor, M. A., & Pérez-Jiménez, M. J. (2015). Simulating FRSN P systems with real numbers in P-Lingua on sequential and CUDA platforms. In G. Rozenberg, A. Salomaa, J. M. Sempere, & C. Zandron (Eds.), *Membrane computing* (pp. 262–276). Cham: Springer International Publishing.
19. Maroosi, A., Muniyandi, R. C., Sundararajan, E., & Zin, A. M. (2014). Parallel and distributed computing models on a graphics processing unit to accelerate simulation of membrane systems. *Simulation Modelling Practice and Theory*, *47*, 60–78. https://doi.org/10.1016/j.simpat.2014.05.005.
20. Martínez-del-Amor, M., Orellana-Martín, D., Pérez-Hurtado, I., Valencia-Cabrera, L., Riscos-Núñez, A., & Pérez-Jiménez, M.J. (2019). Design of specific P systems simulators on GPUs. In: T. Hinze, G. Rozenberg, A. Salomaa, C. Zandron (Eds.),

Membrane computing (vol. 11399, pp. 202–207). Lecture notes in computer science. Springer International Publishing. https://doi.org/10.1007/978-3-030-12797-8_14.
21. Martínez-del-Amor, M.A. (2013). Accelerating membrane systems simulators using high performance computing with GPU. PhD Thesis. Universidad de Sevilla. 2013. https://idus.us.es/handle/11441/15644.
22. Martínez-del-Amor, M. A., García-Quismondo, M., Macías-Ramos, L. F., Valencia-Cabrera, L., Riscos-Núñez, A., & Pérez-Jiménez, M. J. (2015). Simulating P systems on GPU devices: a survey. *Fundamenta Informaticae*, *136*(3), 269–284. https://doi.org/10.3233/FI-2015-1157.
23. Martínez-del-Amor, M. A., Macías-Ramos, L. F., Valencia-Cabrera, L., & Pérez-Jiménez, M. J. (2015). Parallel simulation of population dynamics P systems: updates and roadmap. *Natural Computing*, *15*(4), 565–573. https://doi.org/10.1007/s11047-016-9566-1.
24. Martínez-del-Amor, M.A., Pérez-Carrasco, J., & Pérez-Jiménez, M.J. (2013). Characterizing the parallel simulation of P systems on the GPU. *International Journal of Unconventional Computing 9*(5-6), 405–424. https://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-9-number-5-6-2013/.
25. Martínez-del-Amor, M.A., Pérez-Hurtado, I., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Romero-Jiménez, Á., Graciani-Díaz, C., Riscos-Núñez, A., Colomer, M.A., & Pérez-Jiménez, M.J. (2012). DCBA: simulating population dynamics P systems with proportional object distribution. In 13th International conference on membrane computing (CMC13), pp. 291–310. http://www.sztaki.hu/tcs/proba/cmc13/CMC13-proceedings.pdf.
26. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Gastalver-Rubio, A., Elster, A.C., & Pérez-Jiménez, M.J. (2012). Population dynamics P systems on CUDA. In D. Gilbert, M. Heiner (Eds.) *Computational methods in systems biology* (vol. 7605, pp. 247–266). Lecture notes in computer science. Berlin: Springer. https://doi.org/10.1007/978-3-642-33636-2_15.
27. Martínez-del-Amor, M. A., Pérez-Hurtado, I., Orellana-Martín, D., & Pérez-Jiménez, M. J. (2020). Adaptative parallel simulators for bioinspired computing models. *Future Generation Computer Systems*, *107*, 469–484. https://doi.org/10.1016/j.future.2020.02.012.
28. Pérez-Hurtado, I., Martínez-del-Amor, M. A., Zhang, G., Neri, F., & Pérez-Jiménez, M. J. (2020). A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. *Integrated Computer-Aided Engineering*, *27*(2), 121–138. https://doi.org/10.3233/ICA-190616.
29. Pérez-Hurtado, I., Orellana-Martín, D., Zhang, G., & Pérez-Jiménez, M. J. (2019). P-Lingua in two steps: flexibility and efficiency. *Journal of Membrane Computing*, *1*(2), 93–102. https://doi.org/10.1007/s41965-019-00014-1.
30. Pérez-Hurtado, I., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Colomer, M.A., & Riscos-Núñez, A. (2010). MeCoSim: a general purpose software tool for simulating biological phenomena by means of P systems. In IEEE fifth international conference on bio-inpired computing: theories and applications (BIC-TA 2010), vol. I, pp. 637–643.
31. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M. Á., & Pérez-Jiménez, M. J. (2019). An interactive timeline of simulators in membrane computing. *Journal of Membrane Computing*, *1*(3), 209–222. https://doi.org/10.1007/s41965-019-00016-z.
32. Zhang, G., Pérez-Jiménez, M., & Gheorghe, M. (2017). *Real-life applications with membrane computing*. Berlin: Springer. https://doi.org/10.1007/978-3-319-55989-6.
33. Zhang, G., Shang, Z., Verlan, S., del Amor, M.M., Yuan, C., Valencia-Cabrera, L., & Pérez-Jiménez, M. (2020). An overview of hardware implementation of membrane computing models. *ACM Computing Surveys* (Accepted).
34. Zhang, X., Wang, B., Ding, Z., Tang, J., & He, J. (2014). Implementation of membrane algorithms on GPU. *Journal of Applied Mathematics*,. https://doi.org/10.1155/2014/307617.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Luis Valencia-Cabrera finished his BSc degree in Computer Engineering in 2005, Advanced studies diploma in 2011, MSc in Logics, Computing and Artificial Intelligence in 2013, and Ph.D. in 2015 (Ph.D. Extraordinary Award), in Universidad de Sevilla (Spain). He worked in IT Consulting from 2005 to 2010, obtained a Prof. MSc in Development of Enterprise Applications JEE in 2007, and official certifications by IFPUG (CFPS, 2006) and Oracle (SCJP, SCWCD, SCBCD, 2008). In 2017–2018, he worked in Natural Language Processing, getting 1st prize in category "General Corpus" in II Hackathon of Natural Language Technologies (Ministry of Economy and Enterprises of Spain, 2018). He is an Assistant Professor (certified by ANECA to Associate Professor) in the Dept. of Computer Science and AI (University of Sevilla, Spain). He joined the department in 2011 (from 2010, he belongs to Research Group on Natural Computing). He has supervised many final BSc/MSc final projects, and 2 PhD theses. His main research interests are modeling and simulation of complex systems, natural computing, membrane computing, theoretical computer science, and software engineering. His publications include 36 papers in ISI-JCR-indexed journals, plus other papers in journals, book chapters, invited talks and conference communications, totalling more than 100 scientific publications. He has been involved in more than 10 research projects at regional, national and international levels. He has participated in the organization of a dozen international conferences, and in the edition of the volumes of such conferences.

Ignacio Pérez-Hurtado received his M.S. (2003) in Computer Engineering at Univ. of Seville (Spain). After that, he was working for 4 years as analyst in a software company. He received his Ph.D. (2010) in Computer Science and Artificial Intelligence at Univ. of Seville (Spain). In his dissertation, carried out at the Research Group on Natural Computing (Univ. Seville), he developed the programming language for Membrane Computing P-Lingua. From 2011 to 2014, he worked as researcher in the Group on Natural Computing applying the P-Lingua framework to the simulation of real-life ecosystems. From 2014 to 2017, he received a "Juan de la Cierva" postdoctoral grant, working in the University Pablo de Olavide under several European projects related to social robotics. He is currently an interim lecturer at the University of Seville. His main

research interests are related to P system simulators, programming languages and algorithms for social navigation in robotics.

Miguel Á. Martínez-del-Amor received his Ph.D. (2013) in Computer Science and Artificial Intelligence at University of Seville (Spain), and his M.S. (2008) in Computer Engineering at University of Murcia (Spain). In his dissertation, carried out at the Research Group on Natural Computing (University of Seville), he developed parallel simulators for bio-inspired models of computation using GPUs. From 2014 to 2017, he worked first as an ERCIM fellow, and later as research associate, at the Moving Picture Technologies Department of Fraunhofer IIS (Germany) where he was involved in the standardization of JPEGXS format, parallelization of JPEG2000 codecs with GPUs, and Deep Learning applications in Digital Cinema. Since August 2017, he is an assistant professor at the University of Seville. His main research interest is on the interplay between parallel computing, bio-inspired computing and machine learning. In 2018, he was nominated university Ambassador by the NVIDIA Deep Learning Institute. He has co-authored 37 papers in scientific journals (29 in JCR-indexed journals), more than 52 contributions to conferences, and participated in 9 patents concerning image compression techniques.