SNUPS – a simulator for numerical membrane computing

Article *in* International journal of innovative computing, information & control: IJICIC · June 2011

CITATIONS
21

READS
338

3 authors, including:

Octavian Arsene
Oracle Corporation
33 PUBLICATIONS **372** CITATIONS
SEE PROFILE

Nirvana Popescu
Polytechnic University of Bucharest
99 PUBLICATIONS **702** CITATIONS
SEE PROFILE

International Journal of Innovative
Computing, Information and Control
Volume **7**, Number **6**, June **2011**

 ISSN 1349-4198
pp. **3509–3522**

SNUPS – A SIMULATOR FOR NUMERICAL MEMBRANE COMPUTING

Octavian Arsene[1], Catalin Buiu[1] and Nirvana Popescu[1,2]

[1]Laboratory of Natural Computing and Robotics
[2]Department of Computer Science
"Politehnica" University of Bucharest
Spl. Independentei, No. 313, Bucharest 060042, Romania
{ oarsene; cbuiu }@ics.pub.ro; nirvana.popescu@cs.pub.ro

Received March 2010; revised July 2010

Abstract. *Various phenomena and processes that take place in nature and in living entities have been taken as inspiration sources for developing innovative computing methodologies. P systems represent a branch of natural computing in which the membrane structure and functioning of a biological cell are imitated to provide a powerful computational model. There are numerous theoretical studies on different types of P systems which have been shown to be computationally universal. Various application areas have been successfully proposed, ranging from modeling of biological and biochemical processes, to economics and cryptography. However, there are very few simulators of P systems available and fully documented in the public domain and this fact limits the scope of possible new applications. No numerical P systems simulators have been designed and made available so far and it is the purpose of this paper to present the structure and functionalities of SNUPS, the first and only one simulator of numerical P systems available.*
Keywords: Natural computing, Biomembranes, Numerical P systems, Simulation, Java

1. **Introduction.** Natural computing is an emerging research area in which novel computing paradigms and problem-solving techniques are proposed and implemented by taking inspiration from various natural phenomena and information-processing structures. Natural computing includes among other techniques, evolutionary algorithms, neural networks, molecular computing and quantum computing. Natural computing also refers to those applications in which computer hardware and software are used to replicate or synthesize natural processes and also to the use of natural materials to compute.

Particle Swarm Optimization (PSO) and respectively Ant Colony Optimization (ACO) are two prominent swarm intelligence based problem solving methods and have been used with good results in a variety of application areas. For example, PSO algorithms were used for designing DNA sequence sets [1]. DNA computing (or molecular computing, in general) is a form of natural computing which is based on using DNA, biochemistry and molecular biology, and its power comes from the fact that it has great potential of massive data storage and processing computation over data in parallel [2]. In [3], it is proposed a new biological molecular model using the Adleman-Lipton model. In [4], a new forecasting methodology inspired by natural selection and combining mathematical, computational and biological sciences, which includes fuzzy logic, DNA encoding, polymerase chain reaction and DNA quantification was proposed.

It is considered that the first cell appeared when a membrane formed. Biomembranes regulate the traffic across the boundary between the cell and the rest of the world and divide the internal space of the cell into discrete compartments to segregate processes and components [5]. More, the boundaries are central to energy conservation and cell-to-cell

communication. Membrane computing is an important area of natural computing and is based on using P systems (PS) which were introduced and developed in the context of formal language theory by the Romanian mathematician G. Paun [6]. A PS is inspired by biological cells but do not aim to fully emulate the working of a real cell which of course is not completely understood, being instead based on simple properties of real membranes and information processing as appears to take place inside living cells.

A PS represents a distributed and parallel computing model in which basic data structures are multisets, strings or numerical variables. There are two main categories of PSs: hierarchical and tissue PSs [5]. A hierarchical PS consists of several membranes placed inside a unique skin membrane, see Figure 1.

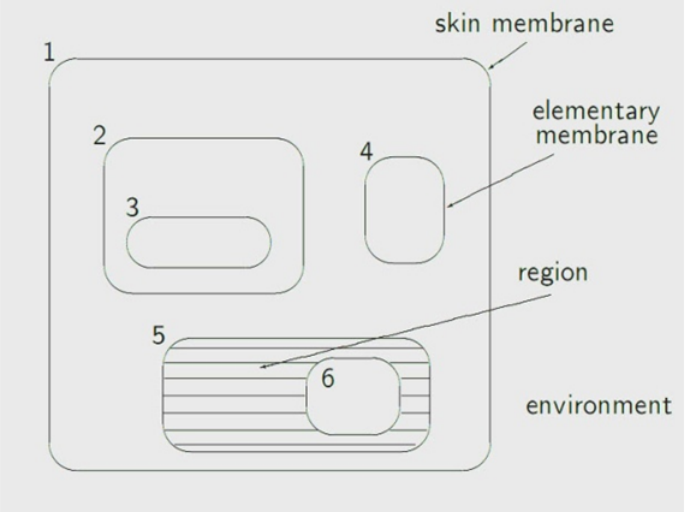


FIGURE 1. Membrane structure in a hierarchical P system

Objects symbolize the molecular species inside a real cell (ions, proteins, etc.) and are usually described by using symbols or strings over a given alphabet [7]. The objects can pass through membranes, which can have a variable permeability, dissolve or divide themselves. Chemical reactions that might take place inside a real cell are symbolized by the evolution rules which are applied in a nondeterministic and maximum parallel way [5]. However, this is not a general rule, as there exist deterministic PSs. The outputs of a PS are given by all possible halting configurations which can be obtained from an initial state. In tissue PSs [8], all the membranes are placed in the nodes of a graph, all of them at the same level.

PSs with active membranes are a subclass of the first category for which powerful results have been proven (ability to solve NP-complete problems in polynomial or even linear time). The same result has been demonstrated for a tissue PS [9]. There are also other PS categories such as evolution communication PSs [10], which were used to simulate some processes that are taking place in living cells, such as respiration and respiration photosynthesis interaction. Timed PS and time-free PS have been introduced in [11]. Metabolic P Systems (MPS) have a discrete deterministic evolution strategy in which the transition to the next state is calculated according to a mass partition strategies [12].

Most of the existing variants of PS are computationally universal, see for example [7]. Up to now, there are no actual implementations of PS (in vitro, in vivo or in silico). Application areas include biology, computer graphics, sorting and ranking, cryptography, linguistics, optimization, economics, process control and robotics.

Claiming an inspiration from economic and business processes, numerical P systems (NPS) have been proposed in which numerical variables evolve from initial values by means of production functions and repartition protocols [13]. Non-deterministic NPSs with polynomial production functions, characterize the Turing computable sets of natural numbers, while deterministic NPSs, with polynomial production functions having non-negative coefficients, compute strictly more than semilinear sets of natural numbers as demonstrated in [13].

Several general membrane computing simulators have been developed and reported in the literature but only a few are available in the public domain in a functional and documented version. There has been used a variety of languages and architectures, such as CLIPS [14], Visual C++ [15], Prolog, C++. Java has been widely used, see SimCM [16] and PSim [17]. P-Lingua was introduced as an easy-to-learn programming language to define P systems in a modular and parametric way. Its syntax is very close to standard scientific notation and parameterized expressions can be used to handle families of P systems [18].

Table 1 summarizes some of the existing simulators including SNUPS, which will be presented in this paper. SNUPS is the first and only one available simulator for the numerical P system class and is delivered as a *ready-to-use* application instead of a framework or language. This uniqueness is the main contribution this simulator brings to the natural computing community.

TABLE 1. Existing P systems simulators

Name	Description	Programming language
PSystemsMF	P System Modeling Framework, simulates the evolution of Multi-compartmental Gillespie algorithm over a hierarchy of compartment structures	Scilab, C
P-Lingua	Programming language for membrane computing	Java
PSim	Simulator for P Systems based on the Metabolic Algorithm	Java
Cyto-Sim	Stochastic stimulator of biochemical processes in hierarchical compartments	J#
SNPS	Spiking Neural P Systems Simulator	C++
SNUPS	Simulator for Numerical P Systems	Java

The existence of SNUPS makes possible new applications of membrane computing in real-world problems where numerical variables are involved, such as process control or mobile robotics.

The next section will present the detailed functioning of a numerical P system, while Section 3 will describe the SNUPS software engineering implementation. Section 4 will present SNUPS use cases and Section 5 describes a concrete example implemented in SNUPS. The last section will give some conclusions and directions for further improvements and researches.

2. **Numerical P Systems – An Overview.**

An NPS uses numerical values in the compartments of the simulated cell. For introduction of computation processes which occur in NPS, in Figure 2, it is shown a simple NPS membrane structure, which is in fact a cell-like two-dimensional hierarchical arrangement of membrane structures.

These variables have initial values (floating or integer), e.g., membrane 1 has only one symbol, variable x_{11}, which has 1 as initial value (which is specified in the brackets) (see Figure 2).

There are functions (production functions) associated with each membrane (e.g., $2x_{11}^2$) in membrane 1, see Figure 2. The membrane's functions take the local variables and compute a number (production). Related to the output of these functions each membrane has a contribution list also (e.g., contribution list of the membrane 1 is: x_{11} and x_{12} , see Figure 2). The output of the function is distributed among the variables from the contribution list (like the commands *here*, *out*, *in* from the standard PS).

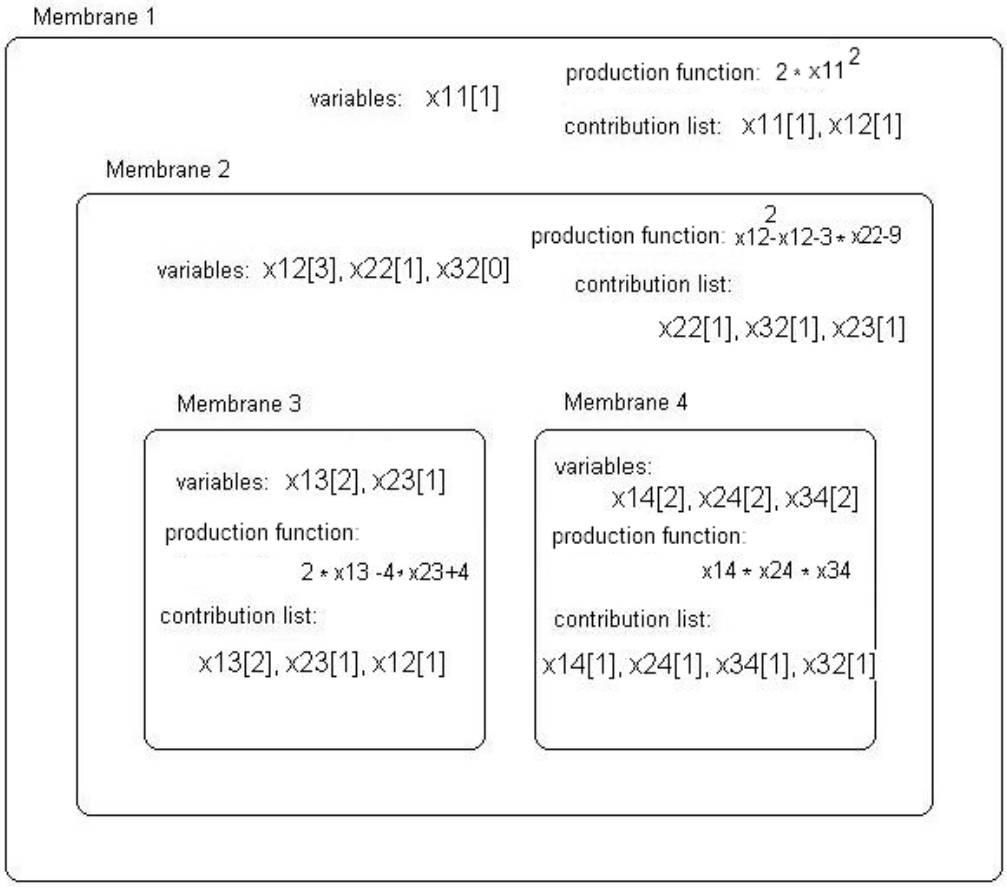


FIGURE 2. A sample membrane structure with numerical variables (with initial values), production functions, and repartition protocols

In each membrane, one function is used at a time, and this happens in parallel in all membranes. In this way, the values of the variables are changed. The process continues iteratively, thus making the system evolving in time.

The repartition procedure is an expression of form: $c_1|v_1 + c_2|v_2 + \ldots + c_n|v_n$, where $c_1, \ldots, c_n$ are natural numbers (they can be also zero) which specify the proportion of the current production distributed to each variable $v_1, \ldots, v_n$ [13]. At time t, the value:

$$q = production / \sum_{i=1}^{n} c_i \tag{1}$$

represents the unitary portion to be distributed to variables $v_1, \ldots, v_n$ according to the coefficients $c_1, \ldots, c_n$, in order to obtain the values of these variables at time $t + 1$ [13]. The variable v_i will receive $q * c_i$ as a new numerical value. If a variable receives such

contributions from several compartments, they are added in order to produce the next value of the variable.

For the purpose of the current paper, it will be considered a *deterministic* PS, having only one production function in each membrane's region. Section 5 will return to this example giving a detailed overview of the respective SNUPS simulation.

3. **SNUPS Implementation.** SNUPS is available for download and testing at http://snups.ics.pub.ro. The implementation has been done using Java Platform, Standard Edition (Java SE). The following reasons were taken into account for choosing Java as a development platform:

1) concurrency higher-level components and low-level mechanisms implemented as core packages in Java platform;
2) soft integration with advanced standards within software industry: enterprise class service-oriented architecture (SOA) and next-generation web applications – Java Platform, Enterprise Edition (Java EE)-, rich internet applications (RIAs) – JavaFX-, mission-critical real-time applications – Java Real-Time System (Java RTS);
3) diversity of devices imposes creation of the applications which should be able to run on mobile and other embedded devices-mobile phones, personal digital assistants (PDAs), TV set-top boxes and printers. Java Platform, MicroEdition (Java ME) provide a robust, exible environment for applications running on mobile and other embedded devices;
4) portability on different operating systems;
5) on-going projects aim for developing of a cognitive architecture for robots using Java Development Framework (JADE) [19] as an agent-oriented middleware and NPSs as computational engines.

The basic concepts of a PS were wrapped into corresponding classes, which are presented in Figure 3. The skin-membrane and elementary membrane (see Figure 1) has been formalized using the Membrane class. It has two components: region and membranes. A membrane can have 0 or at least 1 membrane child (see Figures 1 and 3). Each membrane-object (the instantiation of the Membrane class) has the following attributes: a name, one region-object and a membrane-object children list (see Figure 3).

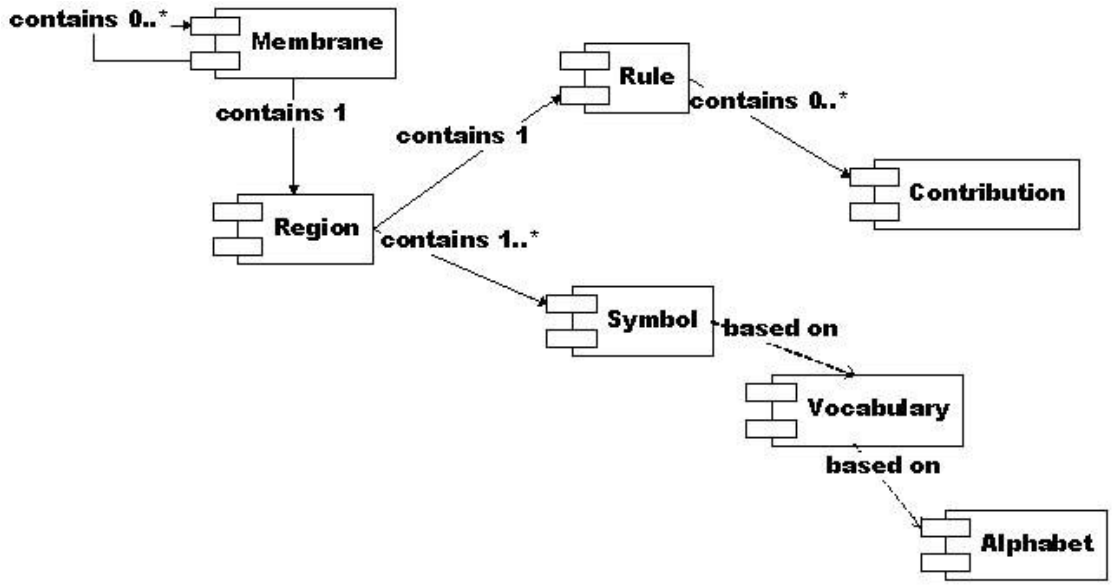


FIGURE 3. Class diagram in SNUPS

The membrane's region has been formalized using the *Region* class. It represents the container for symbols and rules (see Figures 3 and 5). Each region object (the instantiation of the Region class) has the following attributes: a name, a symbol-object list and a rule-object list. The current version is using only one rule for computation.

The symbol of the region concept has been formalized using the *Symbol* class. The symbol-object (the instantiation of the Symbol class) can have one of the following three roles: (1) region's symbol, (2) contribution list symbol and (3) rule's symbol. The contribution list and rule concepts are explained in the following paragraphs. Each symbol-object has the following attributes: a name, an initial value, a current value and a is available boolean value. Both initial and current values are floating numbers. Initial value of the symbol is used for starting computation. After each micro-step (the rule's computation has finished) current value of few symbols can be changed (depends on the output of the rule's and the corresponding contribution list). In case, we want to restart the computation, all current values are reset and the initial values are preserved (unless we changed some of them). After a symbol is assigned to a region it becomes unavailable for the rest of the regions; the is-available attribute is responsible for this purpose. For example, in Figure 4, only symbol x_{11} belongs to region R-1, the rest of the symbols: x_{12}, x_{22} are unavailable because they belong to the regions R-2, R-3 and R-4 (these are gray colored).

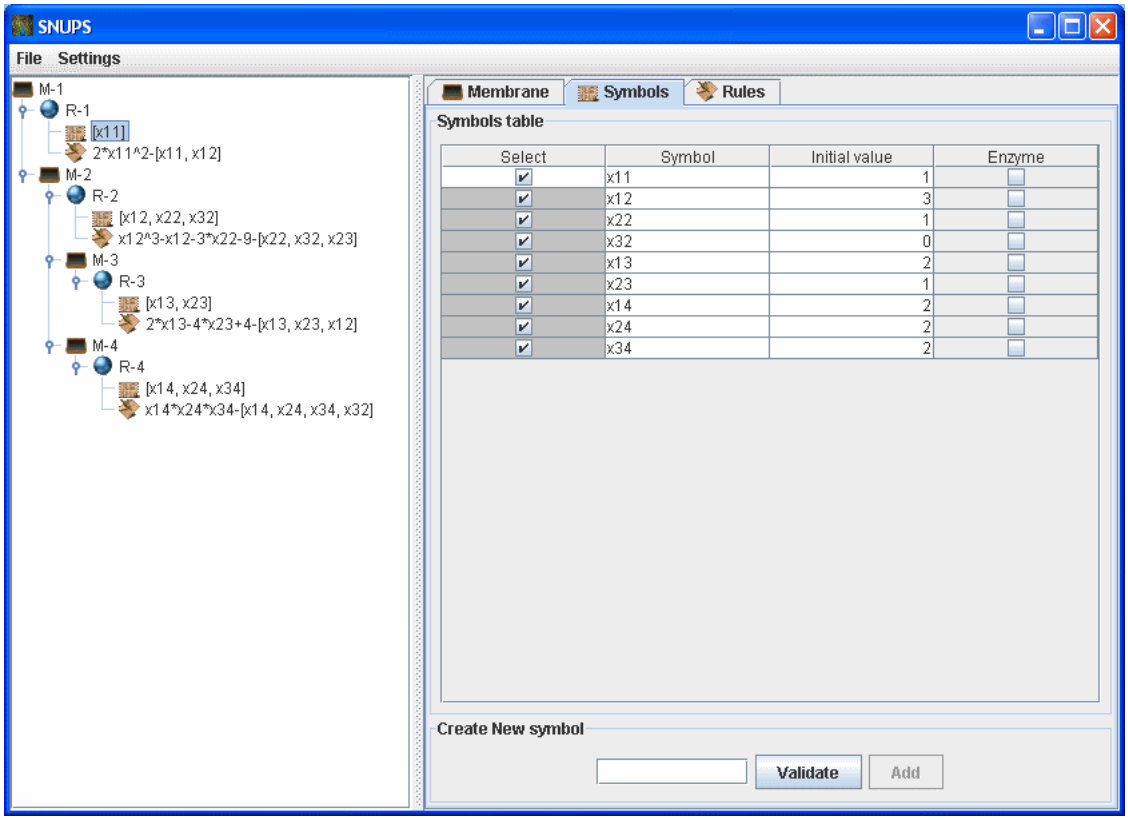


FIGURE 4. Symbols panel

As it was already mentioned, each computation step is based on some mathematical functions (production functions). Their numerical output will be distributed among symbols (they belong to the current region or to the other regions), in order to modify their current values. This distribution process can be seen as many contributions to the some symbols from the regions, the quantitative aspect of the contributions depends on unitary

portion value (see Formula (1)). This is computed based on the values defined in the *Contributions table* (see the *Value* column from Figure 5).

The contribution concept has been formalized using *Contribution* class. The contribution-object (the instantiation of the Contribution class) has the following attributes: a symbol-object and a contribution value (distribution coeficient). A contribution-object should be defined in order to modify the current value of a symbol-object. The contribution value attribute holds the numerical value from the Contributions table region, see right-hand side from Figure 5.

The business rule or mathematical function that computes, has been formalized using the *Rule* class. The rule object has the following main attributes: a production function and a contribution-object list. Contribution-object list can be modified: add new symbols, remove symbols, change symbols (name and/or values) see the *Rules* tab from Figure 5.

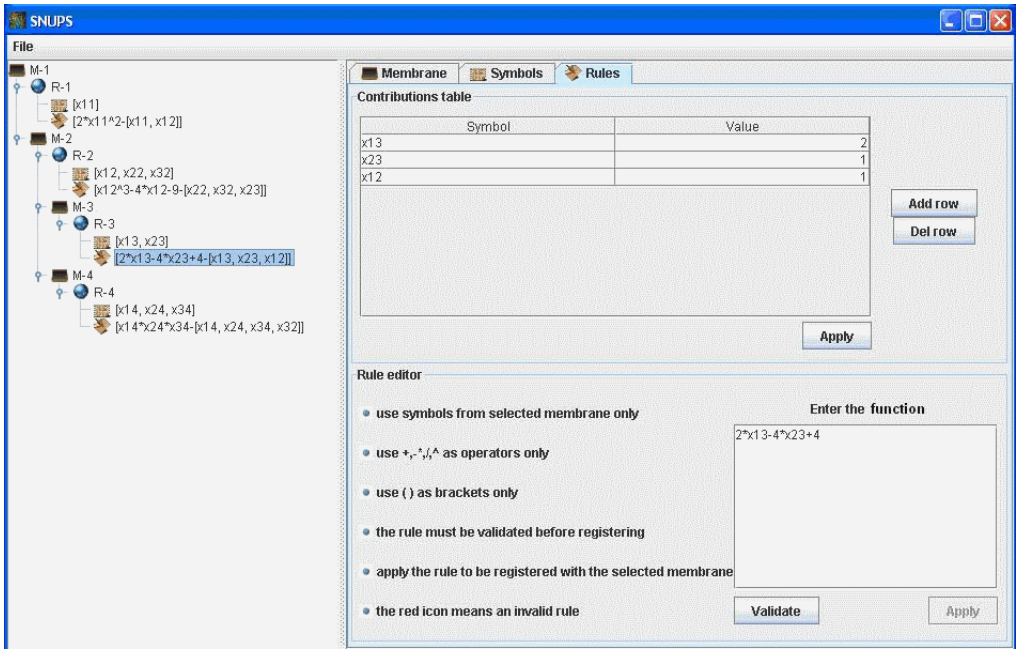


FIGURE 5. Rules panel

The *Rule editor* panel from *Rules* tab, allows editing the function of the rule and its syntax validation. The following math operations are allowed in the rule editor:

- addition (+ sign will be used);
- multiplication (∗ sign will be used);
- subtraction (− sign will be used);
- division (/ sign will be used);
- powers (^ sign will be used).

There are allowed brackets – ( ) signs – for a specific order of mathematical operations. These are some examples for how equations can be written: $2 * x + x^2 * (20 - x)$, $4 * x * (x - (2 - x/2))$; these can be entered and validated inside of the *Enter the function* text area field from right-down corner of Figure 5. If all the symbols used in rule's function belong to the selected region's symbol list (see left hand side tree panel from Figure 5) then this equation is a valid one and it can be assigned to the selected region. Even in the case of an invalid rule (the icon's rule is red), it's still saved, but the computation won't start until the rule is validated. As it was mentioned, this simulator implements the algorithm from [13], where all the mathematical details are given.

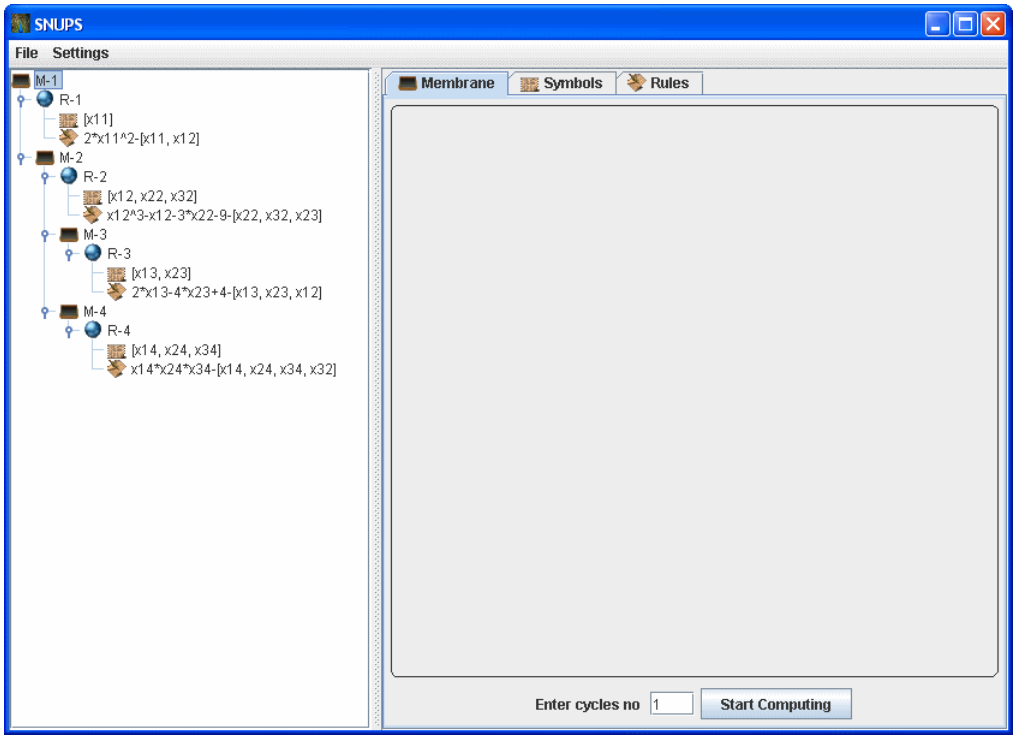


FIGURE 6. Membrane panel

From graphical user interface (GUI) point of view, the loaded/created PS with SNUPS is a tree based on four icons symbols:

- *box*, the membrane;
- *globe*, the region;
- *papyrus*, the symbol list of the region;
- *writing hand*, the region's rule.

The displayed values from the *Membrane, Symbols, Rules* tabs depend on the tree selection value.

The alphabet (formalized as the *Alphabet* class) represents the list of the letters and numbers, which are allowed to be part of the word list within the symbol's vocabulary (formalized as the *Vocabulary* class). Each time we want to add new symbols to the initial list, their letters and numbers' name will be validated against vocabulary and alphabet instantiated objects.

The core classes are organized in a single software package *psys.core*, as follows:

- *Alphabet*, used for vocabulary;
- *Vocabulary*, used for symbol names;
- *Symbol*;
- *Rule*;
- *Contribution*;
- *Region*;
- *Membrane*.

There are two additional software packages: *psys.ui*, used for GUI, and *psys.xml*, used for XML input/output interface (e.g., working with XML documents). From software engineering point of view were used *Observable* and *Model-View-Controller* (MVC) design patterns in order to have a clean code and a smooth development process (e.g., the adoption of the future changes and the integration with new software components).

4. **SNUPS Use Cases.** The simulator's use cases are the following (see Figure 7):

FIGURE 7. SNUPS use cases

1) Create Membrane: user creates a new membrane structure from scratch using the visual tree panel.

2) Import Membrane: user imports a membrane structure from an XML file. The input XML file structure (see Figure 8) is defined based on a schema that is compliant with the membrane structure previously described. The hierarchical structure of the XML file (see Figure 8) has the same meaning as the above tree structure that was explained earlier (see Figure 6). The "⟨membrane⟩" tag defines the *Membrane* node from GUI tree; the *membrName* attribute of the this tag represents the membrane index within the tree.

The "⟨region⟩" tag contains all necessary information for a region: symbol list, symbol values, contribution values and rule's function. Each region's tag should have a symbol list ("⟨symbolsList⟩" tag) and a rule defined using "⟨rule⟩" tag. The *initialValue* attribute of the symbol tag –"⟨symbol⟩"– is the numerical value of a symbol when the application is started, this can be change later from the GUI and can be saved as well (see Figure

ExamplePaunSNUPS.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <environment xmlns="http://www.example.org">
3     <membrane membrName="1">
4        <region>
5           <symbolsList>
6              <symbol initialValue="1">x11</symbol>
7           </symbolsList>
8           <rulesList>
9              <rule>
10                <contrList>
11                   <contribution contrValue="1">x11</contribution>
12                   <contribution contrValue="1">x12</contribution>
13                </contrList>
14                <ruleEquation>2*x11^2</ruleEquation>
15             </rule>
16          </rulesList>
17       </region>
18       <children>
19          <membrane membrName="2">
20             <region> <symbolsLis ... rulesList> </region>
37             <children>
38                <membrane membrName= ... ildren/> </membrane>
57                <membrane membrName= ... ildren/> </membrane>
78             </children>
79          </membrane>
80       </children>
81    </membrane>
82    <allSymbols>
83       <symbol initialValue="1">x11</symbol>
84       <symbol initialValue="3">x12</symbol>
85       <symbol initialValue="1">x22</symbol>
86       <symbol initialValue="0">x32</symbol>
87       <symbol initialValue="2">x13</symbol>
88       <symbol initialValue="1">x23</symbol>
89       <symbol initialValue="2">x14</symbol>
90       <symbol initialValue="2">x24</symbol>
91       <symbol initialValue="2">x34</symbol>
92    </allSymbols>
93 </environment>
```

FIGURE 8. Sample input XML file used for an already defined membrane structure

5). The "⟨contribution⟩" tag has *contrValue* as an attribute; this is the initial numerical value for that contribution's rule an application starts with (can be changed and saved later from GUI). There is a symbols list "⟨allSymbols⟩"– in the bottom of the file also; this is the initial list that can be used later on in the application (changed and saved, see Figure 4).

3) Export Membrane: after a new membrane has been defined or has been updated, the user saves (as an XML file) this structure.

4) Delete Membrane: user deletes a membrane node (the region and the membrane children are deleted also).

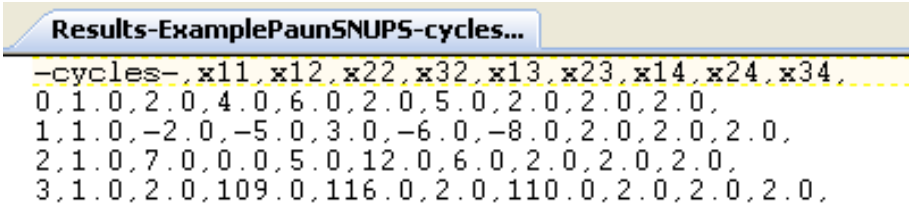
5) Edit Symbol List: user modifies the symbol list of the selected membrane; user can remove a symbol, change the initial value of the symbol and add a new symbol.

6) Edit Contribution List: user modifies the contribution list of the selected rule; one can remove a contribution, add a new contribution, change the symbol of the contribution and change the contribution value.

7) Add New Symbol: user adds a new symbol in the general list (after validation against alphabet and vocabulary); this list is the provider for the each region's symbol and contribution list. The initial value of the new symbol is set as well.

8) Edit Rule: user edits or adds a new rule (after the validation against the region's symbols list).

9) Start Computing: user launches the computation after the all settings are completed within the desired membrane structure and after a desired number of iterations is chosen. The results are persisted into a comma-separated values log file (see Figure 9).


Results-ExamplePaunSNUPS-cycles...

```
-cycles-,x11,x12,x22,x32,x13,x23,x14,x24,x34,
0,1.0,2.0,4.0,6.0,2.0,5.0,2.0,2.0,2.0,
1,1.0,-2.0,-5.0,3.0,-6.0,-8.0,2.0,2.0,2.0,
2,1.0,7.0,0.0,5.0,12.0,6.0,2.0,2.0,2.0,
3,1.0,2.0,109.0,116.0,2.0,110.0,2.0,2.0,2.0,
```

FIGURE 9. SNUPS log file

5. **A Detailed SNUPS Example.** In order to demonstrate SNUPS functionality, an NPS structure considered in [13] is implemented below. The example presented in [13] illustrates the transitions occured in a deterministic NPS. The behaviour shows that only the compartments 1 and 4 will always keep the same values for their variables. A more complex example is presented in [20].

Let us consider a four membranes NPS with the following structure and attributes (see Figures 2 and 10):

- membrane M-1 contains membrane M-2; region R-1 has one symbol: x_{11} having the initial value = 1;
- membrane M-2 contains membranes M-3 and M-4; region R-2 has three symbols: x_{12}; x_{22}; x_{32} having the initial values: 3, 1 and 0;
- membrane M-3 doesn't have children membranes; region R-3 has two symbols: x_{13}; x_{23} having initial values: 2 and 1;
- membrane M-4 doesn't have children membranes; region R-4 has three symbols: x_{14}; x_{24}; x_{34}, having initial values: 2, 2 and 2.

The functions' rules and contribution tables of the above four membranes are the following (see Figures 2 and 11):

- region R-1 has production function: $2 * x_{11}^2$ and the contributions will go to symbols: x_{11}, x_{12};
- region R-2 has production function: $x_{12}^3 - x_{12} - 3 * x_{22} - 9$ and the contributions will go to symbols: x_{22}, x_{32}, x_{23};
- region R-3 has production function: $2 * x_{13} - 4 * x_{23} + 4$ and the contributions will go to symbols: x_{13}, x_{23}, x_{12};
- region R-4 has production function: $x_{14} * x_{24} * x_{34}$ and the contributions will go to symbols: x_{14}, x_{24}, x_{34}, x_{32};

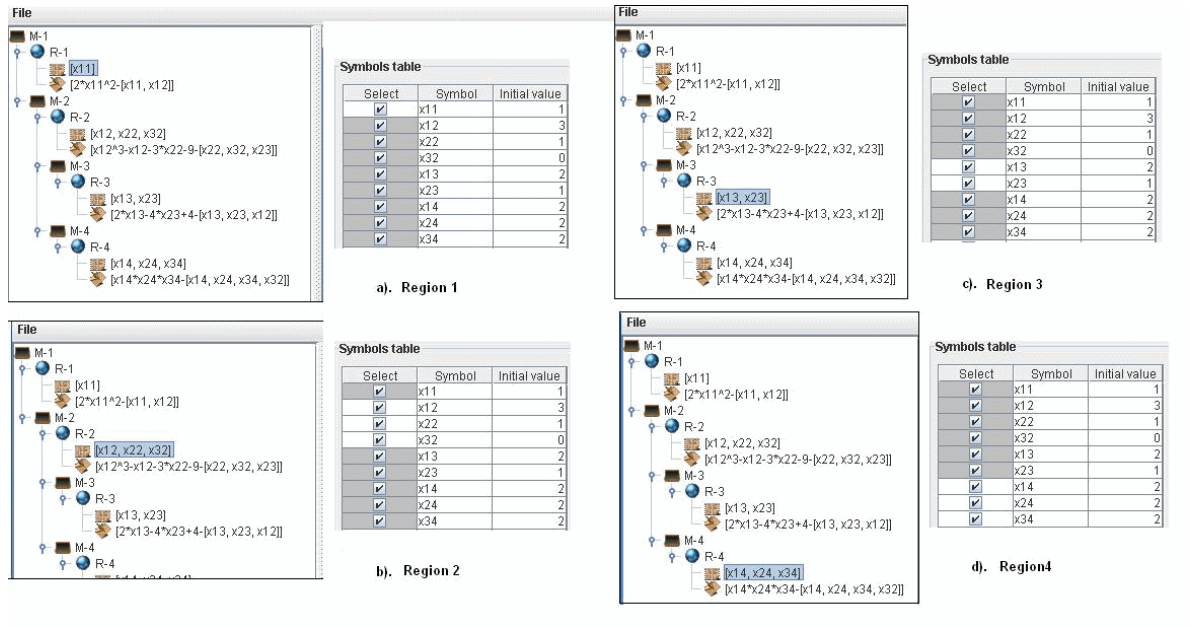


FIGURE 10. Distribution of the symbols in regions

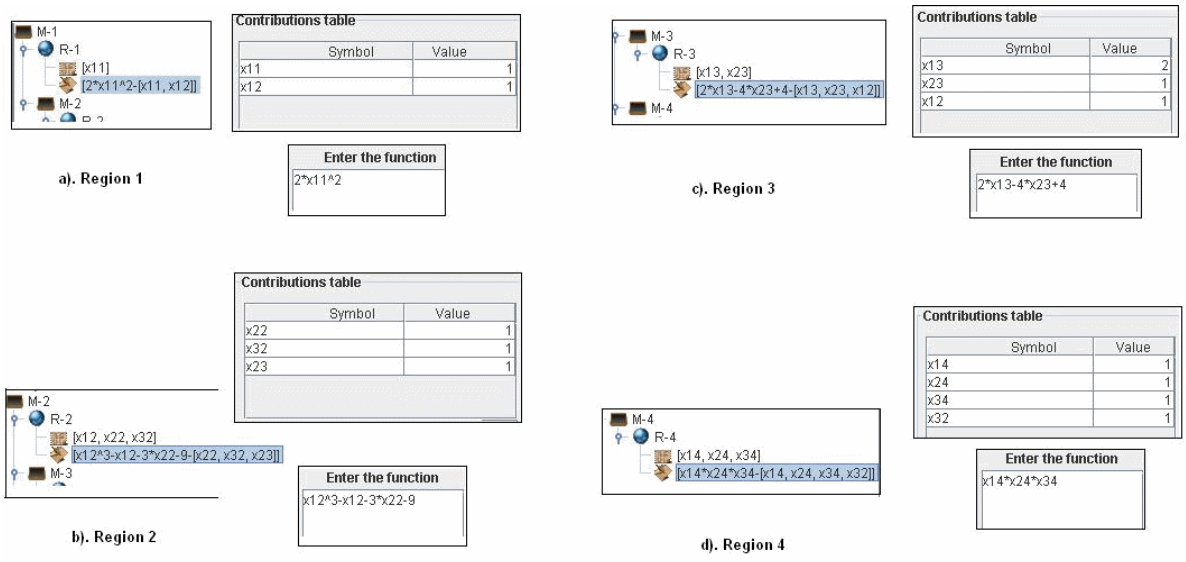


FIGURE 11. Rules definition for regions

We'll consider four computation cycles (four transitions) for our NPS, which is a deterministic one, that is, having only one production function in each region (hence, no branching is possible during evolution). The first transition steps are detailed below.

Step 1: The system evolves (the symbol values are changed) by simultaneously running one function rule in each region; e.g., in region R-1, the function will compute: $2 * 1^2 = 2$. The unitary portion (see the Formula (1)) is $q = 2/(contributionvaluefor\_x_{11} + contributionvaluefor\_x_{12}) = 2/2 = 1$.

The contribution values will be: for $x_{11} = q * contributionvaluefor\_x_{11} = 1 * 1 = 1$ and for $x_{12} = 1 * 1 = 1$. The latter variable also receives one unit from region R-3 ($q = 4/4 = 1$); after considering both repartitions, $x_{12} = 2$.

Step 2: The current values of the symbols are changed according to the received contributions (see Figure 12, the row of cycle 0).

-cycles-	x11	x12	x22	x32	x13	x23	x14	x24	x34
0	1	2	4	6	2	5	2	2	2
1	1	-2	-5	3	-6	-8	2	2	2
2	1	7	0	5	12	6	2	2	2
3	1	2	109	116	2	110	2	2	2

FIGURE 12. Results after four computation cycles

This was first cycle (transition). The both steps are repeated for the other three transitions. During the Step 1 the function will use the new symbol values computed in the previous cycle (e.g., cycle 1 will use as initial values for variables the new ones from cycle 0).

The number of transitions is specified before starting computation (see *Enter cycles* text field from *Membrane* tab in Figure 6). The update of a variable from multiple membranes in the same time is not an issue due to the synchronization access to the variables. The same results as in [13] are obtained using SNUPS, see Figure 12, proving the validity of the proposed software.

6. **Conclusions and Future Work.** The present paper introduced a Simulator for Numerical P Systems (membrane computing) – SNUPS – as a computation engine inspired from the structure and functional mechanism of the living cells, but allowing the direct use of numerical variables. SNUPS is the first (and only one until now) publicly available simulator that allows defining and simulating the structure and functioning of a numerical P system. SNUPS runs on all platforms without any compiling or building prerequisite steps. The GUIs presented in the example from Section 5, Figures 10 and 11, allow people to interact with SNUPS in other ways than typing, the structure can be edited in an easier manner, and the usability is increased leading to a higher usage rate of the software. Even in a particular case when the NPS structure is in an XML format (see Figure 8) being generated automatically by a third party software, it can be easily *consumed* (run, changed, saved) by SNUPS.

New improvements that are being investigated include (1) the integration with SBML (Systems Biology Markup Language) tools, (2) integration with software agent concept (creation of a membrane agent), (3) implementing of a multiple rules per membrane behaviour in SNUPS using enzyme-like variables. It is hoped that SNUPS will allow the development, testing and validation of new applications of P systems.

Acknowledgment. This work was supported by CNCSIS – UEFISCSU, project number PNII – IDEI 1692/2008.

REFERENCES

[1] Q. Zhang, R. Zhang and B. Wang, DNA sequence sets design by particle swarm optimization algorithm, *International Journal of Innovative Computing, Information and Control*, vol.5, no.8, pp.2249-2255, 2009.

[2] C.-N. Yang, K.-S. Huang, C.-B. Yang and C.-Y. Hsu, Error – Tolerant minimum finding with DNA computing, *International Journal of Innovative Computing, Information and Control*, vol.5, no.10(A), pp.3045-3057, 2009.

[3] M. S. H. Ho, J. Lin and C. Chiu, Constructing bio-logical molecular parallel adder and multiplier using the Adleman-Lipton model, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(A), pp.4573-4582, 2009.

[4] D. J. F. Jeng, J. Watada and B. Wu, Biologically inspired fuzzy forecasting: A new forecasting methodology, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(B), pp.4835-4844, 2009.
[5] A. Lehninger, D. L. Nelson and M. M. Cox, *Lehninger Principles of Biochemistry*, W. H. Freeman, 2008.
[6] G. Paun, Computing with membranes, *J. Comput. System Sci.*, vol.61, pp.108-143, 2000.
[7] G. Paun and G. Rozenberg, A guide to membrane computing, *Theoretical Computer Science*, vol.287, no.1, pp.73-100, 2002.
[8] C. Martin-Vide, G. Paun, J. Pazos and A. Rodriguez-Patn, A guide to membrane computing, *Theoretical Computer Science*, vol.296, no.2, pp.295-326, 2003.
[9] G. Paun, M. J. Prez-Jimnez and A. Riscos-Nez, Tissue P systems with cell division, *Int. J. of Computers, Communications and Control*, vol.3, no.3, pp.295-303, 2008.
[10] M. Cavaliere, A evolution-communication P systems source, *Lecture Notes in Computer Science*, vol.2597, pp.134-145, 2002.
[11] M. Cavaliere, Toward asynchronous P systems, *Proc. of the 5th Workshop on Membrane Computing*, pp.161-173, 2004.
[12] V. Manca, The metabolic algorithm for P systems: Principles and applications, *Theoretical Computer Science*, vol.404, pp.1-2, 2008.
[13] G. Paun and A. Paun, Membrane computing and economics: Numerical P systems, *Fundamenta Informaticae*, pp.213-227, 2006.
[14] M. J. Perez-Jimenez and F. J. Romero-Campero, A CLIPS simulator for recognizer P systems with active membranes, *Proc. of the 2nd Brainstorming Week on Membrane Computing*, pp.387-413, 2004.
[15] G. Ciobanu and D. Paraschiv, P system software simulator, *Fundamenta Informaticae*, vol.49, pp.61-66, 2002.
[16] I. A. Nepomuceno-Chamorro, A Java simulator for membrane computing, *Journal of Universal Computer Science*, vol.10, no.5, pp.620-629, 2004.
[17] L. Bianco, V. Manca, L. Marchetti and M. Petterlini, Psim: A simulator for biomolecular dynamics based on P systems, *Proc. of the 5th IEEE Congress on Evolutionary Computation*, pp.883-887, 2007.
[18] A. Riscos-Nez, D. Daz-Pernil, M. J. Prez-Jimnez and I. Prez-Hurtado, A P-lingua programming environment for membrane computing, *Proc. of the 9th Workshop on Membrane Computing*, U.K., pp.155-172, 2008.
[19] F. Belifemine, G. Caire and D. Greenwood, *Developing Multiagent Systems with JADE*, Wiley, 2008.
[20] C. Buiu, O. Arsene, C. Cipu and M. Patrascu, A software tool for modeling and simulation of numerical P systems, *BioSystems*, vol.103, no.3, pp.442-447, 2011.