SIMULATING MEMBRANE SYSTEMS IN DIGITAL COMPUTERS[1]

Fernando Arroyo, Carmen Luengo, Luis Fernandez,
Luis F. de Mingo, and Juan Castellanos

Abstract: *Membrane Computing started with the analogy between some processes produced inside the complex structure of living cells and computational processes. In the same way that in other branches of Natural Computing, the model is extracted from nature but it is not clear whether or not the model must come back to nature to be implemented. As in other cases in Natural Computing: Artificial Neural Networks, Genetic Algorithms, etc; the models have been implemented in digital computers. Hence, some papers have been published considering implementation of Membrane Computing in digital computers. This paper introduces an overview in the field of simulation in Membrane Computing.*

Keywords: Simulation, Membrane Computing, Multiset, Evolution Rules, Membrane Structure.

Introduction

Membrane Computing is inspired in the structure and functioning of living cells. It was considered in October 2003 by Thomson Institute for Scientific Information as a fast emerging research area. This research area was initiated at the end of 1998 by Gh. Paun (in a paper which was published in 2000 in the Journal of Computer and System Sciences) [2]. Nowadays, Membrane computing is one of the most popular research topics in the European community of cellular computing.

The membrane systems, also named P systems, are a class of distributed parallel computing devices of a biochemical type, which can be seen as a general computing architecture where different types of objects can be processed by different operations. The basic model consists of a membrane structure (several membranes hierarchically embedded in a main membrane *skin*). Membranes define regions where different objects are placed. These objects are processed according to given evolution rules, which are associated with the regions. When a rule is applied in a region, the objects present in the region are modified, some of them are sent out or in it. The evolution rules can also dissolve the membrane. In that case, all the objects present in the membrane remain free in the membrane that includes the dissolved one; however, rules associated to the dissolved membrane are removed. The *skin* membrane never is dissolved because then the system can not be considered a system anymore. As can be seen, the system is governed by evolution rules, membranes are considered as separators and communication channels. The application of evolution rules is made in a nondeterministic and maximally parallel manner; at each step, all objects which can evolve must evolve in every region of the system.

This kind of systems compute by passing from a configuration to another configuration by applying evolution rules in the way described above. A computation is considered complete when it halts, e.g. when no further rules can be applied to the objects present in the last configuration. The result of a halting computation can be made in two different ways: by considering the multiplicity of objects presents in the halting configuration inside a determined region, or by concatenating the symbols which are sent out of the system considering the order in which they were sent out. In the first case, vectors of natural numbers are computed while in the second case, languages are generated.

Many variants of P systems have been considered [1]. In some of them, the number of membranes can only decrease by dissolving membranes as result of applying evolution rules. However, many of them the number of membranes can be increased using some biological features of living cells, for example: by division. Some other variants consider membranes not only passive objects of the system, these kind of systems are based in biological processes performed by membranes when chemical compounds pass through the membrane (*protein gates* or *protein channels*).

[1] Work partially supported by contribution of EU commission Under The Fifth Framework Programme, project "MolCoNet" IST-2001-32008.

Membrane Systems with Symbol-Objects: The Simplest Class

P systems with symbol-objects have been presented as a class of non-deterministic parallel computing devices whose their main ingredients are: a membrane structure, multisets of objects, and evolution rules. In this section, they will be formally defined and so, the dynamic of such systems.

The very important ingredient of P systems is the membrane structure. A membrane structure is a three-dimensional arrangement of vesicles in which the only important thing is the mutual relationship between membranes related to be in or out a determined membrane. This relation is defined as an adjacency relation. Figure 1, illustrates several notions about membranes and the adjacency relation. It is important to note the one-to-one correspondence between membranes and regions they enclose. A membrane structure can be pictorially represented in two-dimensional way in form of a Venn diagram and in term of an undirected tree, like in figure 1. The tree representation looses the biochemical intuition, but it makes clear the fact that the position of membranes do not matter

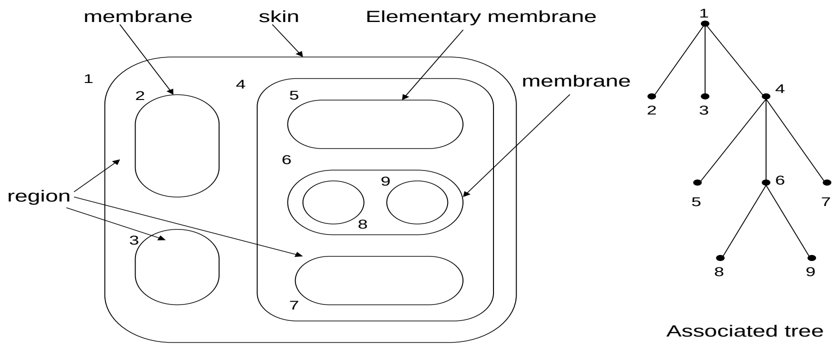

Figure 1: Membrane Structure and its associated tree.

There is one more representation for membrane structure in term of strings of matching parentheses. For example, the membrane structure of figure 1 is represented by the following parenthetic expression:

$$[_1[_2\,]_2[_3\,]_3[_4[_5\,]_5[_6[_8\,]_8[_9\,]_9\,]_6[_7\,]_7\,]_4\,]_1$$

Because the membranes have labels, the pairs of corresponding parentheses also have labels. Of course, the same membrane structure may be represented by different parenthetic expressions.

The second important ingredients are the multisets of objects, which are represented by strings of symbols associated with the objects. In turn, the evolution rules for objects will be given as multiset rewriting rules, of the form $u \rightarrow v$, where u and v are strings (understood as representations of multisets). It is important to note that, working with multisets, means to work with multiple copies of objects; when saying, for instance, that "and object *a* is processed" (for example, passed through a membrane), we understand "a copy of the object *a* is processed".

With these considerations, a membrane system with symbol-objects is a construct

$$\Pi = (V, T, \mu, \omega_1, \ldots, \omega_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_0)$$

Where:

i. V is an alphabet, and its elements are called objects;

ii. $T \subseteq V$ is the output alphabet;

iii. μ is the membrane structure and membranes are labelled in a one-to-one manner from 1 to m, being m the number of membranes in the membrane structure,

iv. $\omega_1, \ldots, \omega_m$ are multisets of objects over V and they are associated to the regions defined by the membrane structure;

v. $R_1,\ldots,R_m$ are finite sets of evolution rules and they are associated to the regions defined by the membrane structure;

- An evolution rule is a pair (u,v) that is usually represented by $u \rightarrow v$ and
  a. u is a multiset of objects over U,
  b. $v=v'$ or $v=v'\delta$, where v' is a multiset of objects over $Ux(\{here, out\} \cup \{in_j \mid j = 1,\ldots,m\})$ and δ is a symbol which is not in U.

vi. $\rho_1,\ldots,\delta_m$ are partial order relations defined over the set of rules of the regions defined by the membrane structure and they specify a priority relation in $R_1,\ldots,R_m$ *respectively,*

vii. i_0 is a label which determines the output membrane of the system.

The symbols: *here*, *out*, in_j, are named targeting commands. They specify the membrane target, e.g. the membrane that have to pass objects produced by the rule. In order to simplify the notation, *here* is omitted when the rules are written.

P systems compute starting from an initial configuration and passing from one configuration to another by the application of evolution rules to objects inside regions until reaching a halting configuration. This configuration is defined as a configuration in which there is no rule applicable to the objects present in the last configuration. Because these systems compute by transition, they are named *Transition P systems.*

A ***configuration*** is defined as a (k+1)-tuple $(\mu,\omega_1,\ldots,\omega_m)$ where μ is the membrane structure having m membranes and $\omega_1,\ldots,\omega_m$ are the objects multiset associated to the regions defined by the membrane structure [3].

The application of evolution rules in every region of the system is made in a non-deterministic and massively parallel way; e.g. every object present in a region that can evolve by the application of an evolution rule must evolve. Moreover, every region in the system evolves in parallel at the same time with the regions of the system.

A transition in a P system is got by passing from a configuration to another consecutive one. More precisely, given two configurations $C_i=(\mu^i_1,\omega^i_1,\ldots,\omega^i_m)$ $C_j=(\mu^j_1,\omega^j_1,\ldots,\omega^j_n)$ of a P system, Π, it is said that there is a transition from C_1 to C_2 and it is represented by $C_i \rightarrow C_j$, if we can pass from C_i to C_j by using the evolution rules from $R^i_1,\ldots,R^i_m$$ in the regions $i_1,\ldots,i_k$.

Transition P systems are devices very powerful. They are computationally complete. There are many other variants of P systems, but the essential set of specification described above act as framework for many of them.

Some Previous Formalizations

There are two main components in the simulation of membrane systems: the static and the dynamic. The first one is related to how to describe the main ingredients of the systems in a static way, the second one is related to give semantic to the static component; e.g. to make evolve the system from a static configuration to the next one by the application of rules associated to the regions of the systems.

The static structure of Transition P systems is related to everything concerning to membranes, and objects to be contained in them [4]. So, it is related to multisets, evolution rules and regions. There are some relations among these elements: multisets will be necessary for defining regions and evolution rules; evolution rules and again multisets will define a region, and finally regions will give the static structure definition of a Transition P system. Therefore, it is very important to determine a data structure for defining multisets, evolution rules and regions and the operations that can be done over such data structures [3] and [7]. Let us to provide some definitions for these elements.

Multisets provide a compact representation for words generated by a given finite alphabet when the order of symbols occurrences does not matter. In fact, multisets are sets of objects in which the number of occurrences of its different elements are taken into account. A multiset over a given finite object set U can be formally defined as follows:

Definition: Let U be an arbitrary finite set and let N the natural number set. A multiset M over U is a mapping from U to N.

$$M : U \to N$$
$$a \to Ma$$

There is another common and very useful representation for multisets of objects in membrane computing, it is the polynomial representation:

$$a^{Ma} b^{Mb} \ldots z^{Mz} \text{ Where } U = \{a, b, \ldots, z\}$$

It is very easy to define some algebraic operation in the set multiset over a given set of object *U*. In particular, addition and subtraction of multiset and multiset inclusion. Moreover, let *M(U)* the set of multiset over the set *U*, then it is easy to prove that *(M(U),+)* is a monoid with identity element (the empty multiset).

Evolutions rules are the active elements of P systems, they make evolve the system until reaching a halting configuration. In this section, evolution rules are described in terms of algebraic structures. They are defined without considering the dynamic implications they have in the evolution of P systems. First, they are defined in terms of multisets of objects over different finite sets of objects. Moreover, evolution rules are here defined independently of P systems and they are considered as static components. After that, some operations are defined and characterized; these operations will be considered in a different section of this paper for described the dynamic of evolution of P systems.

Definition: Let *L* be a label set, let *U* be a set of object, and let *D* $=\{out, here\} \cup \{in_j \mid j \in L\}$ the subset of labels to which rules can sends objects.

An ***evolution rule*** with label in *L* and objects in *U* is a tern (u, v, δ), where *u* is a multiset over *U*, *v* is a multiset over *U x D* and $\delta \in$*{dissolve, not dissolve } = {true, false}*. Let ***R(U,L)*** be the set of evolution rules with label in *L* and objects in *U*. Then, several operations over evolution rules can be defined:

Definition: Addition of evolution rules:

$$+ : M(U,L) \times M(U,L) \to M(U,L)$$
$$((u_1, v_1, \delta_1), (u_2, v_2, \delta_2)) \to (u_1 + u_2, v_1 + v_2, \delta_1 \vee \delta_2)$$

Definition: Product of a natural number by an evolution rules:

$$\bullet : N \times M(U,L) \to M(U,L)$$
$$(n, (u, v, \delta)) \to (nu, nv, \delta)$$

It can be also easily demonstrate that *(M(U,L),+)* is a monoid with identity element. Moreover, linear combination of evolution rules can be defined and they accomplish some interesting properties related to linear dependency.

Regions act as bags for containing multisets of objects and evolution rules. They are defined by the membrane structure of the P systems. Hence, it could be said that a determined membrane structure define a region structure for the system. Moreover, it is very interesting, for developer of simulations of P system, to consider regions as the container elements of P systems and to define an algebraic structure for representing the information of P systems. In this way, regions can be defined by:

Definition: A region with labels in *L* and objects in *U* is a tern ***(l, ω, (R, ρ))*** where $l \in L$, ω is a multiset over *U*, *R* is a set of evolution rules with labels in *L* and objects in *U* and ρ is a partial order relation over *R*.

With the previous definitions, a Transition P system can be defined as a tree of regions in the following manner:

Definition: A ***Transition P system*** with labels in *L* and objects in *U* is a pair of elements whose first one is a region with labels in *L* and objects in *U* and the second one is a set of Transition P systems with labels in *L* and objects in *U*, and regions are uniquely labelled.

$$\Pi = ((l, \omega, (R, \rho)), \Pi\Pi)$$

Where *reg=(l,ω,(R,ρ))* is a region with labels in *L* and objects in *U* and $\Pi\Pi$ is a set of Transition P systems with labels in *L* and objects in *U*.

This static structure of Transition P systems provides a formal and very practical representation for describing P systems configurations. In fact, as it will be show below, it can be used for defining a language for describing P systems in a static manner.

The formalization of the dynamic component of Transition P systems involves the use of algebraic operations defined over evolution rules [5] and [6]. In these papers is described how to obtain linear combination of evolution rules, which are able to make evolve the different regions of the P systems by the application of only such linear combinations –complete multisets of evolution rules- and how the systems evolve using them.

On the Simulation of P Systems

P systems have been demonstrated to be a very powerful computational devices in the theoretical framework (*in info*) but they are today restricted only to this theoretic framework. It is not so clear if they will be implemented in *vivo*, in *vitro* or only in simulations running in digital computer. The straightforward way to demonstrate the computational possibilities for P systems is to developed software simulations to be run in digital computers of general purpose or in specific hardware specifically developed for a specific class of membrane systems. Some published papers deal with software simulations of Transition P systems with different programming paradigms like functional programming, object oriented programming, distributed programming, etc. [8], [9], [10], [11], [12], and [13].

To develop a friendly software application for simulating P systems executions in digital computers it is needed to consider at least three main components, like in Model-View-Controller (MVC) architecture. The Model is responsible for type abstract data, the View is responsible for showing the results to the user through a graphical interface and the Controller is in charge of the requests made by the user. Following with these considerations, three physical components must be implemented in different subsystems: The Graphical User Interface (GUI), which permits the interactions between the user and the application and represents the static structure of the system in a friendly way. The static component describes the P system structure including membrane structure, multisets of objects, evolution rules and their associated priority relations. Finally, the Simulator admits as input the static component of the P system, implements the evolution of the system starting from an initial configuration, and obtains every new possible configuration to which transit.

In order to describe the static structure of P systems, it is needed to represent them very precisely. An specific language have been developed to describe Transition P systems. It was named *Bio-language* for Transition P systems [7] and it is described in term of rewriting rules as follows:

- *Syntax of the bio-language*
  - *V a set of objects*
  - *L a set of labels*
  - *Transition P system (TPS):* $\Pi \Rightarrow$ *[l Region; {*Π*}]l*
  - *Region* $\Rightarrow$ *Objects, Rules, Priorities*
  - *Objects* $\Rightarrow$ *{on} where o* $\in$ *V n* $\in$ *N*
  - *Rules* $\Rightarrow \lambda$ *| rule {, rule}*
  - *Priorities* $\Rightarrow \lambda$ *| ri < rj {, ri' < rj'} where i, i', j ,j'* $\in$ *N*
  - *Addresses* $\Rightarrow$ *here | out | l where l* $\in$ *Labels*
  - *ObjTarg* $\Rightarrow$ *{(o, Addresses)n} where o* $\in$ *V and n* $\Rightarrow$ *1 | 2 | ...*
  - *Rule* $\Rightarrow$ *rm: Objects* $\rightarrow$ *ObjTarg* δ *where* $\delta \in$ *{dissolve, not dissolve}*

As it can be seen, the bio-language uses the representation of Transition P systems as tree of regions. It describes every component of the system including the membrane structure, the multiset of objects, the evolution rules and the partial order relation defined among evolution rules.

This is an example of bio-language for a specific variant of P system. What is important here is to demonstrate the feasibility of formally describing membrane systems and to use this formalization as inputs to the appropriate simulator, which is in charge of implementing the semantic for the membrane system.

The simulator will implement the evolution of the membrane system. It is in charge of parsing the static structure of the P system into abstract data structures and then to transform them in the appropriate way implementing the evolution of the systems. It will provide a set of possible configurations to which the systems could transit starting from the initial configuration. Every new configuration of the system will be described in terms of the static structure using the bio-language for the membrane system. Hence, once has been chosen one configuration to transit the new configuration can be used as input again to the simulator and continue the software execution. Hence, for each variant of P systems, a determined bio-language can describe the static structure of the corresponding system and the specific simulator must implement the dynamic behaviour of the system. An exhaustive list of published papers in Membrane Computing can be found in [14].

Conclusions

Membrane computing is a very active research field, many different variant of P systems are actively modified providing new variant of the model. Simulations of membrane systems are very powerful tools for researchers in order to check their working thesis. Many of the programming paradigms can be use to develop software simulations of membrane system (object oriented programming, functional programming, distributed programming, etc.) exploding different attributes of the membrane system. What is very important is to build different programming approaches for developing simulations and to explore which ones are useful to bring to membrane computing the necessary tools for making it useful for researchers in computer science and biology and to explore new un-conventional computing models.

References

[1] Gh. Paun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002

[2] Paun G.: *Computing with Membranes*. In: Journal of Computer and Systems Sciences, 61, 1. (2000) 108--143

[3] Gh. Paun, G. Rozenberg*, A Guide to Membrane Computing*, Theoretical Computer Science, vol 287, nr 1 (2002), pages 73-100.

[4] A. V. Baranda, J. Castellanos, R. Gonzalo, F. Arroyo, L. F. Mingo, *Data Structures for Implementing Transition P Systems in Silico*, Romanian J. of Information Science and Technology , 4, 1-2 (2001), 21-32

[5] A. V. Baranda, J. Castellanos, F. Arroyo, R. Gonzalo, *Towards an Electronic Implementation of Membrane Computing: A Formal Description of Nondeterministic Evolution in Transition P Systems*, Proc. 7th Intern. Meeting on DNA Based Computers, Tampa, Florida, USA, (N. Jonoska, N.C. Seeman, eds.) Lecture Notes in Computer Science 2340, Springer Verlag, 2002, 350-359.

[6] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo, *A Recursive Algorithm for Describing Evolution in Transition P Systems*, Pre-Proceedings of Workshop on Membrane Computing , Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 19-30.

[7] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo, *Structures and Bio-Language to Simulate Transition P Systems on Digital Computers*, in Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View (C.S. Calude, Gh. Paun, G. Rozenberg, A. Salomaa, eds.) Lecture Notes in Computer Science 2235, Springer-Verlag, 2001, 1-16.

[8] F. Arroyo, C. Luengo, A.V. Baranda, L.F. de Mingo*, A software simulation of transition P systems in Haskell*, Pre-Proceedings of Second Workshop on Membrane Computing, Curtea de Arges, Romania, August 2002 and Proc. of WMC02, Curtea de Arges, Romania, (Gheorghe Paun, Grzegorz Rozenberg, Arto Saloma, Claudio Zandron Eds.) Lecture Notes in Computer Science 2597, Springer-Verlag, 2003, 19-32.

[9] G. Ciobanu, D. Paraschiv: Membrane Software. *A P System Simulator*, Pre-Proceedings of Workshop on Membrane Computing, Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 45-50 and Fundamenta Informaticae, 49 , 1-3 (2002), 61-66.

[10] G. Ciobanu, G. Wenyuan: *A parallel implementation of transition P systems*, Artiom Alhazov, Carlos Martín-Vide and Gheorghe Paun (eds.): Preproceedings of the Workshop on Membrane Computing; Tarragona, July 17-22 2003, 169-184

[11] M. Malita: *Membrane Computing in Prolog*, Pre-proc. Workshop on Multiset Processing , Curtea de Arges, Romania, CDMTCS TR 140, Univ. of Auckland, 2000, 159-175.

[12] Y. Suzuki, H. Tanaka: *On a LISP Implementation of a Class of P Systems*, Romanian Journal of Information Science and Technology, 3, 2 (2000), 173-186.

[13] A. Syropoulos, E.G. Mamatas, P.C. Allilomes, K.T. Sotiriades, *A distributed simulation of P systems* , Artiom Alhazov, Carlos Martín-Vide and Gheorghe Paun (eds): Preproceedings of the Workshop on Membrane Computing; Tarragona, July 17-22 2003, 455-460

[14] http://psystems.disco.unimib.it/

Authors' Information

Fernando Arroyo – Dpto. Lenguajes, Proyectos y Sistemas Informáticos- Escuela Universitaria de Informática; Universidad Politécnica de Madrid - Crta. de Valencia km. 7 - 28031 Madrid SPAIN - farroyo@eui.upm.es

Carmen Luengo – Dpto. Lenguajes, Proyectos y Sistemas Informáticos- Escuela Universitaria de Informática; Universidad Politécnica de Madrid - Crta. de Valencia km. 7 - 28031 Madrid SPAIN - cluengo@eui.upm.es

Luis Fernandez – Dpto. Lenguajes, Proyectos y Sistemas Informáticos- Escuela Universitaria de Informática; Universidad Politécnica de Madrid - Crta. de Valencia km. 7 - 28031 Madrid SPAIN - setillo@eui.upm.es

Luis F. de Mingo – Dpto. Organización y Estructura de la Información - Escuela Universitaria de Informática; Universidad Politécnica de Madrid - Crta. de Valencia km. 7 - 28031 Madrid SPAIN - lfmingo@eui.upm.es

Juan Castellanos – Dpto. Inteligencia Artificial - Facultad de Informática - Universidad Politécnica de Madrid; 28660 Boadilla del Monte - Madrid SPAIN - jcastellanos@fi.upm.es

TEACHING STRATEGIES AND ONTOLOGIES FOR E-LEARNING [1]

Tatiana Gavrilova, Michael Kurochkin, and Victor Veremiev

Abstract. The paper presents one approach aimed at developing teaching strategies based on the principles of ontological engineering. The research framework is targeted on development of methodology and technology that will scaffold the process of knowledge structuring for e-learning. The structuring procedure is the kernel of ontology development. Ontologies that describe the main concepts of the domains are used both for teaching and assessment techniques. Special stress is put on visual design as a powerful learning mindtool. The examples are taken from the courses on the foundations of artificial intelligence and intelligent systems development. These courses are delivered by the authors in St.Petersburg State Polytechnical University at School of Computer Science and in Poland in the First Independent University.

Keywords: E-learning, Ontologies, Visual Knowledge Engineering, Expert Systems Building Tools, Knowledge Acquisition, Knowledge Sharing and Reuse.

1. Introduction

The drawback of e-learning is lack of feedback from the teacher or tutor. That is why the courseware should be more precisely structured that in face-to-face teaching.

The idea of using visual structuring of teaching information for better understanding is not new. Concept mapping [Sowa, 1994; Jonassen, 1998, Conlon, 1997] is scaffolding the process of teaching and learning for more than 20 years. Visual representation of the general domain concepts is facilitative and helps both learning and teaching. A teacher now has to work as knowledge analyst or knowledge engineer making the skeleton of the studied discipline visible and showing the domain's conceptual structure. This structure is now called "ontology". However, ontology-based approach is rather young. It was born in knowledge engineering [Boose, 1990; Wielinga, Schreiber, Breuker, 1992], then it was transferred to knowledge management [Fensel, 2001].

[1] The work is partially suported by Russian Foundation for Basic Studies (grant 02-01-00466)