Visualization and Simulation of Membrane Computing

Student Name: G.B.Newton

Abstract –

Context/Background

Membrane computing is an unconventional computing system in the area of natural computing which builds a model of computation from the function and structure of living cells; it is massively parallel, distributed, and non-deterministic. Research in this field is gaining interest and thus better tools for simulation and visualization of membrane system and their algorithms are required.

Aims

We will pioneer new methods and new tools for specification, simulation and visualization of membrane systems on electronic computers, in addition to suggesting new data structures for simulation, and new paradigms for membrane specification.

Method

We will devise a language to specify Membrane Computing systems, a parser to translate this specification into a membrane structure stored in memory, and a simulator/interpreter to perform all the calculations associated with a transition membrane system using a fast linear algorithm.

Proposed Solution.

We will deliver a specification for a 'Membrane Language', implemented in a fashion similar to that of Object Orientated Programming.

Simulator/Interpreter, implementing fast linear algorithms and data structures for tracking membrane computation.

A visualization engine rendering the state of a membrane system, in addition to providing precise control over playback and debugging features.

Keywords – Natural Computing, P-Systems, Membrane Computing

I. INTRODUCTION

A. I.I Background to membrane computing and problem domain.

Membrane computing is an unconventional computing system in the area of natural computing which builds a model of computation from the functioning and structure of living cells; it is massively parallel, distributed, and non-deterministic. Whilst implementations of P-systems in actual biological cells are not ruled out we will focus purely on the implementation of P-Systems in electronic computers.

Membrane Computing was introduced by G.Paun in (G. Pâun, 2002) under the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called P-systems. Roughly speaking, a P-system consists of a membrane structure, in the compartments of which one places multi-sets of objects which evolve/change according to given rules in a synchronous nondeterministic maximally parallel manner.

While the area offers many exciting possibilities, the process of designing a P system to perform a task is extremely difficult. A small mistake in the description of the initial configuration or in the set of rules can lead to undesired consequences. An attempt to tackle this was made in (Escuela, 2010) which attempted to use genetic algorithms to evolve P-systems rather than using human design.

However this approach ran into problems particularly with respect to the fitness function; 'the fitness function that we considered for this problem, that conforms to a landscape with a unique peak.' i.e. that unlike many fitness functions it could only return yes or no, it had no way of judging any intermediate state, and thus evaluating algorithms that were 'close' to a correct answer. We suggest a secondary problem with the fitness function used the paper. While we realize why it necessary, the function only considered the P-systems correctness, and not the complexity or the efficiency of the P-system producing it. Lastly the paper's goal was to produce an algorithm for producing the squares of numbers (i.e. a simple task), however out of 120 experimental runs they produced 2 successful P-systems. It could be assumed the success rate would decline as the production of more complex algorithms is attempted.

This suggests that human designers are, for the moment, the best option for producing P-systems, and thus we need to provide the tools which will reduce as many of the problems they face as possible. In addition we foresee that should useful automated systems for the design of P-systems be produced it would be extremely useful for human designers to be able to visualize the workings of the systems, and examine them for possible adaptations.

1) I.I.I The Membrane structure

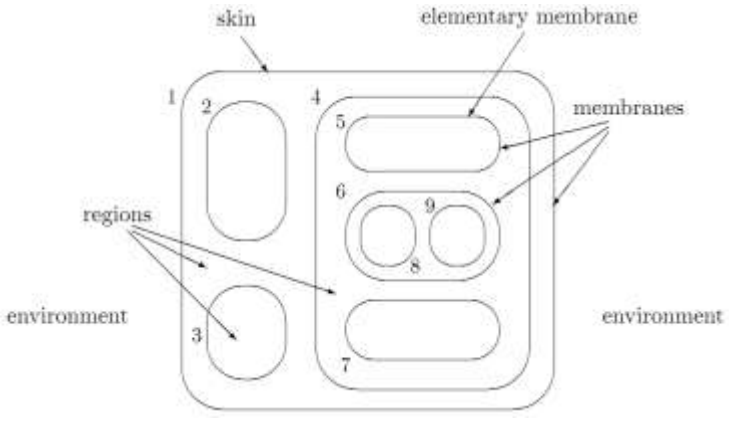

Figure 1

The essential ingredient of a P-system is its membrane structure, a hierarchical arrangement of membranes, like a biological cell. A membrane may be thought of as a separator of two regions, producing a finite inside and an infinite outside. Providing the two regions with selective communication (G. Pâun, 2002).

2) I.I.II Objects in Membranes

Objects are the data which is computed by the P-system. They are analogous of the chemicals in biological membranes and have no inherent order. Objects are specified by multisets of unique symbols and each membrane region contains $n \geq 0$ identical copies of each object. They are consumed, produced and transported by the rules (see below) of a membrane system

Catalysts are a subset of objects which remain unchanged during a reaction, (though with some membrane computing models this is not always the case)

Promoters and Inhibitors are a subset of objects; they are used to control whether a certain rule is executed or not. They are different from catalysts in that a copy is not actually used in the rule. *Promoters* cause a rule only be executed if they are present in the region, *Inhibitors* inversely cause a rules execution to fail if they are present in the region .

3) I.I.II Rewriting Rules in Membrane Systems

Each membrane may be considered a maximally parallel (As many operations should be performed at once as possible.) processor of multisets of data; the processing of these multi-sets is done by non-deterministic multiset rewriting rules. Each rule is made up of one or more of the following types:

- **Evolution** – one set of objects is consumed to produce another set of objects, for example in Eq.(1), each time the rule is run one copy of a and b will be removed from the membrane and one copy of f and g will be added.

$$Eq.(1)\ ab \rightarrow fg$$

- **Promoter/Inhibitor** – The execution of the rules rely on the presence or lack of presence of certain objects. In Eq.(2) the rule could only execute if ‘c’ is present in the membrane, and ‘d ’ is not present in the membrane.

$$Eq.(2)\ ab \rightarrow fg|_{c\neg d}$$

- **Transitional** – These are rules which communicate objects to other membranes with the target indication ‘*in*’ or ‘*out*’, for example in Eq.(3) the object ‘c’ would be passed out of them membrane to its parent membrane, and the object ‘d’ would be passed into a non-deterministically chosen child membrane.

$$Eq.(3)\ \ ab \rightarrow c_{out}d_{in}$$

- **Electro-charge Transition** – These are rules which communicate objects to other membranes using electric charges, when an object is produced it is given a charge, it will then travel inwards until it finds a membrane with an opposite charge.

$$Eq.(4)\ \ ab \rightarrow c_{+}d_{-}$$

- **Variable Permeability** – These rules change the permeability of a membrane, with ‘1’ being normal permeability, allowing normal transitional rules to be applied, a thickness of ‘2’ the membrane is considered a closed membrane where no transitional rules (including electro-charge) may be applied. If a membrane’s permeability is reduced to 0, the membrane is considered to be dissolved, the objects and child membranes contained within become the children of that membranes parent. Rules of the form Eq.(5) reduce a membranes permeability, and of Eq.(6) increase it.

$$Eq.(5)\ \ ab \rightarrow c\delta$$

$$Eq.(6)\ \ ab \rightarrow c\tau$$

- **Symport and Antiport –** The rules represent the natural communication channels which exist between membranes. They are unique in our model of membrane commuting in they are not associated with any particular membrane, but act on the structure as a whole. Symport rules are the transportation of two or more objects from one membrane to another (they need not be adjoining membranes). Anti-port rules are the bi-directional transportation of one or more objects in each direction between two membranes (again they need not be adjoining membranes).

B. Purpose of the project

Membrane computing is a relatively new area of natural computing, it was first conceived by Gheorghe Paun, in his 1998 paper Computing With Membranes, Since then the area has grown with development of SAT solving algorithms, and fast liner algorithms for implementation. From the standpoint of researchers in areas of computability and complexity theory, P-systems offer interesting models for devising distributed and parallel algorithms; however the computations of P-system algorithms are by their very nature challenging to decipher for researchers, as they are abstract from normal programming paradigms, in addition to being non-deterministic and maximally parallel.
An early P system simulator was attempted by (Malita, August 21-25 2000) who created simulation library in Prolog; this however has no support for more advanced membrane systems such as Promoter/Inhibitor systems. The Prolog library was built upon in (D. V. Nicolau, 2002) which offered a C library to implement P-Systems

To aid research into this area we will attempt to provide intuitive tools both for construction and visualization of membrane systems, while at the same time continuing to provide a powerful simulation environment.

C. Specification of deliverables and aims

The project aims to deliver at a minimum

- Basic specification of P-System using a text input interface.
    - Devise a Bio-Language to define a membrane structure, in a text file.
    - A text editor to edit this structure.
    - Parser to move between the file and a structure and based in memory.
- Simulator a for P-System computation including support for:
    - Transitional P-Systems.
    - Symport and Anti-port P-Systems.
    - Promoter/Inhibitor P-Systems.
    - Active Membrane P-Systems.

- Interactive visualization/animation of P-Systems where the computation is simulated at the same time as the visualization.
    - Including support for speeding up, slowing down, and navigating the visualization frame-by-frame.

Advanced Objectives

- Simulation of a subset of P systems using CUDA.
- Intuitive graphical construction of P-Systems.
- Using these tools improve on some existing P-System algorithms.

II. DESIGN

A. System Architecture

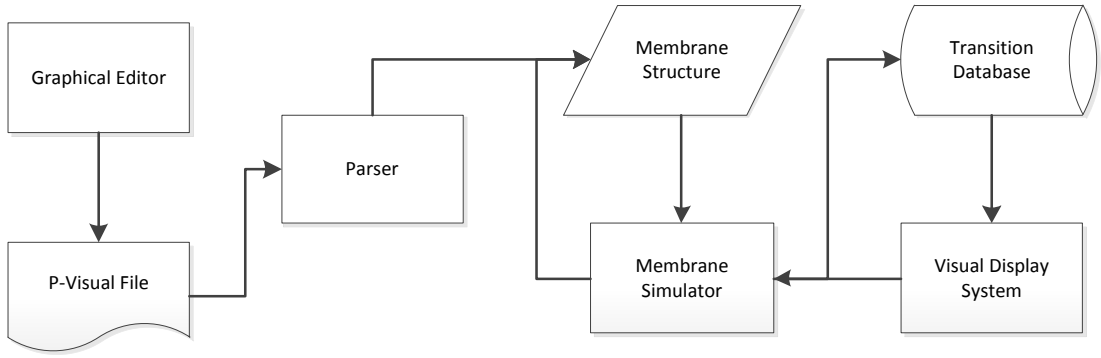

Figure 2

Membrane structure – A tree data structure of nodes representing membranes pointing to implementations of membrane classes and rules.
P-System Simulator – A system of applying the rules of a membrane system, moving from one state to another.
Transition Database – Storage of the computational states the P-System simulator has passed though.
Parser – A lexical system to parse the data from the system file into the membrane structure.
P-Visual File – Written in the P-Visual language (of our devising, see below), a programming language defining a P-System.
Visual Display System – Takes the computational states from the Transition Database subsystem and produces a visual display of the membrane structure, also a graphical user interface for real time interaction with other systems.
Membrane structure builder – Provides a text editor/GUI to build membrane structures.

B. Programming Paradigm

A programming paradigm is a fundamental style of computer programming, providing the concepts and abstractions used to represent the elements of a program, and the steps that compose a computation.
Much as (Shoham, 1993) adapted the paradigm of Object-Orientated Programming (OOP) into Agent-Orientated Programming, We will adapt the paradigm to something akin to Membrane-Orientated Programming. It is clear that many of the aspects of OOP are directly applicable to P-Systems, in particular:

- **Abstraction**- Although few algorithms for membrane computing yet exist, one common theme among those that do is a number of membranes with the same rules working together. Thus it makes sense to the author to abstract the definition of rules and membranes from the actual instances of the membrane. This both reduces programming complexity and data size in memory during simulation.
- **Encapsulation** – The natural presence of encapsulation is clear in the membrane model, as it both restricts access to the objects (data), though rules, in addition to the membrane acting as a construct that facilitates the bundling of data with the methods (rules) operating on that data.
- **Modularity** – Although modularity is not immediately apparent in a membrane system, it is clearly possible, and in the authors opinion something that should be strived for. If we can develop modular ‘black box’ membranes that are known to perform certain functions, then these can be moved between P-systems, allowing us to buildup more complex systems.

By attempting to build our language (P-Visual) in the image of the OOP paradigm we also open up the exciting future possibility of bringing other features of OOP into the whole area of membrane computing such as polymorphism (If for example the ability to create a rule, or membrane, that has more than one form) and inheritance (we consider this a particularly exciting idea, with membranes as an analog for Objects, the ideas of inheritance seems natural, and would allow P-System programmers to build more complex membranes out of the behavior of simpler ones).
The programming paradigm will rely on the flowing concepts:

- A list of defined symbols(objects).
- A set of rule definitions, defined in terms of symbols.
- A set of membrane definitions, defined in instances of rules.
- A initial membrane structure, defined in terms of membrane instances, and initial multi-sets of objects.

C. Language Design

The P-Visual language file will be broken into five major sections defined by a pair of hash tags with the section name inbetween. The sections are alphabet (#ALPHA#), rules (#RULES#), membranes (#MEM#), membrane structure (#STRUCT#), symport/anti-port rules (#PORT#).

1) Alphabet

This section defines the set of symbols which will represent objects within the membrane system. The set of symbols is contained within a set of braces, and separated by a comma e.g.

$$\{a, b, c, d, e\}$$

2) Rules

This section of a P-visual file will define a set of named rules, which will be attached to membrane classes in the 'membrane' section. Each rule is defined with a name followed by a pair of braces containing the rules operation. The left hand side of the rule is separated from the right with a '$->$' symbol as an analog for a $\rightarrow$ symbol. Thus it has the general form

$$name\{LHS -> RHS\}$$

The left hand side contains a set of symbols which will be consumed when the rule is applied. In addition to any Inhibitors (denoted with a '/'), and Promoters (denoted with a '\'). For example a rule consuming one copy of 'a', two of 'b', inhibited by 'c', and promoted by 'd', would be represented by

$$rule\{abb/c\backslash d -> RHS\}$$

The right-hand side contains the set of symbols which will be produced when the rule is applied. In addition it contains targeted outputs (denoted with a '(symbol,target)'), where the target can be 'in' or 'out'. Other targeting systems include electro-charge transition rules, where that targets are denoted by +, and ~ before the symbol. Further symbols that can appear on the RHS are changes in permeability i.e dissolving (denoted with a '!'), and thickening (denoted with a '?').

$$rule\{LHS \rightarrow (a, in)(b, out)\}$$
$$rule\{LHS \rightarrow +a{\sim}b\}$$
$$rule\{LHS \rightarrow ab!\}$$
$$rule\{LHS -> ab?\}$$

3) Membranes

This section of a P-visual file will define a set of named membranes, instances of which will be used in the membrane structure. Each membrane is defined with a name followed by a pair of braces containing the set of rules it uses separated by commas. Thus it has the general form

$$name\{rule1, rule2, rule3\}$$

4) Membrane Structure

The membrane structure is defined by a set of nested braces, each set of braces preceded membrane name defining which class of membrane it belongs to. Braces may contain sets of square-braces, in turn containing a set of parentheses with symbols and quantities separated by commas, having the general form:

```
membrane1{
  [(symbol,quantity)(symbol,quantity)]
  membrane2{}
  membrane2{
            membrane3{}
           }
}
```

5) Symport/anti-port rules

This section of the P-Visual file deals with symport/anti-port rules. These have to be defined separately in despite of breaking from the paradigm, they do not belong to any one class or instance of membrane, thus they must be defined independently. The type of rule defined with a keyword 'SYMPORT' or 'ANTIPORT'(they do not need a name as they are never referenced), followed by a set of braces containing the symbols and membranes involved in the rule, they have the general form:

$$SYMPORT\{symbols|membrane-from|membrane-to\}$$
$$ANTIPORT\{symbols-A|symbols-B|membrane-A|membrane-B\}$$

D. Algorithms

The Algorithm that will form the backbone of this simulator will be the 'Fast Linear Algorithm for active rules application' proposed in (Francisco Javier Gill, 2009). This algorithm however can only provide a core as it only considers Evolution rules. The tools described in this paper aims to provide a much wider range of membrane rules, for use in membrane computation and thus we must develop an algorithm that can properly handle these.

While it is one of our advanced objectives to deliver a system that makes use of the CUDA programming model, we must note the work done in 'Simulation of P-systems with active membranes on CUDA' (Jose M. Cecilia, 2009). The paper provides an elegant method of processing rule applications in parallel; however they restricted the rules used to those with a left-hand side of only one symbol. Such a restriction is not acceptable in the tool we aim to provide. At this time we can see no improvement than can be made to the algorithm presented in the paper which would resolve the problem. Hence if our tool is to use CUDA it must only be as a module used to speed up rules which it is possible to apply CUDA algorithms to.

The Algorithm proposed in (Francisco Javier Gill, 2009), has the following structure:

- List the rules $R\{r \in 'Rules'\}$.
- Calculate the number of times it possible to perform each rule $|r|$.
- Separate rules in those which can be performed at least once R,and those that can not R'.
- Perform a rule a random number of times between 0 and $|r|$.
- Move rules which cannot be performed at least once into R'.
- While $|R| > 0$ repeat.

- When $|R| = 1$ perform the last rule r, the maximum number of times ($|r|$).

Where $|r|$ is the number of times rule r can be performed on the objects contained in the membrane. We shall refer to this in future as the modulus functions of a rule.

The first thing that must be reconsidered is the modulus function, in Gill's algorithm it is a multi-set division of the objects in the membrane by the objects required by the rule. This is still the most important part however we must place additional restrictions on it when it must also return zero.

Firstly we must check that any inhibitors described in a rule are not present, and vice versa that all promoters are present. If one of these conditions is not met the modulus function must return zero.

Secondly with respect to Transitional rules and Electro-Charge Transition rules, because both of these can cause objects to be transported inwards (in a graph representation, to a child node), a check must be performed as weather the membrane executing the rule has a child membrane, if this is not the case the rule cannot be executed, and this has a modulus of zero.

Lastly to better simulate parallel computing on a linear processor, we shall store the result of each membrane's computation as a node on a tree, then resolve these results to produce an up-to-date membrane structure. This is useful for operations where membranes are being dissolved or created. It has a second, more important advantage; as we will discuss in the next section it will reduce the memory problems caused by interactive visualization.

E. Data structures

1) Membrane structure, and transition data

It is clear from the hierarchical nature of a P-System that the tree-like data structure lends its self best to the implementation of a membrane structure in memory. This is discussed at length in (Angel Baranda, 2001), showing it to be optimal in terms of both computer memory, and computational efficiency.

However if our tool is to provide navigation though a simulation of a P-system's computation there is the additional task of storing this data in memory. To do this we will separate out simulator from our visualization engine. The simulator will have a full representation of the P-system stored in memory.

The visualization engine however will store a new data-structure: a linked-list of trees, where the trees are that transitional information required to transform one Membrane Structure into another.

Thus we define

- G_x as tree fully representing a membrane structure.
- G_0 as the Initial membrane structure of computational run C, taking n time.
- G_n as the membrane structure of C at halting time.
- δ_x as the *Transition Graph* between $G_x$and G_{x+1}.
- $G_x + \delta_x = G_{x+1} \qquad G_{x+1} - \delta_x = G_x$as the *Transition Functions* between $G_x$and G_{x+1}.
- $\frac{G_x}{G_{x+1}} = \delta_x$ as the *Transition Computation* between $G_x$and G_{x+1}.

The transition computation is the construction of the transition graph δ_x as the *Transition Graph* between $G_x$and G_{x+1}. The transition graph δ_x will be symmetrical to the G_x, for each node in G_x, The transition graph will represent:

- Any change in Object-Multi-sets, for example if membrane $m \epsilon G_x$ contains 3 copys of object a and $m \epsilon G_{x+1}$ contained 7 copies. $m \epsilon \delta_x$ would contain $a \rightarrow +4$
- Change in permeability .
- If a membrane is dissolved or created a δ_x will contain a pointer information indicating how the membrane is added or removed

A simulation will consist of the linked list of the set $\{\delta_0 \ldots \delta_{n-1}\}$, thus any membrane structure that has been constructed can should be accessible. If G_m Is the membrane structure currently in memory and we want to view membrane structure G_z, then if $m < z$

$$G_m + \delta_{m+1} + \ldots + \delta_{z-1} = G_z$$

else if $m > z$

$$G_m - \delta_{m+1} - \ldots - \delta_{z-1} = G_z$$

2) Representation of multi-sets

Although our tool will provide a outward resonation of multi-sets in terms of symbols, they will be stored in memory as arrays of integers, after being mapped this allows us to perform vector mathematics. If we take an example P-Sytem operating on multi-seta of data consisting of the symbol set $\{a, b, c, d, e\}$. The symbol set will be mapped onto the set of real numbers in this case $\{a \rightarrow 0, b \rightarrow 1, c \rightarrow 2, d \rightarrow 3, e \rightarrow 4\}$, this allows the contents of membranes to be stored as an array, reducing memory usage by mapping symbols to the array index e.g. it could store $\{\{a, 2\}, \{b, 3\}, \{c, 6\}, \{d, 7\}, \{e, 0\}\}$ as;

$$\begin{bmatrix} 2 \\ 3 \\ 6 \\ 7 \\ 0 \end{bmatrix}$$

This is useful as operations such as rule modulus can be performed as vector division. The rule $abbb \rightarrow cdd$ operating on the set $\{\{a, 2\}, \{b, 6\}, \{c, 6\}, \{d, 7\}, \{e, 0\}\}$ has a modulus of

$$|R| = \begin{bmatrix} 2 \\ 6 \\ 6 \\ 7 \\ 0 \end{bmatrix} \Big/ \begin{bmatrix} 1 \\ 3 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

F. Visualization Design

1) Multi-Set presence/quantity representation

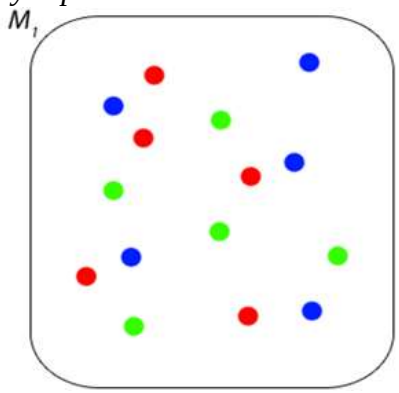

Figure 3

2) Labeling and Structure

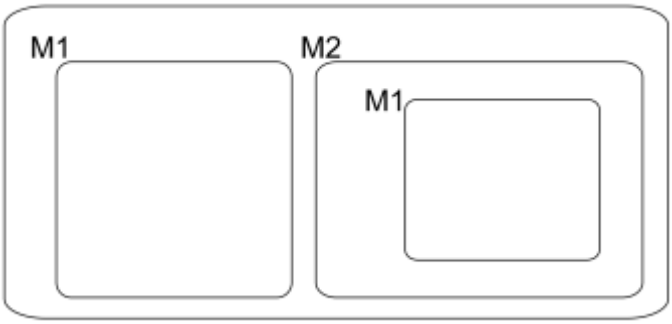

Figure 4

3) Permeability

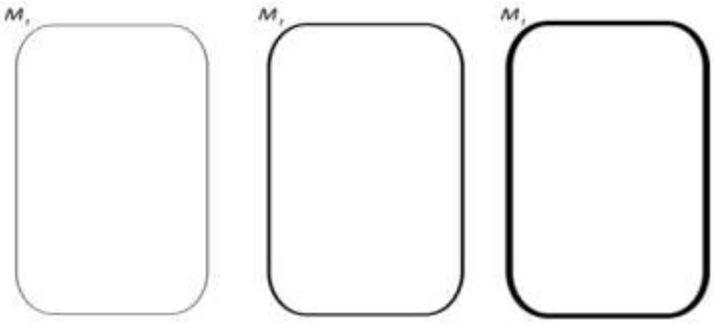

Figure 5

4) Charge

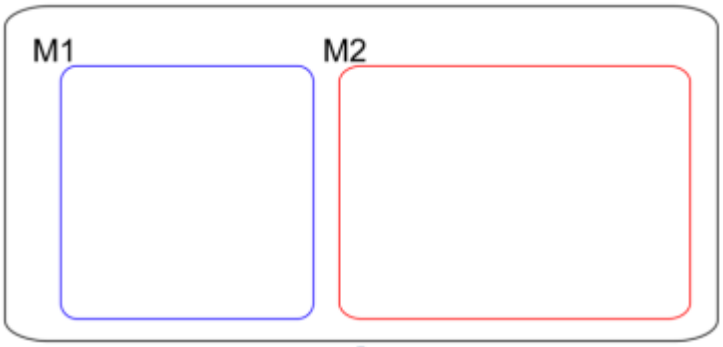

Figure 6

G. Evaluation

To evaluate the effectiveness of the simulators we will need to compare it against other P-System Simulators, this presents problems as there very few of these and most are in varying stages of development.

So it would also be worth comparing algorithms running P-systems to those running in traditional settings such as C++, or Java. To do this we would have to run an algorithm that solved a problem in rounds, as CPU time is not fair comparison. Such examples would be

- Genetic Algorithms
- Ant Colony Optimization

For comparison we will attempted to solve the Traveling Salesman Problem, using both the techniques described a above and comparing the results to the P-System algorithms given in (Gabriel Ciobanu, 2006). We believe the Traveling Salesman Problem is an excellent test of a system being a computationally hard problem (NP-Complete) problem, with real word applications. As well as being easy to scale.

III. References

Angel BARANDA, J. C. (2001). Data structures for implementing Transition P systems in silico. *ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY, Volume 4, Numbers 1-2, ,* 21-32.

D. V. Nicolau, G. S. (2002). A "C" Library for Implementing P systems on the Electronic Computer. *Fundamenta Informaticae. Volume 49* , 1-2.

Escuela, G. a.-N. (2010). An Application of Genetic Algorithms to Membrane Computing. *(Conference)Eighth Brainstorming Week on Membrane Computing* , 101-108.

Francisco Javier Gill, J. T. (2009). Fast Liner Algorithum for Active Rules Application in Transition P-Systems. *Information Therories & Applications Vol.16* , 222-232.

G. Pâun, M. J.-J. (2002). Membrane Computing: Brief introduction, recent results and . *BioSystems* , 1-12.

Jose M. Cecilia, J. M.-d.-A.-H.-J. (2009). Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics* , 1-10.

Malita, M. (August 21-25 2000). Membrane Computing in Prolog. *C. Calude, G. Paun, G.Rozenberg, A. Salomaa.Workshop on Multiset Processing* , 1-16.

Shoham, Y. (1993). Agent-oriented programming . *Artificial Intelligence 60* , 51-92.