

From Membranes to Systems: Self-Configuration and Self-Replication in Membrane Systems

Christof Teuscher

*Los Alamos National Laboratory, Advanced Computing Laboratory, CCS-1,
MS-B287, Los Alamos, NM 87545, USA*

Abstract

Membrane systems are purely abstract computational models far inspired by biological cells, their membranes, and their biochemistry. The inherently parallel nature of membrane systems makes them obviously highly inefficient to execute on a sequential von Neumann computer architecture and in addition, programming a membrane system is often a painstakingly difficult undertaking. The main goal of this paper is to provide some key elements for bringing membrane systems from the abstract model closer to a genuine, novel, and unconventional *in silico* computer architecture. In particular, we will address the mechanisms of self-configuration and self-replication on a macroscopic level and will discuss some general issues related to genuine hardware realizations on the microscopic level.

Key words: membrane system, artificial chemistry, rewriting system, unconventional hardware, cell

1 Introduction and Motivation

More than ever, we observe today a constant need to make computers faster and to improve their overall characteristics, i.e., to make them more robust, more reliable, more adaptable, and smarter. Surprisingly, the von Neumann computer architecture (i.e., the stored program concept)—first expressed in 1945 (von Neumann, 1945)—has survived all efforts and largely dominates computer science in many variants and refinements for more than half a century. Alternative approaches such as for example data-flow architectures, systolic arrays, cellular machines (Macias and Durbeck, 2004; Nagami et al., 1998), and other unconventional computing approaches have only occupied a marginal place, partly because of their limited applicability or because of the difficulty to synthesize, formally analyze, and program them.

Email address: christof@teuscher.ch (Christof Teuscher).

URL: www.teuscher.ch/christof (Christof Teuscher).

The 21st century promises to be the century of bio- and nanotechnology. With the advent of new materials and technologies (Franceschi and Kouwenhoven, 2002; Mathur, 2002) such as self-assembling systems, organic electronics, hybrid electronic-biological machines, etc., and the ever-increasing complexity and miniaturization of actual semiconductors¹ and systems, there is a growing need to fundamentally rethink the way we build computers, the way we organize, train, and program them, and the way we interact with them. This challenge is for example motivated by the hardware-software “design gap,” which has its origin in the silicon evolution roadmap predicted by Moore’s law, and which states that the design methodologies and software tools do not keep up with the growing number of hardware resources (i.e., transistors) that are available to the designers. The quest for novel, alternative, and unconventional design methodologies and computing architectures is definitely on, and it is therefore not surprising that the international research community is fiercely seeking progress in that field to address tomorrow’s challenges in computer science.

Here we are interested in investigating how *membrane systems* (or *P systems*), initiated by Paun in 1998 (Paun, 2000), might help us in this challenge and we will provide some key elements, which we believe might further pave the way to practical applications and might open new perspectives. Membrane systems are a class of abstract computational models afar inspired by biochemistry and by some of the basic features of biological membranes. While the vast majority of the work in the field concentrates on abstract aspects, we will focus on practical issues in the present paper. Paun states in (Paun, 2002, p. 5) that “[c]urrently, we have no idea where to look for an implementation of membrane computing of practical interest, in biology or in existing electronic computers.” The current article shall shed some light on this open question. We are convinced that only genuine hardware will allow to fully benefit from the membrane systems’ theoretical results and their inherently parallel structure. Simulating membrane systems on existing sequential computers or on distributed (sequential) machines is of course always a possibility, but it is neither an efficient (in terms of speed and resources used) nor a scalable approach (which is never the case when one simulates a parallel on a sequential machine). The long-term research goal is to ultimately provide (*in silico*) solutions that (1) minimize the resources (e.g., the number of transistors) used, (2) that do not build upon sequential von Neumann architectures, and (3) that are scalable, adaptable, and reconfigurable. In addition, systems that self-configure, self-replicate, and evolve on such a substrate are believed to play an important role for very large-scale implementations involving huge numbers of components that cannot be individually programmed anymore. In this paper we address in particular the questions of (1) how to self-configure

¹ International Technology Roadmap for Semiconductors, Semiconductor Industry Association, 2004, <http://www.itrs.net/Common/2004Update/2004Update.htm>

or develop a membrane system from an initial “mother cell” that contains a sort of “genome,” and (2) how we can self-replicate a membrane system in a given abstract environment. Whereas these questions are more concerned with the membrane system’s macroscopic aspects, we will address implementational issues on the microscopic level, which are crucial for genuine and efficient hardware implementations, which in turn will eventually allow us to bring membrane systems from theory back to reality. Also, both micro- and macroscopic aspects are best considered together in order to come up with efficient and scalable mechanisms. Conceptual toy examples will provide an illustration and proof of the basic ideas. We believe that these research questions are particularly relevant in the context of biologically-inspired computing machines because cells are central to all living organisms and drawing inspiration from them can only help us to achieve further progress in computer science.

The remainder of the paper is as follows: Section 2 provides a general introduction to cells and membrane systems. In Section 3, we consider the configuration (Section 3.1) and self-replication (Section 3.2) of membrane systems by means of self-inspection. Some rather general hardware and implementational issues are addressed in Section 4. Finally, Section 5 concludes the paper. In order to make the paper accessible to a broader range of readers not necessarily familiar with all the formal and theoretical aspects of membrane systems, we will reduce formalisms to a minimum and will not always stick to the standard membrane systems conventions since our approach is more guided by pragmatic considerations than by rigorous formalisms. Also, we will make use of some biological terms (e.g., cell, genome, development, evolution, etc.) whose meaning here does not necessarily correspond to its exact counterpart in nature.

2 Cells, Membranes, and Membrane Systems: An Overview

The cell is undoubtedly the structural and functional unit of all living organisms. Sometimes, the cell itself is used as a definition of life as all vital functions—including reproduction—of an organism occur within a cell, which is in particular true for single cell organisms. But no cell would exist without its surrounding membrane, which consists of a double layer of lipids (lipid bilayer). The *membrane* forms a boundary between the cell (the “self”) and its environment and controls what moves in and out. But the cell membrane also helps to define compartments and it forms a communication structure. The autonomous organization and self-maintenance of this boundary is an indispensable feature for living cells. Finally, the *cell membrane* is not to be confound with the *cell wall*, which is the rigid structure made up of polysaccharides that provides and maintains the shape of the cell and serves as a protective barrier.

The biological cell has a long influence in computer science: both spiritual fathers of modern computer science, Alan Turing and John von Neumann, had drawn inspiration from cells for their work. In his seminal 1952 paper entitled “The Chemical Basis of Morphogenesis” (Turing, 1952), Turing proposed a mathematical theory of cell-cell interaction via chemical substances (also called *reaction-diffusion model*). Von Neumann, on the other hand, pioneered *cellular automata (CA)* (von Neumann, 1966). The list of work involving cells in computer science is virtually endless and includes basically all aspects of a living cell, either with the goal to faithfully model its biological counterpart or to draw inspiration from it in order to endow machines with new properties. However, fairly little work in computer science has been focused on the cellular membrane *per se*. To the best of our knowledge, Varela, Maturana, and Uribe (Varela et al., 1974) were the first to emphasize the cellular membrane (i.e., the boundary between the “self” and the “environment”) and to model them in a very abstract way in their *autopoietic systems*, where identifiable boundaries are a key element. Much later, in 1998, Paun initiated *membrane computing* or *P systems* (Paun, 2000, 2002; Paun and Rozenberg, 2002) as an abstract computational model afar inspired by biochemistry and by some of the basic features of biological membranes.

A typical membrane system consists of cell-like membranes placed inside a unique “skin” membrane. Multisets of *objects*—usually strings of symbols—and a set of *evolution rules*² are then placed inside the regions delimited by the membranes. Each object can be transformed into other objects, can pass through a membrane, or can dissolve or create membranes. The evolution between system configurations is done nondeterministically by applying the rules in parallel for all objects able to evolve. A membrane system is also commonly considered as a form of an *artificial chemistry*: “An artificial chemistry is a man-made system which is similar to a real chemical system” (Dittrich et al., 2001). Artificial chemistries are a very general formulation of abstract systems of objects which follow arbitrary rules of interaction. They basically consist of a set of molecules S , a set of rules R , and a definition of the reactor algorithm A . In this context, a sequence of transitions in a membrane system is called a *computation*. A computation *halts* when a halting configuration is reached, i.e., when no rule can be applied in any region. A computation is considered successful if and only if it halts. However, in the context of intelligent agents, for example, one might imagine systems with a continuous stream of inputs and outputs and no halting configuration.

Membrane systems are particularly interesting for the creation of hierarchies, which we consider a key for the creation of complex systems. Hierarchical composition is ubiquitous in physical and biological systems: the nonlinear

² This term is commonly used for membrane systems and artificial chemistries, but *developmental rules* would be more appropriate from a biological point of view.

dynamics at each level of description generates emergent structure, and the nonlinear interactions among these structures provide a basis for the dynamics at the next higher level (Scott, 1999). But hierarchies are also a means to divide and “hide” complexity, to save resources as the building-blocks might be shared and re-used, and to create higher levels of abstraction. Among the drawbacks of membrane systems figures the absence of methodologies and tools to “program” and develop larger membrane systems and the lack of practical applications.

Finally, note that a wide variety of membrane system flavors exist today. The interested reader is referred to the P systems website³ or to (Paun, 2002; Paun and Rozenberg, 2002) for further details.

3 A Macroscopic View of Self-Configuring and Self-Replicating Membrane Systems

John von Neumann was not only one of the spiritual fathers of modern computing science but also the first who conceived a *cellular automaton (CA)* able to self-replicate (von Neumann, 1966). Self-replication is definitely a key mechanism for dividing an artificial or living cell into two daughter cells, but also plays a central role in the growth and repair of organisms. In artificial organisms, self-replication exclusively deals with information so far, but things might change with increasing progress in the field of nanotechnology. Following von Neumann’s self-replicating automaton, Langton (Langton, 1984) provided a much simpler, though not universal, self-replicating loop. Recently, several drawbacks of Langton’s loop have been addressed with the *Tom Thumb loop*, initiated by Mange *et al.* (Mange *et al.*, 2004). In the field of cellular automata, there exist basically two ways of realizing self-replicating systems: (1) using universal constructors that require a description of the system itself (Langton, 1984; von Neumann, 1966) and (2) self-inspection based methods, e.g., (Macias and Durbeck, 2004; Morita and Imai, 1996).

Self-replication and -configuration in membrane systems has only recently attracted some interest (e.g., (Csuha-j-Varjú *et al.*, 2005)), but now even figures as a specific problem in (Paun, 2005) (Problem U). As Paun writes, “[t]he motivation is not only related to the reproduction question in cellular automata, but also the complexity issues.” Whereas Csuha-j-Varjú *et al.* (Csuha-j-Varjú *et al.*, 2005) were mainly interested in transforming one membrane system configuration into another, we will concentrate on construction (equivalent to “development” in this context) and self-replication. The next two sections will provide a macroscopic view on how to build membrane systems from a single membrane and how a membrane system can replicate itself in a given environment. Some of the microscopic considerations shall be addressed in Section 4. The motivations for this are twofold: (1) the construction of membrane

³ <http://psystems.disco.unimib.it>

systems has been clearly neglected in the past, (2) construction is key in the genotype-to-phenotype mapping when applying evolutionary algorithms, and (3) self-replication can be used for building complex organisms and plays a crucial rule for dividing cells, growth, and repair.

Unless otherwise stated, we will use the conventions and operating modes of traditional membrane systems.

3.1 Configuring a Membrane System

As an illustrative example, let us assume that we would like to construct the initial membrane system configuration as depicted in Figure 1 (right) from a single membrane system (i.e., called “mother cell”, left). The following questions immediately arise: (1) how do we represent the description of the membrane system in a compacted form (i.e., the genome), (2) how do we build the membrane system’s initial configuration from this description, and (3) how does the system know when the configuration is terminated and when it can start evolving in “normal” mode? We shall name the process of building an initial membrane system configuration a *configuration* or *self-configuration*. Thus *configuring* (or *self-configuring*) the system happens *before* the membrane system starts evolving its initial configuration. From a more biological point of view, both steps together are similar to the development of a cell’s structure and functionality.

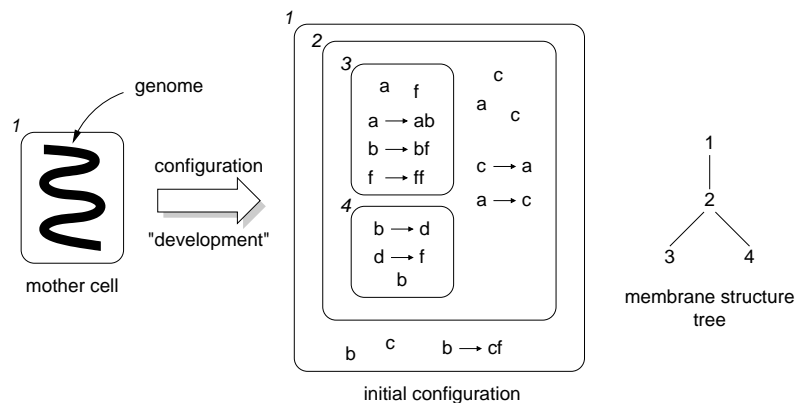


Figure 1. Constructing a membrane system from a single membrane (mother cell), which contains the description of the entire system to be constructed.

As explained for example in (Paun, 2002, p. 301ff), the membrane creation rule $\text{cre} = a \rightarrow [{}_i v]_i$, where a is an object, v is a string representing a multiset of objects, and i is the membrane label, allows to create a new membrane with the label i and the contents as given in v . For our purposes, the string v can not only contain objects but also evolution rules, e.g., $v = ab^2c(a \rightarrow b)(b \rightarrow c)$. For a better readability, the evolution rules shall be put in parenthesis.

The multisets of objects and evolution rules of each membrane compartment

in Figure 1 are as following:

$$\begin{aligned}
v_1 &= bc(b \rightarrow cf) \\
v_2 &= ac^2(c \rightarrow a)(a \rightarrow c) \\
v_3 &= af(a \rightarrow ab)(b \rightarrow bf)(f \rightarrow f^2) \\
v_4 &= b(b \rightarrow d)(d \rightarrow f)
\end{aligned}$$

Now, let us assume that the skin membrane (1) has already been created and that it contains a description in some form of the entire membrane system to be built. What are the initial multisets of rules r_1 and objects w_1 that would have to be placed inside this single membrane system in order to obtain the desired initial configuration after some time? A straightforward solution is as following.

$$\begin{aligned}
w_1 &= \mathbf{M1} \\
r_1 &= (\mathbf{M1} \rightarrow v_1[_2v_2\mathbf{M3}\mathbf{M4}(\mathbf{M3} \rightarrow [_3v_3]_3)(\mathbf{M4} \rightarrow [_4v_4]_4)]_2)
\end{aligned}$$

The symbols $\mathbf{M1}$, $\mathbf{M2}$, and $\mathbf{M3}$ are used as auxiliary symbols for initiating the membrane creation process in each membrane compartment, v_1, v_2, v_3, v_4 are the multisets of rules and objects as specified above. The only symbol contained in the multiset w_1 (i.e., $\mathbf{M1}$) shall be called *seed symbol*, the rule r_1 *genome*. The very first step consists in applying rule r_1 to the only object $\mathbf{M1}$ present in the single membrane system. This step creates membrane number two and puts the “building plans” for the next inner cells within its compartment. The process continues until membranes three and four are created. From a mathematical point of view, this can be compared to a recursive construction of the underlying tree structure that represents the membranes. Note that the construction rules remain in the compartments after a successful construction.

The attentive reader has hopefully noticed that during the construction process, the membranes whose construction has already been completed, will start evolving. Assuming that creating a new membrane can be done within one time step (i.e., one macro-step), the construction time might play a role for deeply nested hierarchies, although one might always take that into consideration while designing the chemistry. Here we propose as an illustrative example another pragmatic solution by introducing a special object \mathbf{D}^T , that prevents the chemistry to evolve in the compartment where this object is present as long as $T > 0$. T , which specifies the number of macro-steps the cell will be inactive, is automatically decremented during each macro-step. One can compare this symbol with a counter that decrements T at each time step until it reaches

0. The symbol is automatically removed from the chemistry once $T = 0$. By expanding the multisets of objects in the genome as following, the membrane system will first be completely built before it starts evolving:

$$\begin{aligned}v_1 &= bc(b \rightarrow cf)D^2 \\v_2 &= ac^2(c \rightarrow a)(a \rightarrow c)D^1\end{aligned}$$

Since the multisets v_3 and v_4 are at the lowest level in the hierarchy (i.e., the membrane structure's tree leaves), no delay is necessary for them and they can start evolving immediately once created. Membrane 2 (v_2) requires a one-unit delay, the outer membrane compartment v_1 two-units.

Note that there are other ways to implement delays and to stop the evolution of the chemistry. In particular, one might convert all rules (e.g., $a \rightarrow b$) into co-operative rules (e.g., $at \rightarrow b$) by adding a special symbol (e.g., t), which has to be present in the compartment for the rules to be applied. These symbols would then be generated by another special rule.

3.2 Self-Inspection and Self-Replication

Csuhaj-Varjú *et al.* (Csuhaj-Varjú *et al.*, 2005) proposed a divide-rule, $[_ha]_h \rightarrow [_{h'}b]_{h'}[_{h''}c]_{h''}$, which replicates objects and membranes in h alike and puts them into two new membranes h' and h'' , at the exception of object a , which is replaced by b in h' and by c in h'' . We feel that such a rule is certainly useful from the theoretical point of view, but that (1) it leaves out very important aspects of self-replication, (2) that the rule is too complex compared to other rules to be implemented with a reasonable number of micro-steps, and (3) that it is too complicated to be directly implemented in hardware. We therefore propose a more pragmatic and hardware-oriented approach in this section, which allows to self-replicate a membrane system (i.e. obtain an identical copy) by means of self-inspection.

In the previous section, the configuration process started from a single membrane (i.e., the *mother cell*) that contained a single *seed symbol* and a single evolution rule called *genome*. In this section, we are interested in how we can obtain the genome and the seed symbol from an existing cell by using a sequence of rather simple instructions, which would be implementable by a reasonable number of micro-steps. The situation is illustrated in Figure 2.

The first step of the process will consist in inspecting the current membrane system and in creating a genome-rule in the skin compartment, which will then be used to create a new mother cell in the environment outside the current membrane. In the following steps, the mother cell will develop, i.e.,

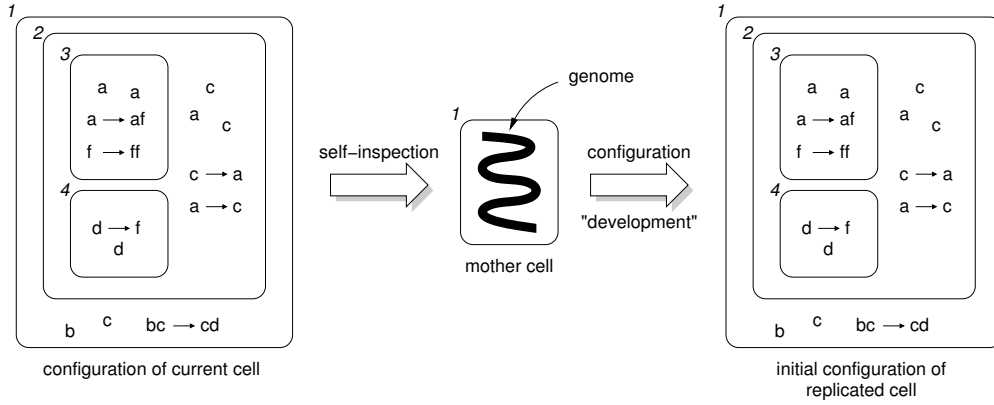


Figure 2. Self-replicating an existing membrane system by self-inspection and subsequent configuration.

self-configure, until the membrane system's initial configuration is reached (as explained in the previous section). As a result, we will end up with two potentially identical membrane systems. Note that depending on whether the initial membrane system was in a halting configuration and whether the replicated membrane system starts evolving during the configuration process already, there might never exist an instant when the two membrane systems are strictly identical. In order to self-replicate a membrane system, we basically need the following macroscopic mechanisms: (1) initiate the process of self-inspection; (2) copy the contents (i.e., objects and rules) of a given compartment; (3) incrementally compose the genome; and (4) create the mother cell outside the current skin membrane.

Let us introduce a new rule, $\mathbf{gen} = a \rightarrow (DC, Mi, out)$, which allows to duplicate all objects and evolution rules in the current compartment and which sends the configuration in the form of a special rule to the outer compartment. More specifically, the \mathbf{gen} -rule does the following in one macro-step:

- (1) duplicates all objects and evolution rules in the current compartment i and puts them together in a temporary and virtual multiset DC ;
- (2) if a rule of the form $Mj \rightarrow [j \dots]_j$ is present in the current compartment, an additional symbol Mj , i.e., the seed symbol, will be added to DC and the rule will be removed from the compartment (but remains in DC);
- (3) it sends the rule, $Mi \rightarrow [{}_iDC]_i$ to the outer compartment; and
- (4) if a rule of the form $Mi \rightarrow [i \dots]_i$ was already present in the outer compartment, it will simply be replaced by the new rule.

In other words, this rule allows to incrementally compose the genome from the bottom of the membrane structure to the top by assembling each membrane's configuration on the way in the form of a rule, which will then be used during the construction process as a genome. Here, the symbol Mi is a symbol, i.e., the seed symbol, not in the symbol alphabet, which is unique for each membrane. The rule of the form $Mi \rightarrow [i \dots]_i$ together with the symbol Mi allows to create

and configure a new membrane i during the construction of the replicated cell. The second step prevents the rule to be already used during the self-inspection process. Also, the rule has to be removed because it is not part of the actual configuration and would otherwise be accumulated in case of multiple replication steps. Figure 3 illustrates the application of the **gen**-rule in a simplified setting. The rule sends the configuration of compartment 2 to the outer compartment. Since no rule of the form $Mj \rightarrow [j \dots]_j$ is present in the current compartment, no Mj -symbol is added.

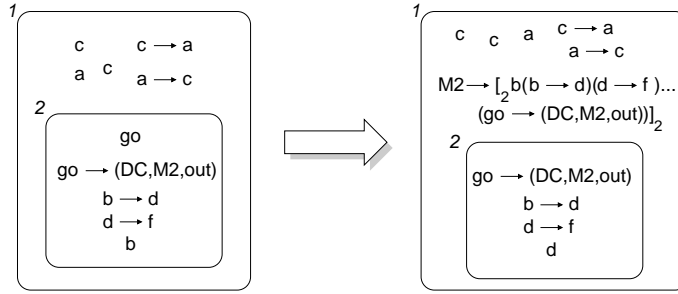


Figure 3. Illustration of the **gen**-rule. The rule basically sends a copy of the current objects and evolution rules in the form of a rule to the outer compartment, which creates a new membrane system in the environment.

If the rule is applied in the top compartment, i.e., the skin compartment 1, the resulting $M1 \rightarrow [1 \dots]_1$ rule, i.e., the complete genome, as well as a seed symbol $M1$ will be sent to the environment. Once outside the skin membrane, the rule will be immediately applied because of the seed symbol, and a new membrane will be created that contains the genome and the seed symbol for the further construction of the new membrane system (see also Section 3.1). Figure 4 illustrates this procedure, which is not entirely biologically plausible as it involves a moment where the seed symbol and the genome are not surrounded by a membrane, but we feel that this represents an acceptable and more straightforward solution than to introduce another rule, such as $a \rightarrow [1 \dots]_1$, for this special case.

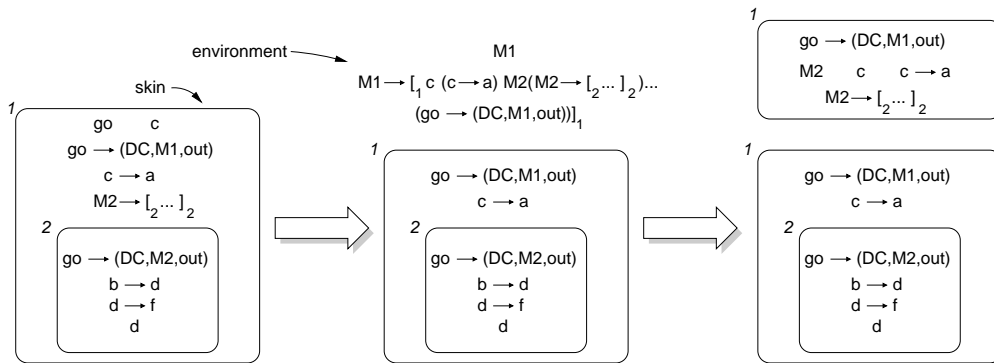


Figure 4. Illustration of the **gen**-rule when applied in the skin compartment.

In order for the **gen**-rule to be applied in the correct order (i.e., from the bottom to the top compartment), we need to pre-configure the membrane sys-

tem with additional symbols and rules. Let us assume that the self-inspection process will be initiated at the bottom of the membrane system, i.e., at the leaves of the underlying tree structure. In Figure 1, the process would thus start in membranes 3 and 4. The entire configuration is then incrementally put together by moving from the leaves to the root of the tree. Here we propose a self-timed approach, which is illustrated in Figure 5. The idea is that each compartment contains a **gen**-rule which is being activated by a special symbol (i.e., *go*). The entire process is initiated by generating a *go*-symbol in each leaf of the tree, which then triggers the sequential activation of the **gen**-rules in each compartment. Obviously, if the membrane system changes its own configuration during its normal evolution (i.e., by adding or removing membranes), the **gen**-rules will have to be modified accordingly, which might not be a trivial undertaking in all cases. For example, if a membrane with label 5 is added in compartment 2 of Figure 5, then the rule would have to be changed from $go^2 \rightarrow \dots$ to $go^3 \rightarrow \dots$ and the new membrane would also have to contain a **gen**-rule. Finally, sending the *go*-symbol from, for example, the skin compartment to the inner-most membranes, might be realized by means of additional rules. This shall not be detailed here.

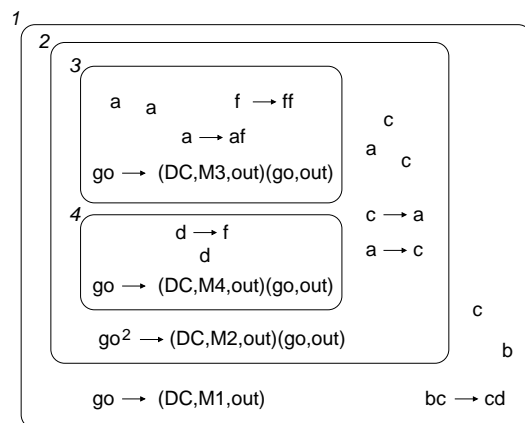


Figure 5. Each compartment has to be pre-configured with a **gen**-rule in order to allow the membrane system to self-replicate by self-inspection. The process is initiated by generating a *go*-symbol in compartment 3 and 4.

4 Towards Hardware Membrane Systems: A Microscopic View

In most computational models driven by abstract considerations, the notions of “space,” “size,” and “resources” is neglected as there is no need for it. For example, it is irrelevant in a cellular automata model or in a neural network what the dimension of a cell or neuron is. It only matters how it is interconnected with its neighbors and what its functionality is. Likewise, the cells of a membrane system have no particular size attributed and they possess a potentially unlimited capacity when it comes to host symbols and reactions, which is of course not biologically plausible. However, considering space in computational models can be beneficial because “computing in the space do-

main” is often more attractive than “computing in the time domain,” it makes the models more realistic, and it is very helpful when it comes to genuine implementations, where space has to be considered and limited resources are a matter of fact.

As Paun states in (Paun, 2002, p. 367), “[...] membrane computing was a theoretical computer science enterprise, aiming to provide new computational paradigms [...]” The question of how much “added value” (e.g., speed, resources, etc.) a genuine hardware implementation yields with regards to a simulation might—and should—of course always be asked. The most important issue not to forget is, however, that no simulation can be done without real hardware. Thus, the question can basically be reformulated as: “What hardware is appropriate for which simulation?” Although implementing a membrane systems on a traditional sequential von Neumann computer or a distributed arrangement of (usually also sequential) machines is straightforward (Ciobanu and Paraschiv, 2002; Ciobanu and Wenyuan, 2003; Syropoulos et al., 2003; Syropoulos, 2004), it remains a simulation as there is not usually parallel hardware involved that could be fully exploited by the inherently parallel membrane systems and their artificial chemistries. As Paun writes, “[i]t is important to underline the fact that “implementing” a membrane system on an existing electronic computer cannot be a real implementation, it is merely a simulation. As long as we do not have genuinely parallel hardware on which the parallelism [...] of membrane systems could be realized, what we obtain cannot be more than simulations, thus losing the main, good features of membrane systems” (Paun, 2002, p. 379). We believe that high-level simulations of membrane systems is a perfectly valid proof of concept, but that the real challenge of “going back to reality” (see also (Paun, 2002, Chapter 9)) consists in building genuine hardware that is optimized for artificial chemistries and membrane systems. In 2004 (Petreska and Teuscher, 2004), Petreska and Teuscher have proposed a very first hardware realization of membrane systems using reconfigurable circuits, namely *Field Programmable Gate Arrays* (FPGAs). The implementation is based on a universal and minimal *membrane hardware component* that allows to very efficiently evolve membrane systems in hardware. Later, Teuscher (Teuscher, 2004) pioneered an unconventional architecture involving membranes, which was inspired by *amorphous computers* and randomly interconnected substrates. The architecture remains theoretical though, and a genuine hardware implementation has yet to be realized. Although the initial approach does make use of a special class of asynchronously operating membrane systems, we shall very briefly outline some key elements here as the basics are valid for all membrane systems.

The main idea is to implement the membranes on a fine-grained computational substrate of particles, which supports massive parallelism and allows to efficiently evolve the artificial chemistries. In principle, this can be done on a regular, cellular automata like arrangement of cells, however, a randomly ar-

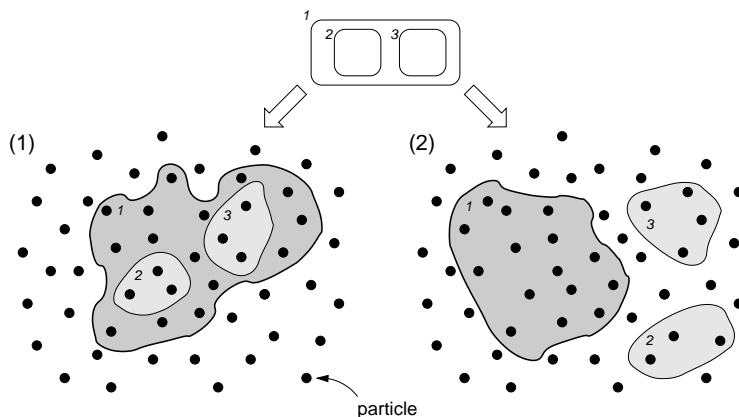


Figure 6. Two possibilities of implementing hierarchical membrane systems on a programmable reactor multitude.

ranged and interconnected set of particles offers several advantages, especially when failures in connections and particles occur and when dealing with redundancy. The approach used in (Teuscher, 2004) consisted in a *Programmable Reactor Multitude* (PRM), an irregular and locally interconnected ensemble of particles, whose main part is a chemical reactor. The objects and rules do stochastically move between all reactors—linked together to form a large, but distributed reactor—of a given membrane compartment. Figure 6 illustrates this as well as to possible ways of mapping hierarchically-organized cells (with a limited capacity for symbols and rules) on a particle ensemble. Possibility (1) seems more natural and simplifies communications, but when an inner cell has to be enlarged, the outer cells have to follow and make room, which can be tricky to implement. For more details, see also (Teuscher, 2004).

5 Conclusion and Perspectives

The present paper described a particular, pragmatic, and hardware-friendly method to self-configure and self-replicate traditional membrane systems by means of self-inspection. The advantage of the self-inspection approach is that there is no need to store the entire membrane system configuration (i.e., the genome) in the system itself, rather this information will be incrementally put together by self-inspection. In comparison to a single replication rule, as for example presented in (Csuhaaj-Varjú et al., 2005), which replicates an entire membrane system in one macro-step, we believe that our approach can be much easier realized in both simulations and hardware implementations because it involves the sequential execution of a series of rather simple rules instead of one single and complex rule.

Here, we have mainly focused on constructional (or developmental) aspects and not on manipulating membrane configurations in general. However, we plan to expand the current approach and to eventually realize most configuration-manipulation rules as presented in (Csuhaaj-Varjú et al., 2005) by a sequence

of more simple operations. Future work will also concentrate on the general challenge of how to program artificial chemistries, which is by all means a painstakingly difficult undertaking. Little work exists ((Busch and Banzhaf, 2003) is one example), but we believe that the most promising approach lies in the field of evolutionary algorithms. To the best of our knowledge, no one has considered evolving membrane systems so far. In combination with self-configuration and self-replication, very interesting settings could arise. Also, we believe that the robustness and reliability issues in membrane systems should be addressed, in particular when considering genuine hardware.

Acknowledgments

While at UCSD, the author was supported by the Swiss National Science Foundation under grant PBEL2-104420. He is grateful to the anonymous reviewers for their helpful comments and discussions.

References

- Banzhaf, W., Christaller, T., Dittrich, P., Kim, J. T., Ziegler, J. (Eds.), 2003. *Advances in Artificial Life. Proceedings of the 7th European Conference, ECAL2003*. Vol. 2801 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg.
- Busch, J., Banzhaf, W., 2003. How to program artificial chemistries. In: Banzhaf et al. (2003), pp. 20–30.
- Ciobanu, G., Paraschiv, D., 2002. Membrane software. A P system simulator. *Fundamenta Informaticae* 49 (1–3), 61–66.
- Ciobanu, G., Wenyuan, G., 2003. A parallel implementation of the transition P systems. In: Alhazov, A., Martín-Vide, C., Paun, G. (Eds.), *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)*. Vol. 28/03. Rovira i Virgili University, Research Group on Mathematical Linguistics, Tarragona (Spain), pp. 169–169.
- Csuhaj-Varjú, E., Nola, A. D., Paun, G., Pérez-Jiménez, M. J., Vaszil, G., 2005. Editing configurations of P systems, submitted.
- Dittrich, P., Ziegler, J., Banzhaf, W., 2001. Artificial chemistries—a review. *Artificial Life* 7 (3), 225–275.
- Franceschi, S. D., Kouwenhoven, L., June 13 2002. Electronics and the single atom. *Nature* 417, 701–702.
- Langton, C. G., January 1984. Self-reproduction in cellular automata. *Physica D* 10 (1–2), 135–144.
- Macias, N. J., Durbeck, L. J. K., March 2004. Adaptive methods for growing electronic circuits on an imperfect synthetic matrix. *Biosystems* 73 (3), 172–204.
- Mange, D., Stauffer, A., Peparolo, L., Tempesti, G., December 2004. A macroscopic view of self-replication. *Proceedings of the IEEE* 92 (12), 1929–1945.
- Mathur, N., October 10 2002. Beyond the silicon roadmap. *Nature* 419 (6907), 573–575.

- Morita, K., Imai, K., 1996. A simple self-reproduction space automaton with shape encoding mechanism. In: Langton, C. G., Shimohara, K. (Eds.), *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. Vol. 1. MIT Press, Cambridge, MA.
- Nagami, K., Oguri, K., Shiozawa, T., Ito, H., Konishi, R., 1998. Plastic cell architecture: Towards reconfigurable computing for general-purpose. In: Pocek, K. L., Arnold, J. (Eds.), *Proceedings of the 6th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM'98)*. IEEE Press, Los Alamitos, CA, pp. 68–77.
- Paun, G., 2000. Computing with membranes. *Journal of Computer and System Sciences* 61 (1), 108–143, first published in a TUCS Research Report, No 208, November 1998, <http://www.tucs.fi>.
- Paun, G., 2002. *Membrane Computing*. Springer-Verlag, Berlin, Heidelberg, Germany.
- Paun, G., 2005. Further twenty-six open problems in membrane computing. Available online: <http://psystems.disco.unimib.it>.
- Paun, G., Rozenberg, G., 2002. A guide to membrane computing. *Journal of Theoretical Computer Science* 287 (1), 73–100.
- Petreska, B., Teuscher, C., 2004. A reconfigurable hardware membrane system. In: Martin-Vide, C., Mauri, G., Paun, G., Rozenberg, G., Salomaa, A. (Eds.), *Membrane Computing*. Vol. 2933 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, pp. 269–285.
- Scott, A. C., 1999. *Nonlinear Science: Emergence and Dynamics of Coherent Structures*. Oxford University Press, Oxford.
- Syropoulos, A., June 14-16 2004. On P systems and distributed computing. In: *Pre-Proceedings of the 5th Workshop on Membrane Computing (WMC5)*. Milano, Italy.
- Syropoulos, A., Mamatas, E. G., Allilomes, P. C., Sotiriades, K. T., 2003. A distributed simulation of P systems. In: Alhazov, A., Martín-Vide, C., Paun, G. (Eds.), *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)*. Vol. 28/03. Rovira i Virgili University, Research Group on Mathematical Linguistics, Tarragona (Spain), pp. 455–460.
- Teuscher, C., 2004. *Amorphous membrane blending: From regular to irregular cellular computing machines*. Ph.D. thesis, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, thesis No 2925.
- Turing, A. M., 1952. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London B* 237, 37–72.
- Varela, F., Maturana, H., Uribe, R., 1974. Autopoiesis: The organization of living systems, its characterization and a model. *BioSystems* 5, 187–196.
- von Neumann, J., June 30 1945. First draft of a report on the EDVAC. Tech. rep., Moore School of Electrical Engineering, University of Pennsylvania.
- von Neumann, J., 1966. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Illinois.