Networks of Evolutionary Processors. *UML* architecture*

MIGUEL ANEL PEÑA
Escuela de Informática
Dept. OEI - UPM
Crta. de Valencia km. 7
28031 Madrid - Spain

MIGUEL ANGEL DIAZ
Escuela de Informática
Dept. OEI - UPM
Crta. de Valencia km. 7
28031 Madrid - Spain

NURIA GOMEZ BLAS
Escuela de Informática
Dept. OEI - UPM
Crta. de Valencia km. 7
28031 Madrid - Spain

Abstract: This paper presents a connectionist model that is widely used to solve *NP*-problems: Networks of Evolutionary Processors (*NEP*). A *NEP* is a set of processors connected by a graph, each processor only deals with symbolic information using rules, that is, symbols in a processor evolve according some rules. Communication through the graph is based on a filter set that permits (or not) symbols to go from one processor to another. This two steps: evolution and communication, makes *NEPs* able to solve *NP*-problems. This paper also proposes an *UML* architecture to implement *NEPs*. This work is actually under development and will be implemented in a distributed way, maybe using *Java Agent Development Enviroment - JADE*, in order to obtain similar results to theoretical ones: solution of *NP*-problems in linear time. Proposed *UML* architecture is based on the general behaviour of *NEPs*.

Key–Words: Natural Computing, Evolutionary Processors, UML Architecture

1 Introduction

Membrane Computing is inspired in the structure and functioning of living cells [1]. 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 commu-

*This work has been partially supported by Spanish Grant TIC2003-09319-c03-03.

nication 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 [2]. 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).

1.1 From membranes to processors

The origin of networks of evolutionary processors (*NEP* for short) is a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [6] as well as the Logic Flow paradigm [3], which consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules, and then local data becomes a mobile agent which can navigate in the network following a given protocol. Only such data can be communicated which can pass a filtering process. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see, e.g., [4, 6].

Each node in *NEPs* may be viewed as a cell having genetic information encoded in *DNA* sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of words (each word appears in an arbitrarily large number of copies), and all the copies are processed in parallel such that all the possible events that can take place do actually take place. Obviously, the computational process just described is not exactly an evolutionary process in the *Darwinian* sense. But the rewriting operations we have considered might be interpreted as mutations and the filtering process might be viewed as a selection process. Recombination is missing but it was asserted that evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration [9].

2 Networks of Evolutionary Processors

A network of evolutionary processors [5, 7, 8] of size n is a construct $\Gamma = (V, N_1, N_2, \cdots, N_n, G)$, where V is an alphabet and for each $1 \leq i \leq n$, $N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:

M_i is a finite set of evolution rules of one of the following forms only

- $a \to b, a, b \in V$ (substitution rules)
- $a \to \epsilon, a \in V$ (deletion rules)
- $\epsilon \to a, a \in V$ (insertion rules)

More clearly, the set of evolution rules of any processor contains either substitution or deletion or insertion rules.

A_i is a finite set of strings over V. The set A_i is the set of initial strings in the i-th node. Actually, in what follows, we consider that each string appearing in any node at any step has an arbitrarily large number of copies in that node, so that we shall identify multisets by their supports.

PI_i and PO_i are subsets of V^* representing the input and the output filter, respectively. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i (w \in PO_i)$.

Finally, $G = (\{N_1, N_2, \cdots, N_n\}, E)$ is an undirected graph called the underlying graph of the network. The edges of G, that is the elements of E, are given in the form of sets of two nodes. The complete graph with n vertices is denoted by K_n. By a configuration (state) of an *NEP* as above we mean an n-tuple $C = (L_1, L_2, \cdots, L_n)$, with

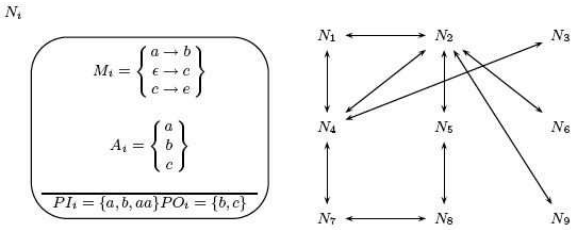

Figure 1: Network of Evolutionary Processors Architecture: (right) communication graph, (left) processor internals.

$L_i \subseteq V^*$ for all $1 \leq i \leq n$. A configuration represents the sets of strings (remember that each string appears in an arbitrarily large number of copies) which are present in any node at a given moment; clearly the initial configuration of the network is $C_0 = (A_1, A_2, \cdots, A_n)$ (see figure 1).

A configuration can change either by an evolutionary step or by a communicating step. When changing by an evolutionary step, each component L_i of the configuration is changed in accordance with the evolutionary rules associated with the node i. When changing by a communication step, each node processor N_i sends all copies of the strings it has which are able to pass its output filter to all the node processors connected to N_i and receives all copies of the strings sent by any node processor connected with N_i providing that they can pass its input filter.

Theorem 1 *Each recursively enumerable language can be generated by a complete NEP of size* 5.

A *NEP* architecture is given by the shape of the underlying graph defined by G, there are some classical architectures such as: complete, star, diamond, etc.

Theorem 2 *Each recursively enumerable lan-*

guage can be generated by a star NEP of size 5.

And a result concerning the universality of Simple *NEP* is given by the *PCP* problem by the following theorem:

Theorem 3 *The bounded* PCP *can be solved by an NEP in size and time linearly bounded by the product of K and the length of the longest string of the two Post lists.*

2.1 Simple NEP

A simple *NEP* of size n is a construct $\Gamma = (V, N_1, N_2, \cdots, N_n, G)$, where, V and G have the same interpretation as for *NEPs*, and for each $1 \leq i \leq n, N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i)$ is the i-th evolutionary node processor of the network. M_i and A_i from above have the same interpretation as for an evolutionary node in a *NEP*, but

- PI_i and FI_i are subsets of V representing the input filter. This filter, as well as the output filter, is defined by random context conditions, PI_i forms the permitting context condition and FI_i forms the forbidding context condition. A string $w \in V^*$ can pass the input filter of the node processor i, if w contains each element of PI_i but no element of FI_i. Note that any of the random context conditions may be empty, in this case the corresponding context check is omitted. We write $\rho_i(w) = true$, if w can pass the input filter of the node processor i and $\rho_i(w) = false$, otherwise.
- PO_i and FO_i are subsets of V representing the output filter. Analogously, a string can pass the output filter of a node processor if it satisfies the random context conditions associated with that node. Similarly, we write $\tau_i(w) = true$, if w can pass the input filter of the node processor i and $\tau_i(w) = false$, otherwise.

Some important results are:

Theorem 4 *The families of regular and context-free languages are incomparable with the family of languages generated by simple* NEPs.

And a result concerning the universality of Simple *NEP* is given by the 3-colors problem by the following theorem:

Theorem 5 *The "3-colorability problem" can be solved in $O(m + n)$ time by a complete simple* NEP *of size $7m + 2$, where n is the number of vertices and m is the number of edges of the input graph.*

3 UML Architecture

Generic evolutionary processors can be defined as $N = (M, A, PI, PO)$, where M represents the evolution rules, A is the strings in the node, PI and PO are the input and output filter respectively. The *UML* class diagram is shown in figure 2.

A string would be represented like symbol, that is the reason why a processor will contain so many symbols as strings it has. The rules are divided in two parts: the antecedent that represents the part that must fulfil for being able to apply the rule, and the consequent that is the

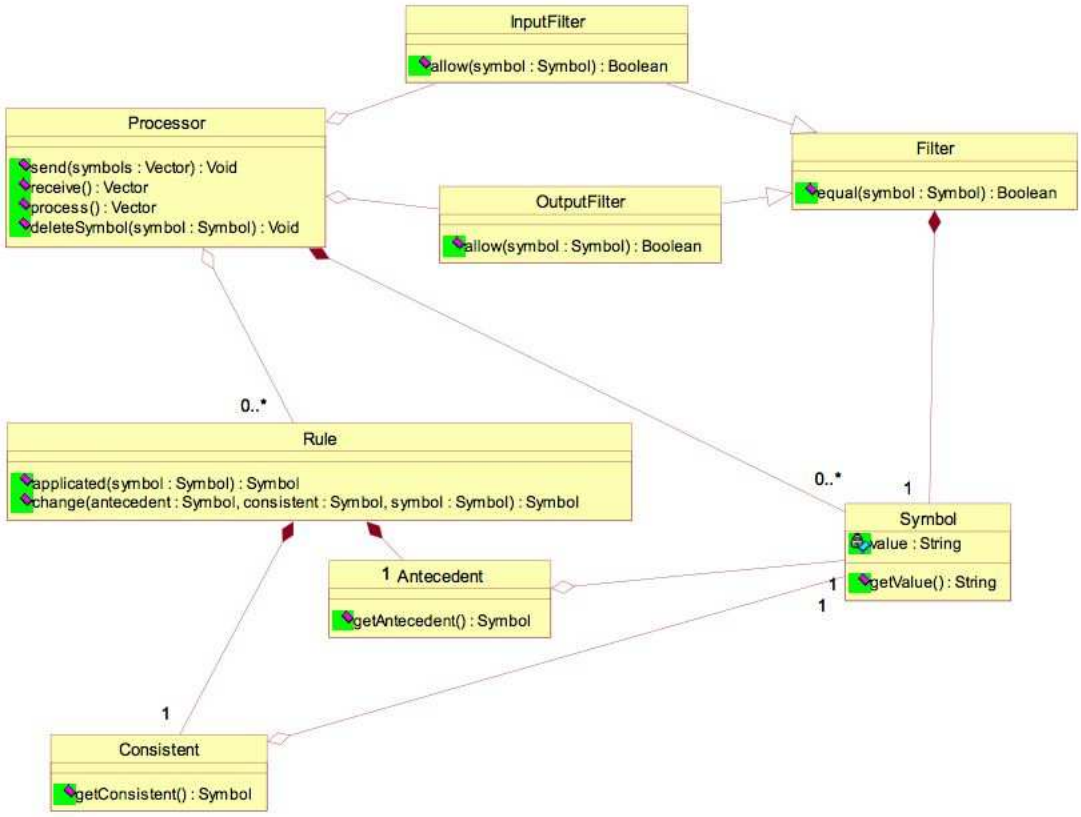

Figure 2: Class Diagram of *NEPs*.

reason why the antecedent will be replaced if the rule is applied. Both, antecedent and consequent in evolutionary processors are not more than a symbol.

Several filters can be implemented in the evolutionary processors. A filter is a system that allows a symbol to go from one processor to another. Normally, the detection system is to compare a symbol with another one. Among the possible filters that an evolutionary processor can have, most common filters are the PI or input filters, and the PO or output filters. A processor can have several filters of each type.

This is the basic composition of an evolutionary processor, nevertheless, there exist *NEP* architectures that have forbidden filters in the input and in the output. Differences in the implementation for the resolution of problems will be defined as types of the generic model for such given kind of problems.

Greater differences among the types of evolutionary processors appear in the form they work. The process that is developed in an evolutionary processor can be divided in 3 parts: to receive the string, to apply the rules and to communicate the new string. At the outset, the processor will contain a series of string. These rules will be applied and will be sent to the rest of processors that are related with this. The process in the processor will begin again, such and as it shows in figure 3.

The processor will receive the chains and send them to the rest of evolutionary processors who is connected. The processor works with a string, this will have to fulfil the input filters. Thus, all the string will be validated with respect to the filters and those will discard strings that do not fulfil the conditions. The PI simply will compare the symbol that receives with those it contains, if both are equal, the string will be valid. On the contrary, the string will be verified with another filter.

There are, according to the objective of the network of evolutionary processors another form of verification of the string. A string can be formed by several concatenated symbols, called substring. The input filter instead of being formed by complete strings can contain substrings. Then, a string is admitted as a valid one for a processor if each one of the symbols or a substring is present in the input filters.

Also, forbidden input filters (FI) can exist, that is, those that contains the substring that the input string cannot contain. If the string is not valid according to a given forbidden filter after the matching process , that string will be immediately discarded. It is also possible that the substring is not present in any PI, in such case all the string would be accepted.

Once the evolutionary processor has all strings, rules start to be applied in a not deterministic form. The rule and the string are selected on a random way. The form to apply a rule is by means of comparison and substitution. The antecedent of the rule is compared to the string, and in the case of equality the string is replaced by the consequent. The antecedent can also be formed by substrings; in such case, it replaces only the substring that matches the antecedent by the consequent. The antecedent and the consequent can be the empty string, in such case the rules are denominated insert and delete rules respectively. Usually, the substitution rules are applied in any position of the chain, whereas those of insertion and delete rules are only applied in the right end, as in the implementation to solve the *3-colorability problem*.

The evolutionary processors can be divided in two types:

- those who apply all the rules in a non deterministic form and they only communicate strings when more rules cannot be applied,
- and those that applies a rule and communicates the string.

Once, all the possible rules have been applied to or only a rule is applied, based on the operation mode, strings are communicated to another evolutionary processors. Before the communication, strings must pass through the output filters and only strings that satisfied these filters are sent out. The forms and types of the output filters are similar to those of the input filters.

4 Conclusions

This paper has introduced the novel computational paradigm Networks of Evolutionay Pro-

cessors that is able to solve *NP*-problems in linear time. The implementation of such model in a traditional computer is being performed and this paper shows an *UML* architecture. This architecture is a generic representation of the behaviour of the *NEPs*, and tanking into account all the possible specific implementations based on this model all the cases that can appear in the article could be contemplated. Obviously, theoretical results will never been achieved due to the sequential operational behaviour of traditional computers, but some improvement can be done in a distributed implementation.

References

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

[2] Gh. Paun. *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, (2002).

[3] L. Errico and C. Jesshope. *Towards a new architec- ture for symbolic processing*. Artificial Intelligence and Information-Control Systems of Robots 94, 3140, World Scientific, Singapore. (1994).

[4] S. Fahlman, G. Hinton, and T. Seijnowski. *Massively par- allel architectures for AI: NETL, THISTLE and Boltzmann machines*. Proc. AAAI National Conf. on AI, 1983:109113, William Kaufman, Los Altos. (1983).

[5] M. Garey and D. Johnson. *Computers and Intractability. A Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, (1979).

[6] W. Hillis. *The Connection Machine*. MIT Press, Cambridge, (1985).

[7] F. Manea, C. Mart?n-Vide, and V. Mitrana. *All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size*. Proc. of DNA 12, in press.

[8] C. Martín-Vide and V. Mitrana. *Networks of evolutionary processors: Results and perspectives. Molecular Computa- tional Models: Unconventional Approaches* 78114, Idea Group Publishing, Hershey. (2005).

[9] D. Sankoff and et al. *Gene order comparisons for phylo- genetic inference: Evolution of the mitochondrial genome*. Proc. Natl. Acad. Sci. USA, 89:65756579. (1992).

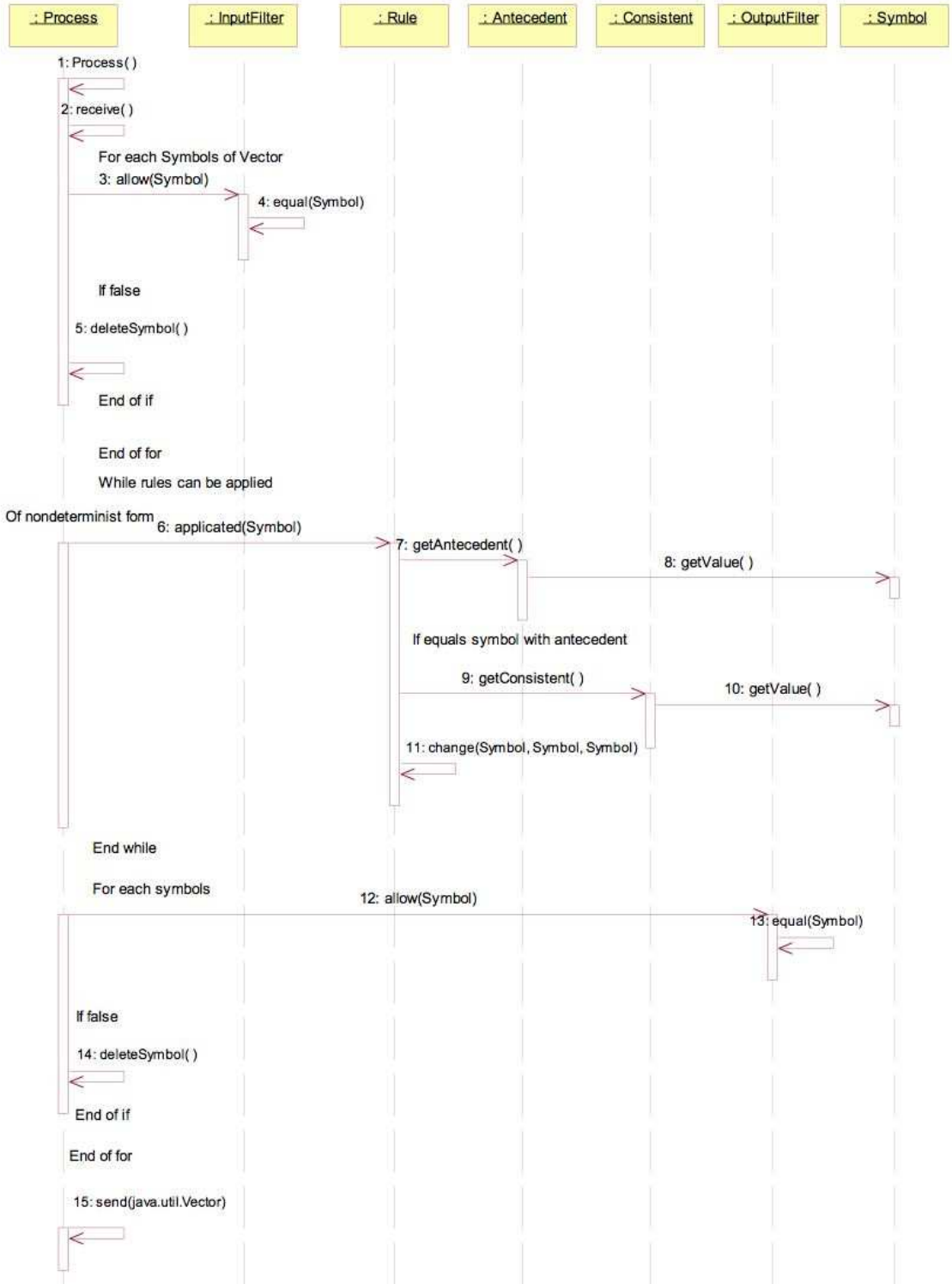

Figure 3: Sequence Diagram of *NEPs*.