USE PROXIES IN TREE TOPOLOGY ARCHITECTURES TO REDUCE THE COMMUNICATION TIME IN MEMBRANES SYSTEMS

Miguel Ángel Peña, Ángel Castellanos, Alberto Arteta, Francisco Gisbert

***Abstract**: There are two architectures that preserve the tree topology of the P system: peer-to-peer and hierarchical peer-to-peer. Architectures used the concept of proxy to reduce the size of messages and communication time. In this paper, we propose that the proxy no only reduces the time needed to sending messages, but also reduce the number of connections between processors. Normally, in one transition, two processors communicate three times, but with this algorithm, the communications are reducing to only one. We present the proxy algorithms for these architectures.*

***Keywords**: Distributed Communication, Membrane Computing, Membrane Dissolution, P-Systems Architectures.*

***ACM Classification Keywords**: F.1.2 Modes of Computation, I.6.1 Simulation Theory, H.1.1 Systems and Information Theory, C.2.4 Distributed Systems*

Introduction

In 1998 Gheorghe Păun introduced membrane computing [Păun, 2000] as a parallel computing model based on the biological notion of the cell. On the original model, there have been several variations in order to solve various problems, and improve computation times, in order to solve complex problems such as NP-complete, in similar times to the polynomial.

The idea behind a membrane system is based on the permeability of the same, and those internal changes whose taking place. Depending on the elements that are working, we can distinguish two main types, those which manipulate objects and those which work with strings. The behavior is similar in both cases. In parallel, each membrane performs a series of rules with its own objects or strings, resulting in other objects or strings, which, using the permeability of the membrane can move to other membrane if it was indicated in their transformation rules. The great advantage of these systems is that each membrane is run independently of the others, so the runtime does not depend on the number of membranes.

We can divide the dynamics of P System in two steps: the internal application to the membrane and communication between membranes. The first step is the allocation of objects in active rules (which imply that they are useful and applicable) and the creation of new objects. In the communication step, the membrane communicates the new items to their corresponding membrane due to the capabilities of membrane permeability and dissolution of the membranes.

To reduce the execution time of P System are used distributed implementations, such as [Syropoulos, 2004] and [Ciobanu, 2004]. In the use of these implementations has been observed network congestion. To solve these problems, several authors have proposed architectures where communication takes place without collisions. The first example is the Peer-to-Peer Architecture [Tejedor, 2008] which was introduced by Tejedor. Based on this

architecture, Bravo proposed the Hierarchical Peer-to-Peer Architecture [Bravo, 2007]. Both architectures make use of the proxy concept introduced by Tejedor [Tejedor, 2008] so that "when a membrane wants to communicate with another one located at a different processor, the first one uses a proxy".

Communication between membranes is done internally in the processor and externally by proxy. Thus, for the communication of new objects will use the proxy, and for the dissolution of membranes will be used again. In addition to determining the useful rules, the membrane needs to know the existence of other membranes in the P System. According [Frutos, 2009] would be necessary to know the total context for the selection of rules. In the case of using any of the above referenced distributed architectures, it is necessary communications between distributed teams to determine the presence or absence of membranes.

Thus, there are three types of communications between processors: object communication, dissolution of membranes and knowledge of whether or not exist other relative membranes to determine the total context. All these communications can be reduced to only three (one of each type) through the use of proxy. In this paper we propose a variant of the proxy, so that the three communications that are made in each evolution of the P System can be done in just one. Because each one of the above architectures has differentiating features, the proxy implementation, even following the same idea, should be done differently. Therefore, it will be explained the behavior of this proxy for each one of the four indicated architectures.

P System definition

The first definition of a P System was published by Păun [Păun, 2000], who defined a Transition P System as:

Definition: A Transition P System is $\Pi = (V, \mu, \omega_1,\ldots, \omega_n; (R_1, \rho_1),\ldots, (R_n; \rho_n); i_0)$, where:

V is an alphabet (composed of objects).

μ is the membrane structure with n membranes.

ω_i are the multiset of symbols for the membrane i.

R_i are the evolution rules for the membrane i. A rule is a sorted pair (u,v) where u is a string over V, and $v = v'$ or $v = v'\delta$ and v' is a string over $V_{TAR}= V \times TAR$ with $TAR = \{here, out\} \cup \{in_j \mid 1\leq j\leq n\}$. δ is a special symbol no include in V, and represent the dissolution of the membrane.

ρ_1 are the priority of rules for the membrane i.

i_0 indicates a membrane, which is the system output membrane or skin membrane.

Proxy: General Idea

A proxy is a program or device located in the processor that carries out an action in representation of another. [Tejedor, 2008] uses this idea to reduce the time when the membrane send objects to other membranes located in other processor. This idea is used in all architectures, but Tejedor did not specify anything about the other data pieces between processors. Figure 1 shows an example of a distributed P system in 4 processors and the relative proxies.

We propose a new proxy that solves the problem of multiple submissions of information between processors, performing a single shipment in every stage of execution. To determine useful rules, each membrane needs to know the total context, i.e. the existence of membranes that can be parents or its daughters. In the application phase, to avoid existence of communication between processors, to determine the presence or absence of

membranes, the proxy must know whether a membrane exists or has been dissolved. To send objects between membranes, the proxy also must know where each membrane, with it wants communicate, is located. The membrane dissolution step is executed after the communication of objects. If a membrane is dissolved, all objects and containing membranes pass to belong to mother. This indication of dissolution also be communicated to its parent membrane and also it is conveyed to all proxies who need to know its existence to the determination of useful rules. Proxies at all times know the total context of the membranes of the processor. In addition, a membrane after the application of rules known whether it will be dissolve or not, but does not dissolve until transferring objects. If the membranes begin to dissolve and the communication with the parent is made before transmission of the new objects, information can be losing. However, with the correct order in the sending of messages it may indicate that the membrane is dissolved while sending objects, reducing the two shipping to only one. The correct order of delivery will depends on the architecture to be used, as shown in the following part.

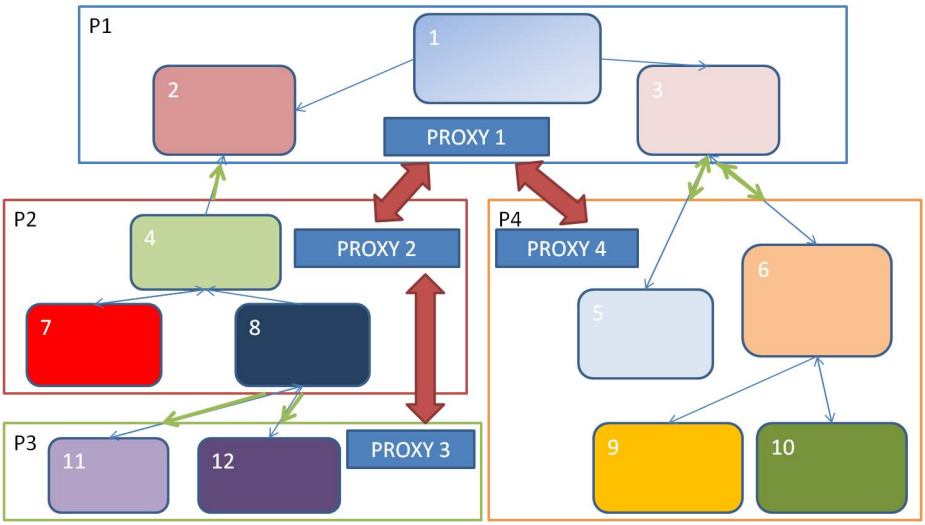

Fig. 1. Example of P system distribution and the associated proxies

Peer-to-peer architecture proxy

The architecture Peer-to-peer (P2P) was introduced by Tejedor in [Tejedor, 2008]. In this architecture the membranes are located in P processors conserving the tree topology. Then a processor only communicates with its parent or children processor, to reduce the time of communication to 2(P-1) communications. When a processor communicates with another, it waits the response.

For this architecture, the proxy must maintain the tree topology of P System (at least for the descendants or ascendants of the membranes that contained membranes, until to the membrane that cannot be dissolved). Thus, for each of these membranes, the proxy must will know where they are located; also it will know the tree processors structure, so that if it has to make a referral to a processor offspring, it knows through which of its children should do it. Finally, it keeps objects to send to other membranes (we named ObjectsToSend), and an indication of the membranes dissolved (MembranesToDissolves), and objects that are contained.

In the example in Figure 2 (based of the distribution of Figure 1), where each membrane is referenced by a circle, the processor contains the membranes 4, 7 and 8, with all its elements (objects, rules...). The proxy contains the structure of other descendants or ascendants to the membrane which cannot be dissolved, indicating only the topology. It also maintains an indication of the processors for the ascending and descending to the membranes

indicated. The proxy has ObjetToSend store, where it stores the objects that should be communicated to another membrane that does not belong to processor, and an indication of the dissolved membranes, whether they are in the processor or in the proxy. In the rules application step, to know that rules are useful, the processor request information to the proxy about existence of other membranes, because it contains such information.

For each outside membrane, the proxy only contains its identifier, the indication of the daughters and parent membranes, and the corresponding proxy reference. Proxy also contains information from other proxies. For each proxy contains its identifier, its address, its parent and children proxies, and the membranes include of the proxy. Then, for *m* membranes and *p* proxies, is necessary:

$$4m + 5p \text{ numbers} \tag{1}$$

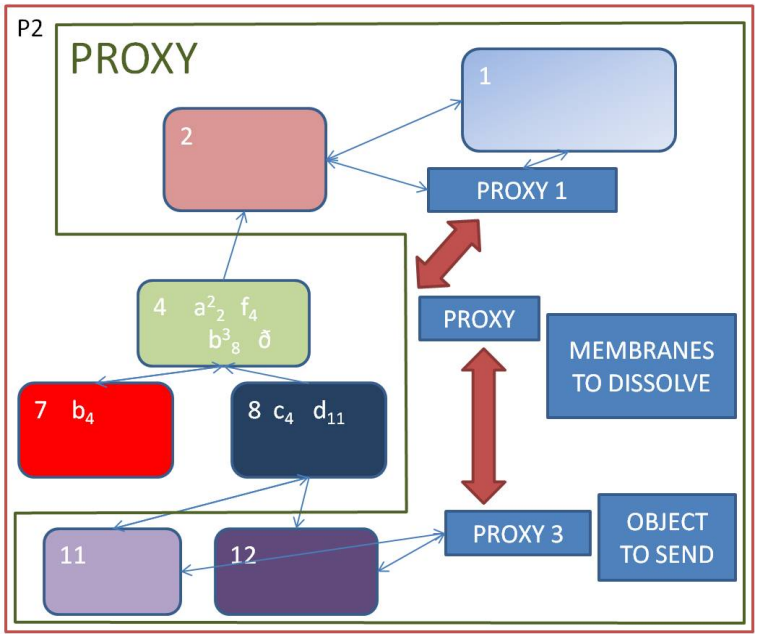

Fig. 2. Proxy of processor 2 (based of P system distribution show in Fig. 1)

The algorithm will run in parallel on each processor, and must contain six parts running sequentially which are running similarly to proxies for all architectures: Local analysis, synchronization, parent receiving message, children communication, preparing the message for the parent with dissolved membranes application, and father sending.

- The **local analysis** is cover all available membranes in the processor and analyst the new object destine. Put objects in the corresponding membrane if it is in the processor, or put it in ObjectToSend, so that later they will send to their respective destinations. In addition, if any membrane will be dissolved, this is added to MembranesToDissolve.
- To avoid collision in the communication, there must be **synchronization** between the processors, so that only one is communicating at all times. To do this, at this point which processors need to communicate with other processors, stop execution until the parent process sends the order to continue. The processor *0* or root will give the order himself, being the only begins to run. The way to give permission to report will be with a message.
- The father, also give an indication that the child should continue its execution, the message send objects and an indication of the dissolved membranes. Therefore, the child, upon **receiving the message from the father**, must process and drop the objects in the corresponding destination (the membrane itself if it

is found in the processor or objectToSend in other case). He will also receive an indication of the membranes that have been dissolved, and add them to his list of dissolved membranes (MembranesToDissolve).

- After receiving and processing information from the parent, it **communicates with the children**. Thus, sequentially, for each child prepares a message, it sends, waits for the child to respond and process the response message. In the message prepared for the child, are attaching all objects in ObjectToSend whose destine is in the child processor or any of their descendants. It adds an indication of the membranes that have been dissolved. It sends this message, telling the child to continue its execution, and waits until the child finishes and passes the order to continue. When it received this order will also receive a message from the child, indicating what membranes are dissolving and what objects are sending.
- At this point, already has all the information to be communicated to the parent. Objects found on ObjectToSend are destined for an ascending of that processor, so they are added to a **new message that is preparing to send the parent**. Also, add the indication of the membranes that are marked to dissolve. It processed all labeled membranes to dissolve and then they are dissolved. This dissolution is to move its objects to the mother membrane if it is on the same processor, or add objects (put as destine the mother membrane) to the message if it is on another processor. Completes the dissolution by removing the membrane, and adjusting the membranes tree (indicates the relationship between membrane).
- Finally **sends the message to the father**, indicating that it has finished executing, and that the parent can continue.

The algorithm is present as Algorithm 1:

Algorithm 1: Proxy algorithm for P2P architecture
1: Local analysis (Algorithm 2)
2: Synchronization (Algorithm 3)
3: Parent receiving message and Process the message (Algorithm 4)
4: Children communication (Algorithm 5)
5: Preparing the message for the parent with dissolved membranes application (Algorithm 7)
6: Father sending (Algorithm 8)

```
Algorithm 2: Local analysis
1:  for all membrane of current processor do
2:    for all new object do
3:      if current processor contain the destine of object then
4:        Put the object in the membrane
5:      else
6:        Put the object in objectToSend
7:      end if
8:    end for
9:    if execute a rule to dissolve the membrane then
10:     Add membrane to MembranesToDissolve
11:   end if
12: end for
```

```
Algorithm 4: Process the message
    1:  for all object do
    2:    if CurrentProcessor contain the destine then
    3:      PUT the object in the membrane
    4:    else
    5:      PUT the object in ObjectToSend
    6:    end if
    7:  end for
    8:  for all membrane in MembraneToDissolve received in the message do
    9:    ADD to MembranesToDissolve if not exists
    10: end for
```

```
Algorithm 5: Children communication
    1: for all child processor do
    2:   Create a message (Algorithm 6)
    3:   SEND (child processor, message)
    4:   WAIT (message)
    5:   Process the message (Algorithm 4)
    6: end for
```

```
Algorithm 6: Create a message
    1: for all object in ObjectToSend do
    2:   if membrane's target is in this child processor or another descendant of this child
processor then
    3:     PUT the object in the message (and remove to the ObjectToSend)
    4:   end if
    5: end for
    6: for all membrane in MembranesToDissolve do
    7:   PUT the membrane in the message
    8: end for
```

```
Algorithm 7: Preparing the message for the parent with dissolved membranes application
    1:  for all object in ObjectToSend do
    2:   PUT the object in the message (and remove to the ObjectToSend)
    3:  end for
    4:  for all membrane in MembranesToDissolve do
    5:    PUT the membrane in the message
    6:    if parent membrane in other processor then
    7:      for all object in the membrane do
    8:        PUT the object in the message, but the target is the parent of membrane.
    9:      end for
    10:   else
    11:     for all object in the membrane do
    12:       MOVE the object to the parent membrane
    13:     end for
    14:   end if
    15:   ADJUST the tree removing this membrane
    16: end for
```

```
Algorithm 8: Father sending
        1: if currentProcessor <>0 then
        2:   SEND (parent processor, message)
        3: end if
```

Hierarchical Peer-to-Peer architecture proxy

This architecture (HP2P) proposed by Bravo [Bravo, 2007] as a variant of the previous. With the same idea, and parallelizing the communication, reduce the time. In this architecture, a processor can communicate sequentially with all its children without waiting for a response to communicate with the following. Later the communication with the children, these communicate with the processor, and finally, the processor communicates with his parent.

The proxy for this architecture has a performance similar to the Peer-to-Peer architecture. In this architecture, communication with children can be done sequentially without waiting for the response of the child. After all the children sent in sequential order, and when they are completed, send a message to the parent.

The proxy only should maintain, for all the membranes of the processor, their descendants and ascendants until the membrane that cannot be dissolved. In the event that contained more information on membranes, the knowledge of them be out of date. This knowledge does not influence the total context of a membrane during the process.

The operation of the proxy presents only this variant, as seen in the algorithm 9:

```
Algorithm 9: Proxy algorithm for HP2P architecture
        1: Local analysis (Algorithm 2)
        2: Synchronization (Algorithm 3)
        3: Parent receiving message and Process the message (Algorithm 4)
        4: Children communication (HP2P) (Algorithm 10)
        5: Preparing the message for the parent with dissolved membranes application
(Algorithm 7)
        6: Father sending (Algorithm 8)
```

```
Algorithm 10: Children communication in HP2P architecture
        1: for all child processor do
        2:   Create a message (Algorithm 6)
        3:   SEND (child processor, message);
        4: end for
        5: for all child processor do
        6:   WAIT (message)
        7:   Process the message (Algorithm 4)
        8: end for
```

Conclusion

The use of proxies in the communication of the membranes is extended regardless of the architecture used. This article has proposed a variant of the proxy introduced by Tejedor. With this new proxy, at each step of evolution, at least three communications between processors are reduced to just one.

The operation of the proxy, which satisfies a general idea, has a small difference if the parallelism of the Hierarchical Peer-to-Peer Architecture is used. Using this proxy, which connects the dissolved phase with the objects shipping, allowing the membranes begin the rules application phase but not all membranes have completed their communication phase, reducing wait times and improving global times in the evolution of the system.

Bibliography

[Bravo, 2007] G. Bravo, L. Fernández, F. Arroyo, and J. A. Frutos. A hierarchical architecture with parallel communication for implementing p systems. In Kr. Ivanova (Ed.) Kr. Markov, editor, Proceedings of the Fifth International Conference Information Research and Applications i.TECH 2007, volume 1, pages 168–174, June 2007.

[Bravo, 2007b] Bravo, G., Fernández, L., Arroyo, F., and Tejedor, J. (2007b). Master-slave distributed architecture for membrane systems implementation. In Aggarwal, A., editor, Proceedings of the 8th WSEAS International Conference on Evolutionary Computing - Volume 8, pages 326-332, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS).

[Bravo, 2008] G. Bravo, L. Fernández, F. Arroyo, and M. A. Peña. Hierarchical master-slave architecture for membrane systems implementation. In M. Sugisaka and H. Tanaka, editors, Proceedings of the 13th International Symposium on Artificial Life and Robotics (AROB 2008), pages 485–490, Beppu, Japan, January 31 - February 2 2008.

[Ciobanu, 2004] G. Ciobanu and W. Y. Guo. P systems running on a cluster of computers. Membrane Computing, 2933:123–139, 2004.

[Frutos, 2009] J.A. de Frutos, L. Fernández, and F. Arroyo. Decision trees for obtaining active rules in transition p systems. In Agustin Riscos-Núñez Gheorghe Păun, Mario J. Pérez-Jiménez, editor, Tenth Workshop on Membrane Computing (WMC10), pages 210–217, 2009.

[Păun, 2000] G. Păun. Computing with membranes. Journal of Computer and System Sciences, 61(1):108–143, August 2000.

[Syropoulos, 2004] A. Syropoulos, E. G. Mamatas, P. C. Allilomes, and K. T. Sotiriades. A distributed simulation of transition p systems. Membrane Computing, 2933:357–368, 2004.

[Tejedor, 2008] J. Tejedor, L. Fernández, F. Arroyo, and G. Bravo. An architecture for attacking the communication bottleneck in p systems. Artificial Life and Robotics, 12:236–240, 2008.

Authors' Information

Miguel Ángel Peña *– Dept. Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660 Madrid, Spain; e-mail: m.pena@upm.es*

Ángel Castellanos *– Departamento de Ciencias Básicas aplicadas a la Ingeniería Forestal. Escuela de Ingeniería Técnica Forestal. Universidad Politécnica de Madrid, Avda. de Ramiro de Maeztu s/n 28040 Madrid, Spain; e-mail: angel.castellanos@upm.es*

Alberto Arteta *–Associate ProfessorTechnical University of Madrid. aarteta@eui.upm.es*

Francisco Gisbert *– Dept. Lenguajes, Sistemas Informáticos e Ingeniería del Software, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660 Madrid, Spain; e-mail: fgisbert@fi.upm.es*