Usefulness States in New P System Communication Architectures

Juan Alberto de Frutos, Fernando Arroyo, and Alberto Arteta

Dpto. de Lenguajes, Proyectos y Sistemas Informáticos
Escuela Unversitaria de Informática - Universidad Politécnica de Madrid
Crta. de Valencia Km. 7 - 28031 Madrid - Spain
{jafrutos,farroyo,aarteta}@eui.upm.es

Abstract. Dealing with distributed implementations of P systems, the bottleneck communication problem has arisen. When the number of membranes grows up, the network get congested. In agreement with this, several published works have presented an analysis for different architectures, which implement P systems in a distributed cluster of processors, allocating several membranes to the same processor. The purpose of these architectures is to reach a compromise between the massively parallel character of the system and the needed evolution step time to transit from one configuration of the system to the next one, solving the bottleneck communication problem.

The work presented here carries out an analysis of semantics of the P systems, in several distributed architectures. It will be shown how to restructure P systems when dissolutions or inhibitions take place in membranes. Moreover, it will be also determined the extra information necessary at every communication step in order to allow all objects to arrive at their targets without penalizing the communication cost. This will be based on usefulness states, presented in a previous work, which allow each membrane of the system to know the set of membranes with which communication is possible at any time.

1 Introduction

Membrane computing [6] is a new branch of natural computing, inspired by living cells. Membrane systems establish a formal framework in which a simplified model of cells constitutes a computational device. Starting from a basic model, transition P systems, many different variants have been considered and many of them have been demonstrated to be equivalent to the Turing machine. Strictly speaking from an implementation point of view and considering only the simplest model transition P systems), there are several challenges for researchers in order to get real implementations of such systems. Today, one of the most interesting is to solve the communication bottleneck problem when the number of membranes grows up in the system. Accordingly with this fact, several works [8], [2] and [3] present an analysis for distributed architectures based on allocating several membranes to the same processor, in order to reduce the number of

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 169–186, 2009.

external communications. These architectures allow certain degree of parallelism in application rules phase, as well as in the communication phase in a transition step during P system execution.

On the other hand, usefulness states were defined in [5] with two main goals. First and foremost, a usefulness state in a membrane represents the set of membranes to which objects can be sent by rules in the current evolution step. This information is essential to carry out a transition correctly. And second, usefulness states are used to improve the first phase (evolution rules application inside membranes) getting useful rules in a faster way. In [8], [2] and [3] the total time for an evolution step is computed, and what is more important is the fact that reducing the application phase time, the system obtains an important gain in the evolution total time.

The goal of this paper is to fit usefulness states into communication architectures presented in [8], [2] and [3], solving the problem of membrane dissolution and membrane inhibition, not considered in those works. Furthermore, it will also be considered the required information for objects to reach their respective target membranes. This information is based on the usefulness state concept.

2 Related Works

In what follows, several distributed architectures for implementing transition P systems are described, and also the usefulness state concept is reviewed.

2.1 Communication Architectures

In order to face the communication problem in P system implementations, in [8] an architecture named "partially parallel evolution with partially parallel communication" is presented. This architecture is based on the following ideas:

1. Membrane distribution. Several membranes are placed at each processor which will evolve, at worst, sequentially. Then, there are two kinds of communications: (i) internal communications between membranes allocated at the same processor, with negligible communication time due to the use of shared memory techniques, and (ii) external communications between membranes placed in different processors.
2. Proxies used to communicate processors. When a membrane wants to communicate with another one allocated at a different processor, uses a proxy. Therefore, external communications are carried out between proxies, no between membranes. This implies that each processor has a proxy which gathers objects from all membranes allocated to it, and after that it communicates with suitable proxies.
3. Tree topology of processors in order to minimize the total number of external communications in the system. Proxies only communicate with their parent and children proxies. Figure 1 shows an example of a membrane structure for a transition P system and its distribution in an architecture with four processors.

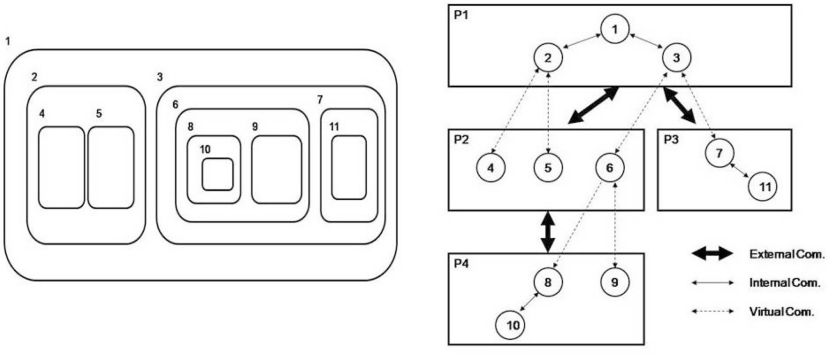

Fig. 1. Membrane distribution in processors

4. Token passing in communications in order to prevent collisions and network congestion. A communication order is established through a token, and then only one proxy tries to communicate at any moment. This token travels through a depth search sequence in the topology of processors tree. In the architecture of Figure 1, the order in communications would be the following: P1 to P2, P2 to P4, P4 to P2, P2 to P1, P1 to P3 and finally P3 to P1.

More recently, Bravo et al. [2] have proposed a variant of this architecture. Membranes are placed in slave processors and a new processor is introduced acting as master. Slaves apply rules and send to the master multisets of objects whose targets are in a different slave. Master processor redistributes multisets to its own slaves. This architecture keeps the parallelization in the application phase obtained in [8], but also it seeks for parallelizing the rule application phase in some processors with the communication phase in others. This produces the reduction of the evolution step time in the system.

An improvement of the last architecture was proposed in [3] by Bravo et al. Now, several master processors in a hierarchical way are used. This fact allows the parallelization of external communication and drastically increases the parallelization of application rules and external communication phases. As a result, a better evolution time of the system is obtained.

2.2 Usefulness States

The usefulness state concept for membranes of a P system was introduced in [5]. This state allows to any membrane to know the set of child membranes to communicate with (membrane context). This information is necessary to determine the set of rules to be applied in a evolution step, and it changes dynamically when membranes are dissolved or inhibited in the P system.

The set of usefulness states for a membrane j in a transition P system can be obtained statically, that is, at analysis time, as it is shown in [5]. One usefulness state in a membrane represents a valid context for that membrane, that is, a

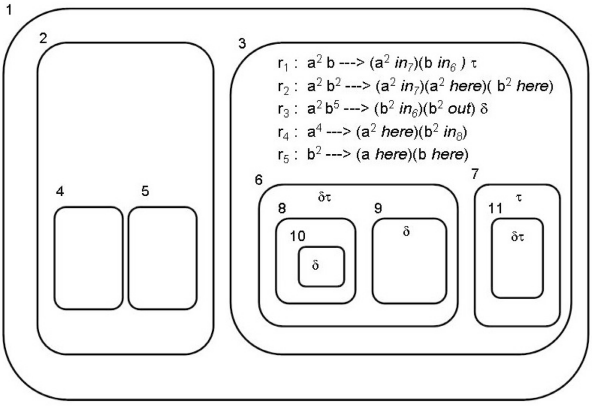

Fig. 2. Dissolving and inhibiting capabilities in membranes

context that can be reached after an evolution step. As the membrane context can change dynamically, transitions among states are also defined in [5].

From a given usefulness state we can obtain the set of useful rules. A rule is useful in an evolution step if all its targets are adjacent, not dissolved and not inhibited, hence the communication is feasible.

Figure 2 represents our example of P system. In this case, only rules associated to membrane 3 are detailed. Symbol δ in membranes 6, 9, 10 and 11 represents the possibility of these membranes to be dissolved by the application of some rules inside them. The symbol τ represents the possibility of inhibiting the communication through membranes 6, 7, and 11. Usefulness states for membrane 3 are depicted in table 1, together with their contexts and useful rules.

Tables defining transitions among states are also defined at analysis time. Suitable transitions take place when a child membrane of the current context changes its permeability in such a way that, during system execution, membranes will obtain the set of useful evolution rules directly from their usefulness states, without any computation.

Table 1. Usefulness states for membrane 3

Usefulness State	**Context**	**Useful rules**
q_0	{6, 7}	r_1, r_2, r_3, r_5
q_1	{6}	r_3, r_5
q_2	{8, 9, 7}	r_2, r_4, r_5
q_3	{8, 9}	r_4, r_5
q_4	{8, 7}	r_2, r_4, r_5
q_5	{9}	r_5
q_6	{7}	r_2, r_5
q_7	$\emptyset$	r_5

From an implementation point of view, problems arise when membranes have a high number of states, which cause transition tables to grow up. That is why in [5] it is proposed to encode usefulness states in order to avoid transition tables. Each usefulness state is encoded depending on its context, hence transitions are carried out directly in the code. Two definitions are introduced:

Total context for membrane j. This is the set of all membranes that eventually can become children of membrane j. Therefore, all contexts are included in the total context:

$$TC(j) = Child\_Of(j) \bigcup_{j_k \in Child\_D(j)} TC(j_k), \tag{1}$$

where $Child\_Of(j)$ is the set of all membrane j children in the initial structure, and where $Child\_D(j)$ is the set of membrane j children that can be dissolved.

Normalized total context for membrane j. This is defined as the $TC(j)$ sorted in depth and in pre-order:

$$TC_{Normal}(j) = (j_1, TC_{Normal}(j_1), \ldots, j_n, TC_{Normal}(j_n)), \tag{2}$$

where $j_k \in Child\_Of(j)$ from left to right in μ, that is, in the initial membrane structure, and $TC_{Normal}(j_k)$ is considered as null if membrane j_k has no dissolving capability. For instance, in our P system, $TC_{Normal}(3) = \{6, 8, 9, 7\}$.

Each usefulness state of a membrane j is encoded by $TC_{Normal}(j)$ depending on its context, with binary logic. The value 1 represents that the membrane belongs to the state context. For example, the usefulness state q_0 for membrane 3, representing the context {6, 7}, is encoded as 1001.

If $q^j(t) = (i_1, \ldots, i_k, \ldots, i_n)$ encoded by $TC_{Normal}(j)$ is the usefulness state for membrane j at time t, the transitional logic will be the following:

1. If membrane i_k at time t is inhibited, then $q^j(t+1) = (i_1, \ldots, 0, \ldots, i_n)$
2. If membrane i_k at time t comes back to be permeable, then $q^j(t+1) = (i_1, \ldots, 1, \ldots, i_n)$
3. If membrane i_k at time t is dissolved, it has to send its usefulness state $q^{i_j}(t)$, encoded by its normalized total context $TC_{Normal}(i_k)$, to membrane j. Considering formula 2, the usefulness state for membrane j can be expressed in a deeper way as $q^j(t) = (i_1, \ldots, i_k, TC_{Normal}(i_k), \ldots, i_n)$. Then, the transition obtained for membrane j is $q^j(t+1) = (i_1, \ldots, 0, q^{i_j}(t), \ldots, i_n)$

In the example, if membrane 3 is in usefulness state $q^3(t) = 1001$, encoded by $TC_{Normal}(3) = \{6, 8, 9, 7\}$ and membrane 6 is dissolved in $q^6(t) = 11$ encoded by $TC_{Normal}(6) = \{8, 9\}$, it is obtained the transition $q^3(t+1) = 0111$.

3 Usefulness States Updating in Membrane Dissolution and Inhibition

In order to fit properly usefulness states updating, we will describe the succession of tasks that are carried out in an evolution step before communications initiate, that is, in the phase of rules application inside a membrane.

1. Active rules are obtained for every membrane of the system.
2. Active rules are applied in a maximally parallel and non-deterministic way in every membrane of the system.
3. Each membrane of the system determines the result of rules application. The following information is obtained:
   - Objects which are produced and remain at the same membrane.
   - Objects which are produced and have as target an adjacent membrane of the P system. These objects will be sent to the proxy of the processor in which the membrane is placed. This proxy will be in charge of collecting objects and sending them to their respective targets, as will be described in Section 5.
   - A new permeability state for the membrane, which is computed following the Figure 3 automaton. This automaton represents transitions among membranes states based on the resulting dissolution and inhibition in the applied rules. The membrane will notify its new permeability state to the proxy only in case of changing.

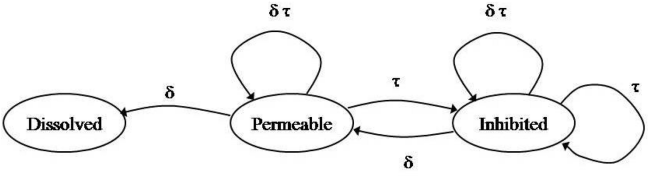

Fig. 3. Membrane permeability states

When a proxy receives the information sent by a membrane (new permeability state and current usefulness state in case of dissolution) it is necessary to carry out the following tasks:

1. The proxy has to find out the father membrane. It is necessary to consider that it can change dynamically, as membranes are dissolved. Furthermore, the father may be allocated to another processor.
2. The proxy has to notify the new situation to the father. The latter will update its usefulness state according to this situation, as it has been shown in Subsection 2.2.

In order to achieve these goals, the proxy must know the membrane structure, as regards membranes allocated in the proxy processor. For each of them, the proxy must know the following information:

- **j:** membrane identifier.
- **D(j):** Dissolved. The value will be true if membrane j is dissolved.
- **TCL(j):** Total context lenght. This value is computed in analysis time following the formula:

$$TCL(j) = \sum_{k=1}^{n} (1 \; + \; TCL'(j_k)) \tag{3}$$

where $j_k \in Child\_Of(j)$ and

$$TCL'(j_k) = \begin{cases} TCL(j_k) \text{ if } j_k \text{ has dissolving capability} \\ 0 \qquad\qquad\quad \text{otherwise} \end{cases}$$

- **PFTC(j):** Position at father total context. This value is the membrane j position at the normalized father total context. This value is computed from the initial structure in analysis time following the formula:

$$PTCF(j_i) = 1 \; + \; \sum_{k=1}^{i-1} (1 \; + \; TCL'(j_k)), \tag{4}$$

  where $j_k \; \in Child\_Of(j)$ at the left of j_i in μ. For instance, as $TC_{NORMAL}(3)$ = {6, 8, 9, 7}, values of $PFCT$ for child membranes are obtained from this total context. Specifically, $PFTC(6) = 1$ and $PFTC(7) = 4$.
- **USM(j):** Usefulness state mask. The membrane j will make use of this mask in the usefulness state updating process.

Following with the example in Figure 1, Figure 4 represents the stored information in proxies. The values for $TLC(j)$ and $PFTC(j)$ are worked out from the corresponding normalized total context, which are obtained taking into account dissolving and inhibiting capabilities of membranes, depicted in Figure 2.

The proxy looks for the father of the membrane which has changed the permeability state, going up in the membrane structure. In this case, it is necessary to use $D(j)$ to find a non-dissolved membrane.

The proxy has also to prepare the suitable information for father membrane in order to update its usefulness state. The updating process is performed by changing the bit representing the membrane which has changed its permeability state and this is done through a XOR between the usefulness state and the $USM(j)$ field. As shown in Subsection 2.2, the following cases can be found:

- Inhibition of a child membrane. The position associated to the child membrane in the usefulness state has to be changed from 1 to 0, which means

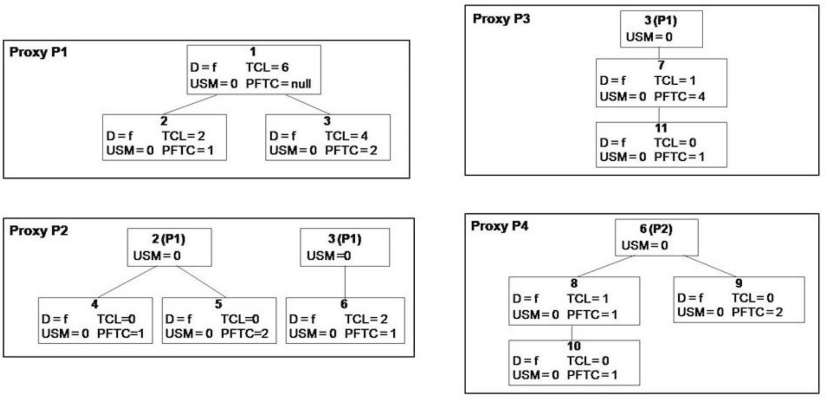

Fig. 4. Information stored in proxies to update usefulness states

that the communication is not possible for the next evolution step. A XOR operation with a bit 1 reaches this change. For instance, let us suppose that membrane 3 in our P system has the usefulness state 1001. As $TC_{Normal}(3) = \{6, 8, 9, 7\}$, this state represents the context {6,7}. Let us also suppose that membrane 7 is inhibited at this time. The usefulness state for membrane 3 is updated as follows:

$USM(3) = 0001$ (bit 1 for membrane 7)
1001 XOR 0001 = 1000 (Context(3) = {6})

- Removing inhibition of a child membrane. The position associated to this membrane has to be changed from 0 to 1. This shows that the child membrane accepts objects for the next evolution step. Again, a XOR operation with a bit 1 reaches the change.
- Dissolution of a child membrane. The bit representing the child membrane in the total context has to be changed from 1 to 0. Moreover, several of the following positions in the normalized total context represent the context of the dissolved membrane, as formula 2 shows, and necessarily these bits have to be replaced with the usefulness state of the dissolved membrane. These changes can be done with a XOR operation between the usefulness state and the mask stored in the $USM(j)$ field. For instance, let us suppose that the usefulness state of membrane 3 is 1001, representing context {6, 7}, and membrane 6 is dissolved in the usefulness state 10. As $TC_{Normal}(6) = \{8, 9\}$, this state represents context {8}. The usefulness state of membrane 3 would be updated in the following way:

  $USM(3) = 1100$ (bit 1 for 6, followed by its usefulness state)
  1001 XOR 1100 = 0101 (Context(3) = {8, 7})

The main problem now is to exactly determine the position of this information in the binary mask, that is, in the field $USM(j)$. In order to do this, it is necessary the $PFTC(j)$ field. The proxy goes up in the membrane structure looking for the father membrane, and simultaneously performing the addition of $PFTC(j)$ fields for every dissolved membranes found in the path.

As an example, let us suppose that membrane 9 is dissolved in a evolution step and membrane 6 was already dissolved in a previous step, in such a way that membrane 3 is the father of membrane 9. The $USM(3)$ field can be obtained in the following way:

Information = 1 (membrane 9, followed by its usefulness state)
Position = $PFTC(9) + PFTC(6) = 3$
Length = $TCL(3) = 4$

$USM(3) = 0010$ (as $TC_{Normal}(3) = \{6, 8, 9, 7\} \Rightarrow 9$ dissolution)

When a membrane j changes its permeability, the algorithm *ChangeUS* (Figure 5) will carry out this process. The operator + represents strings concatenation and 0^n represents a string with n symbols 0.

In an evolution step, it may happen that several child membranes change their permeability, involving the same father. Therefore, the usefulness state of

```
ChangeUS ( j, NewPerm, UState )
(1)   Mask <-- 1
(2)   IF NewPerm = Dissolved THEN BEGIN
(3)           Mask <-- Mask + UState
(4)           Mask <-- Mask XOR (0 + USM (j))
                    (* Send up current USM of membrane j *)
(5)     END
(6)   Mask <-- 0 PFTC(j)-1 + Mask
(7)   Target <-- Father ( j )
(8)   WHILE D(target) AND NOT OutProcessor(Target)
(9)     DO BEGIN
(10)          Mask <-- 0 PFTC(Target) + Mask
(11)          Target <-- Father (Target)
(12)     END
(13)  IF NOT OutProcessor(Target) THEN
(14)          Mask <-- Mask + 0 TCL(Target) - | Mask |
                          (* Fit Mask in the Total Context *)
(15)  USM (Target) <-- USM (Target) XOR Mask
```

Fig. 5. Algorithm used to obtain the USM field for membrane j father

a membrane has to be modified with several masks. No matter the order in which the proxy processes permeability changes, commutative and associative properties of XOR operation allow to obtain the value for $USM(j)$ correctly. Let us suppose, in our example, that membranes 6 and 9 are dissolved in the same evolution step. If proxy processes membrane 6 before, the resulting $USM(3)$ is processed as follows:

$$\begin{cases} 1^{st}\ \text{ChangeUS(6, dissolution, 11)} \left\{ \begin{array}{l} \text{Father = 3} \\ \text{Mask = 1110} \end{array} \right\} \text{USM(3) = 1110} \\ 2^{nd}\ \text{ChangeUS(9, dissolution, -)} \left\{ \begin{array}{l} \text{Father = 3} \\ \text{Mask = 0010} \end{array} \right\} \text{USM(3) = 1110 XOR 0010 = 1100} \end{cases}$$

Both dissolutions have been considered owing to XOR operation in line 15. On the other hand, if proxy processes membrane 9 before, the resulting $UMS(3)$ is processed as follows:

$$\begin{cases} 1^{st}\ \text{ChangeUS(9, dissolution, -)} \left\{ \begin{array}{l} \text{Father = 6} \\ \text{Mask = 01} \end{array} \right\} \text{USM(6) = 01} \\ 2^{nd}\ \text{ChangeUS(6, dissolution, 11)} \left\{ \begin{array}{l} \text{Father = 3} \\ \text{Mask = (111 XOR (0+01)) + 0 = 1100} \\ \qquad\qquad\qquad\quad \uparrow \\ \qquad\qquad\qquad\quad \text{USM(6)} \end{array} \right\} \text{USM(3) = 1100} \end{cases}$$

When membrane 6 is dissolved $UMS(6)$ is inherited by the father membrane, that is $UMS(3)$, through XOR operation in line 4.

Finally, it is important to note that the father membrane may be placed in a different processor; therefore the process is carried out by several processors in a distributed way. The algorithm in Figure 5 deals with this situation in lines 8 and 13, in which *OutProcessor* checks if the target membrane is allocated in other processor.

4 Encoding Targets of Rules within Total Context

Evolution rules in transition P systems have the form $u \rightarrow v$, $u \rightarrow v\ \delta$ or $u \rightarrow v\ \tau$, with $u \in\ O^+$ and $v \in\ (O^+ \times TAR)^*$, where O is the alphabet of objects, and $TAR = \{here, out\} \cup \{in_j \mid j \text{ is a membrane label}\}$. Symbol δ represents membrane dissolution, while symbol τ represents membrane inhibition; u is called the *antecedent* and $v, v\delta, v\tau$ the *consequent* of rules.

Usually, transition P systems implementations up to now [4] [7] require to store a membrane identification for every target in_j in every rule in every membrane. In this paper a compact representation for evolution rules consequent based on the concept of total context is presented. It allows to represent targets without membranes identifications, what reduce significantly the space necessary to store rules. Moreover, this representation allows proxies to find any membrane target in a precise way.

The total context of a membrane is obtained at analysis time, and it encodes any possible in_j target for evolution rules of the membrane. Hence, adding a binary mask of length equal to membrane total context length, it is possible to control if a rule sends objects to a determined child membrane with label j. It is expressed setting to 1 the j position in the binary mask.

In addition, we propose four bits more in order to encode the complete consequent of a rule r_k, two for targets here (b_h^k) and out (b_o^k) respectively and two for representing membrane dissolution (b_δ^k) and inhibition (b_τ^k). Figure 6 shows the proposed encoding for a rule consequent. Besides the sequence of bits, each target has a multiset associated, represented as $M_h^k\ M_o^k\ M_1^k\ \ldots\ M_n^k$.

On the other hand, the antecedent of a rule r_k can be represented with another multiset: M_a^k.

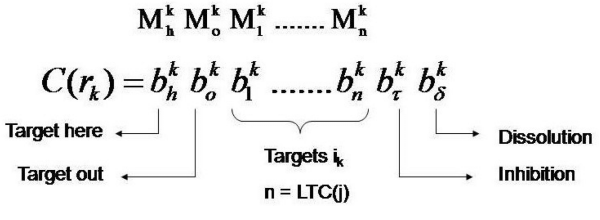

Fig. 6. Encoding a rule consequent

Table 2. Encoding consequent of membrane 3 rules

Rule	**Encoding**	**Multisets**
$r_1: a^2b \rightarrow (a^2\ in_7)(b\ in_6)\ \tau$	00100110	$M_1^1 = b,\ M_4^1 = a^2$
$r_2: a^2b^2 \rightarrow (a^2\ in_7)(a^2\ here)(b^2\ here)$	10000100	$M_h^2 = a^2b^2,\ M_4^2 = a^2$
$r_3: a^2b^5 \rightarrow (b^2\ in_6)(b^2\ out)\ \delta$	01100001	$M_o^3 = b^2,\ M_1^3 = b^2$
$r_4: a^4 \rightarrow (a^2\ here)(b\ in_8)$	10010000	$M_h^4 = a^2,\ M_2^4 = b$
$r_5: b^2 \rightarrow (a\ here)(b\ here)$	10000000	$M_h^5 = ab$

Table 2 contains the encoded consequent of rules in membrane 3 of our example. Let us remind that the normalized total context for this membrane is {6,8,9,7}

In Section 3 it was enumerated the task list to be performed in evolution rules application phase in membranes. Let us explain how can be used and computed the resulting evolution rule using this compact representation of binary mask and multiset of objects. Let $M_R(p) = r_1^{n_1} \ \ldots \ r_m^{n_m} = \sum_{i=1}^{m} n_i r_i$ be the multiset of rules to be applied in the evolution step p, where n_i means the number of times the rule r_i has to be applied. Then, it is needed to compute:

– $C(p)$, the sequence of bits, encoding targets, for the multiset of evolution rules consequent $(b_h\ b_o\ b_1\ \ldots\ b_n\ b_\tau\ b_\delta)$

$$C(p) \ = \ OR_{\forall\, r_i \,\in\, M_R(p)}\ C(r_i) \tag{5}$$

– $M_h(p)$, $M_o(p)$, $M_1(p)$, …, $M_n(p)$, the list of multisets of objects associated to $C(p)$.

$$M_h(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_h^i \tag{6}$$

$$M_o(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_o^i \tag{7}$$

$$M_j(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_j^i \tag{8}$$

– And finally, $M_a(p)$ the antecedent of the multiset of evolution rules.

$$M_a(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_a^i \tag{9}$$

When this process finishes, membranes proceed to data delivery:

– Multisets $M_h(p)$ and $M_a(p)$ will be applied directly to the membrane. Considering w the multiset of objects placed in the membrane at the beginning of the evolution step, w is updated by:

$$w \ = \ w \ - \ M_a(p) \ + \ M_h(p) \tag{10}$$

- $b_o\, b_1\, \ldots\, b_n$, together with $M_o(p)\, M_1(p)\, \ldots\, M_n(p)$, will be sent to the proxy processor.
- $b_\delta\, b_\tau$ will be used to find out changes of permeability, as the automaton in Figure 3 shows. If b_δ is equal to 1, the transition δ is applied; otherwise if b_τ is equal to 1, the transition τ is applied; finally, the transition $\delta\tau$ is applied if both b_δ and b_τ are equal to 1. In the case of permeability change, membrane will notify the new permeability state to the proxy, in order to update the usefulness state of its father, as it is detailed in section 3.

5 Targets Search in Proxies

When a membrane has to send objects to its adjacent membranes, it uses the proxy. The membrane sends to the proxy a pair of data $(Targets, MS)$, where $Targets$ is a binary sequence encoding labels of target membranes $(b_o\, b_1\, \ldots\, b_n)$ and MS is the sequence of multisets associated to each one of the target membranes $(M_o(p)\, M_1(p)\, \ldots\, M_n(p))$. At this moment, the proxy has to perform the following tasks:

1. Target membrane for $M_o(p)$ is the father membrane. Hence the proxy will go up in the membrane structure looking for the first non-dissolved membrane.
2. Target membranes for $M_1(p)\, \ldots\, M_n(p)$ are encoded by $b_1\, \ldots\, b_n$. Hence, the proxy needs to analyze the normalized total context of the source membrane. Considering equation (2) for the normalized total context of a given membrane, for every child membrane the proxy has to keep two important data: the dissolving capability of this membrane, and the length of its normalized total context.

As a consequence, in order to perform targeting search, the proxy has to store the following information related to membranes allocated to its processor:

- **D(j):** Dissolved. The value will be true if membrane j is dissolved.
- **DC(j):** Dissolving capability. Its value will be true if there is any evolution rule which could dissolve the membrane j. It is obtained at analysis time.
- **TCL(j):** Total context length.
- **M(j):** Multiset for membrane j. When proxy determines that membrane j is a target, it stores temporally the suitable multiset in the $M(j)$ field.

Moreover, it is also necessary to note that one or more targets could be placed in different processors. Hence, the proxy has to prepare properly some information to send them, because search of targets must continue on these processors. So, the proxy has to store some data about membranes placed in other processors with which there are established connection (virtual connections in Figure 1). The needed data for the proxy are:

- **DC(j)**, **TCL(j)**, and **M(j)**
- **Targets(j):** To store a sequence of bits encoding a list of targets.

– **MS(j):** To store a list of multisets associated to the sequence of targets. This field and the previous one are needed only for membranes with dissolving capability.

Figure 7 shows the required information by proxies of the processors depicted in Figure 2.

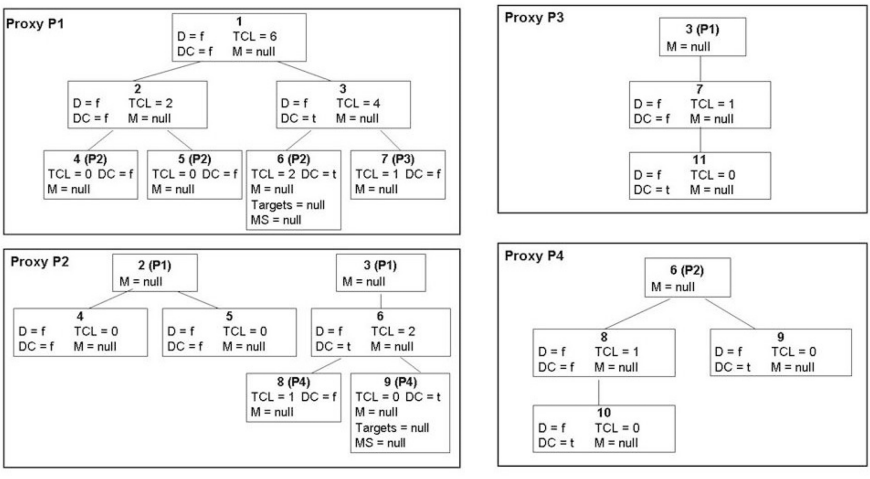

Fig. 7. Information stored in proxies to search targets

5.1 Target Search for $M_o(p)$

The algorithm presented here (*Target_Out*) looks for membrane j father in order to send it the multiset $M_o(p)$. In line 5, $M_o(p)$ is assigned to the temporary field M of the father. Line 3 considers the situation in which the search has to be continued in another processor, then the partial result remains in a field M awaiting to be sent to the appropriate processor. Section 6 of this paper deals with communications in architectures.

```
Target_Out ( j , b_o, M_o)
(1)    IF b_o = 1 THEN BEGIN
(2)        Target <-- Father ( j )
(3)        WHILE D(target) AND NOT OutProcessor(Target)
(4)            Target <-- Father (Target)
(5)        M (Target) <-- M_o
(6)    END
```

Fig. 8. Target search for multiset $M_o(p)$

5.2 Targets Search for $M_1(p)$ to $M_n(p)$

The proxy has to interpret the normalized total context of the source membrane. With this aim, the proxy will go down into the sub-tree of the membrane structure, starting from the source membrane, in depth and in pre-order. When a membrane j has no dissolving capability ($DC(j)$) the analysis of this branch of the sub-tree is finished.

The recursive algorithm in Figure 9 describes the search of targets from the source membrane j, the sequence of bits, encoding targets ($Targets$) and the list of multisets (MS) associated to targets. The algorithm visits child membranes from left to right. When the corresponding bit b_i is equal to 1, the multiset $M_i(p)$ is associated to the child membrane (line 7). Otherwise the child membrane is not a target, but if it has dissolving capability ($DC(j)$) then the search has to be continued from b_{i+1} into the normalized total context of the child membrane, as equation (2) shows. In case that the child membrane were allocated to the same processor, the search continues in the child membrane by making a recursive call in line 17. Otherwise, the information corresponding to the normalized total context of the child membrane is stored in $Targets(j)$ and $MS(j)$ fields in order to continue searching in the appropriate processor (lines 19 and 20).

An additional detail of the algorithm is that if $b_i = 1$ and the child membrane has dissolving capability, the algorithm skips the total context of the current child membrane, because these membranes are not possible targets (line 8).

```
Targets_In ( j , Targets, MS)
(1)   i <-- 1
(2)   FOR EACH k CHILD OF j FROM LEFT TO RIGHT BEGIN
(3)     IF Targets[i] = 1 THEN   (* b_i = 1 ==> membrane j is a target *)
(4)     BEGIN
(5)       IF NOT OutProcessor( k ) AND D( k ) (* k dissolved in current step *)
(6)         THEN Target_Out ( k, 1, M[ i ] )
(7)         ELSE M( k ) <-- M( k ) + MS[ i ]   (* j is the target for M_i *)
(8)       IF DC( k ) THEN i <-- i + TCL( k ) +1  (* Skip the total context *)
(9)                  ELSE i <-- i + 1
(10)    END
(11)    ELSE BEGIN  (* b_i = 0 *)
(12)      IF DC ( k ) (* The TC_NORMAL(k) is represented from Targets[i+1] *)
(13)        THEN BEGIN
(14)               Child_Targets <-- Targets[ i + 1 ] TO Targets[ i + TLC( k ) ]
(15)               Child_MS <-- MS[ i + 1 ] TO MS[ i + TLC( k ) ]
(16)               IF NOT OutProcessor( k )
(17)                 THEN Targets_In ( k, Child_targets, Child_MS)
(18)                 ELSE BEGIN
(19)                        Targets[ k ] <-- Child_targets
(20)                        MS[ k ] <-- Child_MS
(21)                      END
(22)               i <-- i + TCL( k ) +1  (* Skip the total context *)
(23)             END
(24)      ELSE i <-- i + 1
(25)    END
(26)  END
```

Fig. 9. Searching targets for multisets $M_1(p)$ to $M_n(p)$

5.3 Membrane Dissolution

As it has been said before, the proxy has to execute algorithms *Target_Out* and *Target_In* for every membrane placed at the processor which require sending objects. Moreover, it has to execute the algorithm *ChangeUS* for every membrane which notifies a change of permeability.

Nevertheless, membrane dissolution has not been solved yet. Dissolution takes place when an evolution step finishes. Then, objects remaining inside the membrane have to pass to its father. In this way, when membrane j is going to be dissolved, we have to bear in mind the following:

1. Self processing in membrane: Objects remaining in the membrane after rules application will be sent to the father membrane together with $M_o(p)$,

$$M_o(p) \leftarrow M_o(p) + w - M_a(p) + M_h(p),$$

   where w is the multiset of objects in the membrane at the beginning of the evolution step
2. Objects coming from other membranes in the current evolution step. In this case, it is needed to distinguish two possibilities depending on whether objects are processed by proxy: before or after proxy set the membrane as dissolved (field $D(j)$).
   (a) $M(j)$ stores objects arriving the proxy before it has marked the membrane j as dissolved. At the moment the proxy marks membrane j as dissolved, it sends $M(j)$ to membrane j father using *Target_Out* algorithm. This behavior is reached by adding lines 5 to 9 to the *ChangeUS* algorithm, as Figure 10 shows.

```
ChangeUS ( j , NewPerm, UState)
(1)   Mask <-- 1
(2)   IF NewPerm = Dissolved THEN BEGIN
(3)          Mask <-- Mask + UState
(4)          Mask <-- Mask  XOR (0 + USM ( j ))
                (* send up current USM of membrane j *)
(5)          D ( j ) <-- true
(6)          IF M( j ) <> null THEN BEGIN
(7)               Target_Out ( j, 1, M( j ))
(8)               M ( j ) <-- null
(9)       END
(10)  END
(11)  ……..
```

Fig. 10. In case of dissolution, $M(j)$ is sent to membrane j father

(b) In case of objects for membrane j arriving in the proxy after membrane j has been marked as dissolved and before the current evolution step has finished, the *Targets_In* algorithm will send them to membrane j father (lines 5 and 6 of Figure 9).

6 Results Distribution

When system proxies finish all the tasks explained above with algorithms *ChangeUS*, *Target_Out*, and *Targets_In*, their results are stored in $USM(j)$, $M(j)$, $Targets(j)$, and $MS(j)$ fields, where membrane j may be allocated to other processor. In order to deliver these results, the distributed architecture in which the P system is implemented is very important, because external communications are implemented by distributed architectures in different ways.

6.1 Architecture Proposed by Tejedor et al. in [8]

The external communications are established in depth in the processors tree. After receiving information coming from the upper level, a processor P communicates with each descendant processor in both directions and from left to right; finally, P sends data to its ascendant processor. Taking this order into account, the sequence of tasks to be carried out by processor P proxy is the following:

1. P proxy gets all data contained in $M(j)$, $Targets(j)$ and $MS(j)$ coming from ascendant processor proxy.
2. P proxy processes the arrived data using *Targets_In* algorithm for $Targets(j)$ and $MS(j)$ fields. Multisets received in $M(j)$ are placed in the corresponding $M(j)$ field, but if membrane j has been marked as dissolved in the current evolution step, then $M(j)$ has to be sent to membrane j father with *Target_Out* algorithm. In this case, $M(j)$ will come back to the ascendant processor in step 4.
3. for each descendant processor of P from left to right:
   (a) P proxy sends the corresponding information ($M(j)$, $Targets(j)$ and $MS(j)$) to the descendant processor.
   (b) P proxy waits until the descendant processor replies with data composed of fields $M(j)$ and $USM(j)$.
   (c) P proxy continues searching targets upwards from membrane j related to fields $M(j)$ and $USM(j)$ by using *Target_Out* and *ChangeUS* algorithms.
4. Once P proxy has processed all data from all its descendant processors, it sends to its ascendant processor the corresponding fields $M(j)$ and $USM(j)$.
5. Finally and through internal communications, P proxy delivers the definitive fields $M(j)$ and $USM(j)$ associated to inner membranes processor. The evolution step finishes when every membrane j updates its multiset and its usefulness state with this information, as follows:

$$w \leftarrow w + M(j)$$

$$Usefulness\ State \leftarrow Usefulness\ State\ \ XOR\ \ USM(j).$$

6.2 Architectures Proposed by Bravo et al. in [2] and [3]

These architectures make use of one [2] or several [3] master processors which are in charge of controlling communications among slaves processors, while membranes are placed on slaves processors. Hence, a master processor has to store and process $M(j)$, $USM(j)$, $TCL(j)$, $PFTC(j)$, $CD(j)$ and $D(j)$ fields, for all membranes belonging to slaves controlled by the master. As it was said above, $D(j)$ is a dynamic field and it is changed during execution by membranes. Therefore, a problem arises with the $D(j)$ field updating. Proxies associated to slave processors have to notify membranes dissolutions to master proxy.

The sequence of tasks in these architectures is the following:

1. Every slave processor proxy sends data to the suitable master in its corresponding turn. In particular, it sends fields $M(j)$, $USM(j)$, $Targets(j)$ and $MS(j)$ to its master, regardless of the target processor. Additionally, it has to send the list of dissolved membranes in the current evolution step.
2. Master proxy processes the incoming information as follows: $Target(j)$ and $MS(j)$ with *Targets_In* algorithm, $M(j)$ with *Target_Out* algorithm and $USM(j)$ with *ChangeUS* algorithm. Furthermore, master proxy updates $D(j)$ field for all dissolved membranes, taking into account the same questions as in section 5.3.
3. Master proxy sends the corresponding $M(j)$ and $USM(j)$ fields to each one of the slaves processors.
4. Finally, slave proxy sends to each one of its inner membranes their corresponding $M(j)$ and $USM(j)$ fields. Then, evolution step finishes.

7 Conclusion

Membranes make use of usefulness state to determine the set of membranes with which they can communicate. Moreover, when dissolutions or inhibitions are produced in the system, only usefulness states changes in father membranes in order to reconfigure the membrane structure of P systems are necessary.

The work presented here shows that usefulness states can be implemented in several distributed architectures for P systems implementations [8], [2] and [3]. In addition, usefulness states solve permeability changes in P systems for the referred architectures.

In [5], membrane total context concept was defined. This paper shows how to use it in a very useful manner to encode targets in evolution rules, avoiding labels in membranes. This encoding method allows to find any target membrane in a precise way. Moreover, it can be used in several distributed architectures for P systems implementation [8], [2] and [3].

It is also presented here a semantic analysis of P systems for determining what kind of information is relevant for the communications among membranes. In this sense, it was necessary to determine how to solve the targeting problem without producing an overload in the system communication, and how to update and communicate new membranes states all over the systems. The presented solution

based on usefulness states has been proved to be useful at least in distributed architectures presented in [8], [2] and [3].

References

1. Arroyo, F., Luengo, C., Castellanos, J., de Mingo, L.F.: A binary data structure for membrane processors: Connectivity arrays. In: Alhazov, A., Martín-Vide, C., Păun, G. (eds.) Pre-proceedings of the Workshop on Membrane Computing, Tarragona, Spain, pp. 41–52 (2003)
2. Bravo, G., Fernández, L., Arroyo, F., Tejedor, J.: Master-slave distributed architecture for membrane systems implementation. In: 8th WSEAS Int. Conf. on Evolutionary Computing, EC 2007, Vancouver, Canada (June 2007)
3. Bravo, G., Fernández, L., Arroyo, F., Peña, M.A.: Hierarchical master-slave architecture for membrane systems implementation. In: 13th Int. Symposium on Artificial Life and Robotics, AROB 2008, Beppu, Japan (February 2008)
4. Ciobanu, G., Wenyuan, G.: P systems running on a cluster of computers. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 123–139. Springer, Heidelberg (2004)
5. Frutos, J.A., Fernández, L., Arroyo, F., Bravo, G.: Static analysis of usefulness states in transition P systems. In: Proceedings of the Fifth International Conference, Information Research and Applications, I.TECH 2007, Varna, Bulgary, pp. 174–182 (June 2007)
6. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000)
7. Syropoulos, A., Mamatas, E.G., Alliomes, P.C., et al.: A distributed simulation of P systems. In: Alhazov, A., Martín-Vide, C., Păun, G. (eds.) Preproceedings of the Workshop on Membrane Computing, Tarragona, Spain, pp. 455–460 (2003)
8. Tejedor, J., Fernández, L., Arroyo, F., Bravo, G.: An architecture for attacking the bottleneck communication in P systems. In: Sugisaka, M., Tanaka, H. (eds.) Proceedings of the 12th Int. Symposium on Artificial Life and Robotics, Beppu, Japan, pp. 500–505 (January 2007)