Synchrony and Asynchrony in Membrane Systems

Jetty Kleijn[1] and Maciej Koutny[2]

[1] LIACS, Leiden University
P.O.Box 9512, NL-2300 RA Leiden, The Netherlands, email:kleijn@liacs.nl
[2] School of Computing Science, University of Newcastle
Newcastle upon Tyne, NE1 7RU, United Kingdom, email:maciej.koutny@ncl.ac.uk

Abstract. We consider synchrony and asynchrony in the behaviour of various models of membrane systems, which may differ in the way individual reactions are defined as well as in the way multisets of these reactions can be executed in a single computational step. We concentrate on the properties of ongoing computations, including the unbounded ones. Our focus is on the properties of system states involved in such computations as well as on concurrency and causality relationships between executed reactions. This should be contrasted with the approach which investigates different notions of 'results' produced through halting computations of membrane systems. As a formal behavioural model we use Petri nets and their processes which capture the notion of an execution in concurrent contexts. We continue our earlier work reported in [14], where a systematic and structural link has been established between a basic class of membrane systems and Petri nets. Here, we look at some natural extensions of this basic class of membrane systems and investigate the ways in which they can be represented within the behavioural model provided by Petri nets.
Keywords: membrane systems, P systems, Petri nets, localities, causality and concurrency, processes, synchrony, asynchrony, GALS.

1 Introduction

Inspired by the way living cells are divided by membranes into compartments where biochemical reactions may take place, *membrane systems* (also known as *P systems*) have become a prominent new computational model [1, 20, 23, 24]. In a nutshell, a reaction transforms multisets of molecules (or objects) present in the compartment into new molecules, possibly transferring some to neighbouring compartments and the environment. Consequently, all aspects of the dynamic behaviour of membrane systems are determined by the *reaction* or *evolution rules* defined for each compartment and on the way in which these rules may occur. The resulting transformations (or computation steps) are applied starting from an initial configuration (a distribution of objects). Furthermore, a notion of a successful (or halting) computation with its output is defined [23, 24]. Different types of membrane systems have been considered, depending on the form of

the rules and how they are applied, and on input/output definitions. In fact, studies in the field of membrane systems are often concerned with investigating the possible outcomes of the computations, i.e., the computational power of the various models.

The aim of our work, however, is different in that we are interested in describing what is actually going on *during* an execution of a membrane system; alternatively, one might say that we are interested in *computations* rather than *computability*. Thus, we focus on possible system states (configurations) occurring in ongoing computations as well as on the concurrency and causality relationships between executed reaction rules. This emphasis on possible behaviours (runs) rather than input/output relations, further implies that *all* possible computations need to be considered, including non-successful and infinite ones (which are also relevant from a biological/cell point of view).

There are basically two distinguishing features of any model of membrane systems when one is interested in the structural properties of their executions.[3] The first is the degree of synchrony present in a single computation step; in the extreme case, commonly considered in the theory of membrane systems, in a single step the system is transformed by a maximally concurrent execution of reaction rules (no more rules in whatever compartment could have been applied). The second is the definition of individual reactions; in the simplest case, a reaction is supposed only to consume and produce multisets of molecules, but in more elaborate models, its execution can, e.g., be conditional or affect the structure of the cell. In this paper, we will consider different kinds of synchrony as well as different types of reaction rules, and we will indicate how *Petri nets* (see, e.g., [9, 27]) can be used to capture the structural properties of the computations of varying models of membrane systems.

Essentially, Petri nets are bipartite directed graphs consisting of two kinds of nodes, called *places* and *transitions*. Places indicate the local availability of resources (represented by so-called tokens) and thus can be used to represent objects in specific compartments. Transitions are actions which can occur depending on local conditions related to the availability of resources and they can be used to directly represent reaction rules associated with specific compartments. When a transition occurs it consumes resources from its input places and produces items in its output places, thus mimicking the effect of a reaction rule (see Figure 2). Since multiset calculus is basic for membrane systems and also for computing the token distribution in Petri nets [7], some connections between the two models were already established including interpretations of reaction rules of membrane systems using Petri net transitions (e.g., [8, 26]). Petri nets are a *fundamental* modelling tool for elementary relations between occurrences of actions, moreover providing both a language and a method for behavioural analysis through so-called processes formalizing the concept of a concurrent run and a corresponding theory of labelled partial orders. It is worth mentioning that models based on a more architectural view such as process algebras do not

[3] Note that we are not interested here in the exact definition of a successful computation nor in the result it produces.

yield themselves as easily to the modelling of membrane systems since the structure of the latter is relatively simple, and the main advantage of the former, viz. compositionality in system specification and execution, is not needed.

This paper builds on previous work [14, 15], where it has been demonstrated that a structural relationship between Petri nets and membrane systems can be established at the system level. A formal translation has been given from a basic class of membrane systems into a class of Petri nets. The direct correspondence of Petri net transitions together with their input and output places to evolution rules is the key property which makes the translation suitable for dealing with structural aspects of the behaviour of membrane systems. It implies that the causality and concurrency relations between applications of reaction rules are preserved in the relationships between occurrences of the corresponding transitions. Thus also the synchrony in computation steps corresponds to potentially simultaneously occurring transitions. As shown in [14], in case the membrane system evolves in a synchronous fashion (i.e., with a maximally concurrent execution of reaction rules in each computation step), its computations are faithfully reflected in the maximally concurrent step sequence semantics of its Petri net. In *Place/Transition nets with localities* (or PTL-nets), the specific class of Petri nets introduced in [14], each transition moreover belongs to a location, similar to the distribution of the reaction rules over the compartments in a membrane system. Since locality aspects of the resources consumed and produced by transitions is explicitly supported by their underlying graph structure, this locality information is not relevant for the maximal concurrency semantics of a Petri net. However, transitions with associated localities can be used to restrict synchrony to certain locations: in each step, and for each locality actively involved in that step, as many transitions belonging to this locality as possible are executed.[4] Thus the PTL-net model and its *locally* maximal concurrency semantics facilitate the investigation of membrane systems working under the natural assumption that synchrony is restricted to individual compartments. (Observe that this semantics leads to a more general model: maximal concurrency can be studied in the framework of PTL-nets with only one locality). In general, a step sequence semantics for Petri nets provides important insights into concurrency aspects of a system when executed. Such semantics, however, are based on ordered sequences of steps which may obscure the true causal relationships between occurrences of transitions since not all ordering is a consequence of causality. Still information on causal relationships is often highly relevant for system design and analysis. As was recognized a long time ago (see [19]), Petri nets support a formal approach where this information is readily available. Runs (as given, e.g., by step sequences) are unfolded into structures which explicitly represent causality and concurrency (unraveling the steps). For this purpose, labelled occurrence nets, called *processes* are used (see, e.g., [4, 5, 11, 28]). The standard process semantics of Place/Transition nets (based on arbitrary steps) does not work in the PTL case due to lack of information on potential executability of transitions relevant

[4] With this semantics, PTL-nets are an example of so-called 'globally asynchronous locally synchronous', or GALS, systems (see [16]).

for the local maximality of executed steps. To cope with this problem, in [14] the occurrence nets generated by PTL-nets are adapted leading to the notion of barb-processes formally defined and investigated in [15].

Until now we have considered only a very basic class of membrane systems with simple evolution rules and evolving in a (locally) synchronous fashion. In this paper, we will attempt to establish a similar set-up for other existing, more sophisticated, variants and extensions of membrane systems. For each of these variants we intend to define a suitable (extension of) the PTL-net model with a proper semantics. Obviously, we aim at retaining the direct correspondence between (occurrences of) transitions and (application of) evolution rules in order to guarantee that (local) synchrony and asynchrony in the membrane systems have corresponding interpretations in the PTL-net. Note that this work is a preliminary investigation, and technical details are left to forthcoming papers.

2 Preliminaries

In this paper, a multiset (over a set X) is a function $\mathsf{m} : X \to \mathbb{N}$. By $\mathbb{N}^X$ we denote the set of multisets over X. For two multisets m and m' over X, we denote $\mathsf{m} \leq \mathsf{m}'$ if $\mathsf{m}(x) \leq \mathsf{m}'(x)$ for all $x \in X$. Moreover, a subset of X may be viewed through its characteristic function as a multiset over X, and for a multiset m we denote $x \in \mathsf{m}$ if $\mathsf{m}(x) \geq 1$. Multiset m over X is finite if there are finitely many $x \in X$ such that $\mathsf{m}(x) \geq 1$; the cardinality of m is then defined as $|\mathsf{m}| \stackrel{\text{df}}{=} \sum_{x \in X} \mathsf{m}(x)$. The sum of two multisets m and m' over X is given by $(\mathsf{m} + \mathsf{m}')(x) \stackrel{\text{df}}{=} \mathsf{m}(x) + \mathsf{m}'(x)$, the difference by $(\mathsf{m} - \mathsf{m}')(x) \stackrel{\text{df}}{=} \max\{0, \mathsf{m}(x) - \mathsf{m}'(x)\}$, as a total function extending set difference. The multiplication of m by a natural number n is given by $(n \cdot \mathsf{m})(x) \stackrel{\text{df}}{=} n \cdot \mathsf{m}(x)$. Moreover, any finite sum $\mathsf{m}_1 + \cdots + \mathsf{m}_k$ will also be denoted as $\sum_{i \in \{1,\ldots,k\}} \mathsf{m}_i$.

2.1 Basic membrane systems

A *(basic) membrane system* (of degree $m \geq 1$) [20, 24] is a construct $\Pi \stackrel{\text{df}}{=} (V, \mu, w_1^0, \ldots, w_m^0, R_1, \ldots, R_m)$, where:

- V is a finite *alphabet* consisting of (names of) objects;
- μ is a *membrane structure* given by a rooted tree with m nodes, representing the membranes — we assume that the nodes are given as the integers $1, \ldots, m$, and $(i, j) \in \mu$ will mean that there is an edge from i (parent) to j (child) in the tree of μ;
- each w_i^0 is a multiset of objects initially associated with membrane i;
- each R_i is a finite set of *reaction rules* or *evolution rules* r associated with membrane i, of the form $lhs^r \to rhs^r$, where lhs^r — the left hand side of r — is a non-empty multiset over V, and rhs^r — the right hand side of r — is a possibly empty multiset over

$$V \cup \{a_{out} \mid a \in V\} \cup \{a_{in_j} \mid a \in V \text{ and } (i, j) \in \mu\} .$$

Symbols a_{in_j} represent objects a that will be sent to a child node j and a_{out} stands for an a that will be sent out to the parent node. (Nodes represent membranes which in their turn define compartments: compartment c_j is defined by membrane m_j, if it is enclosed by m_j and in-between m_j and its children if any.) We also assume that no evolution rule r associated with the root of the membrane structure uses any a_{out} in rhs^r.

A membrane system Π as above evolves from configuration to configuration as a consequence of the application of (multisets of) evolution rules in each compartment. Formally, a *configuration* is a tuple $C \stackrel{\text{df}}{=} (w_1, \ldots, w_m)$ where each w_i is a multiset of object names; we define a *vector multi-rule* $\boldsymbol{R}$ as an element of $\mathbb{N}^{R_1} \times \cdots \times \mathbb{N}^{R_m}$. Given a vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$, we use as additional notation $lhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot lhs^r$ for the multiset of all objects in the left hand sides of the rules in $\widehat{R}_i$ and, similarly, $rhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot rhs^r$ is the multiset of all — possibly indexed — objects in the right hand sides.

We now come to a point where we need to make precise the execution semantics of the basic membrane system model. As we already mentioned, it can be defined in a number of ways, depending on the balance between *synchrony* and *asynchrony* in the allowed behaviours. We will consider four kinds of execution semantics that have been investigated in the area of membrane systems, i.e., *free* parallelism [25], *minimal* parallelism [10], *maximal* parallelism, and *locally maximal* parallelism.

First, given two configurations, $C = (w_1, \ldots, w_m)$ and $C' = (w'_1, \ldots, w'_m)$, C can *free-evolve* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{free} C'$) if there exists a vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$ such that for every $1 \leq i \leq m$, $lhs_i \leq w_i$ and, for each object $a \in V$,

$$w'_i(a) = w_i(a) - lhs_i(a) + rhs_i(a) + rhs_{parent(i)}(a_{in_i}) + \sum_{(i,j) \in \mu} rhs_j(a_{out}) \ ,$$

where $parent(i)$ is the father membrane of i unless i is the root in which case $parent(i)$ is undefined and $rhs_{parent(i)}(a_{in_i})$ is omitted. Note that any j in the last term must be a child membrane of i.
By the first condition, the configuration C has in each membrane i enough occurrences of objects for the application of the multiset of evolution rules $\widehat{R}_i$, and the second condition describes the effect of the application of the rules in $\boldsymbol{R}$. We further say that C can:

- *min-evolve* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{min} C'$) if $|R_1| + \cdots + |R_m| = 1$;
- *max-evolve* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{max} C'$) if there is no i and rule r in R_i such that $lhs^r + lhs_i \leq w_i$; and
- *lmax-evolve* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{lmax} C'$) if there is no i and rule r in R_i such that $lhs^r + lhs_i \leq w_i$ and $|R_i| \geq 1$.

A *free/min/max/lmax-computation* of Π is then defined to be a sequence of free/min/max/lmax-evolutions starting from $C_0 \stackrel{\text{df}}{=} (w^0_1, \ldots, w^0_m)$, the initial configuration.

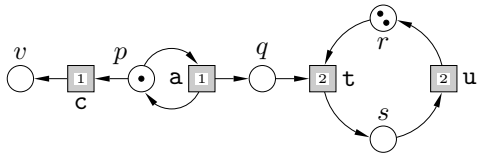

Fig. 1. PTL-net of the one-producer/two-consumers system.

2.2 Petri nets with localities

We first recall the key notions of the standard Petri net model. A *PT-net* is a tuple $N \stackrel{\text{df}}{=} (P, T, W, M_0)$ such that P and T are finite disjoint sets; $W : (T \times P) \cup (P \times T) \to \mathbb{N}$ is a multiset; and M_0 is a multiset of places. The elements of P and T are respectively the *places* and *transitions*, W is the *weight function* of N, and M_0 is the *initial* marking. In diagrams, places are drawn as circles, and transitions as rectangles. If $W(x, y) \geq 1$ for some $(x, y) \in (T \times P) \cup (P \times T)$, then (x, y) is an *arc* leading from x to y. As usual, arcs are annotated with their weight if this is 2 or more. We assume that, for every $t \in T$, there is a place p such that $W(p, t) \geq 1$.

Places represent local states, while markings are global states of systems represented by PT-nets. Transitions represent actions which may occur at a given marking and then lead to a new marking (the weight function specifies what resources are consumed and produced during the execution of such actions).

The *pre-* and *post-multiset* of a transition $t \in T$ are multisets of places given, for all $p \in P$, by: $\text{PRE}_N(t)(p) \stackrel{\text{df}}{=} W(p, t)$ and $\text{POST}_N(t)(p) \stackrel{\text{df}}{=} W(t, p)$. Both notations extend to multisets of transitions U:

$$\text{PRE}_N(U) \stackrel{\text{df}}{=} \sum_{t \in U} U(t) \cdot \text{PRE}_N(t) \qquad \text{and} \qquad \text{POST}_N(U) \stackrel{\text{df}}{=} \sum_{t \in U} U(t) \cdot \text{POST}_N(t) \; .$$

In order to represent the compartmentisation of membrane systems, one can add the notion of located transitions. In the proposed way of specifying locality for the transitions in a PT-net, each transition belongs to a fixed unique locality. The exact mechanism for achieving this is to introduce a partition of the set of all transitions, using a locality mapping $\mathfrak{D}$. Intuitively, two transitions for which $\mathfrak{D}$ returns the same value will be co-located.

Consider the PTL-net depicted in Figure 1. It conveys, in particular, the information that transitions `a` and `c` are assigned one locality, whereas transitions `t` and `u` are assigned another locality. This PTL-net is a model of a producer/consumer system which reflects the view that the producer operates away (at location 1) from the two consumers (location 2).

A *PT-net with localities* (or PTL-net) is a tuple $NL \stackrel{\text{df}}{=} (P, T, W, M_0, \mathfrak{D})$, where $\text{UND}(NL) \stackrel{\text{df}}{=} (P, T, W, M_0)$ is the *underlying* PT-net and $\mathfrak{D} : T \to \mathbb{N}$ is a *location mapping* for the transition set T. In the diagrams of PTL-nets, transitions are shaded rectangles with the locality being shown in the middle. Note that $\mathfrak{D}$ is merely a labelling of transitions, it is not meant as a renaming (as used later for occurrence nets).

We now can introduce execution semantics for the PTL-net which closely reflects the different degrees of synchrony in the behaviours of basic membrane systems.

A *step* is a multiset of transitions, $U : T \to \mathbb{N}$. It is *free-enabled* at a marking M (or $M[U\rangle_{free}$) if $M \geq \mathrm{PRE}_N(U)$. Thus, in order for U to be free-enabled at M, for each place p, the number of tokens in p under M should at least be equal to the total number of tokens that are needed as an input to U, respecting the weights of the input arcs. We further say that U is:

- *min-enabled* at M (or $M[U\rangle_{min}$) if $|U| = 1$;
- *max-enabled* at M (or $M[U\rangle_{max}$) if there is no transition t such that we have $M[U + \{t\}\rangle_{free}$; and
- *lmax-enabled* at M (or $M[U\rangle_{lmax}$) if there is no transition t such that we have $M[U + \{t\}\rangle_{free}$ and $\mathfrak{D}(t) \in \mathfrak{D}(U)$.

Thus localities are only relevant for lmax-enabledness.

Let $\mathfrak{m} \in \{free, min, max, lmax\}$ be a mode of execution. If U is $\mathfrak{m}$-enabled at M, then it can be $\mathfrak{m}$-*executed* leading to the marking $M' \stackrel{\mathrm{df}}{=} M - \mathrm{PRE}_N(U) + \mathrm{POST}_N(U)$. This means that the execution of U 'consumes' from each place p exactly $W(p,t)$ tokens for each occurrence of a transition $t \in U$ that has p as an input place, and 'produces' in each place p exactly $W(t,p)$ tokens for each occurrence of a transition $t \in U$ with p as an output place. If the $\mathfrak{m}$-execution of U leads from M to M' we write $M[U\rangle_{\mathfrak{m}} M'$. A finite sequence $\sigma = U_1 \ldots U_n$ of non-empty steps is an $\mathfrak{m}$-*step sequence* from the initial marking M_0 if there are markings $M_1, \ldots, M_n$ of N satisfying $M_{i-1}[U_i\rangle_{\mathfrak{m}} M_i$ for every $i \leq n$. Such a σ is also called an $\mathfrak{m}$-step sequence from M_0 to M_n, and M_n itself is called an $\mathfrak{m}$-*reachable* marking.

2.3 From basic membrane systems to PTL-nets

We now recall the details of the translation from the basic model of membrane system to PTL-nets introduced in [14]. Let $\Pi = (V, \mu, w_1^0, \ldots, w_m^0, R_1, \ldots, R_m)$ be a membrane system of degree m. Then the corresponding PTL-net is $NL_\Pi \stackrel{\mathrm{df}}{=} (P, T, W, M_0, \mathfrak{D})$ where the various components are defined thus:

- $P \stackrel{\mathrm{df}}{=} V \times \{1, \ldots, m\}$;
- $T \stackrel{\mathrm{df}}{=} T_1 \cup \ldots \cup T_m$ where each T_i contains a distinct transition t_i^r for every evolution rule $r \in R_i$;
- for every place $p = (a, j) \in P$ and every transition $t = t_i^r \in T$,

$$W(p,t) \stackrel{\mathrm{df}}{=} \begin{cases} lhs^r(a) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad W(t,p) \stackrel{\mathrm{df}}{=} \begin{cases} rhs^r(a) & \text{if } i = j \\ rhs^r(a_{out}) & \text{if } (j,i) \in \mu \\ rhs^r(a_{in_j}) & \text{if } (i,j) \in \mu \\ 0 & \text{otherwise} \end{cases}$$

- for every place $p = (a, j) \in P$, its initial marking is $M_0(p) \stackrel{\mathrm{df}}{=} w_j^0(a)$.

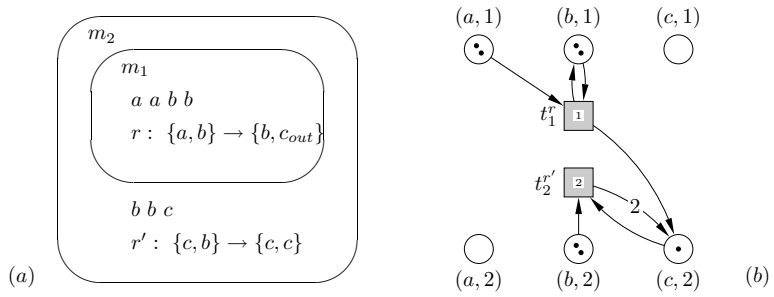

Fig. 2. A membrane system (a); and the corresponding PTL-net (b).

– for every transition $t = t_i^r \in T$, its locality is $\mathfrak{D}(t) \stackrel{\text{df}}{=} i$.

An example is the membrane system depicted in Figure 2(a). It consists of two nested membranes (m_1 and m_2), two rules (rule r associated with m_1, and rule r' associated with m_2; m_1 is the child and m_2 is the root in the membrane structure), and three symbols denoting molecules (a, b, and c). Initially, the compartment c_1 inside m_1 contains two copies of both a and b, and c_2, in-between the two membranes, contains two copies of b and a single copy of c. To model this membrane system as a PTL-net, we introduce a separate place (x, j) for each kind of molecule x and compartment c_j defined by membrane m_j. For each rule r associated with a membrane m_i we introduce a separate transition t_i^r with locality i. If the transformation described by a rule r of membrane m_i consumes k copies of molecule x from compartment c_j, then we introduce a k weighted arc from place (x, j) to transition t_i^r, and similarly for molecules produced by transformations. Finally, assuming that, initially, compartment c_j contained n copies of molecule x, we introduce n tokens into place (x, j). The resulting PTL-net is depicted in Figure 2(b).

Let $C = (w_1, \ldots, w_m)$ be a configuration of Π. Then the corresponding marking $\phi(C)$ of NL_Π is given by $\phi(C)(a, i) \stackrel{\text{df}}{=} w_i(a)$, for every place (a, i) of NL_Π. Similarly, for any vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$ of Π, we define a multiset $\psi(\boldsymbol{R})$ of transitions of NL_Π such that $\psi(\boldsymbol{R})(t_i^r) \stackrel{\text{df}}{=} \widehat{R}_i(r)$ for every $t_i^r \in T$. Note that ϕ is a bijection from the configurations of Π to the markings of NL_Π, and ψ is a bijection from vector multi-rules of Π to steps of NL_Π.

We now can formulate a fundamental property concerning the relationship between the dynamics of the basic membrane system Π and that of the corresponding PTL-net. Let $\mathfrak{m} \in \{free, min, max, lmax\}$ be a mode of execution of membrane systems. Then: $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{\mathfrak{m}} C'$ *if and only if* $\phi(C)\ [\psi(\boldsymbol{R})\rangle_{\mathfrak{m}}\ \phi(C')$. Since the initial configuration of Π corresponds through ϕ to the initial marking of NL_Π, the above immediately implies that the $\mathfrak{m}$-computations of Π coincide with the $\mathfrak{m}$-step sequences of the PTL net NL_Π.

Causality and concurrency The four different modes of execution of PTL-nets provide important insights into the concurrency aspects of the underlying systems. They are, however, still sequential in nature in the sense that steps occur ordered thus obscuring the true causal relationships between the occurrences of transitions. On the other hand, information on causal relationship is often of high importance for system analysis and/or design. Petri nets can easily support a formal approach where this information is readily available as was recognised a long time ago, see [19] where it was proposed to unfold behaviours into structures allowing an explicit representation of causality, conflict and concurrency. A well-established way of developing such a semantics is based on a class of acyclic Petri nets, called *occurrence nets* [28]. What one essentially tries to achieve is to trace the changes of markings due to transitions being executed along some legal behaviour of the original PT-net, and in doing so record which resources were consumed and produced.

Free parallelism Looking at the free-step sequence $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ of the PTL-net in Figure 1, it is not immediate that transition $\mathtt{u}$ could have occurred before the second occurrence of transition $\mathtt{a}$ or, in other words, that the former is not causally dependent on the latter.

Figure 3 illustrates the idea in which we *unfold* the scenario represented by σ. The initial stage shows just the initial marking which includes two separate (labelled) *conditions* (this is how places are called in occurrence nets) to represent the two initial tokens in place r. Executing step $\{\mathtt{a}\}$ consumes the p-condition, creates an $\mathtt{a}$-event (this is how transitions are called in occurrence nets), as well as two new conditions: a p-condition and a q-condition. An important point is to notice that we create a fresh p-condition rather than a loop back to the initial one since we want to distinguish between different occurrences of the same token; as a result the occurrence net being constructed will be an acyclic graph. Another important point is that the environment of the generated $\mathtt{a}$-event corresponds exactly to the environment of transition $\mathtt{a}$; namely, it consumes a p-token and creates a p-token and a q-token. After that, executing step $\{\mathtt{t},\mathtt{a}\}$ consists in consuming three conditions and creating two events and three fresh conditions, and similarly for the last step $\{\mathtt{u},\mathtt{t}\}$. And, as a final result, we obtain an acyclic net labelled with places and transitions of the original PT-net; it is called a *process* of the original PT-net. The process net has a default initial marking consisting of a token in each of the conditions without an incoming arc.

It is now possible to look both at the structure of the process net and the executions which are possible from its default initial marking, making some important observations relating to:

- *Causality.* The causality relationships among the executed transitions can be read-off by following directed paths between the events; for example in Figure 3, the lower $\mathtt{t}$-event is caused by both $\mathtt{a}$-events, while the upper one is caused only by the leftmost $\mathtt{a}$-event.
- *Concurrency.* Events for which there is no directed path from one to another can be thought of as concurrent.

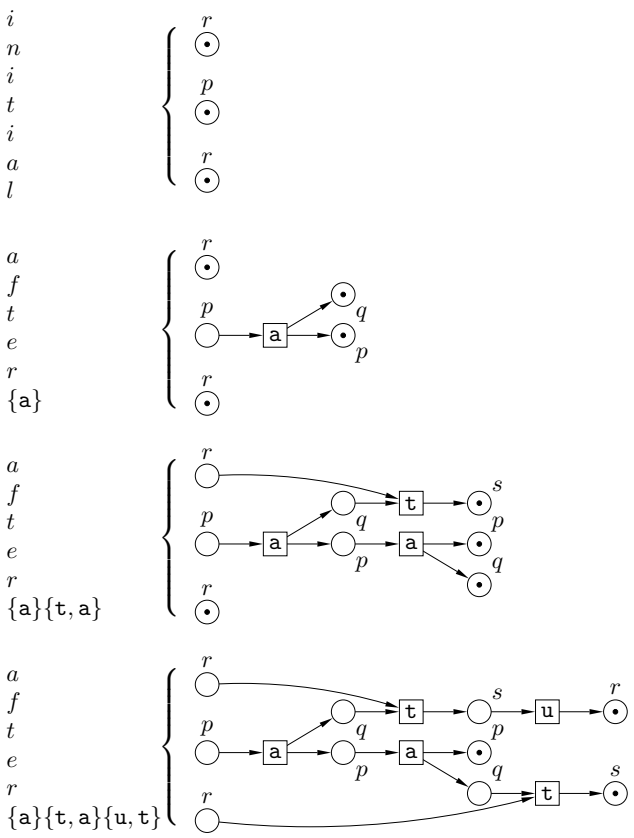

Fig. 3. Constructing a process net corresponding to $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ (localities are omitted as they are not relevant for the free parallelism semantics).

- *Reachability.* Any maximal set of conditions for which there is no directed path from one condition to another corresponds to a reachable marking of the original PT-net.
- *Representation.* The step sequence on the basis of which the process was created can be executed from the initial default marking in the occurrence net. So the original behaviour has been retained. In Figure 3, there are several different free-step sequences generated by the process net defined by $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$, including σ itself.
- *Soundness.* Any step sequence which can be executed from the default initial marking to the default final marking (consisting of tokens placed in each of the conditions without an outgoing arc) of the process net is also a legal step sequence of the original PT-net. Processes provide a highly compressed representation of step sequence behaviours of the original PT-net (this fea-

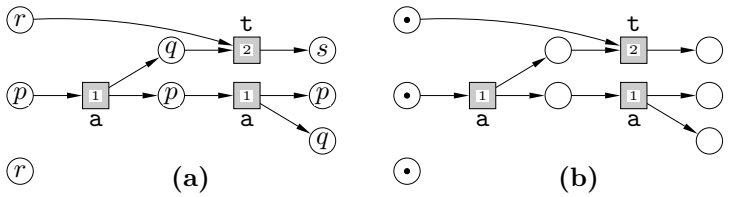

Fig. 4. Process net corresponding to the step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$ (a); and its default initial marking (b).

ture has been exploited to a significant degree in the development of efficient model checking algorithms for PT-nets).

Minimal parallelism From the point of view of causality and concurrency, the minimal parallelism semantics is almost exactly the same as in the case of free parallelism. The only difference is that soundness is formulated with respect to step sequences where each component is a singleton, rather than a general finite multiset of transitions.

Locally maximal parallelism As a first attempt, we simply adopt the unfolding strategy as in the case of free parallelism. We only ensure that the step sequence consists of lmax-steps. Moreover, we preserve the localities of the transitions in the events created while constructing the occurrence net. Figure 4 shows the result for the PTL-net of Figure 1 and the lmax-step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$. Although this is straightforward, we still need an argument that the resulting process is what one would want to take for further analyses. In particular, one would want to retain the soundness of the previous construction. In the case of our example, we can execute the occurrence net and conclude that under the locally maximal parallelism it admits the step sequence $\{\mathtt{a}\}\{\mathtt{a}\}\{\mathtt{t}\}$ which is not a legal lmax-step sequence since after $\{\mathtt{a}\}\{\mathtt{a}\}$, two occurrences of t are enabled. Thus, in general it would be too hasty to accept the standard unfolding routine as satisfactory since information on (additional) enabledness may be lost.

Consider further the PTL-net in Figure 5(a) and its lmax-step sequence $\{\mathtt{t},\mathtt{u},\mathtt{v}\}\{\mathtt{w},\mathtt{z}\}$. Proceeding as in the case of free parallelism, we obtain an occurrence net shown in Figure 5(b). Now the problem is that it has an lmax-step sequence from the default initial marking which corresponds to $\{\mathtt{u},\mathtt{v}\}\{\mathtt{t},\mathtt{z}\}\{\mathtt{w}\}$. The latter, however, is not an lmax-step sequence of the original PTL-net. An intuitive reason is that the standard unfolding 'forgets' that transition `x` was enabled at a stage where transition `w` was selected. Then, delaying the execution of the `w`-event, creates a situation where the executed step (though lmax-enabled within the occurrence net) does not correspond to an lmax-step in the PTL-net.

To cope with the above problem, [15] added to occurrence nets special *barb-events*, represented by darkly shaded rectangles. Barb-events are not labelled with transition names and are not meant to be executed; rather, they are used

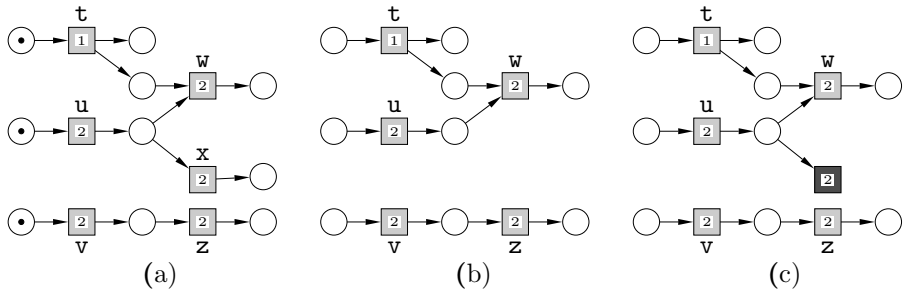

Fig. 5. PTL-net (a); an occurrence net constructed from step sequence $\{\mathtt{t},\mathtt{u},\mathtt{v}\}\{\mathtt{w},\mathtt{z}\}$ (b); and a barbed process (c).

in the calculation of the enabled sets of events. Such occurrence nets are called *barbed* processes. Rather than providing a full formal definition of how barb-events are added during the unfolding procedure, which can be found in [15, 16], we only mention here that it is based on checking for the existence of locally newly enabled transitions not (yet) included in the executed scenario, e.g., since another co-located transition was selected.

Figure 5(c) illustrates the modified construction for the net in Figure 5(a,b). After executing $\{\mathtt{u},\mathtt{v}\}$, it is now impossible to select $\{\mathtt{t},\mathtt{z}\}$ since there is a record in the form of the barb-event that such a step would not be maximal in the locality to which transition $\{\mathtt{z}\}$ belongs. The only way of continuing is to execute $\{\mathtt{t}\}$ and after that $\{\mathtt{z},\mathtt{w}\}$, generating a legal lmax-step sequence $\{\mathtt{u},\mathtt{v}\}\{\mathtt{t}\}\{\mathtt{z},\mathtt{w}\}$.

Maximal parallelism The maximal parallelism semantics of a PTL-net coincides with the locally maximal parallelism semantics of this PTL-net after changing it so that all transitions are mapped to the same locality.

3 Extensions expressible within PTL-nets

In the previous section we outlined the way in which the basic membrane systems can be translated into PTL-nets, and their behavioural properties investigated using processes nets of the latter. In the rest of the paper, we will change focus and investigate what happens if more sophisticated types of reaction rules are allowed. For the sake of simplicity, we will assume from now on that the membrane systems and PTL-nets are executed according to the free-parallelism paradigm (notice that the level of synchrony present in executions is orthogonal to the way individual reaction rules are specified).

We start by considering extensions for which the PTL-net semantics can be used without any, or with only slight, modifications. These extensions have been discussed in [23, 2] and additional references will be provided throughout the text. Note that each extension is motivated by some natural phenomenon in the area of biological systems.

Catalysts In this variant, a subset Cat of objects, called *catalysts*, is distinguished and each reaction rule is of the form $lhs^r \to rhs^r$ with either no catalysts involved at all or with $lhs^r(c) = rhs^r(c) = 1$ for exactly one $c \in Cat$ and $lhs^r(c') = rhs^r(c') = 0$ for all other catalysts c'. In other words, in certain reaction rules a catalyst has to participate, but it is neither destroyed in the process nor can be created. Clearly, since catalysts can be seen as resources for the reaction rules in which they occur (to be returned after application), these rules can be translated into PTL-transitions in exactly the same way as any other rule. Thus, the translation from Section 2.3 is fully adequate. Similarly, other variants of catalysts, such as m-stable catalysts and mobile catalysts, can also be treated by this basic translation.

Rules creation and consumption Within the basic model of membrane systems no assumptions are made with respect to the number of times a reaction rule is available for application in a single execution step. Now, it is assumed that reaction rules are finite resources in the same way as the objects located in compartments [3]. More precisely, each configuration has additional information for each membrane about the number of locally available copies of each rule. Each rule r is of the form $lhs^r \to rhs^r/z$, with z a multiset over the set of rules. Rules are executed in the usual manner with respect to the multisets of objects consumed and created. Moreover, if r when executed is associated with membrane i, then a copy of r is consumed from the multiset of rules currently available in i and the multiset z is added to that pool. Note that, we may assume that each rule is associated with all membranes.

In this case the translation proceeds as in Section 2.3 with two key modifications: (i) for each transition t_i^r corresponding with rule r associated with membrane i, a unique *control place* is added which acts as a counter and indicates the number of copies of r available in the corresponding membrane; (ii) this control place is an additional input place to t_i^r and if r is of the form $lhs^r \to rhs^r/z$, then t_i^r has, for each control place corresponding with a rule $r' \in z$ associated with i, an additional output place with weight $z(r')$.

Systems with i/o communication These systems are defined as in Section 2.1, except that in rules r of the form $lhs^r \to rhs^r$ the right hand side rhs^r is a multiset over $V \cup \{a_{out} \mid a \in V\} \cup \{a_{in} \mid a \in V\}$. The index in of a_{in} means that a copy of object a is to be moved into *any* of the inner membranes of the membrane to which r belongs. Thus every rule represents a set of rules of the original form, each such rule corresponding to a combination of non-deterministic choices of inner membranes for all occurrences of an a_{in}. Thus the translation has to be lifted to a more abstract level and each reaction rule involving sending objects to inner membranes is translated into a set of transitions with the same pre-multisets, but possibly different post-multisets. For example, if 3 and 7 are two inner membranes for the rule $ab \to c_{in}d_{in}$, then this rule is translated into four transitions, with the following post-multisets: $\{(c,3),(d,3)\}$, $\{(c,3),(d,7)\}$, $\{(c,7),(d,3)\}$ and $\{(c,7),(d,7)\}$.

Symport/antiport In a symport/antiport membrane system the rules associated with a membrane i are of one of the forms (x, in) or (y, out) — symport rules — or $(x, in; y, out)$ — antiport rules. Here (x, in) means that the multiset x is moved from the outside of i to its inside and, similarly, (y, out) means that multiset y goes from the inside of i to its outside. Moreover, $(x, in; y, out)$ means that x and y are moved simultaneously. Note, that with the given membrane structure, it must be the case that x moves from the 'location' of the parent of i to i and y in the opposite direction. Consequently, we can again apply the basic translation in case of rules of the first two forms. The third one is somewhat different since it consumes objects from two neighbouring compartments. However, its translation is straightforward and what we simply obtain is a transition taking tokens from places corresponding to different compartments of the original membrane system.

Tissue membrane systems In this case objects are transported through channels rather than membranes. Thus the nested tree-like structure of membranes is replaced by a graph, with its edges representing channels connecting compartments in a completely arbitrary way. Often it is assumed that at most one (symport or antiport) rule associated with a channel is executed at any given moment. Since the actual membrane structure is not relevant for the translation, the first assumption has no effect, and the translation looks as in the case of symport and antiport rules. The second assumption can be addressed by introducing, for each communication channel, a special place marked initially with a single token which is connected by a pair of arcs (pointing in opposite directions) with every transition representing a reaction rule associated with that channel. In this way, there can never occur more than one of these transitions at the same time. As in the case of rules creation and consumption, these additional places are an example of what might be called a 'control structure' which can be used in the Petri net model to implement a specific behavioural aspect of membrane systems.

4 Other extensions

Although it is possible to use the basic class of PTL-nets to analyse various important classes of membrane systems, not all interesting phenomena can be modelled by using purely the features of PTL-nets.

Promoters It is now assumed that a reaction rule can have the form $lhs^r \rightarrow rhs^r|_c$ meaning that c is a promoter object which has to be present for the rule to be executed [6]. It should be stressed that such an object is not a catalyst since catalysts are actively involved in reactions, whereas a single occurrence of c in its role of promoter may enable simultaneously two or more executions of the rule.

It turns out that the standard model of PTL-nets is no longer sufficient for the modelling of promoters because arcs between transitions and places indicate

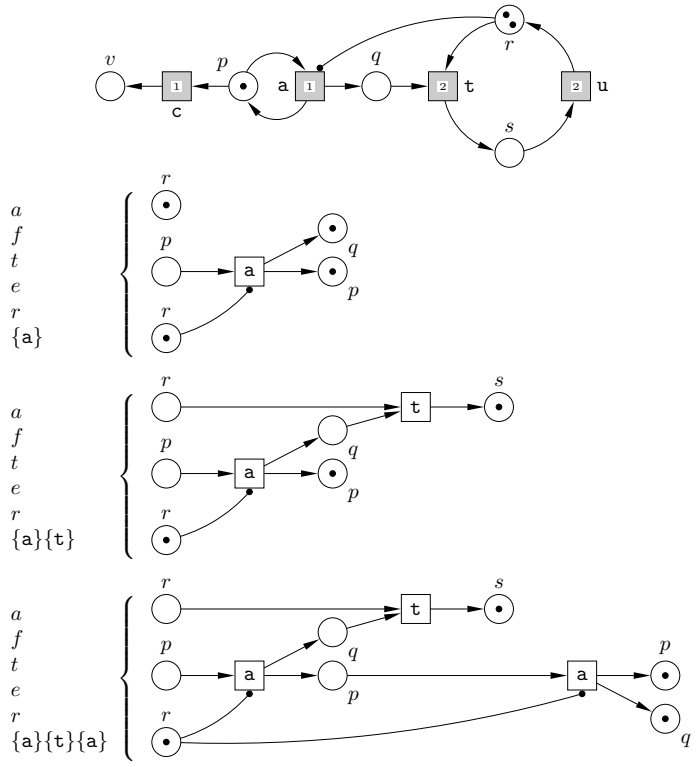

Fig. 6. PTLA-net of the one-producer/two-consumers system with a non-eager producer, and constructing an ao-process corresponding to {a}{t}{a}.

consumption and production. We need an extension with *(weighted) activator* arcs, represented by arcs with small black dots at the end. We call the extended model PTLA-nets. Activator arcs represent 'tests' for the presence of tokens in places. An activator arc of weight n between place p and transition t implies that the latter can only be executed if the former contains at least n tokens. The resulting marking is calculated in exactly the same way as before, i.e., the activator arcs are simply ignored.

The translation of reaction rule $lhs^r \to rhs^r|_c$ associated to membrane i proceeds as the basic translation for $lhs^r \to rhs^r$, and after that an activator arc of weight 1 is added to link the resulting transition with place (c, i). A more elaborate definition of promoters assumes the format $lhs^r \to rhs^r|_u$ where u is a multiset of objects. The translation then proceeds similarly but now a number of activator arcs of weights greater or equal to 1 are added at the end.

The process semantics of the resulting translations can no longer be captured using the standard process semantics of Petri nets. What we use are *activator processes* (or *ao-processes*) which are basic process nets with additional

(weight 1) activator arcs between events and conditions to test for the presence of tokens in the places corresponding to the conditions. With the distinguishing feature of activator arcs being that they do not consume conditions, there may be several activator arcs adjacent to a single condition (in addition, of course, to the standard directed arcs and in contrast with the non-branching of conditions with respect to ordinary arcs).

Consider, for instance, the net in Figure 6 which models a system where the producer only produces items if there is at least one consumer waiting for them. A possible free-step sequence is $\{\mathtt{a}\}\{\mathtt{t}\}\{\mathtt{a}\}$ and the construction of the corresponding ao-process is illustrated in Figure 6. Notice that we have here a condition which is connected by activator arcs to two different events.

The causality semantics of ao-processes is no longer the same as that of the standard processes. Basically, in the latter causality is based on partial orders whereas in ao-processes another relationship, called *weak causality*, is needed. It turns out that the standard partial order treatment of causality can be extended to cover its weak variant as well, and the main results and properties can be recovered [12, 13].

Inhibitors Inhibitors are objects the presence of which makes the execution of certain rules impossible. In this case, a reaction rule can have the form $lhs^r \to rhs^r|_{\neg c}$ meaning that c is an object which, when present in the compartment, inhibits the execution of this rule [6]. Again, and for the same reason as with promoters, we need to extend PTL-nets, in this case with *(weighted) inhibitor* arcs, represented by arcs with small circles at the end (note that activator and inhibitor arcs are existing extensions of the standard Petri net model). We call the extended model PTLI-nets. The meaning of an inhibitor arc of weight $n \geq 0$ between place p and transition t is that the latter can only be executed if the former contains at most n tokens (thus, if $n = 0$ then the place must be empty of tokens). The resulting marking is calculated in exactly the same way as before, i.e., the inhibitor arcs are simply ignored.

The translation of reaction rule $lhs^r \to rhs^r|_{\neg c}$ associated to membrane i proceeds as the basic translation for $lhs^r \to rhs^r$, and after that an inhibitor arc of weight 0 is added to link the resulting transition with place (c, i). A more elaborate definition of inhibitors assumes the format $lhs^r \to rhs^r|_{\neg u}$ where u is a set of object symbols. The translation proceeds then similarly but now inhibitor arcs of any weights can be added.

The process semantics of the resulting translations can be captured using the ao-process semantics as in the case of promoters. Consider, for instance, the net in Figure 7 which models a system where the producer can cancel the production of items only if there is no consumer waiting for them.

A possible free-step sequence is $\{\mathtt{a}\}\{\mathtt{a},\mathtt{t}\}\{\mathtt{t}\}\{\mathtt{c}\}$ and the construction of the corresponding ao-process is illustrated in Figure 7. Note that activator arcs rather than inhibitor arcs are used to test for the holding of conditions. Two activator arcs are used to represent the test for the presence of two tokens in the place s. This ensures that place r is empty since in the PTLI-net of Figure 7 the

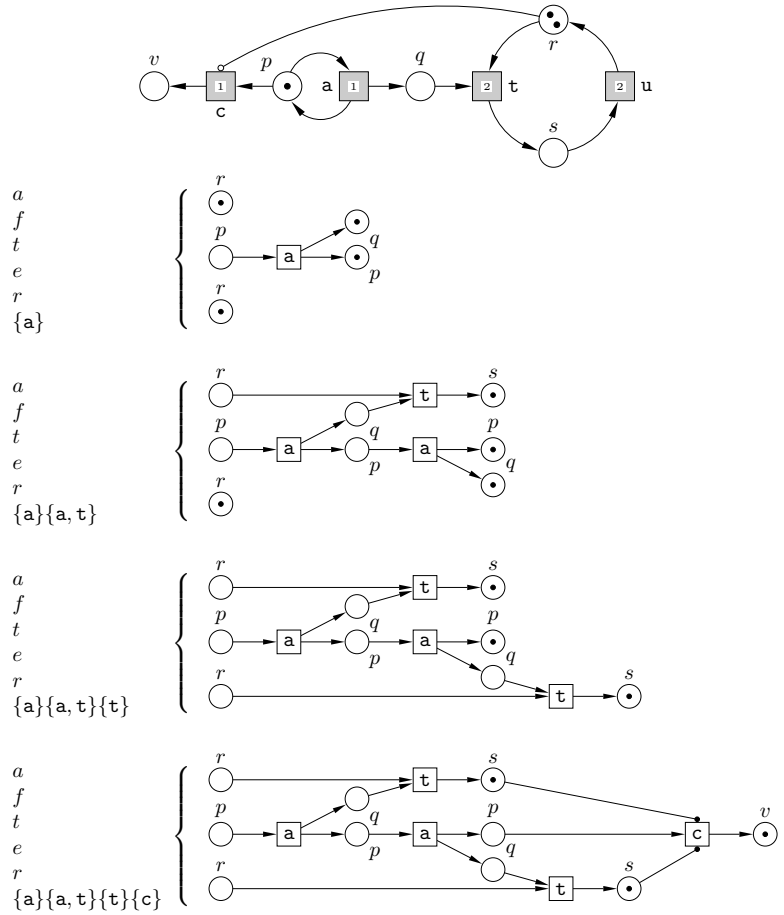

Fig. 7. PTLI-net of the one-producer/two-consumers system with considerate producer, and constructing a process net corresponding to $\{\mathtt{a}\}\{\mathtt{a},\mathtt{t}\}\{\mathtt{t}\}\{\mathtt{c}\}$.

total number of tokens in places s and r is always equal to 2. In Petri net terminology places like that are called *complementary* and the modelling of inhibitor arcs in a process semantics is then rather straightforward. In case a complement for a place like r cannot be found, a more elaborate construction can be used to achieve the desired effect [12, 13]. Since the causality semantics of PTLI-nets is based on ao-processes, it takes into account weak causality as for PTLA-nets.

Permeable membranes The new kind of reaction rule allowed here is $lhs^r \rightarrow rhs^r \,/\tau$. The special symbol τ indicates that this rule, when executed, causes its associated enclosing membrane to become 'thick' or non-permeable, and no object can pass through it anymore [22]. To render this feature within the Petri

net model, we introduce a special, initially empty, place $perm_i$ associated with the membrane i. A directed arc is added to $perm_i$ from those transitions which correspond to rules which make membrane i thick (thus there may be several transitions which can put tokens into the control place $perm_i$). Then each transition t which models a reaction rule r' transferring objects through membrane i, is connected with $perm_i$ using a simple inhibitor arc. Hence, as long as no transition which places a token into the control place $perm_i$ is executed, transitions transferring objects through the membrane i are possible. However, once there is at least one token in $perm_i$, transitions corresponding to rules like r' can no longer be executed as no transition can remove tokens from $perm_i$. As a result, PTLI-nets with the associated ao-processes are sufficient to model the effect of the non-permeability of membranes.

Dissolving membranes To dissolve membranes, reaction rules are used of the form $lhs^r \rightarrow rhs^r/\delta$ with δ a special symbol indicating that execution of this rule causes its associated enclosing membrane (which may not be the skin) to dissolve [22, 21]. Moreover, all objects present in the compartment are incorporated into the immediately enclosing compartment, and all the rules associated to membrane i are rendered inapplicable. The dissolving of membranes may be modelled by a combination of activator and inhibitor arcs. Take, for example a membrane system with four membranes, arranged into a line-like tree $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Assume further that there are three dissolving rules: r' associated with 2, r'' associated with 3, and r''' associated with 4. Then we add three control places for keeping information about dissolved membranes: $diss_2$, $diss_3$ and $diss_4$ which are initially empty and each has an incoming directed arc from the transitions corresponding respectively to r', r'' and r'''.

Suppose now that we need to translate a rule $r : a \rightarrow aa$ associated with membrane 2. To achieve the desired effect, we introduce three transitions with locality 2: t_2^r with pre-multiset $\{(a,2)\}$ and post-multiset $\{(a,2),(a,2)\}$; t_3^r with pre-multiset $\{(a,3)\}$ and post-multiset $\{(a,3),(a,3)\}$; and t_4^r with pre-multiset $\{(a,4)\}$ and post-multiset $\{(a,4),(a,4)\}$. Then we add an inhibitor arc between $diss_2$ and each of these three transitions, as well as three activator arcs: between t_3^r and $diss_3$, t_4^r and $diss_3$, t_4^r and $diss_4$. In this way, t_4^r can be executed only if the dissolution rules r'' and r''' have happened, but r' has not. Note that such a translation can properly render even a simultaneous application of multiple instances of the dissolution rules. Process semantics of the resulting Petri net is a combination of those of PTLA-nets and PTLI-nets.

5 Concluding remarks

A main advantage of process semantics is that it provides a very compact representation of behaviour. This feature has been exploited in the development of efficient model checking algorithms [18], where issues relating to reachability of certain configurations and termination (or deadlock) of a system can be addressed.

A number of relevant behavioural issues can be efficiently investigated given a process notion for membrane systems. For example, one can check for the presence of certain molecules (also in specific compartments), by suitably adapting the notion of reachability in a process net. In some cases, it might be important to know whether local computations within compartments (and across the whole system) are independent of each other, and answering this kind of question could amount to checking for the causal links between various events present in processes.

There are several possible directions for future work. The first is to complete the development of the theory of barb-processes making it fit into the *semantical framework* of [12], and develop suitable process notions and derived causality structures for other membrane systems and execution semantics. Another important question is a complete characterisation of state graphs generated by various models of PTL-nets allowing, in particular, to answer the question whether a given state graph could have been generated by a membrane system of a given kind (such a characterisation has so far been provided for the class of safe PTL-nets [17]). Last but not least, once a sound notion of behavioural characterisation of a membrane system has been provided, one can re-introduce the notion of a successful computation, the result it produces, and the notion of an input-output relation. However, we now can go further and investigate also non-successful computations which the computability oriented approach presently ignores.

Acknowledgment We are grateful to Grzegorz Rozenberg for introducing us to the area of membrane systems and many inspiring discussions. This research was supported by the EPSRC project CASINO.

References

1. Membrane systems web page: `http://psystems.disco.unimib.it/`
2. A.Alhazov (2006). *Communication in Membrane Systems with Symbol Objects.* Rovira i Virgili University, Tarragona, Spain.
3. F.Arroyo, A.V.Baranda, J.Castellanos and Gh.Păun (2002). Membrane Computing: The Power of (Rule) Creation. *Journal of Universal Computer Science* 8, 369–381.
4. E.Best and R.Devillers (1988). Sequential and Concurrent Behaviour in Petri Net Theory. *Theoretical Computer Science* 55, 87–136.
5. E.Best and C.Fernández (1988). *Nonsequential Processes. A Petri Net View.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag.
6. P.Bottoni, C.Martín-Vide, Gh.Păun and G.Rozenberg (2002). Membrane Systems with Promoters/Inhibitors. *Acta Informatica* 38, 695–720.
7. C.S.Calude, Gh.Păun, G.Rozenberg and A.Salomaa (Eds.) (2001). *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View.* Springer-Verlag, LNCS 2235.
8. S.Dal Zilio and E.Formenti (2004). On the Dynamics of PB Systems: a Petri Net View. Proc. of *WMC 2003*, C.Martín-Vide et al. (Eds.). Springer-Verlag, LNCS 2933, 153–167.

9. J.Desel, W.Reisig and G.Rozenberg (Eds.) (2004). *Lectures on Concurrency and Petri Nets*. Springer-Verlag, LNCS 3098.
10. R.Freund (2001). Sequential P Systems. *Romanian Journal of Information Science and Technology* 4, 77–88.
11. U.Goltz and W.Reisig (1983). The Non-sequential Behaviour of Petri Nets. *Information and Control* 57, 125–147.
12. H.C.M.Kleijn and M.Koutny (2004). Process Semantics of General Inhibitor Nets. *Information and Computation* 190, 18-69.
13. H.C.M.Kleijn and M.Koutny (2006). Infinite Process Semantics of Inhibitor Nets. Proc. of *ICATPN 2006*, S.Donatelli and P.S.Thiagarajan (Eds.). Springer-Verlag, LNCS 4024, 282–301.
14. J.Kleijn, M.Koutny and G.Rozenberg (2006). Towards a Petri Net Semantics for Membrane Systems. Proc. of *WMC'05*, R.Freund, Gh.Paun, G.Rozenberg and A.Salomaa (Eds.). Springer-Verlag, LNCS 3850, 292–309.
15. J.Kleijn, M.Koutny and G.Rozenberg: Process Semantics for Membrane Systems. To appear in the Journal of Automata, Languages and Combinatorics.
16. H.C.M.Kleijn, M.Koutny and G.Rozenberg (2006). Processes of Petri Nets with Localities. TR 941, School of Computing Science, University of Newcastle.
17. M.Koutny and M.Pietkiewicz-Koutny (2006). Transition Systems of Elementary Net Systems with Localities. Proc. of *CONCUR 2006*, C.Baier and H.Hermanns (Eds.). Springer-Verlag, LNCS 4137, 173–187.
18. K.L.McMillan (1992). Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of *CAV'1992*, G.von Bochmann and D.K.Probst (Eds.). Springer-Verlag, LNCS 663, 164–174.
19. M.Nielsen, G.Plotkin and G.Winskel (1980). Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science* 13, 85–108.
20. Gh.Păun (2000). Computing with Membranes. *Journal of Computer and System Sciences* 61, 108–143.
21. Gh.Păun (2000). Computing with Membranes. *Journal of Computer and System Sciences* 61, 108–143.
22. Gh.Păun (2000). Computing with Membranes – A Variant. *International Journal of Foundations of Computer Science* 11, 167–182.
23. Gh.Păun (2002). *Membrane Computing. An Introduction*. Springer-Verlag.
24. Gh.Păun and G.Rozenberg (2002). A Guide to Membrane Computing. *Theoretical Computer Science* 287, 73–100.
25. Gh.Păun and S.Yu (1999). On Synchronization in P Systems. *Fundamenta Informaticae* 38, 397–410.
26. Z.Qi, J.You and H.Mao (2004). P Systems and Petri Nets. Proc. of *WMC 2003*, C.Martín-Vide et al. (Eds.). Springer-Verlag, LNCS 2933, 286–303.
27. W.Reisig and G.Rozenberg (Eds.) (1998). *Lectures on Petri Nets*. Springer-Verlag, LNCS 1491, 1492.
28. G.Rozenberg and J.Engelfriet (1998). Elementary Net Systems. In: *Advances in Petri Nets. Lectures on Petri Nets I: Basic Models*, W.Reisig and G.Rozenberg (Eds.). Springer-Verlag, LNCS 1491. 12–121.