Let me try to put down some of the ideas discussed last year;
hopefully it will be useful, if not programmed in the near-future, then at least as a small publication.

1. Simulator. It would be very useful for theory to have a proper tool computing **the set of all possible transitions** from a given configuration.
(Yes, I remember you are more focused on applications like zebra muscles, and you are quite concerned that it does not scale well. However, enough theory is anyway done, and there can be multiple simulators for PLingua. A few times I have been so upset that I thought about programming something like that myself, but what I do is normally not compatible, not user-friendly and definitely not in Java)
Basically, having fixed the current configuration C, for each rule r it is easy to compute the maximal number max(r,C) of times it can be applied in parallel. By dividing, for each object a in lhs(r), |C|_a by |lhs(r)|_a rounding down, and taking the minimum. Same works in a distributed way, assuming proper flattening, possibly on the fly. In the _worst_ case, all possibilities are among the combinations, for each rule r, of applying it from 0 to max(r,C) times. It only takes to verify that the multiset union of lhs(r) times the number of applications of r, summed over all rules r, is contained in r. That would be asynchronous mode. For any other mode, compliance is also to be checked. Of course, for maxpar that would be non-applicability of ANY further rule to the idle objects. Of course, in particular cases the set of possible transitions could be computed more efficiently.

2. Semantics and membranes. The recent advances in PLingua, like defining user rule types, seem to be quite useful. Yet, the main value I see is being able to specify rules other than the most usual ones in the model (and, in particular, being able to combine rules from different models), and a comfortable way to write them is secondary, though also nice.
A thing which is often related to syntax is how to apply it, the most needed versions being "in max. parallelism" and "sequentially". In particular, rules (a) are normally treated as parallel, even though sequential version has been considered, while rules (b),(c),(d),(e),… are normally considered sequential, even though without polarizations alternative semantics has been studied.

Imagining rules involving more than one membrane, we need to be more precise. I proposed to indicate for each user type of rules (how exactly is a secondary question) which membranes are resources and which membranes are context. Then, resources are lhs(r) and contexts are like promoters. Clearly, under usual definitions a rule would be sequential with respect to resources and parallel with respect to contexts.

If that is too difficult, usually it is enough to have implicit semantics: any membrane written completely in lhs(r) is a reactant, a membrane written in rhs(r) is a product, and a membrane containing "→" is a context. Then, [a→u] is a parallel rule, but [a]→[u] would be a sequential rule. Similarly, it automatically follows in [[]→[][]] that the external membrane is a context, while internal membranes are resources, so we already know what is parallel and what is sequential. However, if the user wants such a rule to be sequential also w.r.t. the outer membrane, then it can be written as [[]]→[[][]] .

Unfortunately, this convention alone does not suffice for automatic deduction of parallelism for rules like (b_0), (c_0). Because the context is not outside. (Yes, they could be written as boundary rules, but this syntax is neither universal nor compatible with traditional syntax for active membranes). But of course something can always be invented, e.g., when specifying rule types, write "[p" vs "[s" (parallel vs sequential) or "[r" vs "[c" (resource vs context).

Moreover, I was told that there is some problem with templates without external membrane, except the standard types ☹

3. Dynamic membranes

Clearly, without explicitly indicated semantics []→[[]] would be a seqential membrane creation, while [a→[b]] would be a parallel membrane creation. Then,
[[a]→b] is a membrane dissolution, where the external membrane is a context.
But what is the behavior of other objects, those not specified in the rules explicitly?
The main variant is of course, upon creation the new membrane will only contain b, and upon dissolution all the contents of the old membrane is released in the outer membrane. But of course there are other rules, although less studied. Last year I suggested to use some wildcard, or mask, e.g., $1, to represent other objects (similarly, something like #1 can represent other membranes, and for technical reasons different characters may be chosen; I use these ones to explain the idea how to describe semantics different from the main one).

[a $1→$1 [b]] usual parallel membrane creation
a $1→$1 [b] same without the outer context
[a $1]→[b [$1]] create a new membrane around the existing one and send b there
[[a $1]→b $1] usual membrane dissolution
[[a $1]→b] lose contents of the dissolved membrane
[a $1]→[b $1][c $1] usual membrane division
[a $1]→[b $1][c] create a sibling membrane, without replicating contents
[a $1]→[$1(O)][$1(O')] membrane separation
[a $1[$2]]→[$2 [a $1]] exchange objects in two membranes if the first one contains a

Then, there may be different kinds of non-elementary membrane division
[a]→[b][c] same syntax as for elementary membranes, replicate objects and membranes.
Can be written as [a $1 #1]→[b $1 #1][c $1 #1]
[[][]]→[[]][[]] separating submembranes. But what exactly happens to other submembranes if there are more than these two?
[$1 #1[][]]→[$1 #1[]][$1 #1[]] replicating other objects and membranes
[$1 #1[][]]→[$1(O) #1[]][$1(O') #1[]] separating objects and replicating membranes
[$1 #1[][]]→[$1 #1(H)[]][$1 #1(H')[]] replicating objects and separating membranes
[$1 #1[][]]→[$1(O) #1(H)[]][$1(O') #1(H')[]] separating objects and membranes

Overall, I think there may be some reasonable consistent universal way how to describe the precise evolution not only of dedicated objects and membranes, but also related objects and membranes, because mass action is needed (the first classical example of the mass action is dissolution, of course currently programmed explicitly).

4. Tests
From time to time, researchers consider rules that were not considered in the original model.
I believe many (though not all) of these issues can be captured by the thoughts above.
a) sequential (a), parallel (b),(c),(d),(e),…
b) where other objects and membranes go - division vs separation, outside vs delete, …
c) external rules: a[]→u[], a[]→b, a[]→b[][], …
d) …

5. Other models besides active membranes
With suitable choice of parallel/sequential semantics made clear, $r \in R_i$ can be written as $[r]_i$. In most cases, membrane i must be treated as context, hence, rules are parallel with respect to it.
Antiport: u[v]→ v[u]
Evolutional antiport or boundary rules: u[v]→u'[v']
Transitional: pretty standard, except multiple targets would be represented as multi-membrane context, and dissolution semantics is normally assumed parallel (multiple δ = one dissolution), not sequential.
Spiking: mostly similar, the main difference are additional regular expressions.
Promoters, inhibitors - how much is already captured by PLingua??

Priorities - is there already a well-established syntax for them?
Notice that strong and week priorities can co-exist: these are just additional filters for the set of the next configurations (see part 1: Simulator) besides the derivation mode.
As discussed with Rudi a few days after BWMC19, filters like priorities should be applied BEFORE the derivation mode filter.

6. Other derivation modes.
A new (mostly studied in the last few years) important derivation mode for many models is **set maximally-parallel**. Same as maximally parallel, but in each step each rule may be only applied once. Technically similar to having a dedicated catalyst for each rule.

Some of the classical modes that would be most important to also have are sequential and asynchronous. Asyn is even easier than maxpar - just remove the maximality filter. Sequential is of course the easiest to implement.

7. New ways of rule control.
Activation and blocking (I hope to soon finish formalizing the concept also for zero-delay).

8. One of the "worst" things that could happen.
"Denying"
This is how we call the situation where there exists at least one applicable rule, but there is no valid multiset of rules. An example is ">1 mode" in the situation where only one rule is applicable. This situation has been carefully avoided in the first years of membrane computing, but it does not present a problem (except it is unusual), e.g., this is similar to what happens to partially blind register machines when they try to decrement a register containing zero, which is not allowed by the model.

Finally, a question is - can all of this co-exist in the same context?
I still think it could. If anyone has an example of ANY membrane features that seem incompatible, please let me know, and _maybe_ I will be able to convince you that there is no problem. Reminder - a universal look at P systems models: network of cells, see a few publications on the Formal Framework for a)static structures, b)dynamic structures, c)spiking.

R. Freund, S. Verlan: **A Formal Framework for Static (Tissue) P Systems**. In: Eleftherakis G., Kefalas P., Păun G., Rozenberg G., Salomaa A. (eds) Membrane Computing. WMC 2007. Lecture Notes in Computer Science 4860. Springer, Berlin, Heidelberg, 2007, 271-284. https://link.springer.com/chapter/10.1007%2F978-3-540-77312-2_17

R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez, S. Verlan: **A Formalization of Membrane Systems with Dynamically Evolving Structures**. International Journal of Computer Mathematics 90(4), 801–815 (2013)
https://doi.org/10.1080/00207160.2012.748899

S. Verlan, A. Alhazov, R. Freund, S. Ivanov: **A Formal Framework for Spiking Neural P Systems**. In Proceedings of the 20th International Conference on Membrane Computing, CMC20, Curtea de Argeş (Păun, Gh., Ed.). Bibliostar, Râmnicu Vâlcea, 2019, pp. 523–535.
http://membranecomputing.net/cmc20/pdf/procCMC20.pdf#page=250