Between Expressiveness and Verifiability: P/T-nets with Synchronous Channels and Modular Structure

Lukas Voß[1], Sven Willrodt[1], Daniel Moldt[1] and Michael Haustermann[1]

[1]*University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Department of Informatics*

Abstract

Synchronous channels are a powerful means to structure Petri net models. They enable large, expressive models while maintaining a coherent and well-readable structure. However, the vast number of potential bindings make Petri Nets extended with synchronous channels notoriously difficult to verify. This paper introduces synchronous channels to the basic P/T-net formalism while finding a compromise between the goals of increasing the modeling capabilities and remaining easy to verify.

As part of this paper, a formal definition and an implementation of P/T-nets with synchronous channels are provided. With the provided definition, the semantics and behavior of these models are formally described and well-defined. This forms a foundation for further work based on the formalism, such as verification or formalism extensions. Additionally, transformations are provided to construct equivalent regular P/T-nets, allowing the application of traditional P/T-net techniques. Restrictions on the synchronous channels ensure that these unfolded P/T-nets retain a reasonable size. The implementation furthermore includes a mechanism to partition nets into sub-nets, providing another means to create complex, yet comprehensive models. As a result, the formalism performs a balancing act by providing multiple means to structure large models while keeping the formalism simple enough to be feasible for verification methods developed for P/T-nets.

Keywords

P/T-nets, Synchronous Channels, Structuring Mechanisms, Modeling

1. Introduction

Petri nets are a popular means to model real world systems and explore their many properties. Nets generally suffer from the state space explosion problem, which states that the state space expressed by a Petri net grows much faster than the net itself. While continuous development on verification tools allows handling larger and larger state spaces [1, 2], another problem comes from the practical side: It is the increasing difficulty to model large nets. Algorithms run on Petri nets rarely concern themselves with spatial information of the net's elements, but a human modeler will very much do so. Duplicate structure, overlapping edges, unclear token flow and simply too large nets are some of the problems that make complex nets hard to comprehend.

An approach to improve this are Colored Petri Nets (CPNs) [3, 4], which use *color sets* instead of the indistinguishable black token of P/T-nets. By using color sets, structure can be reused in different modes indicated by the tokens color. One drawback of CPNs, however, is that many

PNSE'22, International Workshop on Petri Nets and Software Engineering, Bergen, Norway, 2022

lukas.voss@informatik.uni-hamburg.de (L. Voß); sven.willrodt@informatik.uni-hamburg.de (S. Willrodt); moldt@informatik.uni-hamburg.de (D. Moldt); michael.haustermann@informatik.uni-hamburg.de (M. Haustermann)

traditional Petri net techniques cannot be applied directly to the more complex CPNs, which is why CPNs must often be transformed into P/T-nets using a method called *unfolding*. These nets are significantly larger, since they still have to be able to distinguish the colors of the original CPN. If color sets are infinite, it might even be impossible to generate equivalent P/T-nets. There is work to generate smarter unfoldings [5, 6]. This paper chooses another approach. Instead of decreasing the difficulty to unfold expressive Petri net formalisms into P/T-nets, it increases the expressiveness of the P/T-net formalism by extensions, while making sure that equivalent P/T-nets remain reasonably small. By staying in the P/T-net formalism, problems that colored tokens present are completely circumvented.

The first and most important extension are synchronous channels. Synchronization is a semantical basis for Petri nets, but explicit communication between transitions has also been explored and introduced in form of synchronous channels [7, 8]. However, they are mostly used in high-level Petri net formalisms [8, 9, 10]. While synchronous channels present their own problems when unfolding, they are a great means to structure nets, allowing to reuse components and eliminate overlapping edges when used. The proposed synchronous channels also receive some restrictions to minimize the size of the equivalent P/T-net.

Another prominent extension that was introduced in many different variants is the division of a single Petri net into related and communicating sub-nets [9, 10, 11, 12, 13, 14]. It is a great means for modelers, as semantically different parts of the net can be modeled and viewed independently. Central is the interpretation of how these sub-nets relate to each other and can communicate. Under the many approaches are hierarchical relationships [11], fixed synchronization options [13] or an object-oriented approach [9, 10]. All of the mentioned approaches come in combination with some notion of synchronous transitions, indicating how well the two concepts interplay with each other. Our approach presented here uses a mixture of the concepts of [13] and [10].

2. Foundations

Definition 1. A Place/Transition net is a directed and weighted bipartite graph $N = (P, T, F, W, m_0)$, where

1. P is a finite set of places,
2. T is a finite set of transitions,
3. $F \subseteq (P \times T) \cup (T \times P)$ is its flow relation,
4. $W : F \rightarrow \mathbb{N}_0$ are its arc weights and
5. $m_0 : P \rightarrow \mathbb{N}_0$ is its initial marking.

A net has a marking m that carries information about the number of tokens on each place. $m(p)$ describes the number of tokens on place p. Both places and transitions have pre- and post-sets. The pre-set of a net element $x \in P \cup T$ is defined as ${}^\bullet x = \{y \in P \cup T : (y, x) \in F\}$. Accordingly, the post-set of a net element $x \in P \cup T$ is defined as $x^\bullet = \{y \in P \cup T : (x, y) \in F\}$.

A transition t is called *enabled* in marking m iff $\forall p \in {}^\bullet t : m(p) \geq W(p,t)$. An enabled transition t may *fire*. Firing t removes all tokens necessary for firing in its pre-set and puts tokens on all places in its post-set, according to the weight function. After firing transition t in marking m, the successor marking m' is defined as $m(p)' = m(p) - W(p,t) + W(t,p) \forall p \in P$. The firing of a transition is denoted by $m \xrightarrow{t} m'$.

Synchronous Channels

Synchronous channels constitute rendezvous synchronization in Petri net formalisms. The main idea is that transitions may be inscribed with channels, which allows them to synchronize if the signatures match. Transitions inscribed with channels can no longer fire alone, but only paired with another synchronizing transition that has a matching signature and is also active. Firing accounts for the locality of both synchronizing transitions. In general, synchronous channels consist of three parts:

1. **Type**: E.g. *!?* and *?!* in [8] or *uplinks* and *downlinks* in [10].
2. **Identifier**, usually in the form of a channel name or a relation.
3. **Parameters** for information-exchange between synchronizing transitions

All three attributes combined define which other transitions a channel transition can synchronize with. Transitions of the same channel type cannot synchronize among themselves, only with transitions of another channel type. Yet, the type does not indicate a direction. The channel identifier is used to further subdivide typed channels. Usually, that comes with a semantic implication of the channel. For example, in a producer-consumer scheme, the string "send" can be used as an identifier for a channel. Lastly, channel parameters are used to transfer information between synchronizing transitions. The term "information" can refer to any kind of information that particular formalism offers. In Colored Petri Nets, information transfer comes in the form of colored tokens, while Reference nets can, for example, also transfer net instance references through channels.

A different concept of synchronizing transitions is used in [13], where *transition fusion sets* are used. They are manually provided and static sets of transitions which must fire synchronously, requiring each one to be activated. They are rather theoretical, as they quickly become impractical to notate with increasing synchronization options. Thus, the former approach for synchronous channels is chosen here.

3. Objectives

The aim of the new formalism for P/T-nets with synchronous channels (PTC-nets) is to provide modeling extensions that allow larger models. The chosen extensions are synchronous channels and net partitioning. Several sub-goals follow:

1. Both a formal definition of the formalism, as well as an implementation, should be provided. That implementation should be as close to the formal definition as possible. It allows to simulate P/T-nets with synchronous channels.

2. Synchronous channels should allow the usage of parameters. These parameters allow to synchronize over the amount of tokens sent through the channel.
3. It should be possible to partition the P/T-nets into several net instances. These net instances can communicate via synchronous channels.
4. A static unfolding of P/T-nets with channels into ordinary P/T-nets should be possible.

The new formalism needs to be formally defined in order to properly describe the semantics of a net. A formal definition is useful for further work with the formalism, e.g. its verification with methods such as model checking. Furthermore, the new formalism combines well-known concepts, for which different interpretations exist, such as synchronous channels [8, 10] and nets consisting of multiple sub-nets [11, 13]. A formal definition gives clarity over the concepts, their nuances and removes any ambiguity. One example is in the definition of synchronous channels; There exist definitions where synchronization may happen over an arbitrary number of transitions and even with cyclic synchronization [10], or limited to exactly two transitions [8].

Providing an implementation for the formalism allows modelers to directly use the formalism and create models of concurrent, communicating systems. Furthermore, an implementation allows to simulate the behavior of such nets. With that, modelers can directly observe the state changes that can occur in a net and students are able to better understand the underlying semantics of a net.

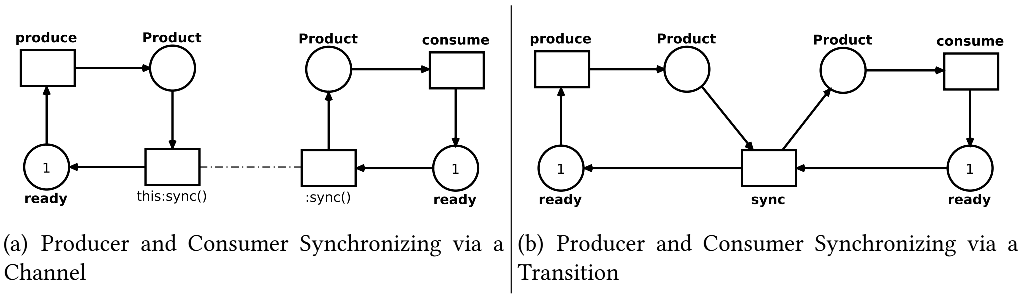
Information exchange between different communicating transitions is enabled via multiple parameters for the channels. Parameters can come in two forms: Either, a parameter is a positive integer. In that case, it indicates how many tokens get transferred to the communication partner. Or, a parameter is a free variable. Then, the variable can be used to inscribe arcs in the transition's pre-/post-set to consume or produce as many tokens as the free variable's value, determined during simulation by the partner transition. Like in [8] and [10], there is no notion of direction in communication.

For modelers, it is very useful to partition nets into several parts. With that, semantically different parts in a net can be separated. That makes the model clearer and easier to extend. For instance, in the well-known example of a producer-consumer system [15], one could separate producers and consumers into their own nets. Each individual net can be extended with complex behavior like different produced goods or complex workflows to consume or order goods, without changing the other net and keeping a direct semantic structuring for viewers of the model. Further, multiple instances of each net can be created for the simulation, allowing to easily simulate different interactions of multiple producers and consumers in several smaller net instances instead of a big, cluttered one. A modeler should be able to control how the communication of different net instances takes place. To control it, they can set hyper-parameters. These specify how many net instances are created and which synchronizations are possible.

Lastly, it should be possible to unfold P/T-nets with channels into ordinary P/T-nets. This is useful for verification and ensures that the expressiveness of the model stays the same. The reason why such an unfolding is possible is that each possible synchronization in a PTC-net can be modeled with one transition. That requirement restricts the formalism such that infinite possible synchronizations or cyclic calls are prohibited.

4. Prototypical Implementations

The implementations for this paper have been developed in Renew[1][16]. Using Renew allows modelers to create graphical representations, use multiple net files for partitioning nets, and simulate the behavior after creation.



(a) Producer and Consumer Synchronizing via a Channel
(b) Producer and Consumer Synchronizing via a Transition

Figure 1: Synchronization of a Single Producer and Consumer

Figure 1 depicts the model of a producer and a consumer synchronizing with each other. On the left side, both synchronize via a channel. Note that the dotted line is only used for visualization purposes and not part of the actual net. On the right side, the synchronization takes place with an ordinary transition.

This example already displays some of the important factors of the new proposed formalism. First of all, the formalism distinguishes between two different channel types, like the "!?-" and "?!-transitions" in [8]. Like there, synchronization can only take place between those different types. Here, they are called *downlinks* and *uplinks*, like channels in the Reference net formalism [10]. Downlinks are denoted with the *this*-keyword before the channel name. Uplinks do not have any descriptor in front of the channel name.

Secondly, the behavior of the synchronizing net can be represented with an ordinary P/T-net. In this example, exactly one synchronization can take place. That synchronization removes one *product* from the producer and the *ready* token from the consumer. It adds the *product* to the consumer and makes the producer *ready* to produce again. That exact behavior can also be represented with a single transition instead. But the example already indicates where the new formalism has its modeling strengths: Using synchronizing transitions, the two cycles are visibly distinct, the edges clearly depict the producer and consumer. If this example were to be extended with multiple producers and consumers, a normal P/T-net would require each *Product* and *ready* place to be connected to multiple transitions to allow communication between all producers and consumers. The net would be cluttered and overlapping arcs would be unavoidable. It would be difficult to structure the net semantically, i.e. that each producer and consumer is directly recognizable. Using synchronizing transitions avoids all these problems and allows to create models with a clear structuring of semantic components.

The net in Figure 2 depicts the idea of parametric, bi-directional information exchange using channels. In this net, there are two distinct downlinks and two uplinks of the same channel. Thus, each downlink has two potential synchronization partners and vice versa. Note that

[1]https://paose.informatik.uni-hamburg.de/paose/wiki/PTCNets

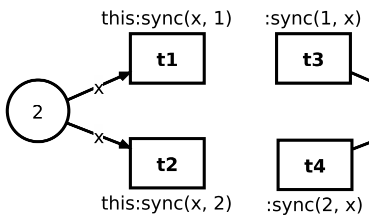


Figure 2: Net with different possibilities to synchronize

the arity of parameters matches for each up-/downlink. If that was not the case, for example one channel having only one parameter, it would have no potential synchronization partner in the net. There are four different possible synchronizations, leading to several different firing sequences and resulting markings. Place p_2 can have 1 to 4 tokens, depending on which synchronizations take place. To understand this net, it is helpful to look at the directions of information exchange and the direction of token flow. As always, indicated by the arc direction, transitions t_1 and t_2 consume tokens from the leftmost place, while t_3 and t_4 create tokens in place p_2. However, it is actually the transitions t_1 and t_2 which specify how many tokens are created in p_2, while t_3 and t_4 specify how many tokens are consumed in p_1: Both t_1 and t_2 consume an amount of tokens determined by their local variable x. This x assumes the value of the first parameter of the synchronization partner, i.e. 1 when synchronizing with t_3 or 2 with t_4. The amount of created tokens is determined analogously by the second parameter of t_1's and t_2's inscription.

For example, if t_1 and t_4 synchronize, t_1's local x is bound to 2 and t_4's local x is bound to 1. Thus, during firing, t_1's variable incoming arc requires 2 tokens from p_1. At the same time, t_4's variable outgoing arc puts 1 token into p_2. The resulting marking is $(0, 1)$ and no more bindings are possible. On the other hand, synchronizing with t_3 consumes only one token from p_1, allowing a second firing with t_3. Firing t_2 synchronized with t_3 twice results in a final marking of $(0, 4)$.

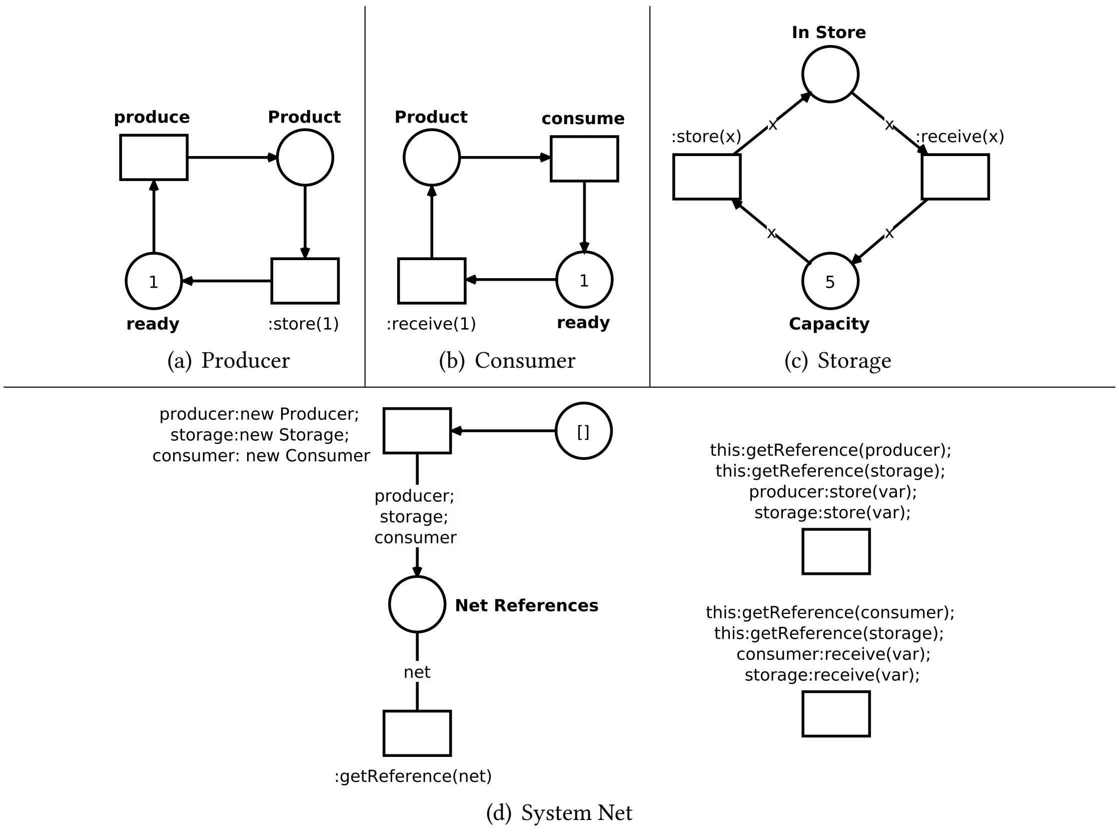
Since this net allows four different synchronizations, a static unfolding of this net has four transitions, one mirroring each effect of the possible synchronizations.

Modular PTC-nets

In both Figure 1 and Figure 2, the communication was between transitions of the same net. Especially for Figure 1, one can see how with the use of synchronous channels, the semantic components of a net are emphasized. For these cases, where clear and distinct (yet communicating) components exist, the emphasis on the componential structure can be further enhanced by partitioning the net into actual components, i.e. sub-nets, also called modules [13].

An example for this can be seen in Figure 3. Here, the producer and consumer are a module each and a storage has been added as well. They all communicate through channels. With the modular design, it is trivial to replace e.g. a producer with another variant or have multiple storage modules at the same time.

For the actual implementation of their synchronization, a system net is used. The general



(a) Producer (b) Consumer (c) Storage

(d) System Net

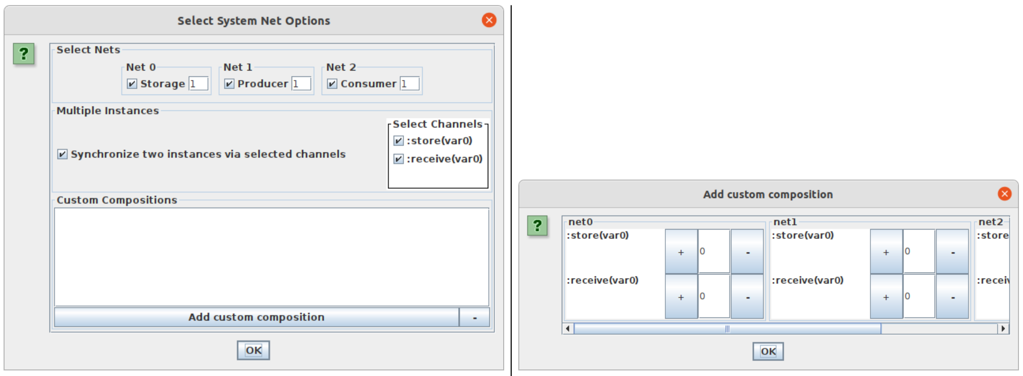
Figure 3: Modular producers and consumers synchronizing via a storage and coordinated by a system net

concept of a system net can be found in [17]. In our approach however, the system net acts purely as a helper net, not meant to be presented to the user or be part of the semantics or definition. In fact, it is not even a PTC-net. We chose to use a (far) more expressive Reference net [10] to simulate our modular PTC-net formalism instead of implementing the semantics in a new simulator. This is why the system net uses inscriptions which are not valid or do not exist in the PTC-net formalism.

As a small and very informal detour into the Reference net formalism, the system net keeps the references of all modular PTC-nets and uses them to initiate communication between two respective PTC-nets. These are the two transitions on the right-hand side of the system net. They both ask for two PTC-nets and then invoke the channel *store* (in the upper case) on both, requiring that their channel parameter is the same. The binding search of Reference nets is also very similar to unification, known from functional and logic programming languages. For the system nets this means in particular that there is also no notion of direction and variables only have their local meaning, i.e. even though in the expression `storage:store(var)` the variable is called storage, it might as well be bound to a producer, as it also offers the `store` channel.

While in total, the modular PTC-nets are simulated by Reference nets, the system net is automatically generated and the implementation provides a full syntax check for the manually created modular PTC-nets.

For the automatic generation, there are hyper-parameters set by the user. While there is a default setting, the user has a fine-grained control over how exactly the synchronization over a channel is possible. Firstly, the channels themselves can be (de-)selected for synchronization. Additionally, it can also be controlled how often a certain channel of one partner must be invoked. Note here that also a synchronous channel may fire concurrently to itself. In the ongoing example, a synchronizing transition in the system net may e.g. require three calls of the *store* channel in one net, together with six calls of the *receive* channel in another net. Each of these compositions of two channels is then added as a transition into the system net. All but the default-setting (where all uplinks can communicate) are syntactic sugar and are just meant to be means to provide some additional modeling simplicity. They are not further considered in the formal description of the modular PTC-nets.



Select System Net Options

Select Nets

Net 0: ☑ Storage 1
Net 1: ☑ Producer 1
Net 2: ☑ Consumer 1

Multiple Instances

☑ Synchronize two instances via selected channels

Select Channels
☑ :store(var0)
☑ :receive(var0)

Custom Compositions

(a) System net creation GUI

Add custom composition

net0: :store(var0) 0; :receive(var0) 0
net1: :store(var0) 0; :receive(var0) 0
net2: :store; :recei

(b) GUI to add custom synchronization transitions to the system net

Figure 4: Both GUIs for the creation of a system net that coordinates the synchronization of modular nets

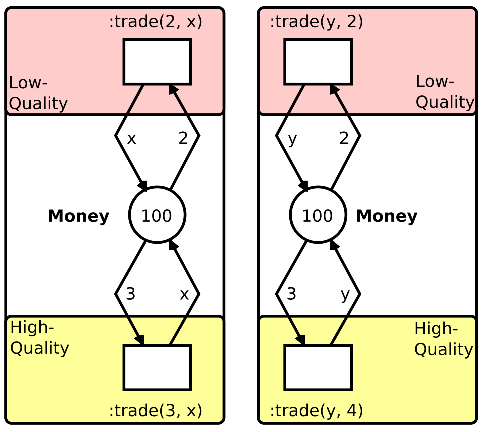
The modeler can set the hyper-parameters through a provided GUI. The GUI for system net creation is depicted in Figure 4. The main GUI, depicted on the left side, consists of three parts. In the upper part, a modeler can select how many and which instances of opened net templates shall be created. The middle part is an optional default setting for synchronization: If selected, all synchronous channels can synchronizes exactly with two different net instances. However, if other synchronization options are desired for modeling purposes, these can be added and created manually through the settings in the lower part. The user is then presented the GUI on the right hand side. Here, synchronizations can manually be specified. For each net instance that is created as defined by the first setting, the number of channel calls required in a synchronization can be defined. With that, a modeler has control over the creation of a system net, while it is guaranteed that the system net is always syntactically correct.

As a final note, modular PTC-nets know two different kinds of synchronization: Synchronization within one net and synchronization between two nets. The former remains unchanged by using matching up- and downlinks, while the latter is realized by uplinks only.

Modeling with PTC-nets

Synchronous channels, parameters, and modularity all are beneficial for modeling. We have already seen an example of this in Figure 3. Leaving the system net aside which is needed for implementation purposes, the modeled system is clearly separated and can be understood quickly. The modularity allows to separate all modeled entities and the interaction between these entities is quickly clear because of the channels. It is also simple to think of a consumer which can consume multiple products at once, e.g. with a storage which can send packages one by one, or as a bundle (which is maybe cheaper, in a more elaborate model), making use of channel parameters.

Another particularity of the PTC-net's synchronous channels and their restrictions are depicted in earlier presented Figure 2. When looking at the meaning of the transitions, it might seem unintuitive at first that a transition knows one parameter, but not the other one, e.g. paying a fixed cost for an unknown reward. But this makes PTC-nets especially useful to model simultaneous actions and processes under uncertainty. Due to the nature of bi-directional information exchange in parametric synchronous channels, different parts of a system (agents) can hold partial information, that is combined when synchronizing. Each agent in a scenario can be modeled independently. These properties make PTC-nets excellent for modeling game theory scenarios, which would not be the first time that Petri net formalisms can be a useful model for economic concepts.



(a) Trader (b) Supplier

Figure 5: Model of Trader-Supplier interaction using modular PTC-nets

Imagine a trader, regularly buying ingredients from a supplier. The ingredients are not always identical: Sometimes, they are of high quality and well worth the price, whereas some other times, they are of low quality. From the perspective of the supplier, they want to sell as many low quality products with lower production costs as possible while maintaining the relations to the customer. If tricked too often, the trader will only buy ingredients at a lower price, assuming that they receive low-quality ingredients. If that is the case, the supplier occasionally needs to send ingredients of high quality, to convince the trader to buy for a higher price again.

A model of this is depicted in Figure Figure 5. Note that the selection of numbers changes the outcome of the model. Currently, if simulated long enough, the trader will profit and make money in the long run. If, however, the production costs for the supplier for low-quality ingredients were even smaller, the supplier would profit as well. The modular concept allows to instantiate multiple traders or suppliers. Also, different suppliers or traders with other ratios could be added, to simulate a whole ecosystem and analyze which participants go bankrupt, remain able to participate, or flourish.

5. Formal Definition

In the following, a single PTC-net and its behavior is defined. The PTC-net definition itself is an extension of Definition 2 by synchronous channels. For the definitions, all of the requirements discussed in Section 3 are considered. Because the synchronization of transition in partitioned PTC-nets orks slightly different, synchronization between different net instances is described in a separate section.

Synchronization within a Single PTC-Net

Definition 2. A P/T-net with synchronous channels (PTC-net) is a tuple $(P, T, F, W_{sync}, m_0, Var, Ch, CE)$, containing

- P: Set of places,
- T: Set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$: Flow relation,
- $W_{sync} : F \to \mathbb{N}_0 \cup Var$: Arc weights,
- $m_0 : P \to \mathbb{N}_0$: Initial marking,
- Var: Set of channel variables
- Ch: Set of channels
- $CE : T \rightharpoonup (type, ch, X)$: Channel expression function with
  - $type \in \{\textit{uplink}, \textit{downlink}\}$
  - $ch \in Ch$
  - X is a (possibly empty) tuple of channel variables and/or positive integers.

Further, we require that variables on arcs connected to synchronous channels must appear in the respective transition's channel variable tuple, or more formally:

$$\forall t \in T : \forall v \in \{W_{sync}(p, t) \mid p \in P\} \cup \{W_{sync}(t, p) \mid p \in P\} : (v \in Var \Rightarrow v \in X)$$

This is simply necessary, because otherwise the value which a variable is bound to cannot be resolved. In [10] this exact scenario is actually possible, free variables that do not occur in the transition inscription on arcs from places to transitions can be used to model a "take anything"-behavior. For the PTC-net formalism, however, we wanted to maintain as much deterministic behavior as possible. Therefore, undetermined variables are forbidden.

Compared to ordinary P/T-nets, this formalism includes synchronous channels for transition communication. For that, a set of channels and channel expressions have been added. Channels themselves are distinguished by their names. The channel expressions further determine how synchronous channels can fire. With channel variables, transitions can communicate the number of tokens being transferred, with which then arcs can also be inscribed. It is an ordered tuple to allow unambiguous binding search. A parameter at position i of transition t_1 simply binds with the parameter at position i of transition t_2. This already implies that the arity of two synchronizing transition's channel variables must be equal, formalized in the next definition.

As a side note, since each transition may be inscribed with at most one channel expression, cyclic bindings are impossible. This is necessary to ensure that the unfolded P/T-nets remain finite.

The aforementioned details indicate that, in addition to the definition how a P/T-net with synchronous channels looks like, it is necessary to define how and when transitions in such nets can fire. Transitions not inscribed with a channel behave as known for P/T-nets and defined in Section 2. Yet, the behavior of channel transitions needs to be defined as well. For that, it is first defined how many tokens are needed on a place in front of two synchronizing transitions to be activated. We refer to the i'th element of a tuple T with $T(i)$, using 1-indexing.

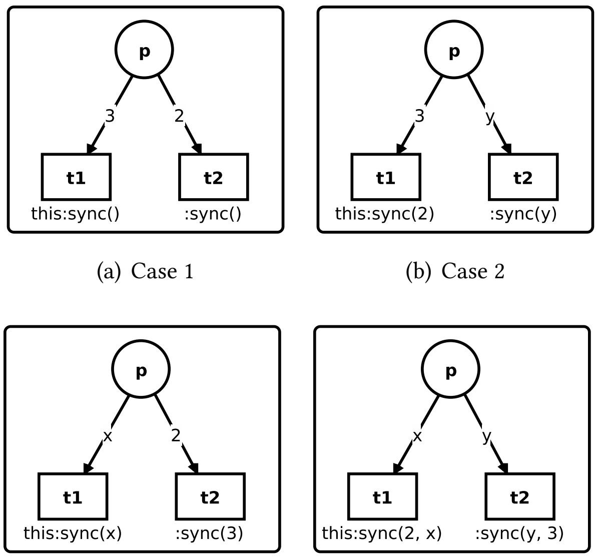
Definition 3. The number of tokens required on a place p in the pre-set of transitions t_1 and t_2 with parameter tuples X and Y in order to synchronously fire them is equal to

$$W^*_{sync}(p,t_1,t_2) := \begin{cases} W_{sync}(p,t_1) + W_{sync}(p,t_2) & W_{sync}(p,t_1) \in \mathbb{N} \land W_{sync}(p,t_2) \in \mathbb{N} \\ X(i) + W_{sync}(p,t_1) & W_{sync}(p,t_1) \in \mathbb{N} \land W_{sync}(p,t_2) \in Var \\ & \land i \in \mathbb{N} : Y(i) = W_{sync}(p,t_2) \\ Y(i) + W_{sync}(p,t_2) & W_{sync}(p,t_1) \in Var \land W_{sync}(p,t_2) \in \mathbb{N} \\ & \land i \in \mathbb{N} : X(i) = W_{sync}(p,t_1) \\ X(i) + Y(j) & W_{sync}(p,t_1) \in Var \land W_{sync}(p,t_2) \in Var \\ & \land i \in \mathbb{N} : Y(i) = W_{sync}(p,t_2) \\ & \land j \in \mathbb{N} : X(j) = W_{sync}(p,t_1) \end{cases}$$

Analogously, the number of produced tokens on each place after synchronized firing needs to be defined:

$$W^*_{sync}(t_1,t_2,p) := \begin{cases} W_{sync}(t_1,p) + W_{sync}(t_2,p) & W_{sync}(t_1,p) \in \mathbb{N} \land W_{sync}(t_2,p) \in \mathbb{N} \\ X(i) + W_{sync}(t_1,p) & W_{sync}(t_1,p) \in \mathbb{N} \land W_{sync}(t_2,p) \in Var \\ & \land i \in \mathbb{N} : Y(i) = W_{sync}(t_2,p) \\ Y(i) + W_{sync}(t_2,p) & W_{sync}(t_1,p) \in Var \land W_{sync}(t_2,p) \in \mathbb{N} \\ & \land i \in \mathbb{N} : X(i) = W_{sync}(t_1,p) \\ X(i) + Y(j) & W_{sync}(t_1,p) \in Var \land W_{sync}(t_2,p) \in Var \\ & \land i \in \mathbb{N} : Y(i) = W_{sync}(t_2,p) \\ & \land j \in \mathbb{N} : X(j) = W_{sync}(t_1,p) \end{cases}$$

$W^*_{sync}(p,t_1,t_2)$ returns the number of tokens that are needed on place p in order to fire both synchronous transitions t_1 and t_2. Index i is used to describe the tuple index of the variable binding: If an arc is inscribed with the variable x and x is the second parameter of the synchronizing transition, then i is equal to 2 and thus refers to the second element of the partner's parameter tuple. If the arcs from p to t_1 and t_2 are both inscribed with integers, these values can simply be added together. If at least one of the arcs is inscribed with a variable, the corresponding value of that variable in the tuple of the communication partner is used. That idea is used analogously for $W^*_{sync}(t_1,t_2,p)$.



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 6: The four cases of Definition 3 visualized

The different cases of W^*_{sync} are visualized in Figure 6:
Case 1: Both incoming arcs are inscribed with integers and not with variables. Thus, ordinary arc

weights can be used and the number of required tokens in order to fire t_1 and t_2 synchronously is $W_{sync}(p, t_1) + W_{sync}(p, t_2) = 3 + 2$.
Case 2: The incoming arc to t_1 is inscribed with an integer, but t_2's incoming arc is inscribed with a variable. Since the variable is the first element of that transition's parameter tuple, the first element of the partner's parameter tuple gets added: $X(1) + W_{sync}(p, t_1) = 2 + 3$.
Case 3: This one is equivalent to Case 2, only that the first parameter of t_2 is used instead of t_1's: $Y(1) + W_{sync}(p, t_2) = 2 + 3$.
Case 4: Both arcs are inscribed with variables. Here, $i = 1$ and $j = 2$, because the first element of t_1's parameter tuple indicates the number of tokens for y and the second element of t_2's parameter tuple indicates the number of tokens for x: $X(1) + Y(2) = 2 + 3$.

The same principle can be applied for outgoing arcs. For cases where only one transition is connected to p, W^*_{sync} still yields, since arc weights for "unconnected" arcs are defined as 0.

Definition 4. In the following, we define the structural requirements for two transitions to be able to synchronize.

For an uplink u with $CE(u) = (uplink, ch_u, Y)$ and a downlink d with $CE(d) = (downlink, ch_d, X)$, we say that u **matches** d iff.

$$
\begin{aligned}
&ch_u = ch_d && (1)\\
\wedge\ &|X| = |Y| && (2)\\
\wedge\ &((X(i) \in Var \wedge Y(i) \in \mathbb{N}_0) \vee (X(i) \in \mathbb{N}_0 \wedge Y(i) \in Var) : \forall i \in \{1, 2, ..., |X|\}) && (3)
\end{aligned}
$$

If u matches d, u and d could potentially fire synchronously, if there are enough tokens at some point during simulation to satisfy W^*_{sync}. Definition 5 contains multiple requirements that need to be fulfilled in order to fire two synchronous transitions. Each line adds an additional requirement:
Requirement 1: Communication is only possible between an uplink and a downlink transition with the same channel.
Requirement 2: Both tuples in the channel expressions need to contain the same amount of parameters.
Requirement 3: For any parameter position, exactly one transition's inscription has an integer at this position and the other has a variable at this position.

By using Definition 3 and Definition 4, we can then give the full definition of when two synchronous channels are activated.

Definition 5. In the following, the requirements for a downlink to be enabled are defined. The requirements for an uplink are identical, merely replacing *uplink* with *downlink* and vice versa.

A downlink transition t_1 with $CE(t_1) = (downlink, ch_1, X)$ is enabled iff

$$
\begin{aligned}
\exists t_2 \in T :\ &CE(t_2) = (uplink, ch_1, Y) && (4)\\
&\wedge \forall p \in {}^\bullet t_1 \cup {}^\bullet t_2 : m(p) \geq W^*_{sync}(p, t_1, t_2) && (5)\\
&\wedge\ t_2 \text{ matches } t_1 && (6)
\end{aligned}
$$

After firing the bound downlink transition t_1 and uplink transition t_2, the net's marking changes. In general, the successor marking is not calculated differently from successor markings of ordinary P/T-nets, but its definition has to account for arcs that are inscribed with variables, and thus their consumption or production of tokens depends on the specific binding. $W^*_{sync}(p, t_1, t_2)$ and $W^*_{sync}(t_1, t_2, p)$ can be used in the definition of the actual successor marking:

Definition 6. The successor marking $m'(p)$ after firing two synchronous transitions is defined as

$$m'(p) := m(p) + W^*_{sync}(p, t_1, t_2) - W^*_{sync}(t_1, t_2, p) \forall p \in P$$

Synchronization Between Different Net Instances

As described in Section 4, the formalism allows to partition P/T-nets into sub-nets called modules. In the following, the adaptations to the formal definitions are described. The modules' communication is similar to the communication within one PTC-net that is constructed by the union of all modules, which is why the definitions for modular PTC-nets are given only as extensions to the PTC-net definition. Since the described system net is only implementation-specific, it is not included in the formal definition. As already mentioned in Section 4, only the default case of communication is considered formally, i.e. every uplink can communicate with each matching uplink from other nets. The major difference is then merely that two uplinks of the same channel can also fire if they are from different nets.

Definition 7. A modular PTC-net is a 1-tuple (PTC), containing

- PTC: A set of PTC-nets, for which we additionally require that
  - the place sets of all PTC-nets are pairwise disjoint, and
  - the transition sets of all PTC-nets are pairwise disjoint.

Definition 8. In the following, we extend the definition of **matching** from Definition 4 by another option to express that two uplinks from different nets can structurally fire together.

In a modular PTC-net $\mathcal{M} = (PTC)$, for an uplink u from net $N \in PTC$ with channel expression function CE and $CE(u) = (uplink, ch_u, Y)$ and another downlink d from net N and $CE(d) = (downlink, ch_v, X)$, we say that u **matches** d if

$$ch_u = ch_v \tag{7}$$

$$\wedge |X| = |Y| \tag{8}$$

$$\wedge ((X(i) \in Var \wedge Y(i) \in \mathbb{N}_0) \vee (X(i) \in \mathbb{N}_0 \wedge Y(i) \in Var) : \forall i \in \{1, 2, ..., |X|\}) \tag{9}$$

For this same uplink u from net $N \in PTC$, we say for another ***uplink*** u' from net $N' \in PTC$ with channel expression function CE', channel variables Var' and $CE'(u') = (uplink, ch_{u'}, X)$, that u **matches** u' if

$$N \neq N' \tag{10}$$

$$\wedge\, ch_u = ch_{u'} \tag{11}$$
$$\wedge\, |X| = |Y| \tag{12}$$
$$\wedge\, ((X(i) \in Var' \wedge Y(i) \in \mathbb{N}_0) \vee (X(i) \in \mathbb{N}_0 \wedge Y(i) \in Var) : \forall i \in \{1, 2, ..., |X|\}) \tag{13}$$

Two transitions match exactly iff they fulfill either one of these cases. Definitions 2, 3, 5 & 6 can easily be transferred using Definition 8 instead of Definition 4 to fully describe the semantics of a modular PTC-net.

6. Unfolding PTC-nets to P/T-nets

As stated before, it is desirable to show that the PTC-net formalism is equivalent to the P/T-net formalism. Thus, it is ensured that even though new modeling techniques are added, no further expressiveness comes with it. Additionally, this always allows to fall back to the well-explored P/T-net verification techniques. We give a proof outline by providing a construction rule to convert an arbitrary PTC-net into an equivalent P/T-net. Since every P/T-net is also a PTC-net with $Var = Ch = \emptyset$, the other direction is trivially true.

We start by providing a construction rule for a single PTC-net and show afterwards how to create a single PTC-net from modular PTC-nets. Because this is more of a sketch of a construction rule, a rather visual and practical approach is chosen to avoid bloating it with small details and mathematical subtleties. We assume w.l.o.g. that the channel variables of each up- and downlink are disjoint, which can be ensured by introducing new channel variables for each transition.

Algorithm 1 Unfolding a PTC-net into a P/T-net

```
for each downlink $d$ do
    for each uplink $u$ such that $u$ matches $d$ do
        Let $X_u/X_d$ be the channel parameters of $u/d$
        Create a new transition $t^*$
        Copy each arc connected to $u$ or $d$, connected to $t^*$ instead
        for each $x \in Var$ at position $i$ in $X_u$ do
            Replace each arc weight $w = x$ in all arcs connected to $t^*$ with $X_d^i$
        end for
        for each $x \in Var$ at position $i$ in $X_d$ do
            Replace each arc weight $w = x$ in all arcs connected to $t^*$ with $X_u^i$
        end for
    end for
end for
Remove all uplinks, downlinks and their connected arcs
```

One can quickly convince oneself that this algorithm constructs a new transition for each possible binding and resolves all variables to fixed integers, leaving only uninscribed transitions and arc weights without variables. A definition for the resulting unfolded net is given in Definition 9.

Definition 9. For a given PTC-net $N = (P, T, F, W_{sync}, m_0, Var, Ch, CE)$ an equivalent P/T-net without synchronous channels is given by $N_U = (P_U, T_U, F_U, W_U, m_{0U})$ with:

- $P_U = P$
- $T_U = \{t \in T \mid CE(t) \text{ is undefined}\} \cup \{t_{d,u} \mid d, u \in T \wedge u \text{ matches } d\}$
- $F_U = F \cap ((P \times T_U) \cup (T_U \times P)) \cup \{(p, t_{d,u}) \mid (p, d) \in F \vee (p, u) \in F\} \cup \{(t_{d,u}, p) \mid (d, p) \in F \vee (u, p) \in F\}$
- $W(x, y) = \begin{cases} W_{sync}(x, y) & \text{if } x, y \in P \cup T \\ W^*_{sync}(x, d, u) & \text{if } x \in P \wedge y = t_{d,u} \\ W^*_{sync}(d, u, y) & \text{if } x = t_{d,u} \wedge y \in P \end{cases}$
- $m_{0U} = m_0$

From here, constructing an equivalent PTC-net from a modular PTC-net is fairly straightforward. It is almost sufficient to simply join all nets into one big net. Only the lack of up- and downlinks in modular PTC-nets needs to be handled. Since every transition can fire with every other transition with the same channel, this is analogous to if every synchronous transition would be an up- and downlink at the same time. Thus, we replace each synchronous channel with both an up- and a downlink.

With the construction rule, we can show that equivalent P/T-net's transitions grow at most quadratically with the number of synchronous channels. For every downlink, an amount of transitions equal to the amount of matching uplinks are created. This number is maximized if every transition is inscribed with the same channel, half the transitions are downlinks and the other half are matching uplinks. Then, the new number of transitions is given by

$$|T^*| = \left\lceil \frac{|T|}{2} \right\rceil \cdot \left\lfloor \frac{|T|}{2} \right\rfloor \leq |T|^2$$

As usual with such considerations, the case where nets only consist of matching up- and downlinks is a rather theoretical one. Just like the case of a PTC-net without any synchronous channels is rather unlikely to be a practical case, one can expect net unfoldings that have less than this maximum number of transitions.

7. Verification

As a final remark on the introduced (modular) PTC-net formalism, we also want to explore some thoughts on their verification. Research on verification methods tailored for Petri Net formalisms which have some sort of hierarchical or modular structure already exists [12, 13, 18, 19, 20]. Modular CPNs with transition fusion sets [13] is one of the more popular approaches for a formalism and methods for verification have also been described [13, 19]. However, their transition fusion sets do not allow synchronization over parameters, which PTC-nets do. Thus, adaptations are necessary to apply the algorithms to PTC-nets. Further, we will discuss how the

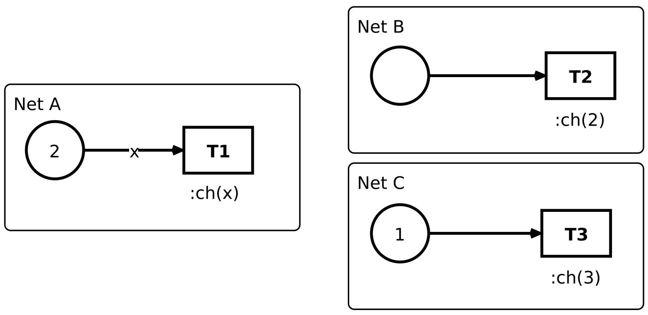


Figure 7: PTC-nets where Net A cannot know on its own if its transition **T1** is activated without knowing the state of other nets.

parametrized, but restricted synchronous channels for PTC-nets can be used advantageously for state space analysis using the example of modular state spaces.

The modular state spaces in [19] can generate state spaces by generating the state space of each module and synchronizing them over their transition fusion sets using a synchronization graph. This has two major advantages: Interleavings of unsynchronized transitions are not explicitly stored, thus the reachability graph can be represented with significantly fewer nodes and edges, the fewer the less synchronization happens. Secondly, the modules can partly be generated in parallel. In the proposed algorithm, the state spaces of each module can be constructed without knowledge of the other modules, until transitions from transition fusion sets are activated. Then, these can be synchronized with nodes from other state spaces which have corresponding transitions activated.

With the parameterized synchronous channels of PTC-nets, a modular state space cannot be constructed directly with the methods of [19]. The central problem is, that a transition might not know, whether it is activated or not. In the Figure 7, the transition T1 of Net A could potentially be activated. But Net A can never know on its own, since the number of required tokens is dependent on the other transitions T2 and T3. But even more, this is not statically predetermined, but can depend on its own state or other net states as well. Here, T1 would be activated if either T2 is activated, or it has a third token in its place. Therefore, a process generating a module's state space cannot simply announce its activated transition.

What it can do instead, is offer a partial binding. This is the first example where the restrictions on synchronous channels make the verification a lot easier. The parameters of synchronous channels are always bound from exactly one side. For each parameter, the module either binds it or "receives" a binding for it, and this is structurally fixed. So instead of announcing that a synchronous channel is activated, a module can announce that it is able to bind all variables from its side, that is has a partial binding. In the example of Figure 7, Net A's process can announce that T1 is now activated with $x = 1$ and $x = 2$. And since PTC-nets know only one kind of token, its sufficient to announce the highest number of tokens for each parameter to encode every possible binding. This would already be harder with CPNs, but another advantage comes with the restrictions on synchronous channels.

The important aspect here, is that all possible bindings are actually structurally known.

Therefore, a module would actually know which other modules are interested in particular bindings and can send them exactly the bindings that could lead to a synchronization, thus minimizing synchronization between processes.

Optimistic Modular State Space Generation

Because no implementation is provided in [19], one can only guess about the general time-efficiency of the algorithms. But the more transition fusion sets exist, the more often will processes have to wait for synchronization. We propose an optimistic variant of the modular state space algorithm, which tries to avoid idle processes at the cost of generating too much. It is especially well-suited for the PTC-net formalisms restrictions on synchronous channels.

The main idea is simple: Whenever a process is waiting, it starts to explore a random binding of one of its synchronous channels. At some point, it will be known if this binding was actually activated and the sub-graph can be kept or discarded. Because the restrictions limit the number of possible bindings to one per partner transition, there are not too many bindings to choose from. Thus, also the chances are a lot higher that a correct binding was guessed. It would be infeasible to purely guess bindings for synchronous channels without restrictions and even more so for CPNs.

There are some aspects that should be considered when generating sub-graphs optimistically. First, exploring transitions which are sure to be active should always have priority. This could be handled by separating markings to explore into two queues based on their certainty, where markings are only polled from the 'uncertain' queue if the other one is empty. Further, by waiting for other processes, [13] avoid constructing infinite sub-graphs which cannot actually occur. Special care has to taken here, and if a covering marking is found, it has to be differentiated if the found cycle lies only in a potential sub-graph or in a certain sub-graph. In the potential sub-graph case, the exploration of this sub-graph should be stopped, but the overall state space generation can be continued until the potential binding that initiated this sub-graph is known to be active. In the other case, the state space generation can stop immediately. To ensure termination, processes need to announce when they are done. Then, other modules can immediately stop exploring transitions which would require an active synchronization partner marked as done.

Finally, since we provide an implementation of the PTC-net formalism, the implementation of the verification algorithms is just a small step ahead.

8. Discussion & Outlook

The definition of the formalism was designed to be as close as possible to ordinary P/T-nets. For that reason, only one arc per direction of a place-transition pair is allowed. An arc is also inscribed with exactly one value. However, for some modeling scenarios, it can be useful to allow multiple variables to take from or put into a place. One example for this is a producer who produces multiple goods, each denoted with its own variable in a channel. If these goods are synchronized with a storage, that storage's capacity decreases by the total amount.

Another desirable addition for modeling purposes is the possibility to synchronize more than two transitions. This is currently only possible in user-specified synchronizations in modular PTC-nets. Such a requirement could be added by allowing multiple downlinks or uplink/downlink pairs on transitions. However, such an addition could lead to cyclic reference calls, resulting in infinite unfoldings. To stay close to the original goals of this formalism, it should only be possible if cyclic calls are explicitly prohibited. With a cycle detection in the implementation, repeated calls of an acyclic tree structure would be possible.

Besides these general extensions, there is also another useful extension only for modular PTC-nets. For specific scenarios, e.g. the dining philosophers problem [21], modular PTC-nets would benefit greatly if directed synchronization would be possible, i.e. one PTC-net trying to synchronize with another specified, chosen PTC-net.

9. Conclusion

This paper provides a new formalism which extends ordinary P/T-nets with synchronous channels. Both a formal definition, as well as an implementation, are provided. The PTC-net formalism allows transitions to be inscribed with channels. Channel transitions need partners to fire and can exchange information using parameters.

Also, the approach allows to partition nets into several comprehensive parts, making modeling easier. Due to the provided implementation, modelers can create and simulate models such as agents with incomplete information, where each agent is its own entity.

We provide an algorithm to unfold PTC-nets into ordinary P/T-nets, together with a definition of the composition of such an unfolding. Further, the PTC-net formalism offers itself naturally to already explored verification techniques based on modular structure, with only small extensions necessary to handle parameters on synchronous channels. Additionally, the discussed restrictions on the parameters provide structural information which can be used to an advantage for state space construction and verification.

Thus, the PTC-net formalism fills a gap between P/T-nets and CPNs, for which many extensions exist, as it offers expressive modeling techniques to model larger systems while remaining feasible for verification due to its structural restrictions.

References

[1] J. F. Jensen, T. Nielsen, L. K. Oestergaard, J. Srba, Tapaal and reachability analysis of P/T nets, in: Transactions on Petri Nets and Other Models of Concurrency XI, Springer, 2016, pp. 307–318.
[2] K. Wolf, Petri net model checking with LoLA 2, in: International Conference on Applications and Theory of Petri Nets and Concurrency, Springer, 2018, pp. 351–362.
[3] K. Jensen, Coloured Petri nets, in: Petri Nets: Central Models and Their Properties, Springer, 1987, pp. 248–299.
[4] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, K. Jensen, Cpn tools for editing, simulating, and analysing coloured Petri nets, in: International Conference on Application and Theory of Petri Nets, Springer, 2003, pp. 450–462.

[5] F. Kordon, A. Linard, E. Paviot-Adet, Optimized colored nets unfolding, in: International Conference on Formal Techniques for Networked and Distributed Systems, Springer, 2006, pp. 339–355.

[6] A. Bilgram, P. G. Jensen, T. Pedersen, J. Srba, P. H. Taankvist, Improvements in unfolding of colored Petri nets, in: International Conference on Reachability Problems, Springer, 2021, pp. 69–84.

[7] E. Jessen, R. Valk, Rechensysteme: Grundlagen der Modellbildung, Studienreihe Informatik, Springer-Verlag, Berlin Heidelberg New York, 1987.

[8] S. Christensen, N. D. Hansen, Coloured Petri nets extended with channels for synchronous communication, in: R. Valette (Ed.), Application and Theory of Petri Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings, volume 815 of *Lecture Notes in Computer Science*, Springer, 1994, pp. 159–178.

[9] C. Lakos, From coloured Petri nets to object Petri nets, in: International Conference on Application and Theory of Petri Nets, Springer, 1995, pp. 278–297.

[10] O. Kummer, Referenznetze, Logos Verlag, Berlin, 2002. URL: http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=.

[11] P. Huber, K. Jensen, R. M. Shapiro, Hierarchies in coloured Petri nets, in: International Conference on Application and Theory of Petri Nets, Springer, 1989, pp. 313–341.

[12] I. A. Lomazova, Nested Petri nets — a formalism for specification and verification of multi-agent distributed systems, Fundamenta informaticae 43 (2000) 195–214.

[13] S. Christensen, L. Petrucci, Modular analysis of Petri nets, The Computer Journal 43 (2000) 224–242.

[14] M. A. Bednarczyk, L. Bernardinello, W. Pawłowski, L. Pomello, Modelling mobility with Petri hypernets, in: International Workshop on Algebraic Development Techniques, Springer, 2004, pp. 28–44.

[15] W. Reisig, Place/Transition systems, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986, volume 254 of *Lecture Notes in Computer Science*, Springer, 1986, pp. 117–141.

[16] O. Kummer, F. Wienberg, M. Duvigneau, L. Cabac, M. Haustermann, D. Mosteller, Renew – the Reference Net Workshop, 2022. URL: http://www.renew.de/, release 4.0.

[17] R. Valk, Object Petri nets – using the nets-within-nets paradigm, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Advances in Petri Nets: Lectures on Concurrency and Petri Nets, volume 3098 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg New York, 2004, pp. 819–848. URL: http://dx.doi.org/10.1007/978-3-540-27755-2_23.

[18] M. Notomi, T. Murata, Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis, IEEE Transactions on Software Engineering 20 (1994) 325–336.

[19] S. Christensen, L. Petrucci, Modular state space analysis of coloured Petri nets, in: International Conference on Application and Theory of Petri Nets, Springer, 1995, pp. 201–217.

[20] P. Kemper, Reachability analysis based on structured representations, in: International Conference on Application and Theory of Petri Nets, Springer, 1996, pp. 269–288.

[21] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.