Some Simple Extensions of Petri's Cycloids[⋆]

Bjarne Jessen and Daniel Moldt

University of Hamburg, Department of Informatics
{5jessen,moldt}@informatik.uni-hamburg.de

Abstract Within the general system theory of Petri some means for modelling concurrent systems are provided. Cycloids belong to the fundamental building blocks to understand basic properties of systems. Up to now cycloids have a very limited scope with respect to general system properties. Valk's modelling proposals of circular traffic queues are first valuable attempts to overcome these restrictions.
In this contribution we extend the modelling in several ways. First of all we introduce conflicts and other concepts to the modelling with cycloids. Second we use reference nets to increase expressibility of the modelling formalism, what eases the way to cover relevant system properties within the models while keeping relevant properties of the underlying processes of these models. As a result of both extensions we can express more interesting behaviour of concurrent systems. At the same time our construction of extended cycloids is designed in such a way that we can map our extended cycloid models to a set of cycloid models.
In future work the transformation of Valk's formal results of the last years should be transferred to our extended versions.
Keywords: Petri nets, Extended Cycloids, Process Models, Modelling.

1 Introduction

Based on Minkowski's ideas about space and time [18] with the notion of *Weltlinie* Petri introduces the *Lebenslinie* of entities for his general system theory [25,26]. Kummer and Stehr discussed in [14] the axiomatic theory of concurrency and causality, especially for the notion of cycloids as one of the fundamental building blocks of Petri's system theory. During the last years Valk developed some closed formulas to describe cycloid models [32,34,35]. Expressibility, however, is rather low, since these models do not cover general conflicts. Overall cycloids cover cyclic behaviour with no choices but with concurrency. The resulting net (process) of an initially marked cycloid model can therefore be characterised by a closed formula [32]. Such a net preserves all kinds of behaviours even under true concurrency semantics.

The unit theory we sketched during the last years [19,20,31,39,6,5,38] proposes to identify any kind of modelling entity on the basis of its behaviour. Coming from Minkowski's Weltlinie and Petri's Lebenslinie for each entity an

[⋆] Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

appropriate Petri net model can found: Behaviour of an entity is described as an infinite Petri net as described in Petri's general system theory. The same holds for a set of entities. Nothing within one system can be lost, so Lebenslinien can be joined and split, however, no cycles exist in the possibly infinite net model. In this contribution all places, transitions, arcs and tokens that are related to the subset of the (possibly) infinite net then describe this entity. In general the unit theory addresses the question of how to model concurrent systems combining the ideas from Petri's concurrency theory and modelling in informatics in general. For this contribution details of the unit theory are not necessary, since the proposal formulated in [10] starts from the same basis as all cycloid models. The relevant part of the unit theory is that we strive for a formal basis of the basic modelling concepts of distributed and concurrent systems.

The extensions proposed here are still restricted with respect to general systems of informatics. However, in [10] a first elaborated proposal for extensions of the cycloids is made, extending proposals in [33]. Some motivation can be found in embedding the restricted set of modelling concepts into our agent-oriented systems modelling [30,3]. Multi-agent systems (MAS) are one prominent kind of distributed and concurrent systems [8], which we cover in our Mulan framework [11]. Agent protocols, describing a fixed set of repeatable behaviour of distributed concurrent entities, can cover (potentially) infinite behaviour. Interpreting agent systems as a set of (possibly nested) entities directly leads to corresponding cycloid models. An observed entity repeats its modeled behaviour over and over again. Due to the missing conflicts, which describe alternative behaviour for an entity, due to some interaction with another entity, cannot be modelled by traditional cycloids. Therefore in [10] extensions are described that where discussed during the last years for the unit theory.

In section 2 we discuss what kind of systems are covered by cycloids so far. Section 3 describes the kind of requirements we have for our intended cycloid applications. Section 4 contains a table of our currently covered extensions. Section 5 provides examples of the previous section. Section 6 covers ideas of our formal mapping of entities to cycloid models and the mapping of sets of cycloids to our extended cycloids. Section 7 relates our models to other kinds of modelling approaches like workflows, MAS and our unit theory. Section 8 summarises our results and sketches an outlook of further topics.

2 Cycloids and their Models so far

To illustrate what cycloids are and where their limits are we introduce some examples, formal definitions and some descriptions.

2.1 Traditional Cycloid Examples

Some prominent examples of cycloids by Petri are [26,22]:

- The four seasons as a basic example for fundamental system properties and minimal illustrations about certain concurrency modelling issues [24,23].

While several basic properties can be studied here, the cycloids are trivial due to strongly sequential behaviour.
- The set of cars that drive within a fixed set of driving slots for an infinite time [32].
Based on this very symmetric model most discussions of cycloid behaviour are made. However, already here the high symmetry of the model is important.
- The firemen example where a group of firemen extinguishes a fire (and repeats this infinitely often) (see example set of [15]). On the left some water supply exists and on the right a fire exists that needs to be extinguished. Full buckets of water are then moved by a fireman to the right and given to his neighbor in exchange with the empty bucket.
This kind of model is e.g. still an observable process. However, a fireman can move forward and backward exchanging full and empty buckets with a neighbor. The gaps are fixed, while the usage by the fireman is not. This can lead to different kinds of behaviour that are not covered in the formulas of Valk [32,33,35].
Given five (or more) gaps and two fireman, one fireman can be slow and move only between the first two gaps while the second fireman move very fast, performing all remaining movements over the remaining three (or more) gaps.
Furthermore, the reason for the fireman is to exchange the full and empty buckets. This process of exchange is usually also not covered by the cycloid, since it is modelled implicitly.

Starting from these examples we provide our extensions. Before that we briefly sketch what cycloids are.

2.2 Cycloids

Cycloids are special Petri nets, more precisely the nets that can be expressed as a folding of a certain infinite Petri net. For the example of firemen or cars that move through space and time Petri considered an infinite causal net that models the movement through space and time and the conditions of causality and concurrency. The infinite causal net is called *Petri Space*.

A concrete model can then be derived from the infinite net by folding it with respect to four particular parameters α, β, γ and δ [32]. These parameters characterise the net pattern, which can be folded / repeated infinitely often to build the net systems, which then can generate all possible runs within such a system. In some way they describe the time and space ranges in discrete values. The folding of the Petri space generates a finite net with cyclic behaviour. For the car example also used in [32] an underlying assumption is that only a certain fixed set of cars moves within a fixed set of gaps. This implies that the first and the last car have a maximal distance with respect to the infinite behaviour. For the firemen example this is natural assumption when considering the firemen standing / moving between the water supply and the fire, since this distance is

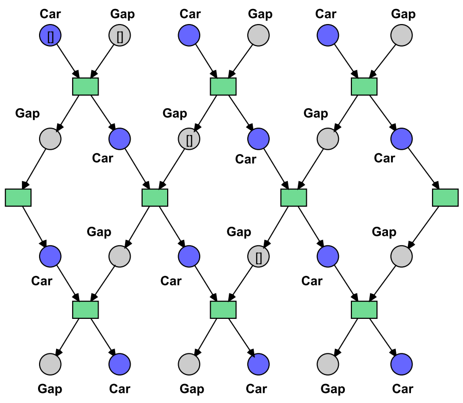

Figure 1. A cutout of the Petri Space

finite and there is a discrete number of places assumed. Even with these strong restrictions different kinds of behaviour are possible: The distances between the cars may vary only within a smaller limit than what would generally be possible considering the set of gaps of the infinite Petri Space, e.g. when they move in column.

Figure 1 shows a cutout of the Petri space. The advance of time is depicted vertically and the advance of space is depicted horizontally. One can see that the cars travel forward in time and space whereas the gaps travel forward in time but backward in space.

To calculate cycloids the Petri space is folded by defining an equivalence relation. The quotient set of the equivalence relation on the Petri space will be the set of net elements of the respective cycloid. The cycloid will have a finite set of net elements since there is a finite number of equivalence classes for the chosen model. Petri space elements in the same equivalence class will collapse which can be seen as the literal folding of the infinite net. For a detailed coverage of these notions we refer to [33].

2.3 Sets of driving Cars

The set of cars that drive within a fixed set of driving slots is an example for a system that exhibits causality and concurrency. In the models that were considered so far it is only possible for a car to move forward if the driving slot in front of it is currently empty. We refer to these slots as *gaps*. In the real world it is possible for a car to move forward even if the slot in front of it is occupied if itself and the adjacent car accelerate simultaneously. However, this is not covered in cycloids, due to the concept of contact in the originally underlying Condition/Event-Nets (C/E-Nets).

Cycloids cover infinite behaviour with a finite set of actions. In our considered models we look at the behaviour as the possible actions of c cars and g gaps

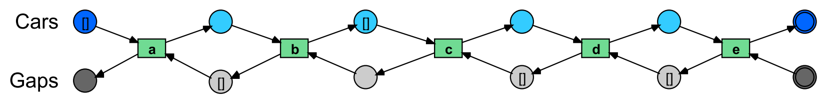

Figure 2. A simple Cycloid (with virtual places)

within $c + g$ driving slots. These driving slots represent the sliding window that is currently looked at in a possibly bigger modelled world. Thus in our models the maximum distance between two cars is $c + g - 1$. Another illustration is that the cars drive in a circuit of length $c+g$ where they cannot overtake one another. If two cars are adjacent to each other the movement of the car in front (meaning that it has to leave the gap, hence releasing it for the next car, indicated by the marked gap place) is the condition for the car behind to be able to drive (causality). If two different cars have each a gap in front of them, they can drive forward independently (concurrency). A Petri net example which portrays a first simple cycloid is shown in Figure 2. It is isomorphic to the net that is created by folding the Petri space with the parameters $(2, 3, 1, 1)$ for $(\alpha, \beta, \gamma, \delta)$. The idea of cars and gaps will play a central role in providing examples for our simple extensions of cycloids which we will do in following sections.

Our considered models have got cyclic net structures. To avoid long and possibly crossing edges in the straight depiction of the lanes we use virtual places which are a feature of our Renew tool [15]. Virtual places enable to create copies of places. In Coloured Petri Nets they are called compound places of fusion sets or fusion place for short [9]. The copies have got a doubled outline and behave semantically identical to the original place. Thus drawing of (preferably) better readable nets becomes possible by putting a copy of some place at the desired location. We use colours in our net drawings to further highlight places and their corresponding virtual copies (virtual places). In Figure 2 places that represent cars are drawn in shades of blue colour, whereas places that represent gaps are drawn in shades of grey colour. Virtual places are drawn in the same colour as their corresponding places. The colour of a place that has got virtual places will only be used once for the place and its copies to maintain a distinct colour scheme. The distinction of transitions is not as important for our considered models so we use a green tone that is set as default in Renew. The initial marking of places is only drawn in the original places but not in the related virtual places.

3 Modelling Requirements for Cycloid Extensions

In the above sections we described what cycloids can do and already mentioned some limitations. This section addresses requirements for the extended cycloids.

First of all we want to decrease the size of the models when modelling cycloids. A similar approach is presented by Valk [35] where he introduces coloured Petri nets for a more compact representation, for which he also provides some

Model	Expresses
m cars, n gaps, one lane	Concurrency
multiple independent lanes, no crossing of lanes	Concurrency
additional overtaking bay of length one, one car overtakes m cars	Conflict
additional overtaking lane of length l, one car overtakes m cars	Conflict
guaranteed cut in	Conflict
forced overtaking	Conflict
guaranteed cut in + forced overtaking	Conflict
multiple cars can overtake	Conflict
multiple cars must overtake	Conflict
multiple cars on a driving slot	Coarsening
overtake can begin on different driving slots (Sliding Window)	Conflict
free crossing of lanes	Conflict
coloured cars	Distinctness
coloured cars + coloured gaps	Distinctness
additional lane with oncoming traffic	Conflict
simultaneous action of cars	Synchronization
representation as a workflow	Workflow

Table 1. A table of our considered extensions

formulas as characterisations. Since we concentrate on the applicability of cycloids for modelling we propose a subset of reference nets [13].

Second, we want to cover more concepts used in modelling. Therefore, conflicts are added. However, the idea is to restrict using them more or less as a consistent extension of cycloids. In which ways this is possible and in which not is discussed in the following sections. In the same context we introduce a special variant of loops, motivated by workflow nets [1] and multi-agent protocols [30,3,11].

4 First Results of Extensions

Table 1 contains our currently covered extensions. We will discuss examples for a selection of our extensions in section 5.

The model with m cars and n gaps on one lane is a simple example that can be expressed as a cycloid. It aids the intuitive understanding of causality and concurrency in distributed systems. Multiple independent lanes without the crossing of lanes increase the complexity of the model but since there are no conflicts it can still be expressed as a cycloid. In our considered models we introduce conflicts in different ways. For example, we add an overtaking bay of length one to the model of cars and gaps. This enables a particular car to overtake other cars which is not possible in traditional cycloid models. An overtaking lane of some length l allows for additional types of behaviour. In models that have an overtaking lane the cut in of the overtaking car can be prevented by another car if it stays in the driving slot intented for cut in. This will be ruled out in a model with guaranteed cut in. The overtaking lane can be unused if the cars do not decide to overtake other cars. We prevent this in a model with forced overtaking. Guaranteed cut in and forced overtaking can also be combined. One can also consider that multiple cars are able to move onto the overtaking lane or even are forced to do so. To describe a coarsened model one can define that multiple cars are able to be located at one driving slot. We model a scenario where an overtaking can begin on different driving slots but has always got a fixed length. This will be called a sliding window. In a model with looser conditions the cars can freely switch between the lanes while they are moving forward. By modelling cars and/or gaps as coloured tokens, we are able to examine the properties of systems containing distinct cars and/or gaps. In the first simple applications of cycloids the cars and gaps are indistinguishable. To further increase the complexity of the system and their models we introduce oncoming traffic on the additional driving lanes. Through the use of synchronous channels provided by the RENEW reference net formalism we model simultaneous actions of cars. Here the actions happen simultaneously in one simulation step. We also consider the presentation as a workflow by unwinding the conditions that regulate the behaviour of the net.

5 Examples for some Extensions

5.1 Conflicts: Overtaking of Cars

At first we introduce a simple conflict by adding another lane to the model which is shown in Figure 3. Please note that the introduced conflict is modelled by virtual places. The lower part of the figure depicts the first lane of driving slots. The upper part of the figure depicts the second lane which we will call the overtaking lane. The net model of each lane contains a row of place pairs which represent the driving slots. In the respective upper row places that represent cars are located, whereas the places the represent gaps are located in the lower rows. If a place representing a car is marked with a token it means that this driving slot is currently occupied by a car. If a place representing a gap is marked it means that this driving slot is not occupied. It makes sense that for each driving slot exactly one of the two places is marked. This is achieved by first choosing an according initial marking of the net (which marks exactly one of the places). Futhermore, the two places for each driving slot are complementary places. This means that each transition that puts a token onto one of the places removes the token from the other one and vice versa. This property also ensures that for all reachable markings each place is marked with at most one token.

The initial marking of the net shows that there are four cars which are located in the first four slots of the first lane. There also is a gap in the fifth slot. Remark that the overtaking lane has only got three driving slots since the places for the first and last slot are virtual places of places that represent slots of the first lane. There are no cars in the overtaking lane which is indicated by the three marked places representing the gaps.

There is cyclic behaviour in the system. For example the cars can drive behind each other in the first lane. The introduced conflict shows when there is a car in the first slot of the first lane and also the second slot of the first lane and the first slot of the overtaking lane are currently empty. In this case the respective first transitions of both lanes (rows) are activated. This means that the car in the first slot of the first lane can decide to either drive forward on the first lane or to sheer out onto the overtaking lane. Having sheered out it can then overtake cars by moving forward on the overtaking lane. The fifth slot of the first lane is also used for cutting in from the overtaking lane which shows another conflict. In this model is it possible that multiple cars decide to move onto the overtaking lane.

5.2 Conflict: Overtaking with Guaranteed Cut In

One can imagine that the conflict at cutting in can be problematic in scenarios of the real world. For example, there can exist compulsive behaviour where one car is not letting another car cut in by blocking the slot used for cutting in. One motivation for modelling with cycloids is seeing them as a specification method for distributed systems. Here the behaviour of systems is described and specified with models that have got cyclic and also live behaviour. Liveness of system means that a continuous possibility of carrying out all the actions of the systems exists. This is a desired system property in many cases. Inspired by the MAS (multi agent system) context, it looks sensible to use cycloids for the specification of protocols that define the interaction of agents with other agents. The rules of a protocol ensure that the interaction of the agents leads to constructive behaviour. In the considered example of compulsive behaviour our rules should ensure that it will always be possible for a car to cut in after having sheered out. A reason for this condition in the real world can be that the overtaking lane ends because a section of no passing follows. If the cars on the first lane block the overtaking car from cutting in it will have to wait on its lane. This will mean a loss of efficiency which should be prevented by specifying that only constructive behaviour should happen. We adapt the simple model shown in Figure 3 to prevent the unwanted compulsive behaviour and guarantee the cut in.

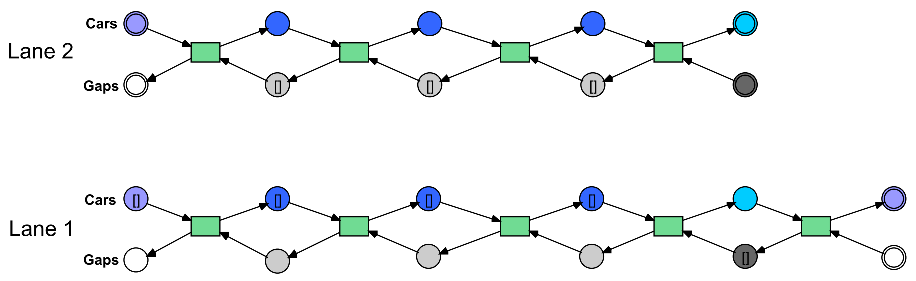

Figure 3. A model containing a conflict

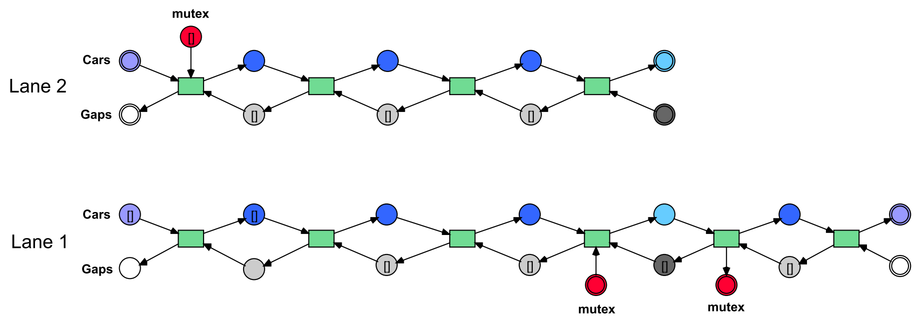

Figure 4. A model for guaranteed cut in

The Petri net model for guaranteed cut in is shown in Figure 4. The graphical representation is similar to Figure 3. The net contains a new place named 'mutex' which regulates a condition of mutual exclusion. The idea of the introduced condition is that cars are simply not allowed to move onto the slot used for cutting in while a car is currently located on the overtaking lane. If a car is currently located at the cut in slot, it is also not allowed for cars to enter the overtaking lane. In the net this is realized by making the mutex available initially which means that it is possible that either a car moves onto the cut in slot or a car moves onto the overtaking lane. In both cases the mutex is made unavailable by consuming the token in order to prevent the other action from happening. After a car leaves the cut in slot the mutex should be made available again. This is done by letting the transition responsible for leaving the cut in slot create a token on the mutex place. This construction leads to only one car being able to be located on the overtaking lane at the same time.

5.3 Conflict: Forced Overtaking

In Figure 3 we have seen that the cars can ignore the option to sheer out and drive forever on the first lane which leaves the overtaking lane empty. This is not quite the behaviour that the modeller intended. We make another change to the net to change its behaviour. We want to achieve that in the case of an empty overtaking lane the first car that is being able to sheer out onto the overtaking lane must do so and is not allowed to drive forward on the first lane. We call this forced overtaking. We also want to enforce that if a car is currently located on the overtaking lane it is not possible for another car to enter the overtaking lane.

The Petri net model for forced overtaking is shown in Figure 5. In essence we still use the familiar color scheme. We introduce two new places that represent new conditions for regulating the overtaking behaviour of the cars. The place 'guard1' und its virtual places are drawn in yellow colour whereas the place 'guard2' and its copies are drawn in red colour. Initially the place 'guard2'

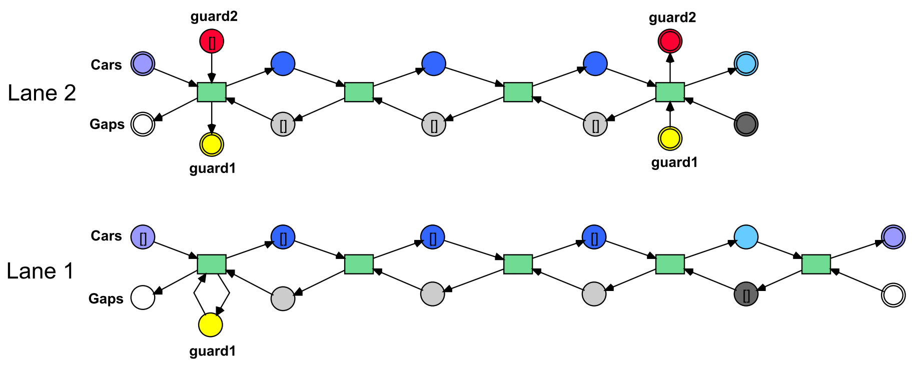

Figure 5. A model for forced overtaking

(second guard) is marked whereas the place 'guard1' (first guard) is not. The first lane initially has got four cars that are distributed on the first four driving slots. The overtaking lane has got no cars initially. The two guards manipulate the behaviour of the two transitions that are in the postset of the place that models the car in the first slot of the first lane.

For this car to drive forward on the first lane it is necessary that the first guard is marked. This is the case if and only if a car is currently located on the overtaking lane. So for an empty overtaking lane it is forbidden to drive forward on the first lane. If there is a car currently located on the overtaking lane and the first guard is marked it will be allowed for the car to drive forward one the first lane. In this case the token of the first guard is consumed and created right after because the condition of the overtaking lane being not empty does not change. For a car on the first slot of the first lane to sheer out onto the overtaking lane it is necessary that the second guard is marked. This is the case if and only if no car is currently located on the overtaking lane. When sheering out, the corresponding transition consumes the token of the second guard and creates a token on the first guard to update the current state of the conditions. When cutting in, the reverse actions happen. The token of the first guard is consumed so cars on the first slot of the first lane are no longer able to drive forward on the first lane and must sheer out again. Also a token is created on the second guard so cars are able to sheer out again. We see that there are three types of using the guards: Consuming the condition (removing the token), providing the condition (creating a token) or claiming the condition (needing but preserving the token).

5.4 Synchronous Actions

So far, concurrent actions in our models can happen simultaneously in one simulation step but can also happen after one another. We introduce further changes to create a type of behaviour where it is forced that particular actions happen in one simulation step. For this we use another concept of Renew, namely synchronous channels. Synchronous channels consist of uplinks and downlinks. Downlinks can be seen as transitions that are calling other transitions whereas uplinks are the transitions being called. However, this metaphor is not exactly correct since the transitions are carried out simultaneously which should not be compared to a method call in programming. We will also use coloured tokens which are supported by Renew as well. We remark here that it is also possible to use multiple tokens on one place in other examples to portray coarsened representations of bigger nets. We refer to [10].

Figure 6 shows a Petri net model with synchronized actions. We use the metaphor that the fuel of the cars is running out as they drive. They have to be refueled simultaneously. While the gaps in our model are still represented by black tokens ([]), we use coloured tokens to model the cars. For futher information concerning the use of coloured tokens in our modelling context we refer to [33] and [10]. We use two types of tokens, a token named 'fuelfull' for a car that has got enough fuel to drive and a token named 'fuelempty' for a car that has not got enough fuel to drive.

The net has got no conflicts and only one lane with five driving slots. Initially there is a 'fuelfull' token on the two places representing the cars on the third and fourth driving slot, respectively. That means that initially there are cars with sufficient fuel on the third and fourth driving slot. There are gaps on the other slots. When moving from the fifth to the first driving slot the type of the respective token changes from 'fuelfull' to 'fuelempty' by the corresponding action inscription. This is done by removing the 'fuelfull' token from the fifth slot and creating a new 'fuelempty' token on the first slot. In other cases 'fuelfull' tokens travel as 'fuelfull' tokens and 'fuelempty' tokens travel as 'fuelempty' tokens. A car that has not enough fuel to drive should not be able to move from the second to the third driving slot. In the net this is realized by adding the guard inscription 'guard x.equals("fuelfull")' to the 'b' transition. The guard states that the variable 'x' which is used for the transfer of tokens must be bound to a token of type 'fuelfull'. To continue driving the cars can get refueled by the two transitions depicted at the upper left hand area of the figure. These transitions carry the uplink inscriptions ':refuelA()' and ':refuelB()'. They each consume a

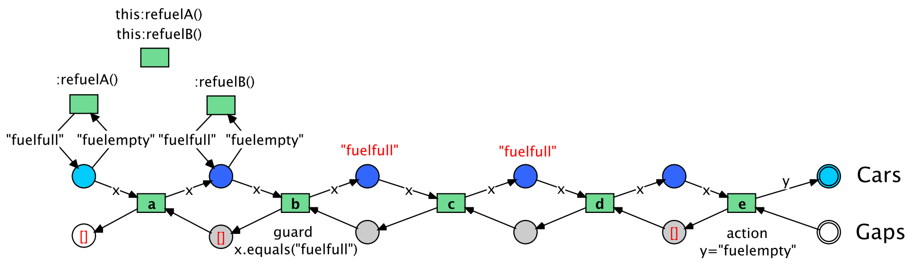

Figure 6. A model with synchronized actions (:refuelX()) (tokens in red)

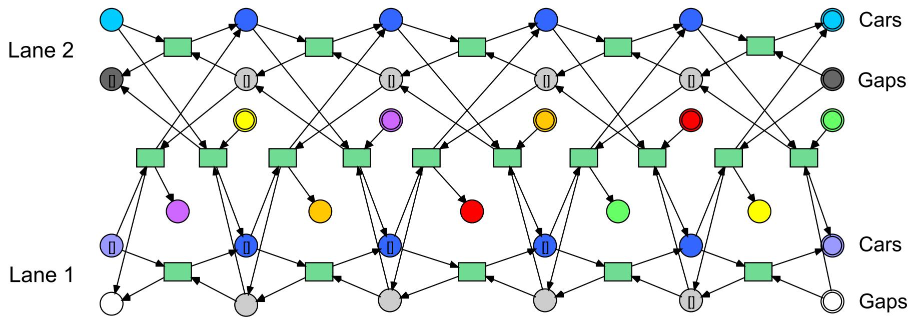

Figure 7. A model for a sliding window overtaking process

'fuelempty' token from the places that represent the first two cars and create a 'fuelfull' token on them, respectively. This models the refueling of the cars. These two transitions are synchronized by the transition depicted above them which carries the two downlink inscriptions 'this:refuelA()' and 'this:refuelB()'. The inscriptions mean that of the current net instance which is referenced by the keyword 'this' the channels 'refuelA' and 'refuelB' are synchronized. After the cars have been refueled synchronously they can continue to drive.

In Figure 3 we see that the action of overtaking always begins at a fixed slot and has got a fixed length. We want to keep its fixed length but let it begin at arbitrary slots. We call this idea *sliding window*. Figure 7 shows a model for a sliding window overtaking process. The model has got two lanes with five slots each. Initially there are four cars on the first four slots of the first lane, whereas there are no cars located on the second lane. In essence the cars can freely switch between the two lanes if a corresponding gap exists at the moment. This is controlled by the ten transitions depicted at the middle of the figure. To enforce that a car cuts in at a certain slot after having sheered out we introduce five new conditions that are represented by the places depicted in yellow, violet, orange, red and light color. These are unmarked initially. The conditions are provided when sheering out and consumed when cutting in. We choose a constellation in which a token is created on the place that needs to be marked for the respective car to cut in right after having sheered out. This leads to the sliding window having an overtaking length of one. However, unwanted behaviour still exists in the model. For example, if multiple cars switch to the overtaking lane the cut in slot to be enforced is not mapped to the sheered out car which enables one of the cars to use the cut in slot being enforced for the other one. Also the cars can stay on the overtaking lane forever and choose to never cut in the first lane again.

5.5 Presentation as a Workflow

We also show a presentation as a workflow by first unwinding the conditions of the Petri net model and then transforming the emerging net into a workflow

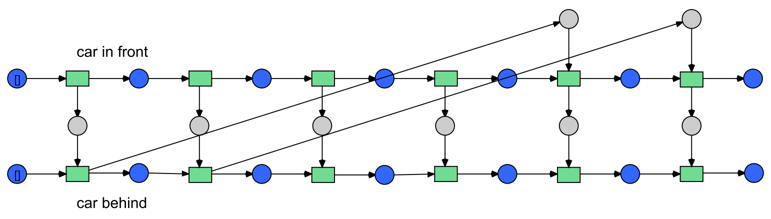

Figure 8. Unwinding the conditions of the model

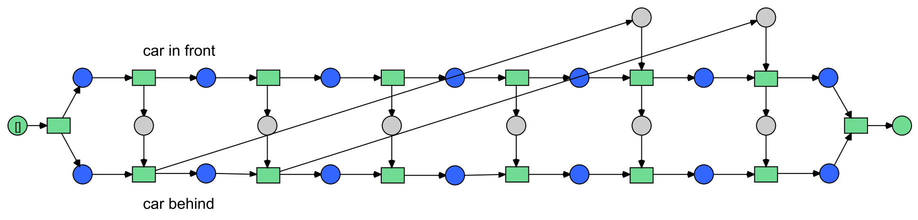

Figure 9. The transformed workflow Petri net

Petri net. We consider a example wherein the maximum distance of two driving cars is five which means that there are six driving slots. In this example we have got two cars. Initially the two cars are located on two arbitrary slots that are adjacent to each other (so they have got a distance of one unit).

The Figure 8 shows a cutout of the causal net that emerges from unwinding the following conditions. Again we use the colour scheme of blue and grey. In order to get a workflow Petri net we add a start place and an end place together with the respective forking and joining transitions. This is depicted in Figure 9.

First of all the car in front needs to move forward to enable the car behind it to move as well. This is modelled by the six gaps that are next to each other in the graphical depiction. Secondly, since the maximum distance must not exceed the value of five slots the car behind needs to move forward before the movement of the car in front would cause the distance between them to exceed five driving slots. This is modelled by the two gaps located at the upper area of the graphical depiction. It is worth to see that the maximal distance can in some circumstances be larger in Figure 8, when the car behind has proceeded by two gaps. Then the car in front can move until it has to wait again for the car behind. This behaviour is *not* possible in Figure 9. Start and end places restrict the possible behaviours. Since e.g. in our MAS models we require to have such starting and ending condition for one of the conversations of two agents, this is not always problematic. What is gained by the restriction is a better understandability of behaviours. One has to start in a single precisely defined initial state and then, after the synchronisation via the end place can restart.

6 Discussion of Mapping of Entities to Extended Cycloids to Traditional Cycloids

After the discussion of extension options above we now discuss the mapping of modelling entities to extended cycloids and of extended cycloids to traditional cycloids. In this section we restrict ourselves to the informal motivation of the mappings. These mappings are most often straight forward, so that we do not provide a formalization. First of all the extension by the use of reference nets needs to be discussed. Second the additional concepts of conflict (and (for-)loops) are treated.

Using virtual places is well known from other formalisms like Coloured Petri Nets [9]. Here also this is treated as a shortcut for modelling. With respect to modelling in general virtual places can be considered as a kind of *goto*. However, when used in a disciplined manner readability increases. At the same time one has to confess that e.g. the visibility of conflicts may be not so obvious for the uninformed reader, see Figures 2 and 3. The problem of crossing arcs can be seen in the net models of Valk [32,35]. The larger the models become the more difficult is to read the net model.

Other shortcuts like the colouring of tokens and the use of variable is nicely demonstrated by the results of Valk in [35]. When modelling complex systems nowadays high-level modelling languages are necessary and wide spread. We use these to cover the mapping of application entities / units to a net. Real world examples often have generalised descriptions and require therefore appropriate models. This becomes possible when we use sets or classes of cars to describe a system. Looking at all entities as individual entities directly leads to C/E-Net models. Each entity like a car is moving through space and time. When classes of cars exist, then we do not want model new kinds of behaviours. In Petri's approach there exists only one kind of car or all cars are reduced / mapped to the class car. This fine if all cars are always treated equally.

When cars need different treatment, e.g. a police car that must have to option to overtake and enforce its cut in, than we need to reflect this in the model. This is covered by the different version we show in Section 5.

Another scenario not discussed before here is that the overtaking of autonomous driving cars can be specified as a kind of required behaviour. Interpreting the overtaking model a specification and ensuring that only correct and complete overtaking scenarios are allowed by the (extended) cycloid, we can ensure that the action of overtaking is safe (if no breakdown of the hardware occurs). An overtaking car can bypass a certain number of cars and then sheer in again before cars on the other lane can collide with them.

Actually when modelling the car on the other lane that approaches a convoy from the opposite direction there is a certain number of free gaps for the overtaking actions. If and only if the speed and the number of gaps is sufficient a car may overtake the other cars.

Modelling extended cycloids to (traditional) cycloids like those used by Petri or in [32] requires a solution for the different concepts (conflict, synchronous channel, colour, loop etc.). The main motivation to e.g. add conflicts to the modelling concepts is to reach a higher expressability of the modelling language. Conflicts are actually one of the basic concepts in general Petri nets [29]. In a scenario not only concurrency and causality need to be covered. Alternative actions also need to be covered by models.

Following the finite set of actions of a modelled system this implies that if the number of actions within a given scenario at least one action must be used more than once. For infinite scenarios this is obvious. Good systems (designed following the ideas of being a correct workflows for each workflow of a system) should react to an external stimuli like a message and then remain in a state until the next external event occurs. This is an ideal scenario, but it allows to specify a reaction of a system more easily. Interference of several concurrent events that have conflicts with respect to actions of resources are explicitly prohibited in our kinds of system models. This restricts the systems that can be modelled, but due to the inclusion of conflicts it still can cover more processes than a traditional cycloid.

However, one can consider the extended cycloid as a kind of description of several cycloids. Each cycloid has an initial marking from which it can perform a certain set of actions until it starts in the beginning again.[1]

Now, if there is exactly a single place with two transitions in conflict within a model then this *only* describes two different cycloid structures that may be used. Given that one alternative is never used than exactly the normal cycloid could be used for its description. The same holds for the other alternative. Both cycloids can be considered to describe the behaviour of a system or unit. The only difference lies in that part of the process that reflects the selection of the alternative. When considering a cycloid model as the blueprint for a unit then the cyclic behaviour can be seen as a composition of processes of this unit. What needs to be seen here is, that the possible behaviours can change, since the possible distances between tokens can be increased due to the larger model structure.

In the case that there is an alternative in the system structure then for each alternative there is different unit that can be used to generate the corresponding process. In some way we have used this technique when we developed the Petri net protocols for our Petri net based multi-agent systems. First models just described a concurrent behaviour of two or more agents interacting. This exactly reflects a cycloid structure. When agents repeat this behaviour then this results in the cycloid process.

To increase the modelling power of our agent protocol modelling technique we introduced alternative. The same was added by the AUML approach of James

[1] This is a simplification, however, the number of variants is strictly limited due to the fact that there is a maximal possible distance between the first and the last token within a cycloid.
When using the idea of workflows with a single initial marking and a single end marking place, then this is easy to see. Nevertheless this is not so simple in the general case for cycloids, since the initial marking is normally larger than a single token.

Odell et al [21]. In both approaches the idea of a set of different, but highly related behaviour of a set of entities describe the behaviour of the overall system unit. Each agent can again be considered to be a unit, being described by cycloids, but this is not deepened here.

For this contribution it is important to notice that we can either assume that we describe a net structure without alternatives to form a basic unit of behaviour. The concatenation of the structure, the unfolding with respect to a given initial marking, results in uniform behaviour that can be described by a cycloid. Adding alternatives within the structure can be seen as the option to concatenate some variants units to form the behaviour.

The other modelling perspective is to look at process and fold these process to structures, as it is done by Petri with his Petri space. The folding can be done for the simple model of cars and gaps for the dimension of space of time. Folding both results in the cycloid. More complex models that have more than two dimensions. These kind of models have been considered in the thesis of Fenske [4]. In the context of the thesis a modelling tool was developed to visualise and to simulate the models. Experiments with these models illustrated that a more compact notion was necessary, what we discussed here.

An interesting perspective is to not use process and fold them to cycloids, but to use branching process and fold them to our extended cycloids. For a restricted set of branching process (with a finite set of regular behaviour patterns) this should be no problem using our proposed mapping. The composition as already used by Valk in [33] illustrates how such a construction is possible. Folding and composition of cycloid(structure)s therefore needs more investigation.

7 Related Modelling Approaches

Compared to the above proposals some modelling approaches were motivating the extensions. In general repetitive behaviour is assumed. This can be found in may application areas of informatics.

Building applications is normally designed for certain scenarios [29]. Embedding the scenario models in an environment to restart a finished scenario the repeated behaviour becomes obvious. In our own models we used this idea for workflow modelling (see e.g. [7,28,37,16,36]), software engineering (see e.g. [17,3]), multi-agent systems (see e.g. [3,12]) or general modelling by units (see e.g. [19,31] [5,38]). Often in these models the first assumption is to have a limited number of behaviours to ease the modelling process. With some simple but powerful generalisations/extensions potentially infinite numbers of behaviours are covered (due to turing completeness by the extensions). These are obviously not covered in the proposed extensions for the cycloids here.

Our proposals of some bounded set of possible behaviours is of course also followed in many other modelling approaches. UML (Unified Modeling Language) with its various modelling techniques, BPMN (Business Process Model and Notation) or other Petri net formalisms and restrictions of Petri nets like workflows use in their methods the idea of simple behaviours that can be executed repetitively. The same holds for multi-agent systems where agents inherit some kinds of behaviour. Usually this behaviour is triggered from the outside and should terminate after the external stimulus or message has been processed. In our Mulan framework this is nicely covered. Due to the usage of higher inscriptions languages like Java again the MAS modelling becomes turing complete. Especially the possibility of self-adaptation (see e.g. [3,12]) is an important feature here. However, to still build controllable systems each single behaviour model should be restricted.

Cycloids provide the means for a single scenario with in fact certain basic blocks of repeated behaviours. There is always a finite number of different marking in a cycloid. Only the variations of such markings provided the possibly infinite number of process that can be build from an initially marked cycloid. Correct workflows [1] without loops are a special case of our cycloid extensions. Even workflow with for loops (for a given model (using coloured Petri nets to model the conditions)) can be considered as extended cycloids, however, we do not deepen the discussion here, since this requires some further investigations for the mappings for easier understandability.

Our extensions more or less adds some further kinds of markings of a set of places. However, the modellers can use conflicts or for loops, what ensures that the number of possible markings remain finite. In addition to the cycloids now not all transitions within an extended cycloid will fire when an initial marking is reached. Obviously alternatives hinder such firings. The same holds if a loop is skipped due to a possibly false condition from the beginning onwards.

With respect to the unit theory a unit describes a any kind of system (see e.g. [19,31,5,38]). This system can contain several actions. Repeated behaviour of such a unit with no alternative but with internal concurrency can be directly described by a cycloid, if the unit can restart from its initial marking once it has reached an end marking. This initial marking and the end marking may be sets of markings, since the shortcut of the model may lead to different kinds of markings of the cycloids and these marking may not exactly match the initial marking. Due to the limited number of markings reachable with a cycloid the options for the initial marking and end marking sets is finite.

8 Conclusion

Modelling repeated behaviour is demanding. As a simple solution cycloids were developed. Restrictions with respect to the modelling power inspired us to extend the cycloids.

The extension were done in two different directions. First new concepts were added to the model more complex systems. Especially conflicts need to be mentioned here. Second reference nets are used to build the models. Using reference nets allows to build more complex models. Applying restrictions on the usage of colours, synchronous channels and other constructs leads to the possibility to map the coloured models to the extended cycloids which again can be somehow described by sets of cycloids.

Future work needs to address the provision of closed formulas as this was done during the last years by Valk. Relations to workflow modelling and other modelling areas like MAS are highly promising. Overall we are sure that the results can be used to provide a formal basis for some parts of the unit theory.

Starting from [22,23], finally, for the fundamental assumptions of a system theoretical perspective of the unit theory, we have to look at the fundamental assumption of Petri that "*. . . if we base our models on the combinatorial concepts of signal flow suggested by informatics, and insist on continuity (as Zuse did), we end up inevitably with a model of a finite universe.* " [27]. Following the principle idea of the general system theory we consider it nevertheless necessary for people modelling for informatics to cover open systems which directly leads to somehow infinite processes. At the same time embedding finite open systems into a finite universe can still impose a kind of understanding of a non-deterministic (infinite?) environment.

Computer scientists, software engineers and other people of informatics urgently need better modelling techniques. Extended cycloids are one option to provide a solid basis on the ground of concurrency theory for more expressive modelling techniques for application areas where a fixed set of scenarios need to be covered in formal manner.

References

1. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) Application and Theory of Petri Nets 1997. pp. 407–426. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
2. Brauer, W. (ed.): Net Theory and Applications, Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, October 8-19, 1979, LNCS, vol. 84. Springer, Berlin Heidelberg New York (1980), `https://doi.org/10.1007/3-540-100016`
3. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (Apr 2010), `http://www.sub.uni-hamburg.de/opus/volltexte/2010/4666/`
4. Fenske, U.: Petris Zykloide und Überlegungen zur Verallgemeinerung. Diploma thesis, University of Hamburg, Vogt-Kölln Str. 30, D-22527 Hamburg (2008)
5. Hewelt, M.: Grundlegende Konstrukte einer einheitentheoretischen Modellierungstechnik. Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (2010)
6. Hewelt, M., Wester-Ebbinghaus, M.: United – a Petri net based framework for modeling complex and adaptive systems. In: Moldt, D. (ed.) Petri Nets and Software Engineering, International Workshop, PNSE'09. Proceedings. pp. 207–226. Technical Reports Université Paris 13, Université Paris 13, 99, avenue Jean-Baptiste Clément, 93 430 Villetaneuse (Jun 2009), `http://www.informatik.uni-hamburg.de/TGI/events/pnse09/`
7. Jacob, T., Kummer, O., Moldt, D., Ultes-Nitsche, U.: Implementation of workflow systems using reference nets – security and operability aspects. In: Jensen, K. (ed.) Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. University of Aarhus, Department of Computer Science, Ny

Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark (Aug 2002), dAIMI PB: Aarhus, Denmark, August 28–30, number 560
8. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence **117**(2), 277–296 (2000)
9. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer-Verlag (Jul 2009)
10. Jessen, B.: Untersuchungen zur konzeptionellen Erweiterung von Zykloiden. Bachelor thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (2020)
11. Köhler, M., Moldt, D., Rölke, H.: Modelling mobility and mobile agents using nets within nets. In: van der Aalst, W., Best, E. (eds.) Proceedings of the 24th International Conference on Application and Theory of Petri Nets 2003 (ICATPN 2003). LNCS, vol. 2679, pp. 121–139. Springer-Verlag (2003), `http://www.springerlink.com/link.asp?id=xf5vqh9cn0q1nukw`
12. Köhler-Bußmeier, M.: Koordinierte Selbstorganisation und selbstorganisierte Koordination: Eine formale Spezifikation reflexiver Selbstorganisation in Multiagentensystemen unter spezieller Berücksichtigung der sozialwissenschaftlichen Perspektive. Habilitationsschrift, University of Hamburg (2009), `http://epub.sub.uni-hamburg.de/informatik/volltexte/2010/144/`
13. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002), `http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=`
14. Kummer, O., Stehr, M.: Petri's axioms of concurrency- A selection of recent results. In: Azéma, P., Balbo, G. (eds.) Application and Theory of Petri Nets 1997. pp. 195–214. No. 1248 in LNCS, Springer-Verlag, Berlin Heidelberg New York (1997)
15. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew – the Reference Net Workshop (Jun 2016), `http://www.renew.de/`, release 2.5
16. Markwardt, K.: Strukturierung petrinetzbasierter Multiagentenanwendungen am Beispiel verteilter Softwareentwicklungsprozesse. Ph.D. thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (2013), `http://ediss.sub.uni-hamburg.de/volltexte/2013/6027`
17. Markwardt, K., Moldt, D., Offermann, S., Reese, C.: Using multi-agent systems for change management processes in the context of distributed software development processes. In: Sadiq, S., Reichert, M., Schulz, K. (eds.) The 1st International Workshop on Technologies for Collaborative Business Process Management (TCoB 2006). pp. 56–66 (2006)
18. Minkowski, H.: Raum und Zeit. Physikalische Zeitschrift **10**, 104–111 (1909), also in: Jahresbericht der Deutschen Mathematiker-Vereinigung, **18**, 75–88 (1909); talk at 80. Naturforscherversammlung zu Köln, 21. September 1908
19. Moldt, D.: Petrinetze als Denkzeug. In: Farwer, B., Moldt, D. (eds.) Object Petri Nets, Processes, and Object Calculi. pp. 51–70. No. FBI-HH-B-265/05 in Informatics Report, University of Hamburg, Vogt-Kölln Str. 30, D-22527 Hamburg (2005)
20. Moldt, D.: PAOSE: A way to develop distributed software systems based on Petri nets and agents. In: Barjis, J., Ultes-Nitsche, U., Augusto, J.C. (eds.) Proceedings of The Fourth International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS'06), May 23-24, 2006 – Paphos, Cyprus 2006. pp. 1–2 (2006)
21. Odell, J., Parunak, H.V.D., Bauer, B.: Representing agent interaction protocols in UML. In: Ciancarini, P., Wooldridge, M.J. (eds.) Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Limerick, Ireland, June 10, 2000, Revised Papers. LNCS, vol. 1957, pp. 121–140. Springer (2000)

22. Petri, C.A.: Cultural aspects of net theory. Soft Comput. **5**(2), 141–145 (2001), `https://doi.org/10.1007/s005000000070`
23. Petri, C.A.: Mathematical aspects of net theory. Soft Comput. **5**(2), 146–151 (2001), `https://doi.org/10.1007/s005000170001`
24. Petri, C.A.: Concurrency. In: Brauer [2], pp. 251–260, `https://doi.org/10.1007/3-540-100016`
25. Petri, C.A.: Introduction to general net theory. In: Brauer [2], pp. 1–19, `https://doi.org/10.1007/3-540-100016`
26. Petri, C.A.: Nets, time and space. Theoretical Computer Science **153**(1–2), 3–48 (1996)
27. Petri, C.A.: On the physical basics of information flow. In: van Hee, K.M., Valk, R. (eds.) Applications and Theory of Petri Nets, 29th International Conference, PETRI NETS 2008, Xi'an, China, June 23-27, 2008. Proceedings. LNCS, vol. 5062, p. 12. Springer (2008), `https://doi.org/10.1007/978-3-540-68746-7_5`
28. Reese, C.: Prozess-Infrastruktur für Agentenanwendungen, Agent Technology – Theory and Applications, vol. 3. Logos Verlag, Berlin (2010), dissertation. Pdf: `http://www.sub.uni-hamburg.de/opus/volltexte/2010/4497/`
29. Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer, Berlin Heidelberg (2013), `http://dx.doi.org/10.1007/978-3-642-33278-4`
30. Rölke, H.: Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen, Agent Technology – Theory and Applications, vol. 2. Logos Verlag, Berlin (2004), `http://logos-verlag.de/cgi-bin/engbuchmid?isbn=0768&lng=eng&id=`
31. Tell, V., Moldt, D.: Ein Petrinetzsystem zur Modellierung selbstmodifizierender Petrinetze. In: Schmidt, K., Stahl, C. (eds.) Proceedings of the 12th Workshop on Algorithms and Tools for Petri Nets (AWPN 05). pp. 36–41. Humboldt Universität zu Berlin, Fachbereich Informatik (2005)
32. Valk, R.: On the structure of cycloids introduced by Carl Adam Petri. In: Khomenko, V., Roux, O.H. (eds.) 39th PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings. LNCS, vol. 10877, pp. 294–314. Springer (2018)
33. Valk, R.: Formal properties of circular traffic queues and cycloids (2019), `http://uhh.de/inf-valk-traffic`
34. Valk, R.: Formal properties of petri's cycloid systems. Fundam. Inform. **169**(1-2), 85–121 (2019)
35. Valk, R.: Circular traffic queues and Petris cycloids. In: tienne Andr, Petrucci, L. (eds.) 41st International Conference, PETRI NETS 2020, Paris, France, June 21-26, 2020, Proceedings. LNCS, vol. accepted, pp. xxx–xxx. Springer (2020)
36. Wagner, T.: Petri Net-based Combination and Integration of Agents and Workflows. Ph.D. thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (2018)
37. Wester-Ebbinghaus, M.: Von Multiagentensystemen zu Multiorganisationssystemen – Modellierung auf Basis von Petrinetzen. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (12 2010)
38. Wester-Ebbinghaus, M., Moldt, D.: Abstractions in actor and activity modeling. In: Nüttgens, M., Thomas, O., Weber, B. (eds.) EMISA. LNI, vol. 190, pp. 195–200. GI (2011)
39. Wester-Ebbinghaus, M., Moldt, D., Reese, C., Markwardt, K.: Towards organization–oriented software engineering. In: Züllighoven, H. (ed.) Software Engineering Konferenz 2007 in Hamburg: SE'07 Proceedings. LNI, vol. 105, pp. 205–217. GI (2007)