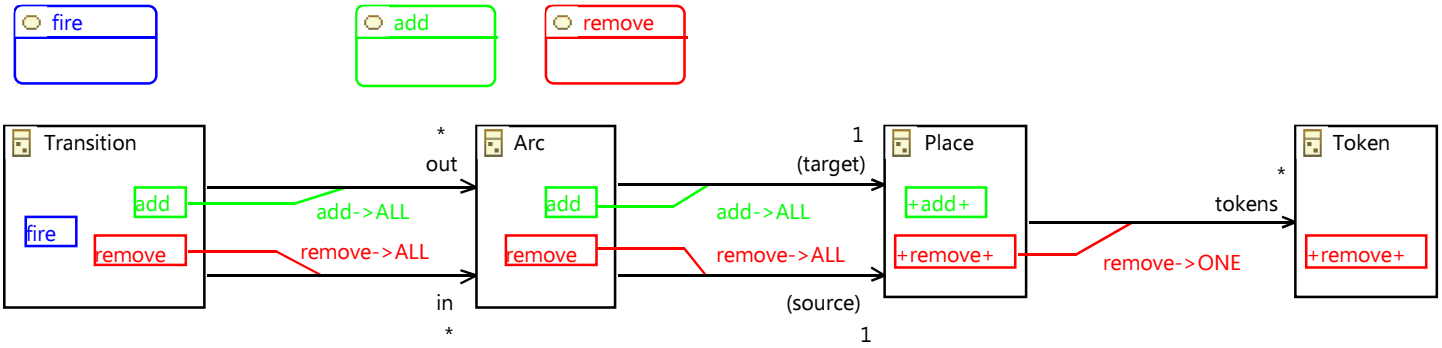
PNNL 81: Cover Picture Story



Cover Picture Story

An ECNO semantics for Petri nets

Ekkart Kindler

Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Lyngby
DENMARK
eki@imm.dtu.dk

Abstract. The *Event Coordination Notation* (*ECNO*) allows modelling the behaviour of software on top of structural software models – and to generate program code from these models fully automatically.

ECNO distinguishes between the local behaviour of *elements* (objects) and the global behaviour, which defines the coordination of the local behaviour of the different elements. The global behaviour is defined by ECNO's *coordination diagrams*, whereas the local behaviour of the different elements can, for example, be modelled by a simple form of Petri nets, *ECNO nets*.

The ideas of ECNO have already been presented in earlier work. In this paper, we will show that the ECNO, in turn, can be used for modelling the behaviour of Petri nets in a simple and concise way. What is more, we will show that the ECNO semantics of Place/Transition Systems can easily be extended to so-called signal-event nets.

Keywords: Model-based Software Engineering, Local and global behaviour modelling, Event coordination, Petri net semantics.

1 Introduction

The cover picture of this issue of the Petri Net Newsletter shows the semantics of Place/Transition Systems – or at least a major part of it – in the *Event Coordination Notation* (*ECNO*). In this Cover Picture Story, we will use this semantics as an example to explain and illustrate the principles, concepts, and modelling philosophy behind the ECNO. Since the ECNO has already been explained in earlier work [1–3], we do not go into its details and do not dwell on a systematic explanation of its concepts. We rather explain the ECNO concepts as far as they are needed to understand the Petri net semantics formulated in terms of the ECNO.

2 Model-based Software Engineering

Before explaining the cover picture and the ECNO semantics of *Place/Transition Systems* (*P/T-systems*) [4] and of *signal-event nets* (*SE-nets*) [5, 6], let us give a brief background on *Model-based Software Engineering* (*MBSE*) by following up a narrative that we started some years ago [7].

One of the main ideas of Model-based Software Engineering is that the complete program code or major parts of it can be generated from models, which essentially capture the software's domain on a high level of abstraction. We had illustrated this idea by modelling the domain of Petri nets and by providing an idea of how a graphical Petri net editor could be generated fully automatically from such a domain model combined with some additional information on the graphical representation of Petri nets [7].

Figure 1 shows a UML class diagram, which captures the main concepts of P/T-systems. A Petri net consists of a number of Nodes and a number of Arcs. Each Arc has exactly one source and one target node; in turn, every node can have any number of in-going and out-going arcs. In fact, nodes can be of two kinds: Transitions or Places, and Places may contain any number of Tokens. For

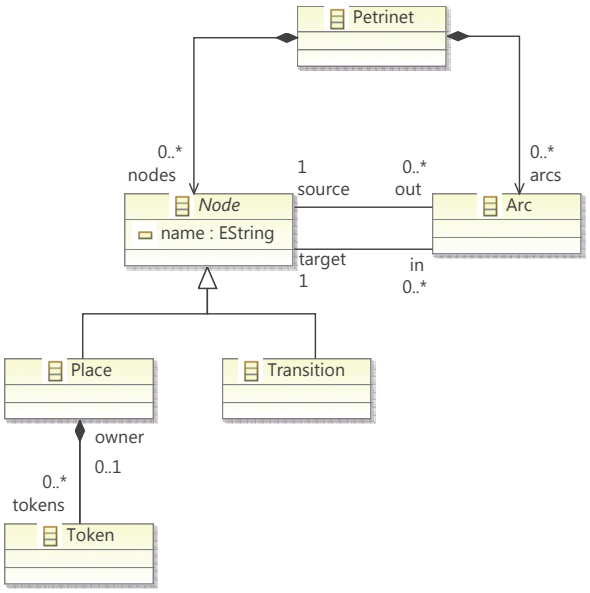


Fig. 1. Domain model for P/T-systems

P/T-systems, there would be an additional constraint to the end that an arc may connect only a place to a transition or the other way round. This could easily be formalized in the *Object Constraint Language* (*OCL*). But, we do not formalize this constraint here for two reasons: firstly, the focus of this paper is

not on modelling the structural or syntactic aspects of a domain in UML or OCL; secondly, we will reuse the same model later in Sect. 4 for signal-event nets, in which an arc may run between two transitions – as so-called *signal arc*.

The model from Fig. 1 defines the concepts of Petri nets, which are models themselves. Therefore, a model such as the one from Fig. 1 is often called a *meta model*: a model of another model or modelling notation. But, we stick with the term *domain model* in this paper. If we had included OCL expressions for the additional constraints, this model would exactly capture what P/T-systems are – from a syntactic point of view. In a sense, this domain model is the software engineering way of rephrasing the well-known mathematical definition of P/T-systems as a tuple $\Sigma = (S, T; F, M_0)$ – leaving out capacities and arc weights from the definition of [4].

From a Petri net point of view, being able to formulate the syntax of some version of Petri net is not too exciting. Formulating the semantics of some version of Petri net, i. e. its *firing rule*, is more interesting. And also from the software engineering point of view, modelling the behaviour of a system so that the program code of the system can be generated from this model fully automatically is the more challenging part. And this is what the ECNO has been made for.

3 Firing rule of P/T-systems

As mentioned earlier, the cover picture of this issue of the Petri Net Newsletter essentially formalizes the firing rule of P/T-systems as an ECNO *coordination diagram*. We will explain this diagram and some supplementary models in this section. In order to make the paper self-contained and to ease following the explanations, the coordination diagram is shown again in Fig. 2. In the explanations, we jump right into the middle of the ECNO diagram – explaining ECNO's concepts and notations on the way.

3.1 Events

In order to be able to coordinate "something" among the different elements of a Petri net, the diagram of Fig. 2 defines some *events*, which are graphically represented by rounded boxes. The most important event for defining the behaviour of Petri nets is the event fire; but, there are two auxiliary events: add and remove, which take the role of actually removing and adding the respective tokens from the respective places when a transition fires. It is the coordination of these events and their joint execution by the different elements that eventually will make up a firing step.

3.2 Coordination and interactions

In order to explain the basic idea of the coordination diagram of Fig. 2, let us assume that a Transition should participate in an event fire – i. e. the transition should fire. Then, the Transition would also need to participate in an event add

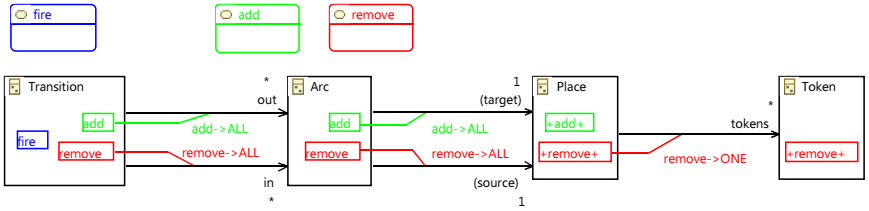


Fig. 2. ECNO coordination diagram for P/T-systems

and an event remove. This is actually not defined in the coordination diagram itself; it is defined by the local behaviour of the Transition, which will be discussed later (see Fig. 4). The coordination diagram, however, says which other elements need to participate in an event when a transition participates in an event. The Transition does not require any other participants for the fire event. For the add event, however, the coordination diagram requires that every out-going Arc must participate in the add event too. This is indicated by the green box with label "add" inside the Transition which is connected to the association out with *coordination annotation* "add->ALL"; this can be read as: if a Transition participates in an add event, all elements at the other end of the links that represent out-going Arcs need to participate in the add event too. We call such a box inside an element a *coordination set*. In the example, there are two coordination sets for the Transition. The coordination set for event remove along with the coordination annotation "remove->ALL" says that every in-going Arc must participate in a remove event, when the Transition participates in a remove event. Summing up the coordination requirements for the transition, we know that, when a transition participates in a fire event, all its in-going arcs need to participate in a remove event and all its out-going arcs need to participate in a add event. The combination of all these participating elements with their events is called an *interaction*. But, we are not yet finished, since the coordination diagram also imposes some requirements on an Arc that participates in an add or in a remove event. Let us have a look at the requirements for the add event for Arcs: it requires that all[1] Places at the target of the Arc also participate in the add event. Likewise, it requires that all[2] Places at the source of the Arc participate in the remove event, when the Arc participates in a remove event. We will see later in the local behaviour of the Place (Fig. 6), that a Place that participates in an add event will, actually, create and add a new token to itself. Note that the place takes responsibility for creating the new token; this task cannot be delegated to the token itself, because the token does not even exist yet. That is why the co-

[1] Note that for P/T-systems, we know from the structure that there is always exactly one target place; therefore, whether we require one or all does not make a difference here. We will see later that it makes a difference, once we have signal arcs, which can run between two transitions. In order to be consistent with the semantics of SE-nets, which will be discussed in Sect. 4, we use the quantifier ALL already now.

[2] Again, we know that it is exactly one in this case.

ordination diagram does not have any requirement on involved other elements in an add event. When a Place participates in a remove event, however, there is another requirement: one of its Tokens must participate in the remove event too, which is indicated by the coordination annotation "remove->ONE". The token does not have any additional requirement in the coordination diagram, but its local behaviour when participating in a remove event (see Fig. 7) will remove itself from the place – and the local behaviour of the token will guarantee that it can remove itself only once (i. e. it can be used only once in its life-time).

Altogether the coordination diagram from Fig. 2 together with the local behaviour, which will be discussed in Sect. 3.3, guarantee that for a transition that participates in a fire event, all the in-coming arcs, and with them all the places in the preset participate in the remove event; and, for each place, one of its tokens participates in the remove event; likewise, all the out-going arcs, and with them all places in the postset participate in the add event. This combination of elements bound to some events is called a *valid* or *enabled interaction*. Figure 3 shows one example of a Petri net, in which one valid interaction is indicated as an octagon with the involved events. The dashed lines from the events to the different elements indicate which elements are involved, and which elements participate in which event. Note that for transition t2, there is no valid interaction in this situation. But, there would be one other interaction, which instead of the top-left token on place p1 would involve the other (bottom-right) token on place p2 for firing transition t1, which however is not shown in Fig. 3 in order not to clutter the diagram.

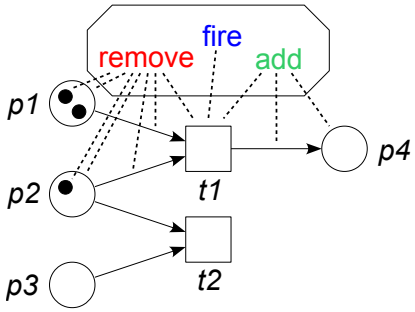


Fig. 3. A Petri net example with an interaction

Note that if such a valid interaction is found, it contains the necessary tokens for firing the transition. And executing all the local actions (see details below) will remove these tokens from every place in the preset and add a new token to every place in the postset of the transition. Each valid interaction corresponds to an enabled transition of the P/T-system, and executing a valid interaction corresponds to atomically firing the transition.

Note that the semantics of ECNO is non-deterministic. In a given situation, any valid interaction could be executed. The actual choice is made outside ECNO by so-called controllers, which are beyond the scope of this paper. But, it is part of ECNO's semantics that, once an interaction is executed, the interaction is executed *atomically* and *in isolation* (without interference of other interactions). Together this exactly reflects the semantics of non-deterministic and atomic firing of Petri nets. In a sense, the existence of exactly two valid interactions in the situation of Fig. 3, reflects what is called the *individual token interpretation* of Petri nets [8]: there is one interaction for each possible choice of tokens for firing the transition.

The ECNO semantics even covers concurrency in Petri nets. Interactions that are independent of each other (i. e. the set of elements participating in the two interactions are disjoint), can be executed concurrently.

3.3 Local behaviour

In the discussion above, we have mentioned already that objects, or *elements* as we call them in ECNO, have a local behaviour. In any given situation, the local behaviour of an element defines in which events it currently could participate in; in some cases, it even defines that it should participate in different events at the same time – as we have seen for transitions in our example. Moreover, the local behaviour of the element defines, what actually happens when the element participates in an event if and when the interaction is executed.

The ECNO does not prescribe any specific notation for the local behaviour. The local behaviour can be defined in different ways. One way of defining the local behaviour is by a simple version of Petri nets, which we call *ECNO nets*. In our example, these ECNO nets are very degenerated nets: most of the transitions have empty presets and postsets, which means that these transitions are always enabled. We discuss the ECNO nets and their meaning below.

Figure 4 shows the behaviour of a Transition. The model of it is an ECNO net with a single transition without any places in its pre- and postset. Therefore, it is always enable. The more interesting part is the annotation (in bold-face letters), which is called an *event binding*: the sequence referring to the three events fire, remove, and add – the order actually does not matter. All three events being in the same transition binding means that, if the local behaviour participates in one of these events, it also needs to participate in the other two events within the same interaction. This way, the local behaviour of the Transition enforces that a fire event always goes together with the add and the remove event.

The local behaviour for an Arc is shown in Fig. 5. The ECNO net consists of two always enabled transitions, which are bound to either the event add or the event remove. This, basically, says that an Arc can always participate in an add or a remove event.

The ECNO net for the local behaviour of a Place is shown in Fig. 6. It is similar to the local behaviour of an Arc. The two transitions with event binding add resp. remove tell that these events are always possible. But, for the transition bound to event add, there is another annotation (in normal font), which defines

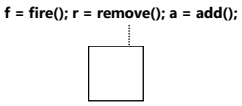


Fig. 4. Local behaviour of a Transition

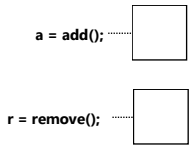


Fig. 5. Local behaviour of a Arc

an *action*. This action is executed if, in some interaction, the Place participates in an add event and when the resp. interaction is executed. The action is defined by a Java code snippet that creates a new token and adds it to the list of the place's tokens (`self.getTokens()`). This Java code uses the API and code that was generated by the Eclipse Modeling Framework (EMF) [9] from the model of Fig. 1. In order to be able to access the EMF generated factory for creating new tokens, the ECNO net uses two other extensions: an import statement and a declaration of an attribute (the constant for the factory, in this case). These details, however, are not relevant here. What is relevant is that this action is one of the two actions, that ultimately makes something happen, when an interaction is executed: adding a token to a place participating in an add event.

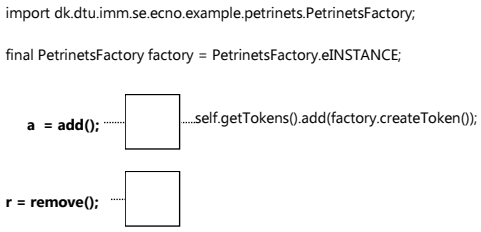


Fig. 6. Local behaviour of a Place

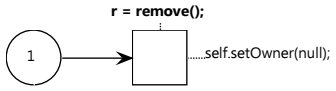


Fig. 7. Local behaviour of a Token

The other local behaviour in which something happens is the behaviour of the Token. The ECNO net for the local behaviour of the Token is shown in Fig. 7. This time, the transition of the ECNO net has a place in its preset, which has one token initially. Therefore, this transition is initially enabled; after it has fired once, however, it will not be enabled anymore. Since the transition is bound to a remove event, this models the fact that a token can initially participate in a remove event; but after a remove event has occurred, the token can never participate in a remove event again. This way, every token can be consumed only once. The action attached to this transition, is the code that actually removes the token from the place. Again, the Java code snippet defining the action makes use of the EMF generated API: setting the tokens' owner to `null`, thereby removing itself (`self`) from its place.

3.4 Variations and details

The definition of the semantics of P/T-systems by ECNO coordination diagrams and ECNO nets has some subtle issues, which we did not explain yet. Some of these subtleties actually concern variation points of the semantics of Petri nets. In this section, we discuss some of these variation points, in order to explain some more details of the concepts behind the ECNO.

Loops First, let us have a look at P/T-systems with *loops*, i. e. with a pair of arcs that connect the same place and transition in opposite directions. Figure 8 shows a Petri net with a loop along with a valid interaction. Note that the

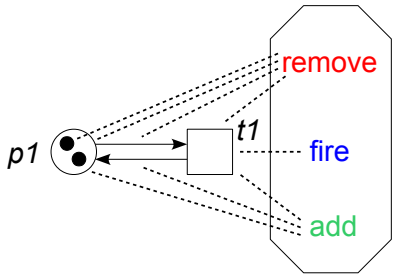


Fig. 8. A Petri net example with a loop

two arcs to and from the transition are required to participate in the event add or the event remove, respectively – as before. The difference now is that these two arcs "end up" at the same place p1, which means that the same place p1 must participate in the add as well as in the remove event. The same element participating in different events within the same interaction, is actually not new. We had seen that before in the local behaviour of the Transition – enforced by a specific event binding of the ECNO net (see Fig. 4). The loop example, however, is different. Two different arcs were requiring two different events to participate for the same place. Now, the question is whether the local behaviour of a Place would be able to participate in the event add as well as in the event remove. A look at the ECNO net for the local behaviour of a Place (see Fig. 6) shows that there is no transition, that is bound to both events at the same time. This might suggest that the local behaviour of a place would not be able to participate in both events. But actually, the two transitions of the ECNO net are completely independent of each other and, therefore, they can be executed in parallel to each other. Therefore, the interaction shown in Fig. 8 is valid – executing both events add and remove together (concurrently) is possible in the local behaviour of the place. This shows that it is not fully by coincidence that we have chosen an extension of Petri nets for defining the local behaviour of elements: one reason for using them is that Petri nets naturally come with a notion of independent or

concurrent execution of transitions. This, makes modelling the local behaviour much easier in many cases.

Now, what would we need to do if we would not want transitions with loops to fire – Elementary Net Systems (EN systems) [10], for example, have this characteristics. In that case, the ECNO net for the local behaviour of the place would have an initially marked place, and both transitions (the one for the add as well as the one for the remove event), would have a loop to this place[3]. This would imply that a place cannot add a token and remove a token within the same interaction – transitions with loops would not be able to fire.

Arc weights (multiple arcs) In our discussion of the domain model for P/T-systems, we suggested that our P/T-system are *ordinary*, which means that all arc weights are equal to 1. But, our domain model for P/T-systems from Fig. 1 allows multiple arcs between the same element in the same direction. This way, an arc with a weight greater than 1 can be represented by multiple arcs between the respective nodes.

Now, the question is what does the ECNO semantics say in this case. To this end, we have a look at another example, which is shown in Fig. 9. It is a P/T-system with two arcs running from place p1 to transition t1. The figure also

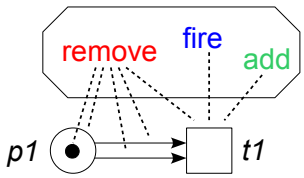


Fig. 9. A Petri net example with double arc

shows an interaction with the bindings of the events to the different elements. If it was not for a feature of ECNO, which we did not explain yet, this would be a valid interaction. For the two arcs bound to the remove event the coordination diagram from Fig. 1 requires that the place is also bound to a remove event; in turn, the requirement for the place says that also one of its token must be bound to a remove event – which is true. Therefore, without additional concepts, the transition with two incoming arcs from the same place could fire – even though there is only one token on the place. In the same way, multiple out-going arcs to the same transition would produce only one token. Whether this is what we want or not is a matter of taste. For ordinary P/T-systems, where all arc-weights are supposed to be one, this semantics would probably be okay, though it would

[3] ECNO nets have P/T-systems semantics; i. e. transitions with a loop can fire, if the respective place has a token – but two transitions connected with a loop to the same place cannot fire concurrently if there is only one token.

be better style if multiple arcs between the same place and transitions would be forbidden syntactically (e. g. by adding some OCL constraints).

For the sake of an example, let us now assume that we want to enforce that multiple arcs between the same place and transition should consume or produce the same amount of tokens on the respective place. How would we achieve that in the ECNO? Actually, it is in the model already, we just ignored this feature up to now. A closer look at Fig. 2 reveals that the labels of the event remove in the coordination sets in the Place and in the Token are shown between two plus signs, which is the key to dealing with multiplicities. This notation indicates that the event remove is treated as a *parallel trigger* or a *counting* event for a Place and for a Token. Let us explain, what that means by the help Fig. 9: both Arcs are bound to a remove event; and according to the coordination diagram of Fig. 2, each of these bindings requires that the source (i. e. place p1) is also bound to the remove event. Since remove is a counting event, this means that the Place must be bound to the remove event twice. The local behaviour of the Place does allow the respective event to occur twice. So, this would be possible. Now, the place is bound to remove twice. This means that the remove event needs to be bound the same number of times to some Token. This could either be two different tokens – if there were more than two tokens on the place – or two bindings to the same token. Since the remove event is also a counting event for tokens, this would mean that the same token would be bound twice to a remove event. But, the local behaviour for tokens does not allow the transition with the remove event to occur twice in parallel to itself (see Fig. 7); actually, it can occur only once in its entire life-time. Therefore, making remove counting events for Place and Token, guarantees that the multiplicity of arcs is properly taken into account. And the interaction shown in Fig. 9 is not valid.

Therefore the semantics of P/T-systems from the coordination diagram of Fig. 2 together with the ECNO nets from Figures 4–7 is the semantics of P/T-systems which properly takes the multiplicity of arcs (arc weights) into account.

3.5 Discussion

Altogether, we have defined the semantics of P/T-systems by ECNO coordination diagrams and ECNO nets. The semantics of ECNO and its coordination diagrams was discussed in other papers [1–3] – though not formalized in mathematics yet. The semantics of the local behaviour (ECNO nets) is basically the semantics of Petri nets (P/T-systems to be precise). Now, the attentive reader might be alert: we have defined the semantics of P/T-systems by something which refers to the semantics of P/T-systems again. A clear definition of a semantics should not have such cyclic dependencies. But, we argue our way out of that problem. First of all, the point of this paper is not to define the semantics of P/T-systems – the point of this paper, is to illustrate the way of how the semantic of Petri nets can be expressed using the concepts of ECNO. Secondly, as pointed out earlier – ECNO provides different ways of defining local behaviour (one of them would be simply programming it). And the ECNO nets that we used for modelling the local behaviour are very basic: synchronize some events,

execute some event – both of which are always possible and can be done concurrently to themselves; and an even more basic behaviour, one event may be executed exactly once. We could have introduced a dedicated notation for that – but we deemed that using Petri nets for the audience of this paper would be more readable. For this paper, we did not deem it worth the while to define such an ad hoc notation.

4 Firing rule of signal-event nets

In this section, we will present an ECNO semantics for another variant of Petri nets, which are called *signal-event nets* (*SE-nets*) [5]. What is more, it only takes a minor twist, to capture the essential idea of *signal arcs*.

A signal arc is an arc running from a transition to a transition. Since our domain model from Fig. 1 already caters for that, we use it also for SE-nets. The meaning of a signal arc is the following: suppose a transition t_1 can fire in some SE-net, and there is a signal arc from transition t_1 to a transition t_2. Then, if transition t_2 can fire in that situation, transition t_2 must fire together with transition t_1. If transition t_2, however, is not enabled, transition t_1 can fire alone. Since transitions now fire together, the kind of semantics used for SE-nets is called a step-semantics, but we do not go into details here. And since there can be chains of transitions connected with signal arcs, it can happen that more than two transitions must fire together due to transitive dependencies.

Figure 10 shows the coordination diagram that captures the semantics of signal arcs. Most of it – in particular the part for Place and Token – is identical to the coordination diagram for the behaviour of P/T-systems (Fig. 2). But,

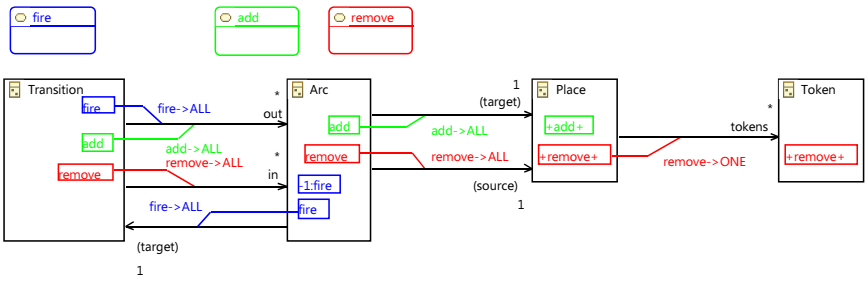


Fig. 10. ECNO coordination diagram for SE-nets

there are now also some coordination annotations concerning the fire event, and the arc is used to "propagate fire events" between transitions. Let us have a closer look at the coordination annotations for the fire event starting from the Transition. The reference from the Transition to its out-going Arcs annotated by "fire->ALL" makes sure that every out-going arc participates in the fire event too (but the arc itself does not take any action itself). If an Arc is involved in a fire

event, the annotation "fire->ALL" at the target reference to the Transition shows that all target Transitions need to participate in the fire event too (unless this is not possible, which will be explained shortly). Note, that an arc could either be a regular Petri net arc (in that case, it would run from a transition to a place) or a signal arc. If it is a regular arc, there would actually not be a transition at the other end, i. e. the requirement that all Transitions at the other end need to participate in fire is trivially valid. Note that the type of the Transition at the other end of the target reference, implicitly reduces the set of elements to be considered – Places are not considered under the ALL-quantification in this case. In order to point out that there is an implicit restriction of possible target elements by the type at the other end, the name of the reference target is shown in parentheses. Actually, the same applied to the source and target references to the Place in the coordination diagram for P/T-systems already – at that time, however, it did not have any bearing. Now it has.

Altogether, the two coordination annotations for fire guarantee that, if a Transition participates in a fire event, all the Transitions to which there are signal arcs will also participate in an fire event. This covers the first case of the semantics of a signal arc. But what, if the transition at the end of the signal arc cannot fire. In that case, it should be ignored. This is actually represented by the coordination set fire of the Arc, which is not connected with any out-going reference – therefore, the Arc can chose not to "propagate" the fire event to the target Transition. The semantics of ECNO is that, if there is more than one coordination set for the same event, any of them could be chosen to follow up alternatively. So, the unconnected coordination for fire takes care of the second case of the semantics of signal arcs: not firing the target transition. In general, the choice between different coordination sets for the same event is non-deterministic. This would mean that the signal arc could chose to involve the target transition or not. This, however, is not exactly the semantics of SE-nets: they require that the target transition must fire, if it is enabled. This is where priorities of coordination sets come into play. The empty coordination set for event fire of Arc has priority -1, which is indicated by the number in front of the colon. All other coordination sets have default priority 0. This way, the target transition must participate in the fire event if possible, since the coordination set propagating the fire event has higher priority.

Basically, the coordination diagram from Fig. 10 captures the semantics of signal arcs. As it is formalized now, it would, however, be possible that a transition with incoming signal arcs fires without one of the source transitions firing. We could change the coordination diagram to cater for that – basically, by introducing requirements in the reverse direction. But, there is a simpler practical solution here. As explained earlier, the actual control of events and interactions for them being issued for elements lies in so-called controllers. If we attach controllers for the fire event only to those transitions that do not have in-coming signal arcs, this has exactly the effect we want in signal-event nets: a transition with in-coming signal arcs, can be executed only by being triggered from other transitions.

Altogether, this shows that with a minor twist of the original ECNO semantics of P/T-systems, we can obtain the semantics of SE-nets. What is more, the extension seems to be a quite natural translation of the two possible cases of the informal semantics of signal arcs.

5 Conclusion

In this paper, we have shown how to formulate the semantics of different variants of Petri nets by the help of the Event Coordination Notation (ECNO). We did not dive into the details of the semantics of ECNO, but rather used the examples to illustrate some of the features of ECNO. This way, we continued the narrative on Model-based Software Engineering and an example of a simple Petri net tool [7], which covers the behaviour of the domain now: the firing rule of Petri nets in our example.

ECNO actually was not specifically made for that purpose and has many more features, which we did not need for formalizing the semantics of Petri nets, such as parameters for events, inheritance on events, and inheritance on elements and their local behaviour. The selection of the concepts and features of ECNO were driven by applications like AMFIBIA [11] and the vision, that it should be able to formulate the semantics of ECNO in its own concept – a report on the ECNO formulation of its own semantics is in preparation. Formulating the semantics of some versions of Petri nets was a finger exercise on that way, which gives some insights into the concepts of ECNO and its modelling philosophy. Moreover, this exercise shows that ECNO allows to concisely formulate the semantics of Petri nets.

ECNO Tool support ECNO is implemented as an extension of Eclipse. The current version of the implementation of ECNO (0.3.0) supports ECNO coordination diagrams on top of Ecore diagrams (a kind of class diagrams) and ECNO nets for modelling the local behaviour. From these diagrams, the ECNO Tool can generate program code. The ECNO Tool includes an ECNO execution engine and runtime environment, which is able to execute the code generated from the ECNO models. The information on how to obtain and install this ECNO Tool, as well the examples discussed in this paper can be found on the ECNO home page at

`http://www2.imm.dtu.dk/~eki/projects/ECNO/`

References

1. Kindler, E.: Modelling local and global behaviour: Petri nets and event coordination. In Duvigneau, M., Moldt, D., Hiraishi, K., eds.: Petri Nets and Software Engineering. International Workshop PNSE'11, Newcastle upon Tyne, UK, June 2011. Proceedings. Volume 723 of CEUR Workshop Proceedings. (2011) 42–56

2. Kindler, E.: Integrating behaviour in software models: An event coordination notation – concepts and prototype. In: Third Workshop on Behavioural Modelling - Foundations and Application (BM-2011), Proceedings. (2011)
3. Kindler, E.: Modelling local and global behaviour: Petri nets and event coordination. Transactions on Petri Nets and Other Models of Concurrency (**6**) to appear in Springer LNCS series.
4. Reisig, W.: Place/Transition systems. In Brauer, W., Reisig, W., Rozenberg, G., eds.: Petri Nets: Central Models and Their Properties. Volume 254 of LNCS., Springer-Verlag (1987) 117–141
5. Starke, P.H., Hanisch, H.M.: Analysis of signal/event nets. In: Emerging Technologies and Factory Automation (ETFA '97), Proceedings, 6^{th} International Conference on, IEEE (1997) 253–257
6. Hanisch, H.M., Lüder, A.: A signal extension for Petri nets and its use in controller design. In Burkhard, H.D., Czaja, L., Starke, P., eds.: Proceedings of the CS&P'98 Workshop. Number 110 in Informatik-Bericht, Berlin, Germany, Humboldt-Universität zu Berlin (1998) 98–105
7. Kindler, E.: Model-based software engineering and process-aware information systems. In Jensen, K., van der Aalst, W., eds.: Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems. Volume 5460 of LNCS. Springer-Verlag (2009) 27–45
8. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures. In: Logic in Computer Science (LICS), Proceedings, 10^{th} Annual IEEE Symposium on. (1995) 199–209
9. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. 2nd edition edn. The Eclipse Series. Addison-Wesley (2006)
10. Thiagarajan, P.: Elementary net systems. In Brauer, W., Reisig, W., Rozenberg, G., eds.: Petri Nets: Central Models and Their Properties. Volume 254 of LNCS., Springer-Verlag (1987) 26–59
11. Axenath, B., Kindler, E., Rubin, V.: AMFIBIA: A meta-model for the integration of business process modelling aspects. International Journal on Business Process Integration and Management **2**(2) (2007) 120–131