Introduction to Petri Nets and Reference Nets

Olaf Kummer
Universität Hamburg, Fachbereich Informatik
Vogt-Kölln-Straße 30, D-22527 Hamburg
kummer@informatik.uni-hamburg.de

Abstract

Petri nets are a well-established means for the description of concurrent systems. Several variants of Petri net formalisms are presented and explained. The main motivation is to provide a basic intuition of the concepts, so that other papers on Petri nets are easier to read.

1 Introduction

After providing some historical and motivational background information, this paper introduces the terminology and the basic effects of Petri nets, both P/T-nets and high-level nets. Afterwards, reference nets are introduced as an extended variant of Petri nets. In all presentations, we will refrain from mathematical notations. The availability of computer tools for Petri nets is emphasized.

2 Petri's Program

The research area of Petri nets dates back to Carl Adam Petri's thesis [6], where he set out to develop a description technique for computational tasks that is firmly based on and realizable with elementary physical phenomena. It was in the subsequent years refined by both Petri and his scholars, who worked on the theoretical foundations, but also on extensions and possible applications of Petri nets.

Already starting with the original thesis, the applications of nets included not only automated procedures, but also human actions and communication. Petri's work is influenced by universal dichotomies: cause and effect, state and change, roles and activities, open and closed, possible and actual. He strives for an integrated and uniform method.

In [7] Petri describes the twelve so-called *communication disciplines* that indicate the problems targeted by his approach: synchronization, identification, copying, addressing, naming, cancellation, composition, modelling, authorization, valuation, delegation, and reorganization.

In [8] a list of interdisciplinary connections of net theory is given, which comprises among others mathematics, physics, engineering, administration, law, pragmatics, and computer science. In the very context of computer science, Petri regards concurrency as the fundamental level and interests of groups and individuals as the highest level in a long sequence of abstractions. He can thus be considered one of the earliest precursors of socionics, if not in terminology, then in spirit.

Many of the items of Petri's program have yet to be tackled, but already now, Petri nets can support the investigation and creation of complex systems.

3 P/T-Nets

Place/Transition nets (P/T-nets for short) constitute a net formalism that is already well above Petri's lowest conceptual level of concurrency. They allow an intuitive representation of the notions of causality, alternatives, parallelism, resource, action, and effect.

Figure 1: Different places: unmarked, marked, multiply marked

A *net* is assembled from places and transitions. *Places* represent resources that can be available or not. Places are depicted in diagrams as circles or ellipses. The availability of a resource if shown by a black dot inside the circle. If more than one resource of a given type is available, more than one dot will be present. When resources are available, we also say that the place is *marked*. Individual resources are abstractly referred to as *tokens*. See Figure 1 for the graphical representation of places and tokens.

Conditions that might be satisfied in a system can also be shown as places, typically carrying no more than one token at a time. The distribution of tokens among the places constitutes the *marking* of the net. A marking represents the current state of a system. Figure 2 shows a very simple transition with two conditions.

Figure 2: A simple transition before and after firing

Transitions are the active elements of a net. Rectangles or squares denote transitions in net diagrams. A transition that *occurs* (or *fires*) can remove tokens from some places and insert tokens into other places. In order to denote the tokens that are moved by a transition, arrows, so-called *arcs*, are drawn from places to transitions and from transitions to places. Each arrow can be annotated by a number that indicates the number of tokens that are moved by this arc.

We will now look at nets that contain many transitions. Figure 3 shows four nets with four elementary patterns that may appear in Petri nets. The topmost net must be executed in a *sequence*, one transition after the other. The second net shows *synchronization* where one transition merges two otherwise unrelated branches of execution. Synchronization is enforced, because a transition can only fire if all its input places are marked. The middle transitions of the third net are in *conflict*, because they require a common resource and cannot both fire at the same time. In such a situation, it is undetermined which transition fires. In the fourth net, *concurrency* is introduced by a transition that spawns two sequences of transitions. The middle transitions may fire simultaneously or in any order.

Figure 4 exemplifies a firing of a transition that involves multiple tokens. In a chemical reaction, two molecules of hydrogen and one molecule of oxygen are processed into two molecules of water, i.e., $2\,H_2 + O_2 \rightarrow 2\,H_2O$. One molecule of hydrogen remains in the system.

Figure 5 provides a more complex net for the running example of this paper. Here we represent resources (money that is required to buy a ticket to commute to work) and

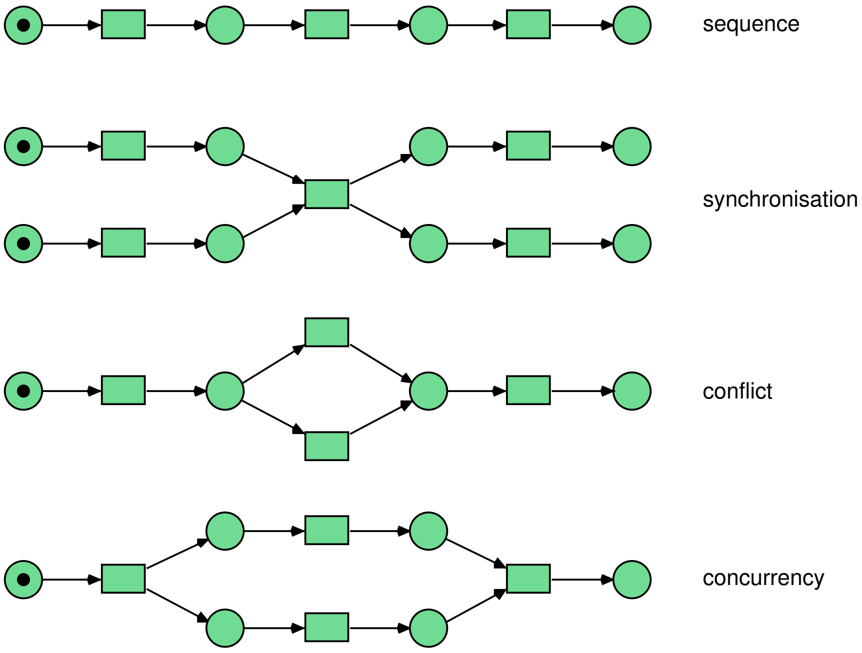

Figure 3: Elementary situations in Petri nets

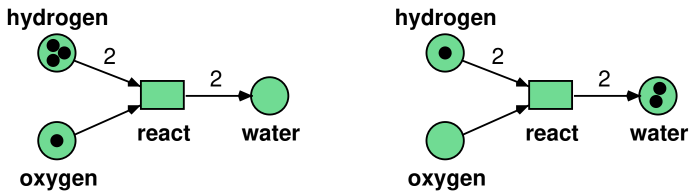

Figure 4: A chemical reaction

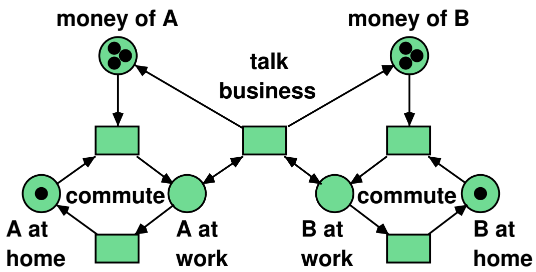

Figure 5: The life of two business persons

conditions (whether a person is at home or at work) in one net. The transition `talk business` allows both parties to earn new money by making a good deal. We observe the *double arcs* that connect the transition with the `at work` places. Such arcs remove one token, but reinsert it immediately afterwards. Further transition allow the persons to commute to work and back home.

4 High-Level Nets

It is apparent that the subnets for the two business persons shown in Figure 5 are almost identical. It will simplify the net, if represent the joint structure only once, but we have to keep track of the different state of both sides regarding money and position.

High-level Petri nets as described in [4] tackle this problem. Here tokens are no longer unstructured black dots, but can represent arbitrarily complex data. In the simplest case, a data item is moved around unchanged. In Figure 6 we can see the commuting part of the business person example.

Figure 6: A high-level net before and after firing

Here, the two persons A and B are initially located at two different positions, but there are two transitions that can move tokens from the left to the right and vice versa. Both arcs of the upper transition are inscribed with the variable `x`. During the firing of a transition, every variable must be bound to a fixed value and the value of the variable inscribed to an arc determines which token value is moved by this arc.

After the upper transition fires with `x="A"`, both tokens are located in the right place. Now only the lower transition can fire. It uses the variable `y` for clarity, but since variables are strictly local to a transition, we might have reused `x`. Two firings of the lower transition move both tokens to the left. The two firings require different bindings of the variable `y`, which is allowed.

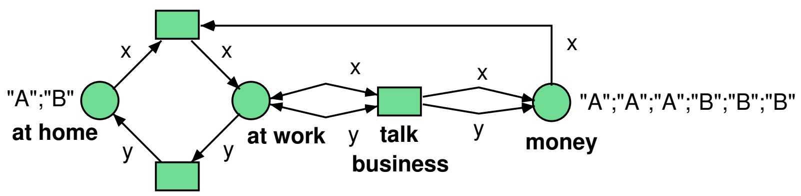

Figure 7: The joint life of two business persons

Figure 7 extends the previous example and shows the handling of money in addition to the movement of the business people. Here, a token `"A"` in the place `money` indicates that one monetary unit is owned by person A. Note that it would be very easy to add further persons to this net. In Figure 5 it would have been quite tedious to introduce more than a few persons.

We described the persons' bank accounts as collections of tokens: one token for each monetary unit. This can be inconvenient. It would be more natural to denote a bank account by a pair of name and current balance. This is done in Figure 8. In the sequel, we will show how a similar net can be used within the running example.

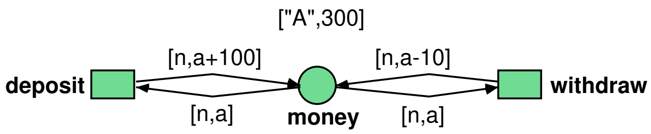

Figure 8: A simple bank account

The `withdraw` transition subtracts `10` units from the balance, whereas the `deposit` transition deposits `100` units. In general, all sorts of mathematical and other precisely defined operations are possible for the manipulation of tokens. The tokens may belong to arbitrary domains, e.g. pairs of character strings and numbers as show here.

5 Channels

A valid objection against the high-level business person net would be that it mixes the movement of money and the movement of persons too much. At first sight, it might not be obvious which parts of the net belong to which aspect of the system.

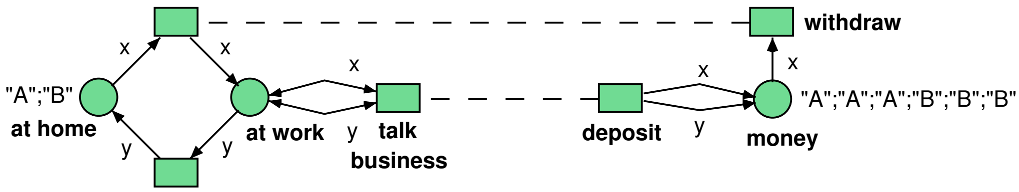

Figure 9: Splitting persons and accounts

In Figure 9 we prepare a split of the system that was last shown in Figure 7. Two transitions were drawn twice in different parts of the net diagram. In order to fire the transitions correctly, they would have to be merged again.

We go one step further and indicate the intended meaning of the synchronization using a textual annotation. In Figure 10 the annotation `this:withdraw(x)` means that in *this* net there should be another transition that can handle the withdrawal of money from `x`'s account. On the other hand, `:withdraw(x)` designates the transition that can perform this task. We provided the variables at the end of the annotations to make clear which information needs to be passed around.

Such annotations are called synchronous channels, because they force the transitions to act synchronously and because they channel the flow of information. The authors of [2] introduced this concept to high-level Petri nets.

Because we used textual annotations for the channels, we can request many synchronizations at once while keeping the diagram readable. See Figure 11, where we have restructured

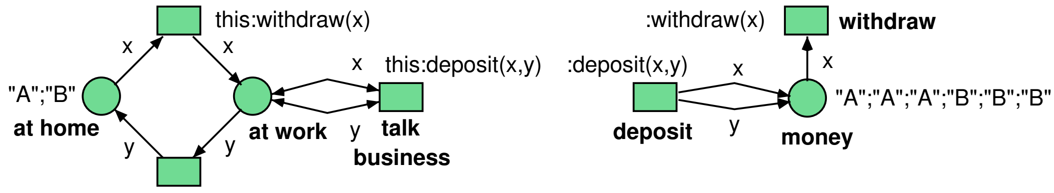

Figure 10: Using textual annotations for synchronization

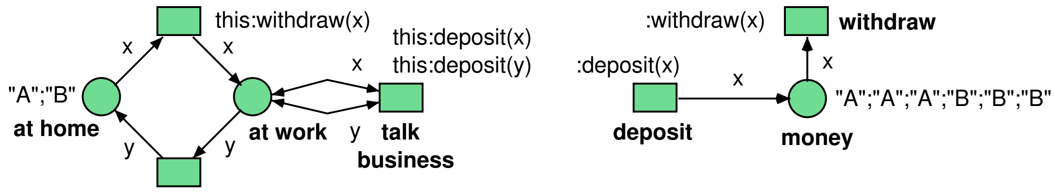

Figure 11: Reusing a channel

the net from Figure 10 in such a way that the transition `talk business` contains two invocations of the synchronous channel `deposit`. That causes the transition `deposit` to be executed twice, which is the intended behaviour in our example.

We can now combine the solution from Figure 11 with the high-level model of a bank account and represent each account as a single pair token. Figure 12 shows the solution. Notice how we pass the number of monetary units to be deposited or withdrawn through a second parameter of the channel.

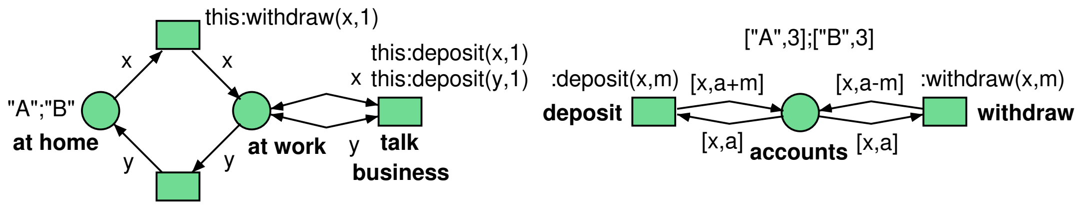

Figure 12: Using arithmetic for bookkeeping

Note that channels add to ordinary Petri nets the ability to talk about simultaneous actions, not only about sequential or concurrent actions. They still require us to think about the communication discipline of synchronization. They do not solve the associated problems by themselves, but they allow the net designer to abstract from the underlying communication in a clean way.

6 Net Instances

For classical Petri nets, the places and transitions that are drawn exist exactly once during the execution of the net. We may generalize and allow multiple *instances* of each net. So that different instances can be distinguished and processed, we allow *references* to net instances as tokens in nets. This leads us to the *reference net* formalism.

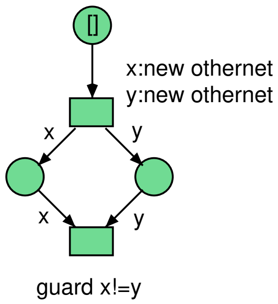

Figure 13: The main net `creator`

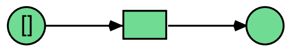

Figure 14: The net `othernet`

In Figs. 13 and 14 we provide a simple example of net references. The main net creates two instances of the net named `othernet` and stores the references in two different places. Ultimately, a transition checks whether the created nets are unequal, which is always the case. By the way, you can see that the black token is represented as an empty list `[]` in reference nets.

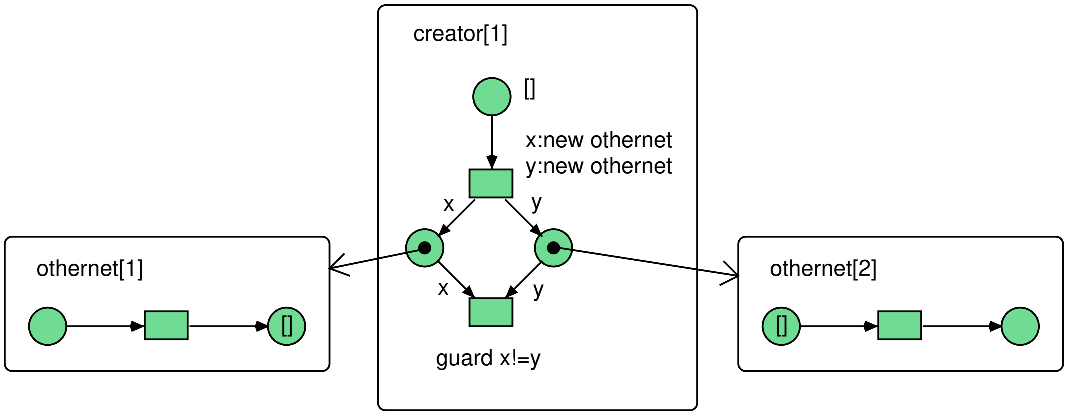

Figure 15: Net instances containing reference tokens

In Figure 15 you can see a snapshot during the execution of the example from Figs. 13 and 14. You can see that the main net has already created the two other nets and that one of these has already performed a firing that moved a token to its right place. After the second transition of the main net fires, no more references to the other nets exist, but the nets themselves are not destroyed. There can still be local transitions that are allowed to fire, e.g. in the rightmost net.

Returning to our running example, it is already clear that persons and bank accounts are pretty much independent of each other and only connected via channels. It would thus make sense to split the entire net into two. So that the persons can contact their accounts, they need to know about the other net, however. Nets that are referencing nets are supported by the reference net formalism, which extends high-level Petri nets.

Figure 16 describes the new net for the two persons. It contains an explicit setup of the bank accounts of both persons. The initialization transition on the right puts a reference to the created net into the place `bank`. Net references can be used just like every other data item in the net once they are produced, but being active objects, their creation differs from other operations.

The synchronous channels can no longer target the same net (`this`), but they have to target the account net that is responsible for the person in question. This is only possible if the transitions access one of the tokens in the place `accounts`.

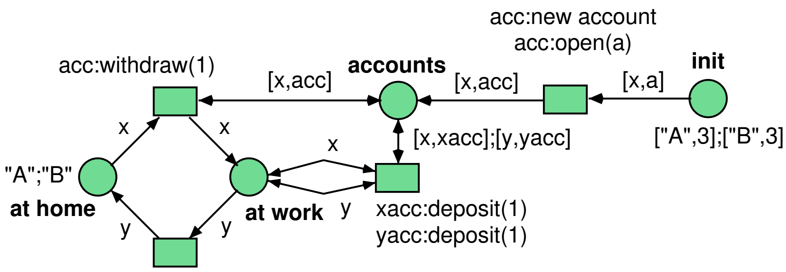

Figure 16: The person net

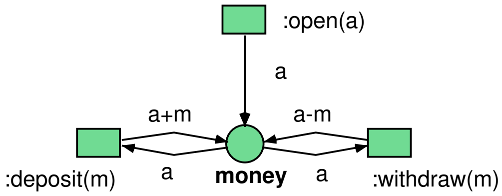

Figure 17: The account net

Note that we have progressed to a state where we might add plenty of details to both the persons and the bank accounts without many problems. Note that the ability to split nets and to introduce net references can help to keep individual nets simple and clean.

We might go further to introduce one net instance for each individual person. In that case, we would have more than one instance of a net, which is allowed. It would also be allowed to introduced cyclic references when, e.g., a persons references an account, but the account references its owner in turn.

Net instances are an important concept, allowing us to reason about the *identity* of an object of the real world, whereas ordinary high-level nets merely talk about *equality* of values. They provide a new level of dynamics where a system that is represented by mutually referencing net instances is allowed to change as the result of an action which adds, moves, or removes net references.

Without net references, it would merely be possible to change the state of system components, but not to re-organize the system entirely. As you can see, the communication disciplines of identification, composition, delegation, and re-organization come into play when dealing with net instances and net references.

Reference nets owe much to the object-based programming paradigm found in computer science. An *object* is characterized by its identity, by its encapsulated state and by the operations (there called methods) that it supports for external access. How the well-known concept of *inheritance of behaviour* should be transferred from object-oriented programming to Petri nets is still subject to debate.

7 Tool Support

All nets of this paper have been generated with the Petri net tool Renew [5], whose main window is depicted in Figure 18. As indicated in the introduction, Petri imagined physically realizable, i.e. ultimately executable, models. Executability cannot replace fundamental insights into a system, as also emphasized by Petri, but it is a necessary precondition for precise models of the real world.

Renew [5] is used for the execution of the nets. A fixed annotation language has to be chosen before implementing a tool. In our case it was the programming language Java [3], which offers many features that nicely complement Petri nets and which is widely available.

Figure 18: The toolbar of the Renew application

Execution is also referred to as simulation in the Petri net literature. This is meant as 'simulating the abstract mathematical definitions in the real world' and not as 'simulating the real world'. The latter can be done, of course, but an executing net may actually be part of the real world, not just simulating it.

Besides ordinary reference nets, Renew supports models with simulated time and with specialized arcs between places and transitions that may for example move multiple tokens at once. The user is aided in the creation of nets by specialized tools and by an online syntax check. Execution states may be saved and restored and breakpoints may be set to stop a net's execution at a predefined point of time. The execution of a net may utilize elementary visualization support to convey the current system state to the user.

8 Conclusion

In the previous sections we discussed the basic concepts of both high-level and low-level Petri nets. Basic phenomena and some modelling practices were hinted at in passing. The interested reader may find further information about various Petri net types in the monographs [1], [4], and [10] or at the WWW sites [5] and [9].

The many reasons why Petri nets match nicely with socionics should have become obvious by now. Petri nets allow to model human behaviour and to specify automated behaviour. In the extended variants, they may represent resources, data, communication, identity, structure, distribution, movement, behaviour, conflicts, and concurrency. In the simple variants they rely solely on the concepts of active and passive objects.

References

[1] Falko Bause and Pieter Kritzinger. *Stochastic Petri Nets – An Introduction to the Theory.* Advanced Studies in Computer Science. Vieweg Verlagsgesellschaft, 1996.

[2] Søren Christensen and Niels Damgaard Hansen. Coloured Petri Nets Extended with Channels for Synchronous Communication. Technical Report DAIMI PB–390, Aarhus University, 1992.

[3] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification.* Addison Wesley, 1997.

[4] Kurt Jensen. Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use. *EATCS Monographs on Theoretical Computer Science*, 1992, 1994, 1997. Three volumes.

[5] Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – The Reference Net Workshop. WWW page at `http://renew.de/`. Contains the documentation of Renew and a more technical introduction to reference nets.

[6] Carl Adam Petri. Kommunikation mit Automaten. Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, 1962.

[7] Carl Adam Petri. Communication Disciplines. In B. Shaw, editor, *Computing System Design: Proc. of the Joint IBM University of Newcastle upon Tyne Seminar, Sep., 1976*, pages 171–183. University of Newcastle upon Tyne, 1977.

[8] Carl Adam Petri. Introduction to General Net Theory. In W. Brauer, editor, *Lecture Notes in Computer Science: Net Theory and Applications, Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979*, volume 84, pages 1–19, Berlin, Heidelberg, New York, 1980. Springer-Verlag.

[9] Petri Nets World. WWW page at `http://www.daimi.au.dk/PetriNets/`. The central source for all information regarding Petri nets.

[10] Wolfgang Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

I wonder where they get those tokens.
Walt Whitman, 'Leaves of Grass'