Contents

1 Introduction ................................................................. 1

2 Basics ................................................................. 2
2.1 Basic definitions of Petri Net Theory. ......................... 2
2.2 Condition-Event-Nets. ............................................. 6

3 Place-Transitions-Nets. ................................................. 12

4 Application of Petri Nets for the Modelling. .................... 20
4.1 Methodology for the Modelling with Petri Nets .......... 20
4.2 Examples for the Modelling with Petri Nets ............... 22
4.2.1 Sender-Receiver-Model .................................. 22
4.2.2 Five Philosophers Dining Problem ................. 27
4.2.3 Farmer Wolf Goat Cabbage Problem .............. 33
4.2.4 Transformation of a Program Flow Chart into a Petri Net 36
4.2.5 Modelling of Production Schemata .................. 39
4.2.6 Representation of non-procedural Circumstances with Predicate-Rule-Nets 43
4.3 System Design using Petri Nets ............................... 46
4.3.1 Example ..................................................... 46
4.3.2 Refinement and Embedding ............................... 49
4.3.3 Design and Schedule .................................. 51
4.3.4 Consistency of a Design ............................... 53
4.3.5 Completeness of a Design .............................. 55

5 Methods of Petri Net Analysis ......................................... 60
5.1 Safety / Boundedness / Freeness of Contact. ............. 60
5.2 Reachability / Coverability. .................................. 62
5.3 Markable Places and Place Sets. ............................. 64
5.4 Liveness of Transitions, Markings, Places and Nets. .... 64
5.5 Deadlocks / Livelocks ........................................ 66
5.6 Reversibility / Reproducibility ............................. 67
Literature


1 Introduction

In our time it is seldom that a theory is given the name of its initiator (originator). But the theory of Petri nets is such a case. The father of this theory is Carl Adam Petri (http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri.html) with his dissertation “Kommunikation mit Automaten (Communication with Automata), Diss. A Bonn, 1962 (1966 appears translation in USA). The thesis developed a mathematical concept to describe systems with distributed states. This theory was temporarily forgotten. But with the development of complex concurrent systems it was detected that all this can be described and analysed with this theory. The second important paper of Carl Adam Petri to enhance his theory was the article “Grundsätzliches zur Beschreibung diskreter Prozesse” (Fundamentals (Basics) for description of discrete processes), 1967.

The concept of Petri Nets and Net Theory is a result of searching methods for describing and analysing information and control data flow in information (data) processing. The Petri Net Theory is well suited to describe the dynamic behaviour of complex concurrent systems based on graph theoretical concepts, and it is possible to analyse properties of such systems. This theory is simple, modular, flexible, clear and easy to learn.

Petri Nets are appropriate
- for visualisation of workflows which are working non sequential,
- to demonstrate explicitly causal dependencies and independencies in a set of events,
- for modelling data banks, real time operating systems, manufactoring, systems, and computer networks,
- to illustrate the behavior of a system on different levels of abstraction without changing, tools and methods (e.g.: from the modification of a single bit to the embedding of a computer system into its user interface).

In addition to the first properties of your system using simulation of the net model you can get with the help of Petri Net theory the verification of correctness of your design, boundedness, reproducibility, liveness, reachability or non-reachability of system states, invariants, and deadlocks.

The main domain of Petri Nets is the modelling of systems where the system events occur causal independent (concurrent), but the occurrence is dependent on some constraints (e.g. availability of common used resources).

Numerous software tools for Petri Nets exist. (Here we are working with Netedit (W. Nauber, TU Dresden) and INA (P. Starke, Humboldt-Universität zu Berlin).
2 Basics

2.1 Basic definitions of Petri Net Theory

The net model of a system consists of a set of **passive elements** which is assigned to state parameters (components of states, system conditions, materials, statements). These elements are called **conditions** (germ. *Bedingungen*) or **places** (germ. *Plätze, Stellen*). Graphical symbol for a condition or a place is a **cycle**. Fulfilling a condition respectively the availability of a resource with a fixed number is expressed by marking the condition or place with thick black dots (**tokens**, germ. *tokenn*). The tokens describe the state of the net.

Dual to passive elements there are **active elements** which are assigned to activities of the system (events, transitions, production steps, proofs). They are called **events** (germ. *Ereignisse*) or **transitions** and the graphical representation is a **rectangle** or thick **bar**. The activities in a system have mostly local character. We need for this some resources or it must be fulfilled some conditions, and after then new resources are created or new conditions hold. These causal dependencies are described in our net model by a **flow relation** and graphically represented by **directed arcs**. An arc connects a condition (place) with an event (transition) and vice versa. Never exist an arc between two elements of the same type.

If an arc goes from a condition to an event, then this means that the condition is necessary for the occurrence of the event. In this case we say that we have a **pre-condition**. Analogously an arc from an event to a condition means that the occurrence of the event makes this condition true, and it is called a **post-condition**. If we connect places with transitions and vice versa than we speak about **pre-places** and **post-places**.

We want now to illustrate our knowledge with an example. We have to model a **job processing system** with one processor and parallel working input and output stations.

![Fig. 2.1.1: Job processing with one processor](image)

We want now to illustrate our knowledge with an example.
The elements of net from figure 2.1.1 have the following interpretation

**conditions**

- jobwait - a job is waiting for the beginning of the processing
- procfree - the processor is free
- procwork - a job is processed by the processor
- jobready - the job is ready

**events**

- jobinput - a job is inputted
- beginexe - processing of a job begins
- endexe - processing of a job ends
- joboutpt – a job is outputted

In figure 2.1.1 the only one token characterizes the situation in which the processor is free and there are no jobs in the system.

We come now to the formal specification of a net.

**Definition 2.1.1**

A triple \( N=(P,T,F) \) is called a **net**, if holds

1. \( P \) and \( T \) are some finite nonempty disjoint sets
2. \( F \subseteq (P \times T) \cup (T \times P) \) is a binary relation,
3. \( \bullet F \cup F\bullet = P \cup T \), where \( X = P \cup T \) is the **set of nodes**

\( \bullet F = \{ x \mid \exists y \in X \text{ with } xFy \} \) is the **pre-domain of relation** \( F \)

\( F\bullet = \{ y \mid \exists x \in X \text{ with } xFy \} \) is the **post-domain of relation** \( F \).

Condition (3) needs that each node has at least one input or output arc, i.e. he is **not isolated**.

The elements of \( P \) are called **P-elements** of \( N \), the elements of \( T \) **T-elements** of \( N \), and \( F \) as **flow relation** or **set of arcs** of \( N \).
For $\bullet F$ resp. $F\bullet$ often used is the notation \textit{domain} resp. \textit{codomain} of $F$.

In graph theoretical terms a net is a bipartite graph without isolated nodes.

\textbf{Definition 2.1.2}

If we interprete the P-elements as passive elements resp. the T-elements as active elements, then we call they also as \textit{channels} resp. \textit{instances} (or \textit{agencies}) bezeichnet. Then the interpreted net is called Channel-Instance-Net (abbr.: \textit{CI-Net}) or Channel-Agency-Net (abbr.: \textit{CA-Net}).

For the job processing net from fig. 2.1.1 holds:

$N = (P,T,F)$ with

$P = \{\text{jobwait, procfree, procwork, jobready}\}$

$T = \{\text{jobinput, beginexe, endexe, joboutpt}\}$

$F = \{(\text{jobinput, jobwait}), (\text{jobwait, beginexe}), (\text{procfree, beginexe}),$

$(\text{beginexe, procwork}), (\text{procwork, endexe}), (\text{endexe, procfree}),$

$(\text{endexe, jobready}), (\text{jobready, joboutpt})\}$

\textbf{Definition 2.1.3}

$N$ is a net, $x \in X = P \cup T$.

$x^\bullet = \{y \mid yFx\}$ is the \textit{pre-domain} (P.H. Starke: \textit{pre-set}) of $x$,

$x^\bullet^\bullet = \{y \mid xFy\}$ is the \textit{post-domain} (P.H. Starke: \textit{post-set}) of $x$.

\textbf{Definition 2.1.4}

$N$ is a net.

$N$ is called \textit{simple} (germ.: \textit{schlicht}) iff

for $x, y \in X \land x^\bullet = y \land x^\bullet^\bullet = y^\bullet \implies x = y$.

$N$ is called \textit{pure} (germ.: \textit{rein}) iff there are no cycles of length 2.

That means it doesn’t exists a \textit{loop} (germ.: \textit{Schlinge}).
Fig. 2.1.2: not a simple net and not a pure net
2.2 Condition-Event-Nets

In this chapter we consider a very simple type of Petri Net which is characterised that the P-elements contain at least one token. If we assign a token to a P-element it means that this resource is available or that the condition represented by this P-element is fulfilled.

A net of this special type is called **Condition-Event-Net**.

**Definition 2.2.1**

A mapping from set P into the set \{0,1\} is a **0/1-marking**.

If the conditions are numbered then we write markings often as row vectors.

**Definition 2.2.2**

A 4-tuple \(N=(P,T,F,m_0)\) is a **0/1-marked net** iff the initial marking \(m_0\) is a 0/1-marking and \((P,T,F)\) a net.

**Definition 2.2.3**

A 0/1-marked net \(N=(C,E,F,m_0)\) is a **Condition-Event-Net** (**CE-Net**; germ.: **Bedingung-Ereignis-Net**) iff

\((C,E,F)\) is a pure net where the elements of \(C\) are called **Conditions** and the elements of \(E\) **Events**.

For \(e \in E\): •e is a **pre-condition** of event e.

For \(e \in E\): e• is a **post-condition** of event e.

For \(c \in C\): •c is a **pre-event** of condition c.

For \(c \in C\): c• is a **post-event** of condition c.

A 0/1-marking of a CE-net is called a **case** (germ.: **Fall**).
A net characterises the static structure of a system but the initial marking describe the concrete dynamic. We want now to explain the dynamic behaviour of a CE-net.

**Definition 2.2.4**

The dynamic of net \( N = (C,E,F,m_0) \) follows the **safe firing rule**: i.e. an event \( e \) is enabled (can be fired, has concession) in marking \( m \) iff

all pre-conditions of \( e \) in \( m \) are marked (have a token) and no post-condition of \( e \) is marked in \( m \).

If an enabled event \( e \) fires then the marking \( m \) is changed to marking \( m' \) where

\[
\begin{align*}
m'(c) &= \begin{cases} 
m(c) - 1, & \text{if } c \in \bullet e \\
m(c) + 1, & \text{if } c \in e \bullet \\
m(c), & \text{otherwise.}
\end{cases}
\end{align*}
\]

From part (2) of definition 2.2.3 we can conclude that an event without pre-conditions and with non marked post-conditions can fire just as well an event which has marked pre-conditions and no post-conditions.

In figures 2.2.1 and 2.2.2 we have some general examples of firing and not firing of an event.

**Fig. 2.2.1:** Firing an event

**Fig. 2.2.2: Event e1 cannot fire**
We want now model some simple examples of condition-event-nets.

Examples:

1) FIFO-memory for 3 memory cells

Fig. 2.2.3: FIFO-memory

2) LIFO-stack for 3 memory cells

Fig. 2.2.4: LIFO-memory

3) RANDOM-memory for 3 memory cells

Fig. 2.2.5: RANDOM-memory
4) Counter modulo 3

We want to get the behaviour schemata \( e_1, e_1, e_1, e_2, e_1, e_1, e_1, e_2, \ldots \)
with events \( e_1 \) and \( e_2 \) (it is allowed to fire some auxiliary events between).

![Diagram of Counter modulo 3](image)

**Fig. 2.2.6**: Counter modulo 3

The above solution realises the requirements. But we have 2 variants of sequences of fired events:

a) \( e_1, h_1, e_1, h_2, e_1, e_2, \ldots \) and

b) \( e_1, h_2, e_1, h_1, e_1, e_2, \ldots \)

If \( e_1 \) has fired then we have a token in \( b_2 \) so that \( h_1 \) and \( h_2 \) are enabled.

After firing event \( h_1 \) event \( h_2 \) is no more enabled and vice versa. Such a situation is called a **conflict** between events \( h_1 \) and \( h_2 \).

To enforce a unique firing sequence we can add to our net a so called **regulation cycle**. The cycle consists of conditions \( b_5 \) and \( b_6 \) and arcs from and to the events \( h_1 \) and \( h_2 \) which was in conflict:
Fig. 2.2.7: Counter modulo 3 with regulation cycle

The job processing system of figure 2.1.1 is also a condition-event-net. What happens if a job was already inputted (a token in condition jobwait) and we want to read in a new job? Using the firing rule from definition 2.2.4 it is not allowed to do it. To model such circumstances we can now add some more conditions for waiting jobs for processing and analogously for ready jobs. We want to model the processing system with up to 3 waiting jobs and up to 4 ready jobs. The output of jobs should now be done in such a way that two jobs can be outputted in parallel. Furthermore we want to change our system from a one processor system to a two processor system (parallel processor system) where we can randomly take a waiting job and execute the job by a free processor independently from the other processor. A possible realisation of these requirements as a condition-event-net shows figure 2.2.8.

Fig. 2.2.8: Job processing system with two processors as condition-event-net
The changes from the one processor system to the two processor system show already the important rising number of net elements. If we have to model a manufacturing system with hundreds or thousands particularly resources like bolts and nuts or other single parts then the corresponding condition event nets are not more manageable. Hence it is obvious not to distinguish single elements but to consider only the number of elements. A box for bolts corresponds with a place which is assigned a number. But this requires an extension of our net concept what we are doing in the next chapter. This lead us to the concept of *Place-Transition-Nets*. 