"What Is a Petri Net?"
Informal Answers for the Informed Reader

Jörg Desel and Gabriel Juhás

Katholische Universität Eichstätt
Lehrstuhl für Angewandte Informatik
85071 Eichstätt, Germany
{joerg.desel,gabriel.juhas}@ku-eichstaett.de

Abstract. The increasing number of Petri net variants naturally leads to the question whether the term "Petri net" is more than a common name for very different concepts. This contribution tries to identify aspects common to all or at least to most Petri nets. It concentrates on those features where Petri nets significantly differ from other modeling languages, i.e. we ask where the use of Petri nets leads to advantages compared to other languages. Different techniques that are usually comprised under the header "analysis" are distinguished with respect to the analysis aim. Finally, the role of Petri nets in the development of distributed systems is discussed.

1 Introduction

What is a Petri net? Very often, the thesis of Carl Adam Petri [23] written in the early sixties is cited as the origin of Petri nets. However, Petri did of course not use his own name for defining a class of nets. Moreover, this fundamental work does not contain a definition of those nets that have been called Petri nets later on. In fact, there are hundreds of different definitions and extensions in the literature on Petri nets since then. Most authors did not mean to define something completely new when coming up with a new definition. They use the term "Petri net" to express that the basic concept of a notion is the one of Petri nets, no matter how this notion is formulated mathematically or which extensions of standard definitions are used. In this contribution we try to identify central aspects of this basic concept of Petri nets. In other words, we aim at providing characteristics of Petri nets that are common to all existing and future variants. It should be clear that this can only be done in a very subjective manner. So we like to place the following disclaimer at the very beginning of the paper: We do not consider our list of important aspects of Petri nets complete, and for each aspect claimed to be common to all Petri net variants there might exist very reasonable exceptions.

This paper is not an introduction to Petri net theory. Instead, we assume that the readers have some knowledge about Petri nets and preferably even know different Petri net classes. For an overview of Petri net theory we refer to the proceedings of the previous advanced course on Petri nets [25,26]. The other

H. Ehrig et al. (Eds.): Unifying Petri Nets, LNCS 2128, pp. 1–25, 2001.
© Springer-Verlag Berlin Heidelberg 2001

contributions in this book should also be helpful, although the present paper is meant to be an introductory note to this book. In particular, the work of the "Forschergruppe Petrinetz-Technologie", represented by the papers [35,17,13,36], show how different variants of Petri nets can be subsumed and structured in a unified framework.

There are also examples of modeling notions which do not carry "Petri net" in the name but apparently stem from Petri nets. Among these notions are event-driven process chains (EPCs) [31] (originally called "Ereignisgesteuerte Prozessketten" in German), a standard notion for modeling business processes in the framework of the "ARIS-Toolset". The first publications on this model explictly refer to Petri nets. Still, the central idea is the one of Petri nets although there are some significant differences. Another example is given by activity diagrams, a language within the Unified Modeling Language (UML). These diagrams more or less look like Petri nets and have an interpretation which is very similar to Petri nets but have some additional features such as "swim lanes", associating each diagram element to an object. Although people from the UML community insist that activity diagrams have nothing to do with Petri nets, there already exist a number of publications establishing close connections between these two languages [14,18]. Actually, Petri nets are suggested for a formal semantics of activity diagrams – this notion has evolved to a standard without having any fixed semantics by now. So this paper is about Petri nets and those related formalisms which are based on the same concepts as Petri nets.

Many papers defining or using Petri nets emphasize the following characteristics of the model; Petri nets are a *graphical notion* and at the same time a *precise mathematical notion*. So we take it that these two properties are the most important ones and we devote the following two sections to them. The next important characteristics of Petri nets is described by their *executability*, their *semantics*, their *behavior* or the like. Whereas it seems that the first two characteristic features do not rise any dissension, there is no common agreement what the semantics of a Petri net should look like, i.e., what the behavior of a Petri net formally is. We split the consideration on behavior in two parts; behavior is constituted by the *occurrence rule* – which defines under which conditions a transition is enabled and what happens when it occurs – and by derived formal descriptions of the entire *behavior*, given by the set of occurrence sequences, partially ordered runs or any kind of trees or graphs representing all runs of a net. These parts constitute the topics of sections four and five. *Analysis* of Petri nets is the next important subject, addressed in section six. This term comprises many different concepts; analysis by simulation, by employing structural properties of the net, or by analysis of the exhibited behavior of a net. We distinguish between analysis techniques that automatically provide useful information for a given net (like deadlock-freedom), techniques that automatically verify a given property (like mutual exclusion) and techniques that help in manually proving the correctness of a net with respect to a given specification. The last section is concerned with topics that are not explicitly addressed in most other papers on Petri nets. Each Petri net is a model of a system, if it is not just a counter-

example or an illustration of a proof. There are many different languages for modeling systems, most of them not comparable with Petri nets (consider, e.g., models of the architecture, models of the data structure etc.). Therefore we have to be more precise; a Petri net models the behavioral aspects of a system. The same can be said about differential equations. So we should add that the behavior is constituted by discrete events. Again, there are more prominent languages for this task, namely the variety of automata models. The core issue of Petri nets is that they model behavioral aspects of *distributed systems*, i.e., systems with components that are locally separated and communicate which each other. Surprisingly, neither components nor any notion of locality appears with the usual definition of a Petri net. The section on *distributed systems* discusses aspects of this gap.

Each section header is an answer to the question raised at the beginning of the paper.

2 A Graphical Notation

Most modeling languages have graphical notations, and this has good reasons. Models are used as a means to specify concepts and ideas, and to communicate them between humans. Nearly everybody would use some kind of graphics to express his or her understanding of a system, even without using any explicit modeling language. We asked our first semester students to give a model of the enrollment procedure of our university. The result was a very interesting variety of models emphasizing surprisingly many different aspects of the procedure. All these models were supported by graphics. It does not need psychological research to state that graphics employing two dimensions allow for a better understanding of complex structures than one-dimensional text. Since specification of systems and communication of models are the main applications of Petri nets in practice, understandability for humans is among the most crucial quality criteria for modeling languages. Petri nets have a nice graphical representation using only very few different types of elements, which is a good basis for an easy understandability of a model and for the learnability of the language. These two criteria for modeling languages belong to the most important ones recognized in the "Guidelines of Modeling" [3].

Many modeling languages are supported by graphics that possibly abstracts from some details of a model. Petri nets are not only supported by graphics but each Petri net *is* a special annotated graph. One could argue that the annotations of a Petri net are as essential as the graphics. In fact, for some high-level Petri net classes it is possible to represent any model equivalently by a trivial net structure, putting all the information about the model into the annotations of a single place, a single transition and the connecting arcs [32]. In general, often one has to trade off between specification by graphical means and specification by textual means in the annotations. It is a typical feature of Petri nets that the semantics of textual annotations can be given in terms of nets, i.e. of graphs. As an example, consider the low-level unfolding of a high-level Petri net [32]. In

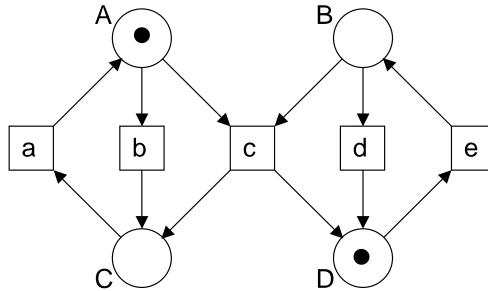

Fig. 1. A picture of a Petri net

this sense, annotations can be viewed as shortcuts for more complex graphical representations, employing, e.g., symmetries of a net. Hence it is justified to claim that a Petri net is a graph.

In the previous paragraphs we confused mathematical graphs with graphical notations. So what is a Petri net, a mathematical object representing the components of a graph or a picture? It is important to notice that by definition the way a net is drawn does not carry any semantic information. This is different for languages such as SADT [28] where it makes an important difference whether an arc touches a node at its right, left, upper or lower side. Also the relative position of Petri net nodes carries no formal information. However, the topology of a drawn Petri net is important from a pragmatic perspective. The modeler might place the elements representing a single system component on a cycle if this helps to understand the net. In this case, additional knowledge about the model and its relation to the system is put in the picture. Alternatively, a tool can calculate a nice way to draw a net; then the figure carries information about the net itself and about some analysis results. So a Petri net picture can be more than a mathematically defined graph. The difference is irrelevant for analysis tools. But it is significant when the net is used as a means for human communication. Even simple models can be drawn in a spaghetti style such that this picture does not help much (compare for example two pictures of the same Petri net in Figures 1 and 2). The topology of a net drawing is an important topic in the context of interchange standards for Petri nets [20]. The exchange information of a picture might contain information about the relative position of the nodes, about their shape etc.

It is often emphasized that Petri nets are bipartite graphs, because each directed arc either leads from a place to a transition or from a transition to a place. This is not exactly true; Petri nets are more than that. In bipartite graphs the two sets of nodes play a symmetric role whereas places and transitions are dual concepts. Exchanging places and transitions leads to a completely different net. The existence of places and transitions and their distinction, is one of the fundamental ingredients of Petri nets. Therefore this formalism is neither primarily based on actions (like data flow diagrams), represented by transitions, nor is it primarily based on states (like automata), represented by places. Instead, the

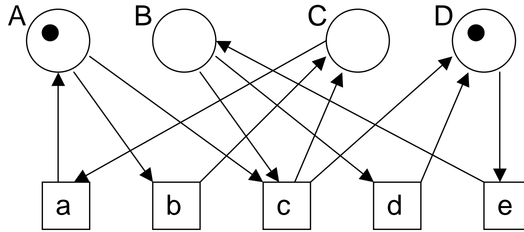

Fig. 2. Another picture of the Petri net from Figure 1

mutual interplay of local activities and local states constitutes the basic components of each net, as will be discussed later in more detail. The equal footing of actions and states is reflected in nets by the definition of places and transitions on the same level. So the following definition is the most common answer to the question "What is a Petri net?"

> **The usual definition of a "core" Petri net**
> A Petri net is a directed graph with two kinds of nodes,
> interpreted as places and transitions,
> such that no arc connects two nodes of the same kind.

Places of a Petri net are usually represented by round graphical objects (circles or ellipses), and transitions by rectangular objects (boxes or squares), as shown in Figures 1 and 2. There is a standard arc type between vertices of different type representing the flow relation, as shown in the figures. This convention makes it easy to guarantee a rough understanding of any Petri net without additional legend. One of the main advantages of the Unified Modeling Language (UML, see [30]) is that it unifies the shape of vertices and arcs in its diagrams that have been used in a contradictory way in different languages before. Likewise, the consistent use of graphical symbols for Petri net objects is one of the main reasons for the worldwide and long standing success of Petri nets. Whenever someone acquainted with Petri nets is confronted with a new variant, the general interpretation of places and transitions does not have to be explained and gives no rise to misunderstandings. So Petri nets play the role of a "Unified Process Language" since a long time.

Sometimes the use of only circles and squares is considered a disadvantage. Instead of circles or squares, special symbols representing the actual type of the represented system component can be used. Branches at transitions or at places can be substituted by special branching nodes. These variants are – among others – implemented in many commercial Petri net tools. The vendors claim that the readability of their models is improved by the graphical extensions. This might sometimes be true, but there is the danger of inconsistency between different products. Moreover, an increasing number of features leads to an increasing number of modeling errors. Someone only familiar with such a specific application-dependent notion can not understand an example net given in another proprietary notation. However, as long as additional graphical notions have

a unique and easy translation to traditional Petri net components, a good knowledge about Petri nets will help to understand any such model. In this way, Petri nets – including their graphical representation – can be seen as an "interlingua" for many different related modeling languages.

3 A Precise Mathematical Language

One might say that this answer to the question raised in this paper is a matter of course. However, often the mathematics, and particularly its presentation, is the reason to consider Petri nets difficult for users. As mentioned in the previous section, it is the graphical representation of a net which is actually used in practice and which can easily be understood. So why do we need any further mathematical foundation? And what does it mean for a modeling language to be precise? The answer to these questions concerns two parts: *syntax* and *semantics*.

There are different ways to specify a class of nets syntactically. Well-known examples are restricted classes such as free-choice Petri nets [5] where the local vicinities of net objects are restricted in a characteristic way. Another frequently used possibility is given by the class of all nets that are generated from an initial one using a given set of production rules. Such Petri net grammars can be used for a syntactic formulation of a Petri net class, defining exactly which Petri nets belong to that class.

We consider here another way to deal with the syntax of Petri nets; each Petri net should have a precise syntax. In other words, it should be clear what kind of objects belong to a given Petri net and which objects do not belong to it. This syntax differs for different classes of Petri nets. It turns out that this kind of formal syntax can be more conveniently be given in terms of simple mathematics than in terms of the graphical representations. So for definition purposes, Petri nets are syntactically defined as annotated graphs in a mathematical setting. The usual notions equip tuples of sets, relations and mappings. The following definition shows an example.

The mathematical definition of a place/transition Petri net
A place/transition net is a tuple (S, T, F, M_0, W, K), where
S is the set of places,
T is the set of transitions,
F is the flow relation,
M_0 is the initial marking,
formally given as a mapping from S to the nonnegative integers,
W maps arcs to positive numbers (arc weights) and
K maps places to positive numbers (capacity restrictions).

These objects have to satisfy some restrictions, such as $M_0(s) \leq K(s)$ for each place s (no capacity restriction is violated initially). All these objects and restrictions are very easy to explain using the graphical representation. For example, "no arc connects two places or two transitions" might be more plausible than the usual expression $F \subseteq (S \times T) \cup (T \times S)$. But all the used objects and restrictions

have to be precise enough that an equivalent mathematical formulation can be given in a simple and obvious way and such that there is no doubt how this formulation would look like. The tuple notion induces an order on the objects (in the above example, places before transitions before arcs before ...). This order does not imply valences of the used objects. It is just arbitrarily fixed for convenience sake; the formulation "given a place/transition net (A, B, C, D, E, F)" is the shortest way to define all components of a place/transition net. But the tuple-notion never represents the core idea of a Petri net. When teaching Petri net theory one should be careful not to emphasize this notion too much – it unnecessarily complicates the matter.

When are two Petri nets identical? Using the mathematical definition, the answer is obvious: two nets are identical if and only if all their objects are pairwise identical. This implies in particular that a different graphical representation of a Petri net does not change the Petri net. Conversely, two Petri nets which look the same, i.e. which have identical graphical representations, are not necessarily identical, assuming that the graphical representation does not include the identity of each single element. This is often not exactly what one would like to have. Instead, Petri nets that look the same should sometimes not be distinguished. Imagine for example the net with a single (unmarked) place, a single transition and one arc from the place to the transition (arc-weights, capacities etc. are ignored for this example). Putting this description into mathematics one needs to define a place s, a transition t and an arc (s, t). There is no unique net that matches the above description, because the identity of the place and the identity of the transition is chosen arbitrarily. The net with place s' (where $s \neq s'$), t and arc (s', t) is different to the one defined before. This difference is only meaningful if the net models something; then s and s' model different objects of the system domain. But syntax does not distinguish what is modeled. So, intuitively one is interested in the class of all nets which can be obtained from the original one by consistent renaming. In other words, the syntactical definition of a Petri net comes with the notion of an isomorphism relation.

Isomorphism of Petri nets
Two Petri nets are isomorphic if there are bijections
between their respective sets of objects (places and transitions)
which are respected by all annotations, relations and mappings
that belong to the syntactical definition.

The simple but important distinction between equality and isomorphism of Petri nets is only easily possible on a mathematical level. Intuitively, a single (graphical) Petri net is mathematically given by an isomorphism class of tuples, where each single tuple of the class has the same Petri net as its graphical representation. Isomorphism classes are particularly important for labeled Petri nets, i.e. nets where each element carries a label which establishes the connection to the modeled world. In a labeled Petri net, two distinct places can represent the same object and two distinct transitions the same action. For example, process nets representing partially ordered runs of other Petri nets are labeled Petri nets.

The same run can be represented by many isomorphic process nets (see the next section).

What is the semantics of a Petri net? Taking the original meaning of the word "semantics", the answer should associate objects of a net to objects of the modeled system. Considering also the dynamics of the net, the behavior of the net should correspond to the behavior of the modeled system. In the context of modeling languages, the term "semantics" is used in a different way, usually together with the prefix "formal". A class of Petri nets has one (or several) formal semantics although the world of modeled systems is not considered at all when defining such a class. The formal semantics generically defines the behavior of each Petri net that belongs to the class, i.e. the role of each possible ingredient of a net with respect to behaviour is precisely defined by the semantics. Since Petri nets are defined by mathematics, so are their formal semantics. At this stage, we do not discuss different variants of semantics, because this will be the topic of the next section.

Many modeling notions used in practice do not have a precise semantics. Defining a formal semantics is only possible for a notion possessing a formal syntax. Hence, without explicitly defining the syntax it is impossible to formalize semantics. Some languages do have a formal syntax, with or without a mathematically given description, but no fixed semantics. These notions are frequently called "semi-formal". It is often claimed that semi-formal modeling languages allow more flexibility and are hence better suited for practical applications than formal modeling languages like Petri nets. Moreover, semi-formal models are said to be easier to understand and easier to learn. We claim that the opposite is true. The theory of Petri nets offers classes of nets where specific details of the model are left open. For example, channel/agency nets define only the structure given by places, transitions and arcs together with the interpretations of these elements, but no behavior [24]. Place/transition nets identify all tokens and thus abstract from different token objects. Conflicts, i.e., different mutually exclusively enabled transitions, can be interpreted as incomplete specifications – the vicinity that decides which alternative will be chosen is missing. Most nets abstract from all notions of time. So there are various ways to express different kinds of vagueness. The important point is that it is always very clear which aspects are expressed by the net and which aspects are not. Many modeling notions outside the Petri net world exhibit moreover a kind of meta-vagueness. For these models, it is a matter of interpretation to decide which aspects are represented in the model and which are not. So flexibility concerns not only the model itself but also its interpretation – a feature that we do not consider desirable. Instead, it is much easier to understand a model and also the modeling language if there is a precise understanding about *what* has been modeled and *how* it is modeled. Only a precise mathematical language, such as given by most variants of Petri nets, provides sufficient clarity.

As an example for a semi-formal notion, consider event-driven process chains (EPCs) [31]. This language is a derivative of Petri nets. In the application field of business processes it has emerged to a quasi-standard. The major benefit of EPCs

is that they are integrated in a larger context containing additionally a data model and a structure model (the "house of ARIS" [31]). An EPC has three types of nodes, two comparable to places and transitions, and an additional node type for the logical connections AND, OR, and XOR (exclusive or). Not surprisingly, the OR connector raises severe problems. A binary OR-split is interpreted as follows. Either one of the output arcs or both arcs are chosen for forwarding the control. A binary OR-merge cannot be interpreted in such a simple way. After receiving the control from one input arc, either one has to wait for the control from the other arc or one can continue immediately which corresponds to the different possible decisions at the OR-split. This technical problem has led to quite a number of research activities (see e.g. [29]), but there exists no really satisfying solution yet. The problem is that EPCs have no formal semantics. When asking experts in EPC modeling about the correct interpretation of an OR-merge in a difficult example, they come to very vague (and different) answers. Surprisingly, it is often claimed that EPCs are more compact, more appropriate and easier understandable than Petri nets in the application area of business processes. The paper [2] proves that they are not smaller than equivalent Petri nets in general. Nonetheless, the Petri net community should learn from EPCs which kind of concepts and which kind of links between concepts are necessary for successfully selling a modeling language together with an associated tool.

4 A Structured Set of Activities that Remove and Add Tokens

Most Petri net variants are equipped with a notion for behavior. Some variants, however, are not. For example, channel/agency nets do not have an explicit behavioral definition [24]. They are used as a first step when developing a Petri net model. Refinement and completion of a channel/agency net leads to a more detailed model, which can then be equipped with behavior.

In this section, we restrict our considerations to nets that do have a behavior.

In contrast to all automata models and transition systems, a (global) state of a net is not a fundamental concept but it is constituted by local states of all places of the Petri net. States are formally represented by markings. A marking associates a set, multi-set, list etc. of tokens to each place, where tokens are elements of some domain. So a global state is only a derived concept (with the exception that the definition of a Petri net often contains initial or final global states).

Principle of Distributiveness
States are associated to places and thus distributed.
A global state is constituted by all local states.

In most cases the behavior of a net is formulated by means of a rule stating under which conditions a single transition can occur and stating the consequences of its occurrence, the so-called occurrence rule. It is one of the central principles of

Petri nets that both the enabling conditions and the consequences only concern the immediate vicinity of a transition. In other words, if the occurrence of a transition is related to the state of a place then there must be some arc connecting the transition and the place.

> **1. Principle of Locality**
> The conditions for enabling a transition, in a certain mode if applicable, only depend on local states of (some) places in its immediate vicinity.

> **2. Principle of Locality**
> The occurrence of an enabled transition only changes the local state of (some) places in its immediate vicinity.

We formulated the locality principle in two parts because the relevant sets of places for enabledness and for change in the vicinity of a transition are not necessarily identical. For place/transition nets, all places in the pre-set (i.e., sources of arcs leading to the transition) are relevant for enabling the transition, and places in the post-set (i.e., targets of arcs from the transition) only play a role when capacity restrictions are involved. The state is only changed for places which are either in the pre-set or in the post-set but not both (as long as arc weights are not considered). Moreover, the new state of a place depends on its previous value in place/transition nets, because a token is added. However, the relative change of the state of a place does not depend on its previous state. Given a transition, we can distinguish:

a) places where the local state is relevant for enabling but is not changed (such as read places or inhibitor places),
b) places where the local state is relevant for enabling and is changed by the transition occurrence (places in the pre-set in case of place/transition nets without capacity restrictions), and
c) places where the local state is not relevant for enabling but is changed by the transition occurrence (places in the post-set in case of place/transition nets without capacity restrictions).

Orthogonally, places where the local state is changed by the transition occurrence (cases (b) and (c)) can be divided into:

1) places where the new local state depends on its previous value (places in the pre-set and places in the post-set in case of place/transition nets), and
2) places where the new state does not depend on the previous one (such as places reset by the transition occurrence in case of nets with reset arcs).

Often, the different role of the places is depicted by different arc types such as inhibitor arcs or reset arcs. When talking about the vicinity of a transition, we mean all places connected with the transition by an arbitrary arc.

It might be worth mentioning that the majority of Petri net formalisms considers test-and-set-operations elementary, i.e. reading a local state and changing it depending on the previous value is considered one atomic action. These Petri

net formalisms have no difficulty with simultaneous access to different places, even if these places model conditions at different locations. The general paradigm is the one of removing and adding tokens. It can even be phrased as:

The Token Flow Paradigm
Tokens flow with infinite speed from place to place,
sometimes they mutate, join or split in transitions.

Perhaps surprisingly, read actions (case (a)) and write actions (case (c),(2)) are not that usual in the Petri net literature. As explained above, reading the state of a place means that this place is relevant for a respective transition but the state of this place is not changed by the occurrence of the transition. Although concurrent read is an essential operation in most areas of computer science, many semantics of Petri nets do not allow any concurrent access to the tokens of a place (see [11]). Likewise, writing is a central issue in other areas of computer science but there is hardly any corresponding concept in the Petri net literature. Petri nets with reset arcs are an exception, but they model only the special case that a place looses all tokens when the corresponding transition occurs. More generally, writing the local state of a place means changing the state arbitrarily without taking the previous state into account. The only way to model writing with Petri nets is by synchronous removing the old tokens and adding new ones. It needs a special variant of high-level nets to perform arbitrary removing with a single transition, such that the previous local state has no influence on any new state (see [6]).

There are generalizations of the occurrence rule concerning the simultaneous occurrence of many transitions. These variants still obey the principle of locality, because the vicinities of all simultaneous transitions have to be considered.

5 A Compact Way to Specify Behavior

The behavior of a Petri net does not only concern occurrences of single transition but sets of occurring transitions which can be in different relations such as causal relationship, concurrency, choice, or being totally ordered. The behavior can also include intermediate local or global states or the final global state and possible continuations from these states. Different ways to describe the behavior of Petri nets are given by different semantics of the respective Petri net classes.

Given a model of a dynamic system, the behavior of the model should be in a close relationship to the system's behaviour. If the model is executable, i.e. if it has a defined semantics, then runs of the model can be generated. These runs correspond to the runs of the system. Analysis of the model's behaviour yields information about the system's behaviour. In this section we concentrate on the question how to formalize the behavior of a net. Since the behavior is the most interesting aspect of a model, one can phrase this question also as: What kind of behavior is represented by a Petri net?

We will not discuss different semantics in detail. Other contributions to this book are devoted to this topic [4,11,21,22]. Instead, we provide a rough landscape

of different behavioral notions that can be formulated for arbitrary Petri net variants that are equipped with dynamic behaviour, formulated by means of an occurrence rule.

We distinguish different ways to formalize single runs, namely sequences, causal runs and arbitrary partially ordered runs. Orthogonally, we distinguish single runs, tree-like structures representing more than one run, and graphs, representing all runs and taking cyclic behavior into account.

5.1 Runs

Given a Petri net with initial marking, not only a single transition can occur but also sets of transitions, constituting a run. We call the occurrence of a transition in a run an event. In the sequel, runs for different semantics will be sketched. For each semantics, we provide a Petri net notation for its runs.

> **The behavior of a net is a net**
> Runs of Petri nets consist of events and pre- and post-conditions
> that generate a (partial) order.
> Runs can always be represented by nets.

For sequential semantics, representing runs by nets is not usual. Instead, often words or sequences are used to formalize totally ordered runs. Automata-like trees and graphs represent the entire behavior. In this paper, we represent all types of runs by Petri nets. An obvious advantage is that, using this unifying approach, different semantics can be easier related and compared which each other. However, we do not claim that this representation is better readable than alternative graphical or textual representations.

In the sequel, the Petri net modeling a system will be called system net, to avoid confusion.

An *occurrence sequence* describes a sequential view on a single run. In the initial state, some transition can occur yielding a follower state. In this state, again some transition occurs, and so on. Hence the events of an occurrence sequence are totally ordered and can be represented by a sequence of transition names (as the name occurrence sequence suggests): $t_1\, t_2\, \ldots\, t_n$ for finite occurrence sequences with n events or $t_1\, t_2\, t_3\, \ldots$ for infinite occurrence sequences. Notice that, for $i \neq j$, t_i and t_j might denote the same transition. Sometimes all intermediate global states are represented as well. However, they do not provide any additional information because each global state can be calculated from the subsequence leading to it and the initial state, using the occurrence rule.

A sequential run can also be conveniently represented by a very simple Petri net, where places represent tokens and transitions represent events. An example is shown in Figure 3. In general, each place in the pre-set of a transition represents a token of the marking enabling that transition, and similarly for post-sets. In this example, the number of tokens is two for all markings, but this is not the case in general. The net representation of an occurrence sequence is unique up to isomorphism.

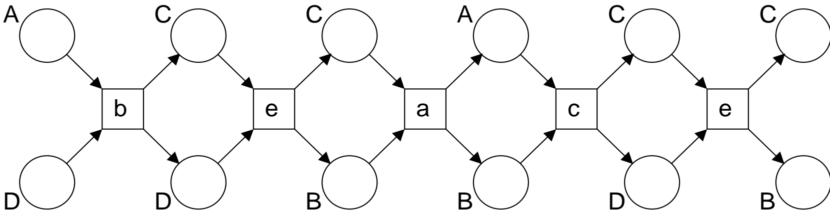

Fig. 3. A Petri net representing the occurence sequence $b\,e\,a\,c\,e$ of the system net from Figure 1

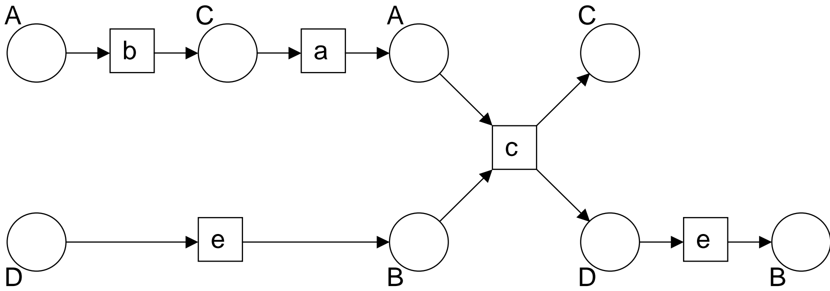

Fig. 4. A process net of the system net from Figure 1

A *process net* is a Petri net representing all events of a run and their mutual causal dependencies. Any such dependency states that a transition can only occur after another transition has occurred. General dependencies are generated by immediate dependencies, stating that a transition occurrence creates a token that is used to enable the other transition. These tokens are represented by places of the process net. Reasons for immediate dependencies are always explicitly modeled in the system net. So there is a close connection between the vicinities of a transition representing an event of a process net and the vicinity of the corresponding transition of the system net. Process nets have specific syntactic restrictions:

– Each place has at most one input transition and at most one output transition, representing the creation and the deletion of a token instance in one single run.
– The places with empty pre-set correspond to the initial token distribution which is given by the initial state.
– The relation "connected by a directed path" is a partial order, i.e., a process net contains no cycles. This is due to the fact that this relation represents the dependency relation, which obviously is acyclic.

Figure 4 shows an example of a process net.

The next semantics under consideration is given by arbitrary *partially ordered runs*. Process nets induce partially ordered sets of events. Occurrence sequences induce totally ordered sets of events. Sometimes arbitrary partial orders which

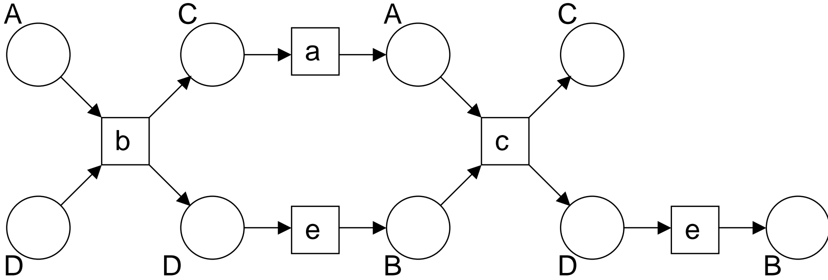

Fig. 5. A Petri net representing the process term $(b\,;(a+e)\,;c\,;e)$ of the system net from Figure 1

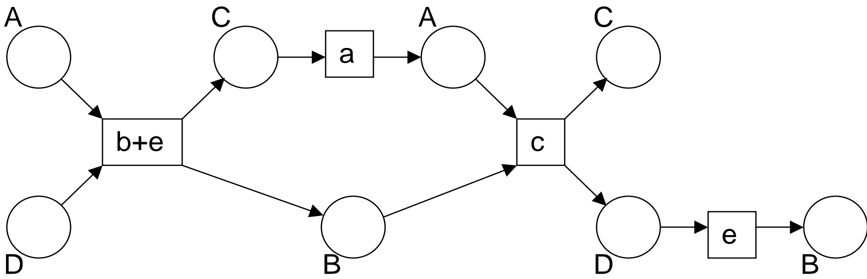

Fig. 6. A Petri net representing a run with the step $\{b, e\}$

define more dependencies than a process net and less dependencies than an occurrence sequence are useful. For example, when a Petri net variant contains timing information then it might be useful to define a relation "later than". This relation can express that an event occurs after another event, even when there is no token that constitutes an explicit dependency between the events.

Another example is given by a so-called process term semantics (see e.g. [11]). A process term such as $(b\,;(a+e)\,;c\,;e)$ is a generalization of the sequence representation of a sequential run. It describes that transitions a and e occur concurrently, both after b, and both before c, which occurs before e. A Petri net representation of this term is given in Figure 5. As discussed in [11], process terms do not have the expressive power to describe arbitrary process nets. However, sets of process terms can be used to specify an arbitrary process net.

Steps represent sets of simultaneous events. Simultaneous occurrences and concurrent occurrences of transition are different in general. Being simultaneous is a transitive relation whereas concurrency is not (in the above example of process nets, the events labeled by b and e are concurrent, the events labeled by e and a are concurrent but the events labeled by b and a are not concurrent). In general, concurrent events can occur in a step but not each step refers to a set of concurrent events. A Petri net representation of the run given by the process net of Figure 4 using the step $\{b, e\}$ is shown in Figure 6.

Since a run can be infinite, all the mathematical objects corresponding to the above representations of a run can be infinite as well.

5.2 Trees

Two different runs can start identically and then proceed differently. A compact representation of these runs contains the common prefix only once and then splits for the different continuations. This representation also explicitly shows after which events there exist alternative continuations (in Petri net theory, alternatives are also called choices or conflicts). This construction can be performed for arbitrary sets of runs and for all representations of runs listed above. Taking the Petri net representation of occurrence sequences and our above example, the occurrence tree of Figure 7 is obtained this way. Notice that this net, seen as a graph, is not really a tree but only a tree-like structure which we call tree by abuse of notation. When markings are represented by single vertices, which is the usual way to draw occurrence sequences, then the resulting graph actually is a tree.

If the reason for constructing an occurrence tree is only to identify the set of reachable markings, then it is not necessary to consider any event leading to a marking that was already identified as reachable before. In our example shown in Figure 8, it suffices to consider the occurrence sequences $e\,b$ and b because the marking reached after $e\,d$ or $b\,a$ is the initial marking, the marking reached after $e\,c$ is also reached after b and the marking reached after $b\,e$ is also reached after $e\,b$ (these are all possible continuations). In other words, we can cut the complete tree after the occurrence of $e\,b$ and after the occurrence of b.

In the example, any sequential construction of the occurrence tree will stop after three events if the above cut criterion is used. In general, a Petri net can have infinitely many different reachable markings. Then there still exists a finite tree-like structure that provides at least some information on the reachable markings: If the above stop criterion is changed to: "stop if a transition that occurred previously produced a marking that is smaller than the one produced by the current transition" then the so-called coverability tree is obtained (see [8]).

Tree-like structures can also be constructed from process nets. The resulting nets are called unfoldings of the system net. Again, cut criteria can be used to obtain finite representations of the behavior. For unfoldings representing all process nets, these criteria are given by cut-off transitions, as defined in [15]. A similar concept can be used for unfoldings obtained from an arbitrary subset of process nets [10].

When process terms do not only have operators for sequential and concurrent composition but moreover allow to express alternatives, then the corresponding Petri net representation is a tree-like structure obtained by glueing common prefixes of their Petri net representations. Likewise, it is not difficult to define a corresponding concept for steps.

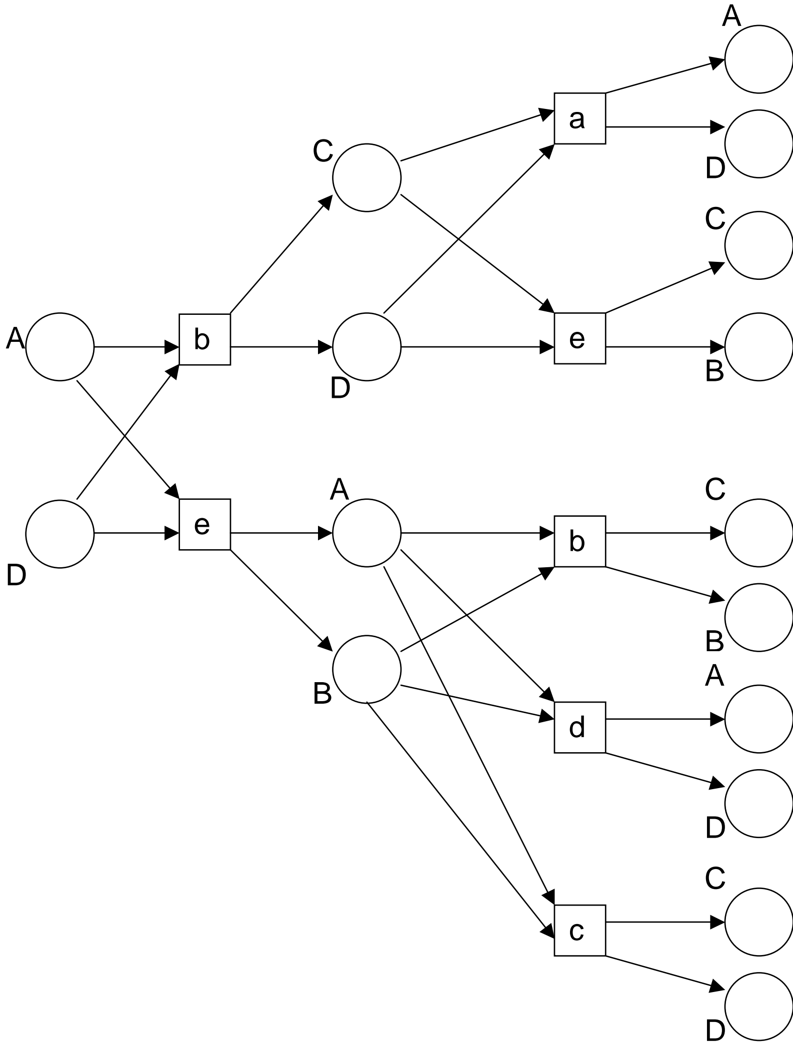

Fig. 7. A Petri net representing an occurrence tree of the system net from Figure 1

5.3 Graphs

In addition to the glueing of common prefixes of runs, one can identify sets of places that represent the same marking, to be explained next. In the previous subsection we suggested to stop the tree construction when the post-set of an event represents a marking that is already represented by the places of the post-set of another event. The next step to obtain graphs is simply performed by adding the new event and drawing arcs from it to all places of the set of places that represent the reached marking. The graph obtained this way is the reachability graph of the system net. Actually, the usual definition of a reachability

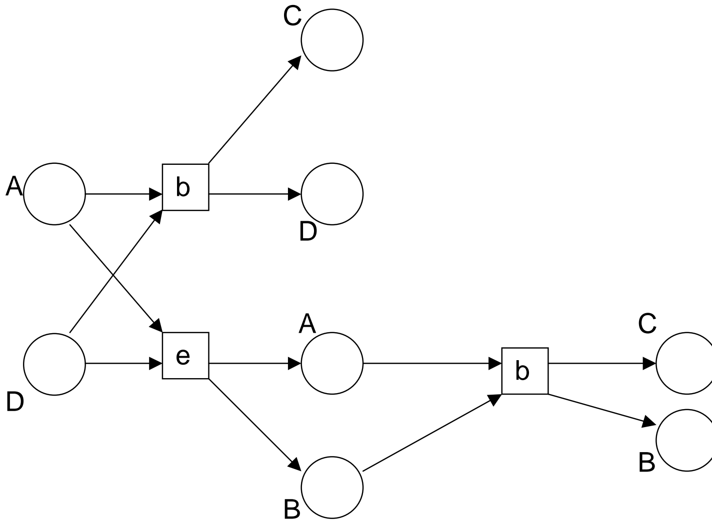

Fig. 8. A Petri net identifying all reachable markings of the system net from Figure 1

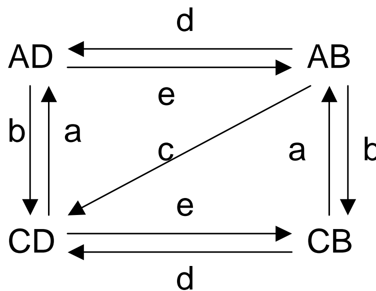

Fig. 9. The reachability graph of the system net from Figure 1

graph employs markings as nodes and transitions as arc labels, see Figure 9. It is not difficult to see that our Petri net notion of reachability graphs is equivalent, see Figure 10.

Similarly, one can construct coverability graphs from coverability trees to obtain a smaller representation of the entire behavior. Steps can also be taken into account in reachability graphs in the obvious way. However, for process nets and other Petri nets describing partially ordered runs or trees there is no obvious way to construct graphs representing the entire behavior. The reason is that markings of these nets are properly distributed. In fact, glueing all places and transitions with respective equal label usually resembles the original net, and in this case nothing is gained by the construction.

For process terms, loops in the corresponding graphs correspond to additional operators for iteration.

Fig. 10. The Petri net representing the reachability graph of the system net from Figure 1

6 A Formalism Equipped with Analysis Methods

A considerable amount of the huge number of Petri net articles published in the past thirty year is devoted to the analysis of Petri nets. Clearly, here is no space to give a survey of all these results. Instead, we provide a classification of different methods that are often summarized under the word “analysis”.

6.1 Simulation

Simulation means creation and investigation of runs. In most cases, not all runs of a model can be generated. Runs might be infinite, and the set of runs might be infinite. But even in the case of finitely many finite runs their number often is too large to allow a complete simulation of all runs of a given Petri net. Therefore, simulation usually considers only a part of the system's behavior. Like testing of programs, simulation can thus only identify undesirable behavior but cannot prove the correctness of a model, as long as not all possible runs of a model are simulated.

Simulation can be performed by playing the token game by hand or even in mind. This procedure is quite error-prone. When finding an undesirable behavior, it is hard to say whether the identified error is due to a design error or to a simulation error. Therefore simulation is usually done by computer tools.

Many simulation tools just offer a visualization of the token game such that the constructed run is represented only implicitly. Other tools create runs which

can be represented to the user and also can be input to further analysis. For example, the VIPtool [10] creates process nets which can be analyzed with respect to a given specification.

There are other applications of simulation approaches in the context of performance evaluation, where quantitative measures, e.g. about the average throughput time of a system, are derived in a simulative way.

6.2 Analysis

Analysis in a narrow sense means to gain information about a Petri net model. For example, results of analysis can be the information on deadlock-freedom, liveness, boundedness and the like. Analysis can also yield information which is useful for proof methods. For example, an analysis tool can calculate a set of place invariants (see below).

Analysis of syntactical properties such as the free-choice property [5], strong connectedness etc. are based on the structure of the net, whereas analysis of behavioral properties such as deadlock-freedom usually needs the construction of a tree or graph representing the behavior. For many properties of general Petri nets it can be proven that essentially there does not exist any more efficient way to decide the property [16]. Exceptions only exist for subclasses of Petri nets such as free-choice nets.

Quantitative analysis provides quantitative results. In contrast to simulation, quantitative analysis computes these results from quantitative parameters and attributes associated to a Petri net.

6.3 Verification

The term "Verification" often comes along with the term "specification". Verification finds out whether a given specification holds true. There are numerous ways to formulate a specification. Like analysis, a typical approach to verification is based on the construction of the reachability graph of a Petri net which is then further investigated. More efficient approaches construct reduced reachability graphs that still carry all the information that is relevant for verification. For example, reachability graphs can be reduced by employing symmetries and methods reducing the redundancy which is caused by concurrency. In particular, the so-called stubborn set method [34] has proved to yield significant reductions of reachability graphs without spoiling information about the possible enabledness of transitions. An alternative efficient approach to verification employs unfoldings of nets, i.e. a behavior description based on process nets, see [15].

6.4 Semi-decision Methods

A semi-decision method is a method for verifying a given property which has either the possible answers "yes" and "don't know" or the possible answers "no" and "don't know". Thus, one possible output provides a useful information

whereas the other one just states that this method is of no use in the current case.

A well-known example for an efficient semi-decision method for deciding reachability of a marking of a place/transition net is given by the marking equation: For every reachable marking this equation system has a nonnegative integer-valued solution. A marking is proven to be unreachable if it is shown that no such solution exists, whereas the existence of a solution does not prove anything. Weaker but more efficient methods look for arbitrary integral solutions, for nonnegative rational-valued solutions or even for rational-valued solutions. For a discussion of the respective expressive power and the complexity of these approaches, see [9].

6.5 Proof Methods

The most prominent formal concepts for Petri net analysis are place invariants for almost all variants of Petri nets and siphons and traps for place/transition nets. Perhaps surprisingly, these concepts are of little use for analysis in the narrow sense of 6.2. Usually the existence of a specific place invariant is not a property relevant for a user. In general, the number of minimal (non-negative non-zero integral) place invariants, the number of siphons and the number of traps can grow exponentially with the size of the net. So even an enumeration of these objects is not feasible.

Instead, place invariants, siphons and traps can be used very elegantly for proving that a desirable property holds. However, the user has to find the necessary invariants, traps and siphons first. Tools can help to verify that a suggested invariant actually is an invariant of the investigated net. Place invariants have close relations to the semi-decision method based on the marking equation. Namely, there is a place invariant for proving a property if and only if the marking equation has no rational-valued solution. A similar result holds for so-called modulo place-invariants and integral solutions of the marking equation [7]. In this sense, the proof methods based on these concepts can be viewed as nondeterministic semi-decision algorithms; if the right place invariant, siphon, or trap is guessed, then its characteristic property can esily be verified and it can be used to prove the desired property.

Place invariants, siphons and traps are based on simple arguments on assertions; the associated properties are preserved by arbitrary change from a (not necessarily reachable) marking to a follower marking that is allowed by the occurrence rule. Actually, only the changes from reachable markings are relevant. The restricted expressive power is due to the possibility that the property under consideration is not preserved by an unreachable marking change, and hence the argument cannot be used, although it might be preserved for all reachable changes. However, the restriction to reachable changes is not easy because it requires the construction of all reachable markings. In most cases this construction is very time consuming and provides an immediate proof of the desired property without using assertions.

6.6 Validation

Whereas verification checks a model against a given specification, validation checks a model against the modeled system or against desired properties of the system. If the model is not correct then analysis and verification of the model is of little use because in this case the behavior of the system might significantly differ from the behavior of the model.

Validation of a model means to compare the model with either the existing system or a planned system where some properties are known. It can be done on a structural level by comparing all the components (elements and connections) of the model with reality. A further step in validation uses simulation: Simulated runs of the model should correspond to runs of the system and vice versa. Verification and analysis can also be used; when applied to the model, the results should coincide with corresponding properties expected from the system.

The investigation of a system by analysis of its model only makes sense if the model can be assumed to be valid. So it is useful to proceed in two steps: First the above mentioned methods are applied to ensure that the model is correct with respect to the modeled system, i.e., that it is valid. After that, it can be assumed that the model's behavior and the system's behavior are closely related, and further application of the above methods to the model provides information about the system's behavior.

7 A Model of a Distributed System

Complex distributed systems with a large number of connected components exhibit a very complex behavior. Every component might depend in some way on each other component. The set of global states reachable by consecutive transition occurrences often grows exponentially in the size of the system. The central feature of Petri net theory is that

Petri nets can manage the complexity of large systems.

Instead of yielding rapidly growing state spaces, the number of places grows linearly with the size of the modeled system. The reachable states do not have to be represented explicitly but are implicitly given by the many combinations of local states. Instead of explicitly stating all direct or indirect dependencies, only the immediate dependencies are represented – other dependencies follow transitively in runs of the model. It does not matter whether transitions and their vicinities are taken as elementary building blocks, as discussed in Section 4, or whether places and their vicinities representing the relevant actions are considered. The result is the same: a Petri net. This way of modeling has not only the advantage of keeping the complexity of the model manageable, it also resembles the modular structure of the modeled system.

However, in general the single components of a system and their connections cannot be identified in its Petri net model. Petri nets are not equipped with notions for physical distribution, channels, messages or locality (at least, this

holds for the most common Petri net variants). The lack of these apparently important concepts is often claimed to be a disadvantage of Petri nets. Other modeling languages are based on local components, have means for communication such as message passing or synchronization and provide elegant ways for composing components, refinement of components etc.

Comparing Petri nets with such notions, it turns out that Petri nets support all these concepts as well. Since Petri nets constitute a very general language, different concepts for locality, refinement, composition and communication can be expressed.

When using Petri nets for modeling distributed systems of a specific kind, this model is easier to understand when its components, the communication between components etc. are easily identified in the model. Since in conventional Petri nets the information about these concepts is lost, it is useful to define languages that are based on Petri nets but restrict to certain macros defining possible building blocks in a given paradigm. Petri nets are general enough such that this kind of macros can be easily defined for different modeling paradigms. On this macro level, it is easy to understand the model from a behavioral view, because the model is still a Petri net. It is also easy to identify components and communication because they are formulated in terms of macros. By definition of the macros, restricting to certain sets of macros ensures that the model obeys the rule for the given paradigm. For example, a model of a message-passing system can not use a macro representing a shared variable. Here are some examples for suitable macros:

A *local component* can be given by a subnet which is connected to other subnets in a very restricted way. Different states of a component can be represented by different places or by a single high-level place (i.e., a place of a high-level Petri net). It is useful to give a graphical representation for the subnets that represent components. In a more compact representation, a single subnet of a high-level net might represent a set of similar components [27].

A *variable* can be represented by a special kind of place that is only connected to transitions that read or write the variable (see Section 4). It is useful to give a special graphical representation for variables.

A *message channel* can be represented by a specific place. Only transitions of components that actually have access to the channel can remove a token. Sometimes a channel is represented as a chain of places and transitions. In this case it is useful to provide a coarser view by a single place that is refined to this chain.

Synchronous communication can only be applied to transitions that model interfaces of components. It is useful to provide a graphical representation for these transitions. When synchronized, two transitions occur together. This can either be defined as part of the semantics or an additional common transition is introduced.

The concept of *Asymmetric Synchronization* means that a transition can only occur together with another transition, which in turn can only occur alone if the first one is not enabled [33,19,12]. This concept frequently occurs when

modeling modular technical systems. There exist translations to traditional Petri nets. However, the number of transitions in such translation can grow rapidly. Also, a macro notation using special event arcs keeps the nets more readable.

8 Conclusion

This paper presented the author's selection of possible readings of Petri nets, commenting on them from the personal perspective. It was not meant to be technical and attempted no comparisons between models, nor between different variants of nets. Instead, it tried to concentrate on the common grounds of Petri net variants. There would have been hundreds of opportunities to add references to other work but the authors avoided to create an annotated bibliography. So also the selection of pointers to the literature was a (sometimes biased) personal choice.

Some readers my feel that some topics should have been treated in greater detail, or in a more technical fashion. We will end the paper with a couple of links to further readings which we left out because their respective topics concern only a part of the world of Petri nets.

There exists prosperous research on Petri nets equipped with *time*. Time can be associated to transitions, to places, or to arcs. Time can be deterministic, i.e., the occurrence of a transition always lasts the same amount of time, or it can be stochastic. Timed Petri nets and the concept of concurrent runs are not a very good match but they do not totally exclude each other. The major part of research on timed Petri net is considered with performance evaluation, i.e. with the calculation and estimation of throughput time, delays etc. of the modeled systems.

As mentioned at some places above, there are different *levels* of Petri nets – from low-level to high-level. Actually, this dimension allows for many more variants than suggested by these terms. Different high-level Petri nets emphasize a syntactial view or a semantical view or a functional view etc. On the highest level in this classification, a Petri net represents an entire class of models which all satisfy a syntactically given specification. These nets are called algebraic Petri nets. They involve algebraic specifications. Any interpretation of such a specification leads to another concrete Petri net.

There exists very many translations and correspondences between Petri nets and *other formalisms* for concurrent systems, some of them mentioned above. A related topic is the integration of nets and other formalisms. For example, transitions can be inscribed by expressions of a programming language. Then every occurrence of a transition corresponds to a run of the respective program, taking the tokens as input and output values. Other integrating approaches combine Petri nets with formal data models. When Petri nets are used in the process of system design then there is no way of using them totally separated from other methods. So integration concepts, as well as respective tools, are necessary. Although some solutions in this directions have been developed in the

last years, we consider this research direction most urgent to further disseminate the very idea of Petri nets in practical applications.

Acknowledgement

We a grateful to anonymous referees for their helpful remarks.

References

1. W.M.P. van der Aalst, J. Desel and A. Oberweis (Eds.) *Business Process Management*. Springer, LNCS 1806, 2000.
2. W.M.P. van der Aalst. Formalization and Verification of Event-Driven Process Chains. *Information and Software Technology*, 41(10):639-650, 1999.
3. J. Becker, M. Rosemann and C. von Uthmann. Guidelines of Business Process Modeling. In [1], pp. 30–49.
4. R. Bruni and V. Sassone. Two Algebraic Process Semantics for Contextual Nets. In this volume.
5. J. Desel and J. Esparza. *Free Choice Petri Nets.* Cambridge Tracs in Theoretical Computer Science 40, Cambridge University Press 1995.
6. J. Desel. How Distributed Algorithms Play the Token Game. In C. Freksa, M. Jantzen and R. Valk (Eds.) *Foundations of Computer Science: Potential–Theory–Cognition*, Springer, LNCS 1337, pp. 297–306, 1997.
7. J. Desel, K.-P. Neuendorf and M.-D. Radola. Proving Nonreachability by Modulo-Invariants. *Theoretical Computer Science*, 153(1-2): 49–64.
8. J. Desel and W. Reisig. Place/Transition Petri Nets. In [25], pp. 122–173.
9. J. Desel. *Petrinetze, lineare Algebra und lineare Programmierung.* Teubner-Texte zur Informatik, Band 26, B. G. Teubner, 1998.
10. J. Desel. Validation of Process Models by Construction of Process Nets. In [1], pp. 110–128.
11. J. Desel, G. Juhás and R. Lorenz. Petri Nets over Partial Algebra. In this volume.
12. J. Desel, G. Juhás and R. Lorenz. Process Semantics and Process Equivalence of NCEM. In S. Philippi (Ed.) *Proc. 7. Workshop Algorithmen und Werkzeuge für Petrinetze AWPN'00*, Fachberichte Informatik, Universität Koblenz-Landau, pp. 7–12, October 2000.
13. C. Ermel and M. Weber. Implementation of Parametrized Net Classes with the Petri Net Kernel. In this volume.
14. R. Eshuis and R. Wieringa. A formal semantics for UML activity diagrams, 2000. Available at http://www.cs.utwente.nl/ eshuis/sem.ps.
15. J. Esparza. Model checking using net unfoldings. *Science of Computer Programming* 23(2): 151–195, 1994.
16. J. Esparza. Decidability and Complexity of Petri Net Problems –An Introduction. In [25], pp. 374–428.
17. M. Gajewsky and H. Ehrig. The PNT-Baukasten and its Expert View. In this volume.
18. T. Gehrke, U. Goltz and H. Wehrheim. Zur semantischen Analyse der dynamischen Modelle von UML mit Petri-Netzen. In E. Schnieder (Ed.): Tagungsband der 6. Fachtagung Entwicklung und Betrieb komplexer Automatisierungssysteme (EKA '99), pp. 547–566, Beyrich, Braunschweig, 1999.

19. H.-M. Hanisch and M. Rausch. Synthesis of Supervisory Controllers Based on a Novel Representation of Condition/Event Systems. In *Proc. IEEE International Conference on Systems Man and Cybernetics*, Vancouver, British Columbia, Canada, October 22-25, 1995.
20. M. Jüngel, E. Kindler and M. Weber. The Petri Net Markup Language. In S. Philippi (Ed.) *Proc. 7. Workshop Algorithmen und Werkzeuge für Petrinetze AWPN'00*, Fachberichte Informatik, Universität Koblenz-Landau, pp. 47–52, October 2000.
21. J. Meseguer, P. Ölveczky, and M.-O. Stehr. Rewriting Logic as a Unifying Framework for Petri Nets. In this volume.
22. J. Padberg and H. Ehrig. Parametrized Net Classes: A uniform approach to net classes. In this volume.
23. C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Univ. Bonn, 1962.
24. W. Reisig. *A Primer in Petri Net Design.* Springer, 1992.
25. W. Reisig and G. Rozenberg (Eds.). *Lectures on Petri Nets I: Basic Models.* Advances in Petri Nets, Springer, LNCS 1491, 1998.
26. W. Reisig and G. Rozenberg (Eds.). *Lectures on Petri Nets II: Applications.* Advances in Petri Nets, Springer, LNCS 1491, 1998.
27. W. Reisig. Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets. Springer, 1998.
28. D. Ross and K. Schoman. Structured analysis for requirements definition. IEEE Transactions on Software Engineering Vol. SE-3, No. 1, pp. 6-15, 1977.
29. F. J. Rump. *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten.* Teubner-Reihe Wirtschaftsinformatik, B. G. Teubner, 1999.
30. J. Rumbaugh, I. Jacobson, G. Booch. *The unified Modeling Language Reference Manual.* Addison-Wesley, 1999.
31. A.-W. Scheer and M. Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In [1] pp. 376–390.
32. E. Smith. Principles of high-level net theory. In [25], pp. 174–210.
33. R. S. Sreenivas and B. H. Krogh Petri Net Based Models for Condition/Event Systems. In *Proceedings of 1991 American Control Conference*, vol. 3, pp. 2899-2904, Boston, MA, 1991.
34. A. Valmari. The State Explosion Problem. In [25], pp. 429-528.
35. H. Weber, H. Ehrig and W. Reisig (Eds.). *Proc. Colloquium on Petri Net Technologies for Modelling Communication Based Systems.* Berlin, October 1999.
36. H. Weber, S. Lembke and A. Borusan. Petri Nets Made Usable: The Petri Net Baukasten for Application Development. In this volume.