Modeling Distributed Systems and Processes with Petri Nets – the Big Picture

Jörg Desel[1,*,†], Julia Fleischer[1,†]

[1]FernUniversität in Hagen, Germany

Abstract

This is another introduction to Petri nets, focusing on modeling as a discipline and on distributed systems as the target of modeling. Based on Stachowiak's fundamental characterization of modeling, we discuss what Petri net models actually model. This leads to a strict distinction between modeled systems and their models, but also between system behavior and model behavior. In a "Big Picture" we locate the typical practices of Petri net modeling of systems. Furthermore, we add the concept of process in this terminology.

Keywords

education of modeling systems, model theory, Stachowiak,

1. Introduction

This contribution is not an introduction to Petri nets per se, but rather an introduction to Petri net modeling. We approach Petri nets not merely as a formal language, but as a modeling tool for complex dynamic systems and their behavior. Furthermore, we ask not only how systems are modeled, but also why. We will provide one big picture that includes the roles of systems in the real world, Petri net models of systems, behavior of systems and of Petri nets, properties of systems and of models, and their analysis. We will identify processes (in the sense of business processes) as specific systems that are embedded in larger systems but have their own behavioral properties. This overall picture will prove very helpful in defining various practices in the field, such as modeling itself (to better understand a system), model validation, model synthesis, and the different variants of system and process mining.

In the following section, we review the state of the art in Petri net modeling with a special emphasis on different paradigms for behavior modeling, and also present our perspective. Section three applies Stachowiak's modeling theory to Petri net modeling in detail and provides precise terminology to distinguish important aspects of Petri net modeling. The Big Picture announced in the title of this work will be presented in Section four. It places all the prepared ingredients in their relationships with each other. This image can be used to clarify and differentiate between different Petri net modeling practices. Section five provides a closer look at the modeling process itself. Finally, Section six is devoted to business processes and their Petri net models. Processes will be interpreted as systems and thus process models as particular system models, which, however, have crucial differences to other models.

2. Petri net models

Most introductions to Petri net theory begin by emphasizing that Petri nets are particularly well-suited for modeling distributed systems, though not exclusively. These introductions typically proceed with the formal definition of nets and the token game, which can be formalized in a seductively simple way in the form of transition sequences. The modeled system, together with its behavior, is not considered further and is often even identified with its model.

PeNGE'25: Petri Net Games, Examples and Quizzes for Education, Contest and Fun. June 24, 2025, Paris, France

*Corresponding author.

†These authors contributed equally.

joerg.desel@fernuni-hagen.de (J. Desel); julia-anna.fleischer@fernuni-hagen.de (J. Fleischer)

https://www.fernuni-hagen.de/sttp/team/joerg.desel.shtml (J. Desel)

Such definitions often overlook several essential facts:

1. Considering a model without its original may be useful for theoretical considerations, but it makes little sense for modeling. The model itself, as well as its behavior and properties, are only valuable if they can be related to what is being modeled.
2. The modeled system is, in most cases, distributed, which means that it consists of components that operate independently of each other, except when they communicate. Furthermore, each modeled system communicates with components that have not been modeled. Therefore, a system model – or, better yet, a model of each system component – must explicitly provide interfaces through which this communication can take place.
3. The core of Carl Adam Petri's original idea was not limited to places, transitions, edges, and tokens, but fundamentally included concurrency as a central assumption. This focus on concurrency is the foundation of the theory that Petri developed. Concurrency is not primarily a property of the behavior of a system model, but of the behavior of the system itself, resulting from the assumption that the system is distributed. Therefore, it is unavoidable to explicitly distinguish and relate the behavior of the modeled system and the behavior of its model, with a particular focus on concurrency.
4. Like any model, a Petri net model of a system is the result of a modeling process and is by no means unique. Stachowiak's general modeling theory teaches us that the relationship between a model and its original follows certain rules, including a pragmatic perspective that states that modeling serves a specific purpose that we must make explicit.
5. Stachowiak's work also shows that modeling always abstracts, omitting aspects of the original (and, if possible, adding nothing that doesn't correspond to the original). The mere decision to model with Petri nets already implies that we will emphasize discrete dynamic aspects and abstract from other aspects. The selected section of the observed world, which we call a system, means abstracting from everything else. And finally, the chosen level of abstraction is one of the decisions made during modeling. Therefore, there is almost never a unique single correct model.

Our aim is to address and consider these aspects. In this paper, we particularly focus on the relationships between the system and the model, as well as between the system and its behavior, which of course should have a strong connection with the model's behavior. As mentioned above, we postulate that the behavior of a distributed system is concurrent (which includes cases where it happens to be sequential, namely, when each individual step depends on its predecessor). A Petri net model, on the other hand, is not distributed in this sense, and the occurrence rule immediately tempts one to define transition (occurrence) sequences, which describe sequential views of behavior. However, since the quality of a model can be measured by the extent to which the behavior of the model matches the behavior of the modeled system, the respective behavioral concepts must match.

Therefore, it is often assumed that, also in system runs, all activities, or more precisely, their occurrences or executions, are ordered. This is justified by a global timeline: every activity occurs at a specific point in time, and if scaled precisely enough, any two activity occurrences can be objectively sequenced. We will call the occurrence of an activity an *action* in the sequel.

This approach is questionable for several reasons: First, there is no publicly accessible and infinitely precise global clock, and it moreover would be of little use for distributed systems. Ultimately, it contradicts modern physics, even though the systems under consideration are usually not based on such phenomena. The assumption that such a clock can be safely assumed, even if it does not exist, contradicts the modeling principle that one may only abstract, not add. Furthermore, it only creates new problems that would not otherwise exist: For example, one must then ask in which order independent activities are carried out and treat this formally like a selection from various alternatives.

Other groups of scientists, including Petri himself, are attempting the desired alignment of behavioral concepts on the model side. They do not represent model behavior by occurrence sequences, but rather by partially ordered structures whose elements represent actions together with pre- and post-comditions and whose dependencies represent the causal structure in the system's behaviour. In particular, concurrency at the system level is expressed by the lack of order between these elements.

These partially ordered structures are often nicely formulated as Petri nets, so that this approach could also be used as an example for Petri net morphisms. However, this approach hasn't been widely adopted because the definition of partially ordered distributed system runs is already too complicated. Moreover, these partially ordered model runs are actually constructed sequentially, so that an occurrence sequence is secretly considered beforehand. We also believe that the arguments for distributed, partially ordered runs that apply at the system level are not transferable to the model level, because a model is just a model, usually not actually physically distributed, but rather only a mathematical structure or a suitable syntactical representation of it.

Our current approach combines both perspectives on behavior modeling: The behavior of a system is given by partially ordered runs (we intentionally avoid the term "process" here, as it will later refer to a specific subset of systems, each of which has its own runs). However, it is fine to consider occurrence sequences as the primary behavioral concept of a Petri net. But what about the relationship between these concepts? There are two possible answers, both described in detail in the literature (albeit in different contexts). First, we can consider linearizations of partially ordered runs at the system level, i.e., ordered sets of activity occurrences that do not contradict the partial order. These sequences are often called observations, since an assumed – naturally non-distributed – observer of all system activities could note these events in such an order (another observer might detect a different linearization of the same run). We can then relate these observed sequences to the occurrence sequences of the model. Second, as already mentioned, from occurrence sequences of a Petri net one can construct according partially ordered representations of runs. This is done either step by step, by adding the next transition occurrence to an already existing partial order and placing this new element only "after" those transition occurrences that generated tokens that are now consumed [1]. Or we apply concepts from trace theory [2], which transfer an independence relation between transitions of a Petri net to their occurrences in sequential runs.

Finally, we take up the topic of time once again. As already mentioned, the assumption of a generally available global clock, which leads to a linearization of all actions of a system run, is neither helpful nor realistic. However, there are other dependencies related to time in many systems. Some examples:

- At a railway barrier, one expects that a train can pass safely one minute after it is closed.
- An airbag opens in time after a collision to save lives.
- A processor with a known clock speed will make the result of a multiplication available in a timely manner if this value is subsequently used by other processors.

In all of these cases we see a dependency related to time, which is not related to causality, i.e., the train does not wait for crossing cars, the driver's head does not wait for the airbag etc. One is tempted to express these relationships in terms of numbers that represent time. In Petri nets, however, such dependencies are strictly distinguished from causal dependencies, because their validity depend on the functioning of cars, airbags or processors. If they do not function properly, there will be another behavior, usually an undesirable one. However, this behavior is not excluded by definition of the model and still visible. Instead, it is possible to express such dependencies through additional assumptions (specifications) and to distinguish behavior that satisfies these assumptions from general behavior. This is admittedly somewhat cumbersome, but it is a consequence of the fundamentals of Petri nets in their original form. Some authors have proposed alternative approaches to integrating time into Petri nets and their semantics, but usually at the cost of raising more questions than answers regarding other considerations.

3. Modeling Theory

A Petri net can be thought of as a graph, a mathematical element, or a syntactic unit that can be entered into an analysis tool [3]. However, this view does not take into account that a Petri net is a model and therefore something is modeled by the Petri net. Since this work is about modeling systems and

processes with Petri nets, we will take a closer look at models and the relationship between a model and its corresponding original, i.e. the thing modeled by the Petri net.

The General Model Theory (translated from German "Allgemeine Modelltheorie"), introduced by the philosopher and mathematician Herbert Stachowiak about 50 years ago (see [4], unfortunately only available in German) provides us with the following basic model properties that properly characterize models in computer science and related disciplines:

Fundamental Model Properties

- **Mapping.** A model is always a model of something, i.e. an image or a representation of an original, which can be natural or artificial. The original can be constructed using syntactic means, can be an idea or something from the physical world. It can be naturally occurring, engineered, or given in some other way. When models and originals are interpreted based on their attributes, model attributes are mapped to the attributes of the original.
- **Reduction.** A model does not capture all attributes of the original but only those relevant to the model creator and/or model user. In this sense, something is reduced. In particular, an original is usually not a model of itself. Stachowiak's did not use the word *abstraction*, which is one of the computer scientist's favorite terms. When we abstract from something of the original then we intentionally do not include it in the model.
- **Pragmatism.** A model is not per se uniquely assigned to its original. It fulfills its replacement function for a specific (human or artificial) user for a specific purpose and does so within a certain time interval.

Let us try to instantiate these rules for Petri net modeling. For illustration, and because it is obligatory for this workshop, we will use an example, the well-known vending machine [5]:

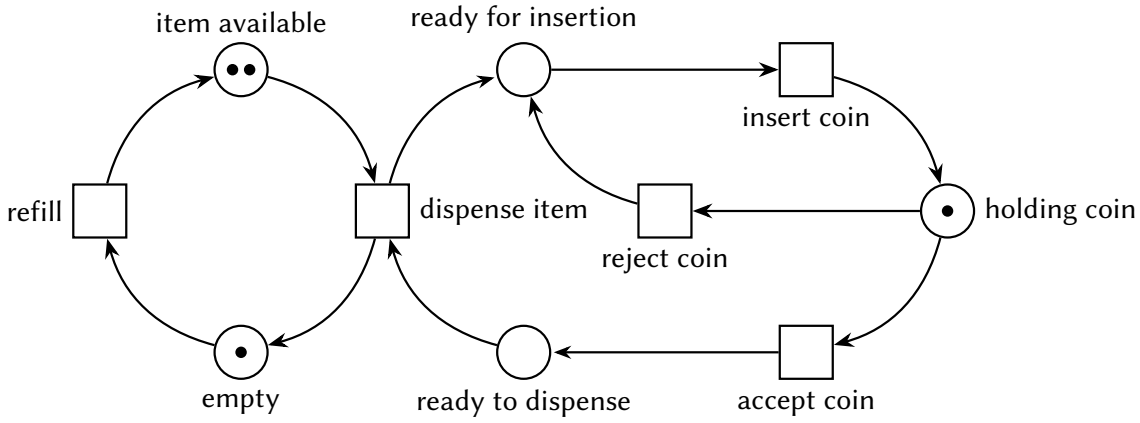

Figure 1: A model of a vending machine

Mapping: What do we model with Petri nets?

Each Petri net model has an original, and there are several examples for this original. In particular, a Petri net model can refer to something that exists or it is used as a blueprint for something that will exist (or at least could exist). The original can be physical, just an idea, a piece of mathematics or something formulated syntactically, like an algorithm. All these originals have in common, that they have a behavior. Therefore, there are runs (or executions) and at least local conditions whose values change during a run. Notice that behavior is not only given by a set of possible runs; for example, points of decisions also play a role for behavior specification. However, in this contribution we restrict to the view that behavior is defined by the possible runs.

Each original is embedded in a bigger original, say the real world. It is essential to identify the interfaces between the modeled and the unmodeled. The modeled segment will be called *system* in the sequel (notice that, in the Petri net world, this term was also used for Petri nets with initial marking, but here we need it for a better purpose). The concept of an interface directly enables the definition of subsystems of a system and interfaces between subsystems as well as between a system and its subsystem and between a system and is environment.

In the vending machine example, we model its physical unit (on the left hand side) and its control (on the right hand side). Both parts can be viewed as subsystems with respective interfaces "dispense item". Possible users belong to the unmodeled real world, with interface "insert coin", whereas operating staff will interact e.g. via "refill".

You will often encounter Petri nets whose elements have abstract names like *a*, *b*, *c*. These may be useful within theory, but they are not models in the true sense.

Reduction: What do Petri nets abstract from?

Since each Petri net models only a part of the real world, one could argue that it abstracts from the rest. More importantly, like any modeling language, Petri net models abstract from everything that cannot (or can hardly) be expressed with nets. For example, the vending machine model makes no mention of the color, weight, or price of the machine. Therefore, the choice of modeling language predetermines certain abstractions due to their representational bias.

Even when considering behavioral aspects, Petri nets typically abstract from all aspects of time. This does not apply to the global clock mentioned above, since it does not exist and is therefore not subject to abstraction. However, in Petri nets transitions model timeless state changes, which seldom refer to real changes. In particular, Petri nets abstract from continuous behavior, as expressed, for example, by differential equations.

A concrete Petri net model can also abstract from the data if it is not relevant to the behavior under consideration. And it always represents its original on a certain level of abstraction, thereby abstracting from irrelevant details.

Pragmatism: What is the purpose of a Petri net model?

Let us answer this question by an enumeration of possible purposes. A Petri net model might help to

- understand the behavior of the modeled system,
- understand the (relevant) structure of the modeled system,
- gain insights into the behavior of the modeled system by analysis,
- gain insights into the behavior of the modeled system by simulation,
- prove properties of the modeled system,
- specify a system,
- run a real system, if the model is an interpreted part of it,

and sometimes the purpose of a Petri net is purely didactic.

4. The Big Picture

For our further considerations, it is helpful to summarize the terminology which consistently distinguishes between systems and their models and between the respective structure of system and model and their behaviors, see Table 1.

The Big Picture, presented in Figure 2, comprises a system within the real world with subsystems, models, and their respective behaviors. It allows to illustrate and structure typical problem scenarios and corresponding modeling practices:

Table 1
Terminology in system and model behavior

Level	Elements	Relations	Ordering
system structure	activities (and conditions)	potential dependencies	
system run	actions (occurring activities)	causal structure	partially ordered
event log	observed actions	sequence	totally ordered
Petri net model	transitions (and places)	flow relation (arcs)	
Petri net run	events (transition occurrences)	causal dependencies	partially ordered
occurrence sequence	transition occurrences	sequence	totally ordered

1. **Improving the model.** If discrepancies exist between expected system behavior and model behavior, the model must be revised. This requires comparing both behaviors. (Validation)
2. **Exploring system behavior.** The system is known, but its behavior is unknown or cannot be observed directly. In this case, a model is used to simulate the behavior, which serves as a proxy for the actual system behavior. (Simulation)
3. **Deciding system properties.** If the model correctly reflects the system, properties like deadlock-freeness can be assessed by analyzing the model. (Analysis)
4. **Mining a model.** If the system itself is unknown, but behavioral data is available (e.g., event logs), a model can be derived from this behavior. (Process mining)
5. **Synthesizing a model.** If the system does not yet exist but desired behavior is known, a model can be constructed to generate a system with that behavior. (Synthesis)

When we refer to "behavior" of systems or models in these practices, we always have to distinguish between sequential behavior (e.g., occurrence sequences of Petri nets) and partially ordered behavior. As mentioned above, these two concepts are related by observation (from partially ordered behavior to occurrence sequences) and construction of partial orders from sequences.

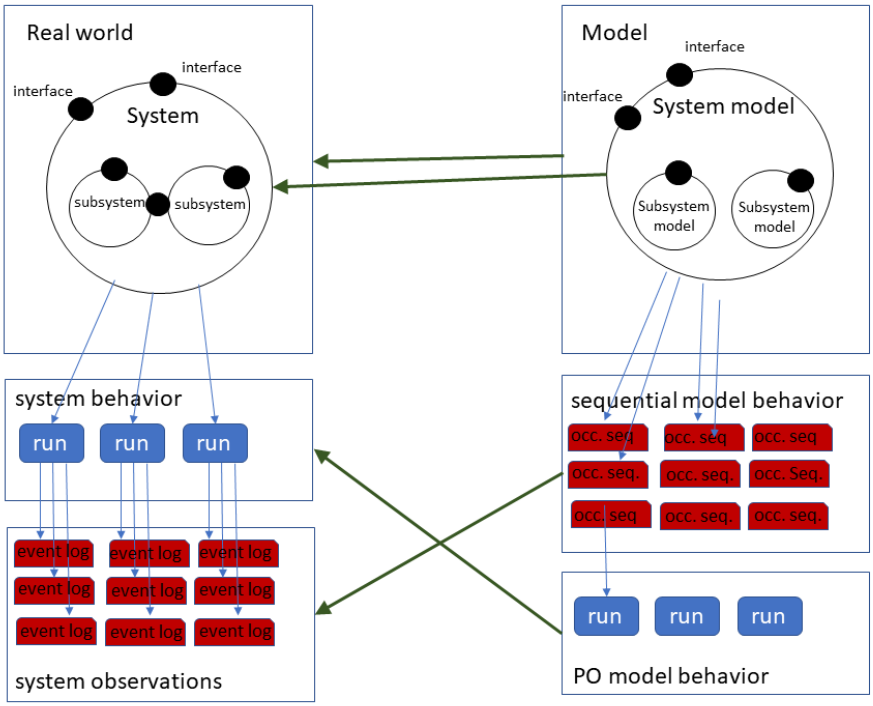

Figure 2: The Big Picture

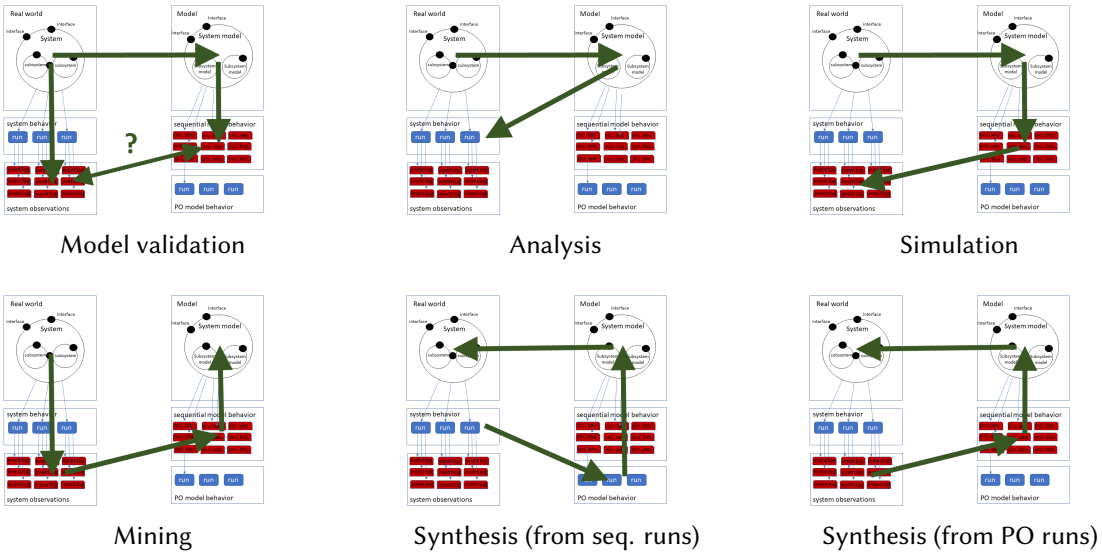

Figure 3: Practices on Petri net models

5. Modeling

The reader might have noticed that the first mentioned practice was validation of models, instead of the practice of modeling (i.e., construction of a model) itself. The Big Picture contains a representation for "System", the original of the model considered. Of course, this system representation does not exist in modeling applications. By drawing any symbol that represents the original of the model under consideration, this system representation itself becomes a model.

For models created by a human modeler, a *cognitive model* of the system, encompassing the aspects to be modeled according to the *modeling purpose*, exists before a system *abstraction* is *formalized* in a suitable language. This cognitive model depends both on the *modeling purpose*, which determines the abstraction, and on the expressive power of the *modeling language*, which restricts the aspects that can be modeled. As shown in Figure 4, meaning can also be assigned in the opposite direction: The cognitive model describes the *perception* of the system, and the *interpretation* of the formal model leads back only to the cognitive model and can only be compared with it. So a (formal) model is a deliberate twofold abstraction: it suppresses phenomena deemed irrelevant for the modeling purpose and codifies the remaining ones in a formal vocabulary.

This refined view is relevant when it comes to model quality and model validation. If a model is incorrect with regard to the system to be modeled, the error can lie in the abstraction - in which case the cognitive model is also incorrect - or in the formalization - in which case the cognitive model is correct.

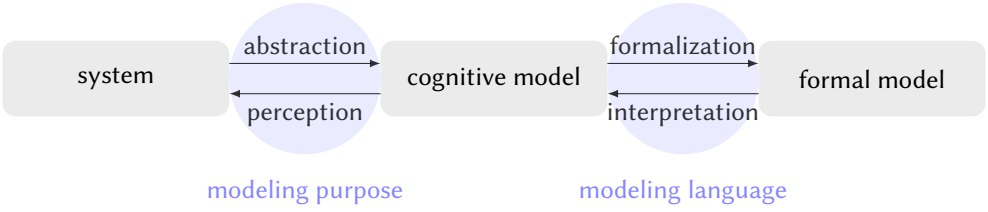

Figure 4: The cognitive model

6. Processes as Systems

One of the most important applications of Petri net modeling is modeling of business processes. In particular, process mining has become a well-established field of research. How do the concepts developed specifically for process models fit into our Big Picture? We consider a process to be a specific subsystem, whose supersystem is typically an enterprise information system or something similar. Since business processes, by definition, have a start and an end, interfaces always exist, both for the system and for its model, because a process is started by its environment and its end is identified by the environment

Whereas for other systems liveness and boundedness are often desirable properties, in the case of processes, we aim for *soundness* [6] as a correctness criterion. In fact, the soundness property refers to the liveness and boundedness of the supersystem of a process. Therefore, it can be analyzed by studying a system that includes the process, with the rest of the supersystem abstracted by a single transition.

Process mining is the most common practice in this context — very often, neither the process nor a corresponding model is initially known. The result of process mining is one or several process models, which already indicates that the existence of a unique process original is debatable. In practical applications, when asked to define "the process itself" one is typically referred to descriptions, rules, or documents -— all of which are models themselves. Therefore, the entire field is reminiscent of The Emperor's New Clothes — and the upper-left element in the Big Picture figure is, in a sense, fake for business processes.

7. Conclusion

The main contribution of this paper is a figure that relates models to their original, which we call the system, as well as to system and model behavior. We discussed these relations in depth and showed how typical practices of Petri net modeling can be interpreted as paths within this figure.

The approach could be refined in various directions. For example, we could explicitly mention places, transitions, and the flow relation and their counterparts in systems, thereby reflecting the very nature of Petri nets. Another direction is to study Stachowiak's modeling theory in greater depth.

This paper is intended as a contribution to Petri net didactics. Often, students only learn token games and analysis applications, each of which gains its deeper meaning only within the full picture. Therefore, it might be desirable to begin with this Big Picture and to add this orientation to all topics related to Petri nets.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] W. Reisig, A primer in Petri net design, Springer Compass International, Springer, 1992.

[2] V. Diekert, G. Rozenberg (Eds.), The Book of Traces, World Scientific, 1995.

[3] J. Desel, G. Juhás, What is a Petri net?, in: H. Ehrig, G. Juhás, J. Padberg, G. Rozenberg (Eds.), Unifying Petri Nets, Advances in Petri Nets, volume 2128 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 1–25.

[4] H. Stachowiak, Allgemeine Modelltheorie, Springer, Wien, 1973. https://glossar.hs-augsburg.de/Stachowiak,_H._(1973):_Allgemeine_Modelltheorie.

[5] J. Desel, J. Esparza, Free choice Petri nets, Cambridge University Press, USA, 1995.

[6] W. M. P. van der Aalst, K. M. van Hee, Workflow Management: Models, Methods, and Systems, Cooperative information systems, MIT Press, 2002.