From Human Knowledge to Process Models

Jörg Desel

Angewandte Informatik, Katholische Universität Eichstätt, Germany
joerg.desel@ku-eichstaett.de
www.informatik.ku-eichstaett.de

Abstract. This contribution suggests a novel approach for a systematic generation of a process model in an informal environment. It is based on the claim that the knowledge about the process to be modelled is distributed in several involved people's minds. Some people have knowledge about the general process where the single activities are on a high level of abstraction and have to be refined. Other people only know something about some detail of the process, i.e., about the refinement of an activity of the general process which defines a subprocess. Moreover, it is assumed that these domain experts can more easily define instances, i.e. runs, of the general process (of a subprocess, respectively) than the process itself. The approach employs new techniques developed for process mining and Petri net synthesis and adapts these techniques to generate processes from example runs. It is based on a formal definition of partially ordered processes, which allows to proceed in a modular way: Subprocesses and general processes are generated independently. Finally, it is argued that the approach is a suitable first step in a general method for process definitions which is followed by validation techniques.

Key words: Business Process Modelling, Petri nets, Synthesis

1 Introduction

During the last decades, information system modelling emphasized the modelling of data as a prerequisite of data base design. The modelling of processes was also considered, but only in the last years process models received attention comparable to data models [21]. This development is closely connected to the raise of Business Process Management [4, 34], of workflow management systems [1], of process-centric ERP systems and finally of web services.

It is well known that for both, data and process models, the first phase of modelling is of particular importance. Only if models that faithfully present the part of reality (or of intended reality) to be modelled are used in system design, the final system can be expected to behave correct according to the requirements. Conversely, any error in an early stage of modelling will cause very costly redevelopments in later phases.

For data models, suggestions for a systematic analysis of requirements have been developed since a long time and have proven to be very useful in practice [5, 27]. In particular, the research done within the project KCPM (Klagenfurt

Conceptual Predesign Model) in Klagenfurt shows how to extract formal model components from natural language documents provided by the system experts and users [30].

For process models, there is no such generally agreed procedure. Again, the KCPM people made suggestions since a long time (see e.g. [28], where pre- and post-conditions of actions are extracted from text and used as local vicinity of corresponding transitions of Petri net models; more recent developments are given in [29, 23, 31]).

The purpose of this paper is to tackle the very same question: How can we derive a process model in/from an informal environment? So instead of deriving a process model from another model by any kind of transformation (where correctness is a matter of verification), we do not assume any previous representation of the model to be constructed but only distributed knowledge about the process in people's minds.

Very often, semi-formal modelling languages are suggested to support this first step in process modelling. However, there is no clear definition of "semi-formal". Sometimes, a language is called semi-formal if it has a defined syntax but no (or more than one) semantics. Whereas people often praise missing unique semantics as flexibility, I cannot see any advantage of semi-formality if not even author and recipient of a model are sure to communicate the same object. In particular, for this kind of conceptual models it should be clear what exactly is modelled and what is left open [12]. This approach, starting with an informal model and adding semantics in a fuzzy way will never guarantee that the final, formal model matches the initial intuition.

The approach sketched in this paper is different. It starts with formal objects from the beginning but keeps these objects as simple as possible. In the area of processes (including alternatives, variants etc.) the simplest concept is on the instance level: a single run of a process, sometimes called a case or a run of a case. We assume that the relevant persons can more easily describe runs than starting with a process description (which is on the type level and can be as complicated as an arbitrary algorithm). Whereas the definition of a run for a given process is often obvious, the converse direction is more involved: Given a set of runs, what is the corresponding process?

A related question is answered by Petri net synthesis theory [22, 9, 6, 26] which either starts with a reachability graph or with a set of sequential runs or with a set of partially ordered runs. However, synthesis aims at a construction of precisely the process with the specified behavior or – if such a process does not exist – a process with minimal behavior including the specified one. Thus, often an unnecessarily complicated process is constructed.

Another related research area is process mining [2, 32]. Mining techniques aim at constructing a simple process from a large set of recorded behavior, given by so-called process logs. This is also not exactly our setting. We start from some representative examples and try to construct a process which is as simple as possible, can behave like the example runs and does not have too much additional behavior. [3] shows how techniques from synthesis theory can

be used for process mining. The same techniques will be applied in our approach to generate process models from sequential or from partially ordered runs.

A second assumption of this paper is that relevant people know something about the runs of the general process on an abstract level. In other words, the single activities occurring in the runs can be refined to subprocesses. Other experts might not know the entire process but have knowledge about runs of a subprocess.

An obvious way to deal with this kind of vertical modularity is to replace each occurrence of a coarse activity in a given run by one (or all) of the runs of the subprocess and then to apply synthesis techniques to the set of all runs refined in this way. Another way to deal with the same problem is to first generate subprocesses from their runs, then to generate the general process on the abstract level, and finally to refine its activities by the subprocesses. It could be assumed that in both ways we end up with the same process definition. However, this does not hold if only sequential runs are considered, see the following example: Assume we have to activities x and y which refine as follows: x has subactivities a followed by b whereas y has subactivities c followed by d. Assume moreover that x and y occur concurrently. In a sequential setting, the general process is specified by example runs $x\,y$ and $y\,x$. Specifying x by the run $a\,b$ and specifying y by the run $c\,d$ and replacing x and y by these runs yields the following possible runs of the refined process: $a\,b\,c\,d$ and $c\,d\,a\,b$. Taking the other approach, we first construct a process with two concurrent activities x and y. Replacing then these activities by processes allowing only for sequences $a\,b$ and $c\,d$ respectively leads to a process where the subactivities are mutual concurrent. So several more runs are possible, for example $a\,c\,b\,d$ and $a\,c\,d\,b$. Taking the Petri net formalism, the process is shown in Figure 1. We argue in this contribution that this problem is solved by using nonsequential semantics. Instead of restricting runs to sequences we consider partially ordered runs, i.e., runs where concurrent occurrences of activities are not ordered by a causality relation.

The following section will provide some necessary formal definitions. Section three shows that partial order semantics indeed solves the above problem. Section four uses this observation and introduces our approach to process modelling. In Section five we relate the approach to a previously introduced validation approach, i.e., we argue that both approaches nicely work together. Finally, the concluding Section six provides some details about our further research plan on this topic.

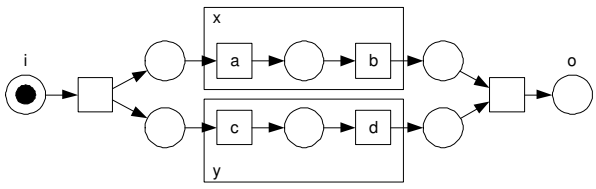

Fig. 1.

2 Formalities

We use the Petri net formalism (see e.g. [33])to represent processes. Petri nets have been a standard formalism for the representation of processes since a long time (see e.g. [10]) and they are the underlying formalism for many other languages [15, 19].

We use the usual notions and definitions of a Petri net (S, T, F, M_0). Capacity restrictions, arc weights and inhibitor arcs are not considered. A Petri net is called 1-bounded if no marking reachable from M_0 assigns more than one token to a place.

Definition 1. *A process is a Petri net (S, T, F, M_0) with two distinguished sets of input- and output-transitions $T_i, T_o \subseteq T$. A process is called safe if the Petri net*

$$(S \cup \{s\}, T, F \cup (\{s\} \times T_i) \cup (T_o \times \{s\}), M_0 \cup \{s, 1\})$$

is a 1-bounded Petri net, where s is a new place.

We assume for the rest of the paper that all processes are connected Petri nets.

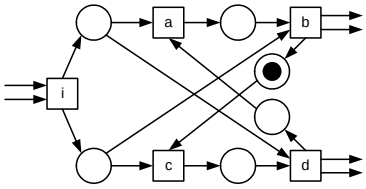

Fig. 2. Example process with input transitions $\{i\}$ and output transitions $\{b, d\}$

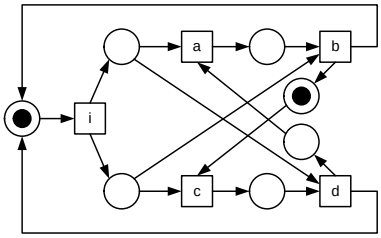

Fig. 3. The process is safe if this Petri net is 1-bounded

1-boundedness of the place s implies that transitions of T_i and of T_o occur strictly alternatingly, starting with a transition of T_i. 1-boundedness of all other places implies that no transition can ever occur concurrently with itself: Since the net is connected, every transition has at least one place in its pre-set or in its post-set, which carries at most one token.

In the sequel we will distinguish main processes and subprocesses which are formally both processes in the above sense.

A main process is considered connected to an additional place s which is initially marked with one token. s has an empty pre-set. The post-set of s is T_i. Sometimes another place s' with pre-set T_o is considered.

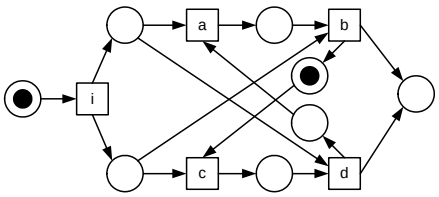

Fig. 4. The process of Figure 2 considered as a main process

A subprocess refines a transition of another process. It replaces the transition, the transition's input places are connected to the input transitions whereas the output transitions of the subprocess are connected to the post-set of the transition. For a formal definition, let us first notice that we will not distinguish isomorphic processes, i.e., we are free to consistently rename places and transitions of processes.

Definition 2. *Let $P = (S, T, F, M_0)$ be a process with a transition t and let $P^t = (S^t, T^t, F^t, M_0^t)$ be a process with input-transitions T_i^t and output-transitions T_o^t. Assume w.l.o.g. that the elements of P and of P^t are disjoint. The refinement of P w.r.t. transition t and process P^t is defined as*

$$(S \cup S^t, T \cup T^t \setminus \{t\}, F \cup F^t \cup ({}^\bullet t \times T_i) \cup (T_o \times t^\bullet), M_0 \cup M_0^t)$$

where ${}^\bullet t$ and $t^\bullet$ refers to the process P.

Figure 4 shows a process and Figure 5 shows the same process when transition t is refined by the process of Figure 2.

Different transitions of a main process can be refined by the same subprocess. As mentioned above, we do not distinguish isomorphic processes and assume that instead disjoint copies of the subprocess are used.

Proposition 1. *Assume a process P with two transitions u and v and respective refinements P^u and P^v. Then the processes obtained by first refining u and then v and by first refining v and then u are isomorphic.*

Hence the order of refinement does not matter. So we can speak about refinement of all refinable transitions in one step. Since a subprocess can again have transitions to be refined, an arbitrary refinement hierarchy can be constructed.

The intuition of the behavior of a safe process is that first one of its input transition fires and after some internal behavior one of its output transition fires, provided the process does not run in a deadlock. It is not possible that two input transitions occur without an intermediate output transition. Hence the external behavior of the subprocess resembles the behavior of a transition,

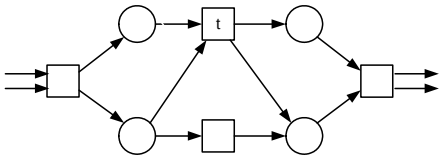

Fig. 5. An abstract process...

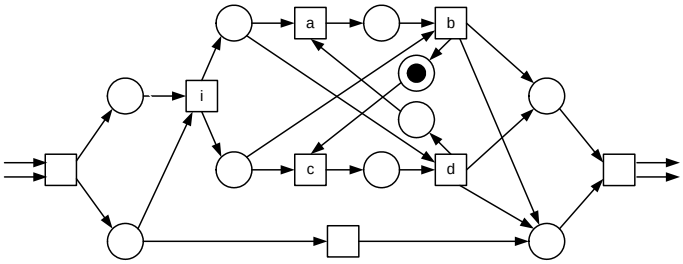

Fig. 6. ...and its refinement w.r.t. t and the process of Figure 2

with the difference that first the input tokens are consumed and later output tokens are produced. Conversely, the surrounding net resembles the behavior of the additional place s shown in Figure 3 with the difference that the input tokens show up after the output tokens have been produced. This results in the following observation:

Proposition 2. *If processes P and P^t are safe, then the refinement of P w.r.t. one of its transitions t and P^t is safe, too.*

3 Concurrent Runs and Refinement

There are several suggestions how to define the behavior of a Petri net. The best known is based on sequential runs, formalized by occurrence sequences, where an occurrence sequence is a sequence of transition names that can occur one after the other. A marking graph can be viewed as a condensed version of all occurrence sequences, together with the intermediate markings, where additionally the branching behavior is represented.

Partial order semantics of Petri nets have originally be defined by means of so-called occurrence nets which are mapped to the original net (see e.g. [33]). An occurrence net together with this mapping is often called process. This instance use of "process" contradicts today's use of "process" in "business process" on the type level. Therefore, the term "run" is used instead for a single instance of a process in this paper.

The main aim of this section is to motivate and show that the diagram of Figure 7 commutes for partial order semantics. In the introduction it was shown that this is not the case for sequential semantics.

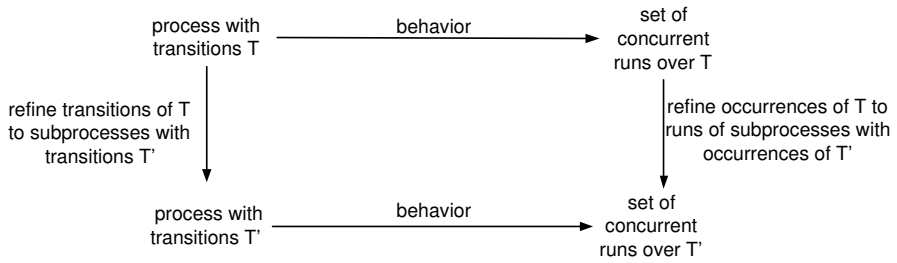

Fig. 7. Behavior and refinement

The following definition is nearly the usual one for partial order semantics. However, we require that each run leads to the end of the process. In other words, we consider maximal runs and not arbitrary prefixes and we rule out runs that run into a deadlock.

Definition 3. *Let $P = (S, T, F, M_0)$ be a safe process with input-transitions T_i. A (concurrent) run of P is a Petri net $R = (B, E, G, I)$ together with mappings $\rho_B \colon B \to S$, $\rho_E \colon E \to T$ satisfying the following properties:*

- *places of R are not branched, i.e., $|{}^\bullet b| \leq 1$ and $|b^\bullet| \leq 1$ for each b in B,*
- *R is acyclic, i.e., $G^* \cap id_{|B \cup E} = \emptyset$,*
- *the flow relation G of R agrees with the flow relation F of P, i.e., for each e in E, ρ_B induces bijections from ${}^\bullet e$ to ${}^\bullet \rho_E(e)$ and from $e^\bullet$ to $(\rho_E(e))^\bullet$,*
- *exactly one transition e in E is mapped by $\rho(e)$ to a transition of T_i,*
- *exactly one transition e in E is mapped by $\rho(e)$ to a transition of T_o,*
- *$I(b) = 1$ for each b satisfying ${}^\bullet b = \emptyset$ and $I(b) = 0$ for all other places b.*
- *initial places of the run are mapped to initially marked places of the process, i.e., ρ_B induces a bijection from $\{b \in B \mid {}^\bullet b = \emptyset\}$ to $\{s \in S \mid M_0(s) = 1\}$.*

Since the process P is assumed to be safe and by the general assumption that each transition has at least an input place or an output place, there are no transitions with empty pre-set except input transitions. Moreover, the fifth item could be relaxed to"at least one transition..." because safety ensures that no output transition occurs more than once.

We aim at comparing runs of a process with runs of its refinement. Places and transitions of runs are related to places and transitions of processes respectively by the mappings ρ_B and ρ_E. We also have to relate elements of a subprocess to the refined transition. Therefore, given a process P with a transition t and a process $\overline{P}$ resulting from a refinement of t by a subprocess, we define a mapping ϕ from elements of $\overline{P}$ to elements of P [7]. ϕ maps all elements of the subprocess to t and is the identity function for all other elements of $\overline{P}$. Abusing terminology, we will call the process constructed by several subsequent transition refinements still a refinement of the original process. Its associated mapping ϕ is defined as the composition of the mappings associated to the stepwise refinements. In other words, ϕ maps each element of the refinement to the associated transition of the original process.

Transitions in runs represent occurrences of transitions in processes. If a transition is refined by a subprocess, the corresponding refinement of the transition occurrence in the run is done by a run of the subprocess. Formally, we use the same definition as above. Therefore, we have to define runs as particular subprocesses. This is done in the obvious way: The set of input transitions consists only of the unique transition of the run which is mapped to an input transition, and likewise for output transitions.

Proposition 3. *Let P be a process with a transition t and let P^t be a refinement of t. Let $\overline{P}$ be the resulting process. Each run $\overline{R}$ of $\overline{P}$ is a refinement of a run R of P where each occurrence of t (i.e., each transition of R mapped to t by ρ_E) is refined by a run R^t of P^t. Conversely, each refinement of a run R of P with respect to all occurrences of t (refined by a run R^t of P^t each) is a run of $\overline{P}$.*

In other words, the diagram given in Figure 7 commutes.

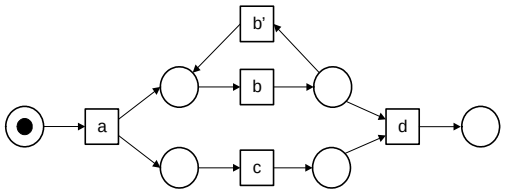

Fig. 8.

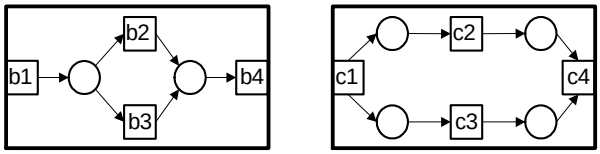

Fig. 9.

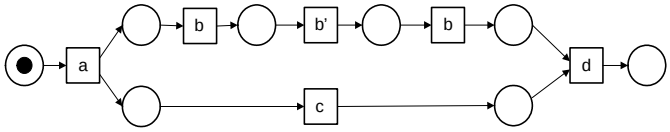

Fig. 10.

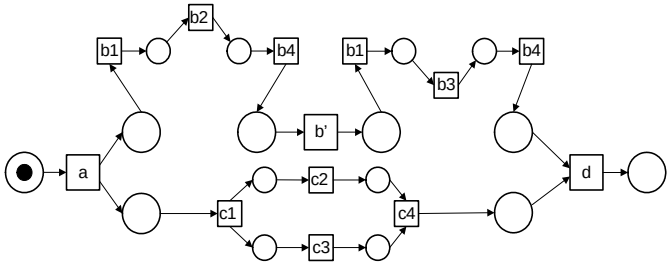

Fig. 11.

Figure 8 shows a process (considered a main process here, hence the input and output place), Figure 9 shows refinements of transitions b and c, Figure 10 shows a run (where for the sake of completeness also a marked input place and an output place is added) and Figure 11 shows both, a run of the refined net and a refinement of the run of Figure 10.

There exist other notions of concurrent runs, given by partial languages. Each run of a process defines a partial order on the transitions occurring in the process in an obvious way: A transition t precedes a transition t' if there is a directed path leading from t to t'. Each partial order over the same transitions including this order is sometimes considered a possible execution of the process. It is known that linearizations of the partial orders given by all runs correspond to all occurrence sequences, when transitions are replaced by their respective images according to the mapping ρ_E. It is well known how to construct runs from a given process [33] and it is straight-forward to check whether a Petri net is a run of a process. The same is less obvious for partial language executions, but [24] (supported by the tool described in [25]) shows that there is an efficient way to answer this question, too. For runs as defined above, [20] (based on [13]) shows how to obtain a process by folding a given set of runs such that these runs are actually runs of the process. For partial languages, a synthesis algorithm is provided in [26].

4 Process Construction using Synthesis and Mining Techniques

Although this section is the core of the paper, it only roughly describes how to construct a process from given runs of different abstraction levels, provided by experts. The motivation was given in the introduction.

1. First, identify start conditions, start actions and end actions of the process to be defined.
2. Let relevant people define runs of the process on an abstract level.
3. Agree on the abstract actions that occur in these runs.
4. Synthesize a process from the runs using the techniques from [20, 26, 3].
5. Validate this process by generating runs (see next section).
6. Identify actions that have to be refined.
7. Identify experts that can provide information (namely runs) for these actions.
8. Agree on actions that occur in these runs.
9. Synthesize a process from the runs using the techniques from [20, 26, 3].
10. Validate this process by generating runs (see next section).
11. Iterate the procedure if there is a need for a higher refinement hierarchy.
12. Construct the entire flat process by refining all transitions on the abstract level (possibly to be repeated for the next levels).
13. Analyze this process w.r.t. appropriate correctness criteria, such as liveness, using known techniques for Petri nets.

5 Validation by Generation of Runs

As seen in the previous section, the constructed processes are validated at various steps. Whereas verification means to automatically or semi-automatically prove desired properties, validation aims at checking correctness w.r.t. the reality to be modelled [16, 17]. Starting with the VIP project [8], we have developed a validation technique which is based on the construction of runs from a given process. Additionally, desired properties can be edited in the process and checked for all generated runs. This way, the process as well as the specification of properties can be validated by the user of the VIP-tool. For example, he can see which runs are ruled out by a behavioral specification and which runs agree with the specification. If the process is modelled correctly, this way the specification is validated. If the specification can be assumed to hold for the process, this way the process is validated.

In our setting, the VIP approach is used to validate the generated processes on different levels. Users are the respective experts. If it turns out that the generated process can generate undesired runs then an according specification is formulated, validated as described above and finally implemented by an according modification of the process. This procedure is iterated as often as necessary.

Finally, the specifications used in the last step of the procedure described in the previous section is first validated by the VIP tool before it is used to prove correctness of the final process.

The process definition of this paper is closely related to the definition of workflow nets in [1]. However, an important difference is that workflow nets are assumed to have no memory. They are executed starting with an empty marking. In contrast, we assume that processes can remember their past. For example, the process of Figure 2 will execute c and d in its first run, and a and b in its second run. So also the specification of runs has to take additional conditions into account which determine which variant of the process has to be executed.

6 Conclusion

This contribution presents the idea of a systematic stepwise construction of a process net in a completely informal setting. Roughly speaking, this idea copied ideas from conceptual data modelling. In particular, I looked at the KCPM project and its predecessors.

Refinement and the relation between processes and related runs only rapport for partial order semantics, as illustrated in the paper. The idea also heavily depends on techniques to automatically derive processes from runs, techniques which have been developed for synthesis and for mining in the last years. Only recently, such techniques have been developed for partially ordered runs, which is necessary for this approach.

Since we only presented the idea, most of the work has still to be done. There are various technical questions. For example, it is important to reduce the complexity of the synthesis techniques. We are quite confident that the modular

approach presented in this paper helps to keep the single processes sufficiently small. Whereas this modularity is vertical, a horizontal composition of modules would be useful in this setting, too.

Another question which appears to be of importance is how to deal with different granularity of actions formulated by different experts and related terminology problems. Since these questions are not specific to process modelling, results from data modelling might be reusable.

The most important and last point to be mentioned is that the research shall not be guided by theoretical questions but rather by application examples and solid evaluation. Only this way the really important problems will show up, and only this way a really useful concept can arise.

References

1. van der Aalst, W.M.P., van Hee, K.: Workflow Management – Models, Methods and Systems. MIT Press (2002)
2. van der Aalst, W.M.P.: Finding structure in unstructured processes: the case for process mining. 7th International Conference on Application of Concurrency to System Design (ACSD), IEEE (2007) pp. 3–12
3. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. 5th International Conference on Business Process Management (BPM), LNCS 4714, pp. 375–383, Springer (2007)
4. Business Process Management. LNCS 1806, 2678, 3080, 4102, 4714. Springer (2000-2007)
5. De Antonellis, V., Demo, G.B.: Requirements Collection and Analysis. Methodology and Tools for Data Base Design, pp. 9–24. North-Holland (1983)
6. Darondeau, P.: Synthesis and control of asynchronous and distributed systems. 7th International Conference on Application of Concurrency to System Design (ACSD), IEEE (2007) pp. 13–22
7. Desel, J.: On abstractions of nets. Advances in Petri Nets 1991, LNCS 524, Springer (1991) 78–92
8. Desel, J., Oberweis, A.: Validierung von Informationssystemen durch Auswertung halbgeordneter Petrinetz-Simulationslufe. Formale Grundlagen für den Entwurf von Informationssystemen. Informatik-Berichte der Universität Hannover Nr.03/94, pp. 132–138 (1994)
9. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Informatica 33, pp. 297–315 (1996)
10. Desel, J., Oberweis, A.: Petri-Netze in der Angewandten Informatik. Wirtschaftsinformatik 38 Nr. 4, pp. 359–367 (1996)
11. Desel, J., Oberweis, A., Reisig, W., Rozenberg, G. (eds.): Petri nets and Business Process Management. Dagstuhl-Seminar-Report Nr. 217 (1998)
12. Desel, J.: Formaler Umgang mit semi-formalen Ablaufbeschreibungen. Angewandte Informatik and Formale Beschreibungsverfahren, Teubner-Texte zur Informatik 29, pp. 45–60, Teubner (1999)
13. Desel, J., Erwin, T.: Hybrid specifications: looking at workflows from a run-time perspective. International Journal of Computer System Science & Engineering 15 Nr. 5, pp. 291-302 (2000)

14. Desel, J.: Validation of process models by construction of process nets. Business Process Management: Models, Techniques and Empirical Studies, LNCS 1806, Springer (2000,)pp. 108–126
15. Desel, J., Juhás, G.: What is a Petri net? Informal answers for the informed reader. Unifying Petri Nets, Advances in Petri Nets, LNCS 2128, Springer (2001) 1–25
16. Desel, J.: Model validation – a theoretical issue? 23rd International Conference on Applications and Theory of Petri Nets, LNCS 2360, pp. 23–43, Springer (2002)
17. Desel, J.: Validierung and Verifikation von Prozessmodellen. Promise 2002 – GI-Workshop über Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen, LNI P-21, pp. 78-80, Gesellschaft für Informatik (2002)
18. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and validation with VIP-tool. Business Process Management, June 2003, LNCS 2678, pp. 380–389, Springer (2003)
19. Desel, J.: Process modelling using Petri nets. In [21], pp. 147-177 (2005)
20. van Dongen, B.F., Desel, J., van der Aalst, W.M.P.: Aggregating causal runs to workflow nets. BETA Working Paper Series, WP 173, Eindhoven University of Technology (2006)
21. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems – Bridging People and Software through Process Technology. Wiley (2005)
22. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures; Part I and II. Acta Informatica 27 (1990) 315–342, 343–368
23. Fliedl, G., Kop, C., Mayr, H.C.: From scenarios to KCPM dynamic schemas – aspects of automatic mapping. Natural Language Processing and Information Systems, 8th International Conference on Applications of Natural Language to Information Systems, LNI P-29, pp. 91–105. Gesellschaft für Informatik (2003)
24. Juhás, G., Lorenz, R., Desel, J.: Can I execute your scenario in my net? 26th International Conference on Applications and Theory of Petri Nets, LNCS 3536, pp. 289–308, Springer (2005)
25. Lorenz, R., Bergenthum, R., Desel, J., Juhás, G.,: Can I execute my scenario in your net? VipTool tells you! 27th International Conference on Applications and Theory of Petri Nets, LNCS 4024, pp. 381–390, Springer (2006)
26. Lorenz, R., Bergenthum, R., Desel, J., Mauser, S.: Synthesis of Petri nets from finite partial languages. 7th International Conference on Application of Concurrency to System Design (ACSD), IEEE (2007) pp. 157–166
27. Mayr, H.C., Dittrich, K.R., Lockemann, P.C.: Datenbankentwurf. Kapitel 5 in: Datenbank-Handbuch, pp. 481–557. Springer (1987)
28. Mayr, H.C., Schnattler, M.: Vorkonzeptuelle und konzeptuelle Dynamik-Modellierung. Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen. Informatik aktuell, pp. 3–18. Springer (1993)
29. Mayr, H.C.: Towards business process modeling in KCPM. In [11], pp. 27, 1998.
30. Mayr, H.C., Kop, C.: A user centered approach to requirements modeling. Modellierung 2002, LNI P-12, pp. 75–86. Gesellschaft für Informatik (2002)
31. Mayr, H.C., Kop, C., Esberger, D.: Business process modeling and requirements modeling. First International Conference on the Digital Society (ICDS'07). Los Alamitos, CA, pp. 8–11, 2007
32. Process Mining Group, Eindhoven University of Technology: www.processmining.org
33. Reisig, W.: A Primer in Petri Net Design. Springer (1992)
34. Weske, M.: Business Process Management – Concepts, Languages and Architectures. Springer (2007)