Verification of Workflow Nets

W.M.P. van der Aalst

Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
wsinwa@win.tue.nl

Abstract. Workflow management systems will change the architecture of future information systems dramatically. The explicit representation of business procedures is one of the main issues when introducing a workflow management system. In this paper we focus on a class of Petri nets suitable for the representation, validation and verification of these procedures. We will show that the correctness of a procedure represented by such a Petri net can be verified by using standard Petri-net-based techniques. Based on this result we provide a comprehensive set of transformation rules which can be used to construct and modify correct procedures.

1 Introduction

At the moment more than 250 *Workflow Management Systems* (WFMSs) are available. This number signifies the importance of the workflow paradigm ([3,4,6,13,18,23]). Unfortunately, today's WFMSs suffer from a number of serious drawbacks. A theoretical basis for workflow management tools is missing. As a result there are, even at a conceptual level, important differences between the tools. Despite the efforts of the Workflow Management Coalition [26] there are no real standards. Moreover, the absence of tools to support the analysis of workflows is a serious drawback for the success of today's WFMSs. In this paper we present a Petri-net-based approach to overcome some of these problems. In particular we will address the problem of analyzing the correctness of *workflow procedures*.

Business processes supported by a WFMS are centered around procedures. A procedure is the method of operation used by a business process to process *cases*. Examples of cases are orders, claims, travel expenses, tax declarations, etc. The procedure specifies the set of *tasks* required to process these cases successfully. Moreover, the procedure specifies the (partial) order in which these tasks have to be executed. The goal of a procedure is to handle cases efficiently and properly. To achieve this goal, the procedure should be tuned to the ever changing environment of the business process. A WFMS supports the definition and modification of procedures. Unfortunately, most of the WFMSs allow for the specification of incorrect procedures. We have evaluated many WFMSs. None of these WFMSs allow for the verification of essential properties such as the absence of deadlock and livelock.

In this paper we focus on the use of *Petri nets* ([20–22]) as a tool for the representation, validation and verification of workflow procedures. It is not difficult to map a procedure onto a Petri net. As it turns out, we can even restrict ourselves to a subclass of Petri nets. Representatives of this class are called *WorkFlow nets* (WF-nets). A WF-net is a Petri net with two special places: i and o. These places are used to mark the begin and the

end of a procedure, see Figure 1. The tasks are modeled by transitions and the partial ordering of tasks is modeled by places connecting these transitions.

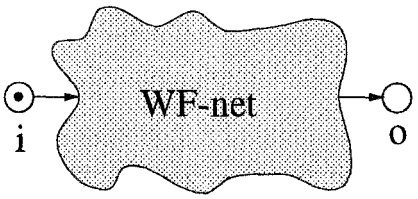

Fig. 1. A procedure modeled by a WF-net.

The processing of a case starts the moment we put a token in place i and terminates the moment a token appears in place o. One of the main properties a proper procedure should satisfy is the following:

> *For any case, the procedure will terminate eventually, and at the moment the procedure terminates there is a token in place o and all the other places are empty.*

This property is called the *soundness property*. In this paper we present a technique to verify this property using standard Petri-net tools. If we restrict ourselves to *free-choice Petri nets* (cf. Best [8], Desel and Esparza [12]), this property can be verified in polynomial time.

WF-nets have some interesting properties. For example, it turns out that a WF-net is sound if and only if a slightly modified version of this net is live and bounded! We will use this property to show that there is a comprehensive set of transformation rules which preserve soundness. These transformation rules show how a sound procedure can be transformed into another sound procedure.

The remainder of this paper is organized as follows. In Section 2 we introduce some of the basic notations for Petri nets. Section 3 deals with WF-nets. In this section we also define the soundness property. In Section 4 we present a technique to verify the soundness property. A set of transformation rules that preserve soundness is presented in Section 5.

2 Petri nets

Historically speaking, Petri nets originate from the early work of Carl Adam Petri ([22]). Since then the use and study of Petri nets has increased considerably. For a review of the history of Petri nets and an extensive bibliography the reader is referred to Murata [20].

The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 1 (Petri net). A *Petri net* is a triple (P, T, F):

- P is a finite set of *places*,
- T is a finite set of *transitions* ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation)

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing p as an input place. Note that we restrict ourselves to arcs with weight 1. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \rightarrow \mathbf{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$.

For any two states M_1 and M_2, $M_1 \geq M_2$ iff for each $p \in P$: $M_1(p) \geq M_2(p)$. $M_1 > M_2$ iff $M_1 \geq M_2$ and $M_1 \neq M_2$.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.
(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

Given a Petri net (P, T, F) and an initial state M_1, we have the following notations:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n, i.e. $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$.

We use (PN, M) to denote a Petri net PN with an initial state M. Let us define some properties for Petri nets.

Definition 2 (Live). A Petri net (PN, M) is *live* iff, for every reachable state M' and every transition t there is a state M'' reachable from M' which enables t.

Definition 3 (Bounded). A Petri net (PN, M) is *bounded* iff, for every reachable state and every place p the number of tokens in p is bounded.

Definition 4 (Strongly connected). A Petri net is *strongly connected* iff, for every pair of nodes (i.e. places and transitions) x and y, there is a directed path leading from x to y.

In this paper we use a restricted class of Petri nets for modeling and analyzing workflow procedures. As we will see in Section 4.3, it often suffices to consider Petri nets satisfying the so-called *free-choice property*.

Definition 5 (Free-choice). A Petri net is a *free-choice Petri net* iff, for every two places p_1 and p_2 either $(p_1 \bullet \cap p_2 \bullet) = \emptyset$ or $p_1 \bullet = p_2 \bullet$.

Free-choice Petri nets have been studied extensively (cf. Best [8], Desel and Esparza [12,11,14], Hack [17]) because they seem to be a good compromise between expressive power and analyzability. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist.

3 WF-nets

3.1 What is a WFMS?

A WFMS is a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications (WFMC [26]). A WFMS can be seen as the operating system of administrative organisations. The WFMS takes care of the 'office logistics'. At the moment more than 250 WFMSs are available. Examples are: COSA (Software-Ley), Staffware (Staffware), Flowmark (IBM), InConcert (XSoft), Open Workflow (Wang), TeamWARE (TeamWARE), Visual WorkFlo (Filenet) and Workparty (Siemens). These systems focus on *business processes* and are centered around the definition of a business process, often referred to as workflow.
The objective of a workflow process is the processing of *cases* (e.g. claims, orders, travel expenses). To support the processing of these cases using a WFMS we need to specify two aspects of the workflow ([5]):

(i) *Procedure*
For each workflow process we have to specify the procedure that is used to handle a case. In essence the procedure defines a partially ordered set of *tasks*. Cases are routed through the procedure using AND-splits, OR-splits, AND-joins and OR-joins. As a result it is possible to specify sequential, selective and parallel routing and iteration. In the procedure there are no explicit references to the resources (persons) that will execute the tasks.

(ii) *Resource classification*
Besides the procedure we need to specify who is going to do the work. Therefore, resources (persons, departments, machines, etc.) are classified into resources classes. A resources class represents a role (a group of people with a specific set of attributes, qualifications and/or skills) or an organisational unit (department, company or team).

Tasks are associated with resource classes. This way it is possible to link the resource classification to the procedure. The workflow engine (i.e. the workflow enactment service) allocates resources to tasks given the constraints specified in the procedure and resource classification.

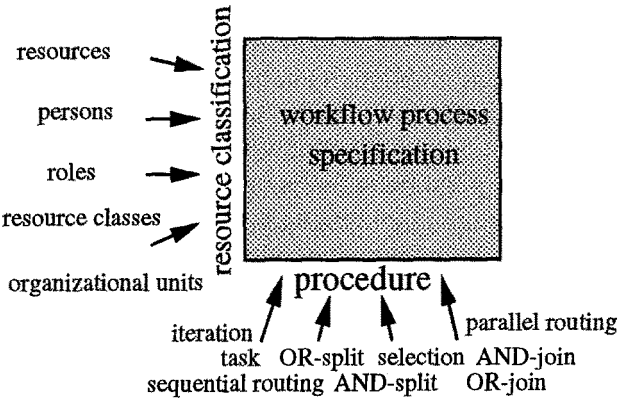

Fig. 2. The two dimensions of a workflow process specification.

Figure 2 illustrates the fact that the specification of a workflow process is composed from two elements: a procedure and a resource classification. Many WFMSs provide two workflow definition tools: one for the specification of procedures and one for the specification of resource classes. In this paper we *abstract from resource classes* and focus on techniques to support the construction and analysis of procedures. We also abstract from case data. In a WFMS it is possible to use case attributes when routing a case. We abstract from this information by introducing non-deterministic choices. Since case attributes are set and modified by applications and may involve human decisions, we have no other choice but to abstract from case data.

3.2 What is a workflow procedure?

The procedure specifies the set of tasks required to process cases successfully. (Synonyms for task are process activity, step and node.) Moreover, the procedure specifies the order in which these tasks have to be executed. (Tasks may be optional or mandatory and are executed in parallel or sequential order.) The allocation of resources to tasks is required to decide who is going to execute a specific task for a specific case. Each resource (e.g. a secretary) is able to perform certain functions (e.g. typing a letter) and each task requires certain functions. A resource may be allocated to a task, if the resource provides the required functions. Recall that we concentrate on modeling workflow procedures, i.e. we abstract from the resources required to execute these procedures. To illustrate the term (workflow) procedure we will use the following example. Consider an automobile insurance company. The workflow process *process_claim* takes care of the processing of claims related to car damage. Each claim corresponds to a case to be handled

by *process_claim*. The workflow procedure that is used to handle these cases can be described as follows. There are four tasks: *check_insurance*, *contact_garage*, *pay_damage* and *send_letter*. The tasks *check_insurance* and *contact_garage* may be executed in any order to determine whether the claim is justified. If the claim is justified, the damage is paid (task *pay_damage*). Otherwise a 'letter of rejection' is sent to the claimant (task *send_letter*).

3.3 Modeling a procedure

We use Petri nets for modeling and analyzing workflow procedures. Basically, a procedure is a partially ordered set of tasks. Therefore, it is quite easy to map a procedure onto a Petri net. Tasks are modeled by transitions and precedence relations are modeled by places. Consider for example the workflow procedure *process_claim*, see Figure 3. The tasks *check_insurance*, *contact_garage*, *pay_damage* and *send_letter* are modeled by transitions. Since the two tasks *check_insurance* and *contact_garage* may be executed in parallel, there are two additional transitions: *fork* and *join*. The places *p1*, *p2*, *p3*, *p4* and *p5* are used to route a case through the procedure in a proper manner.

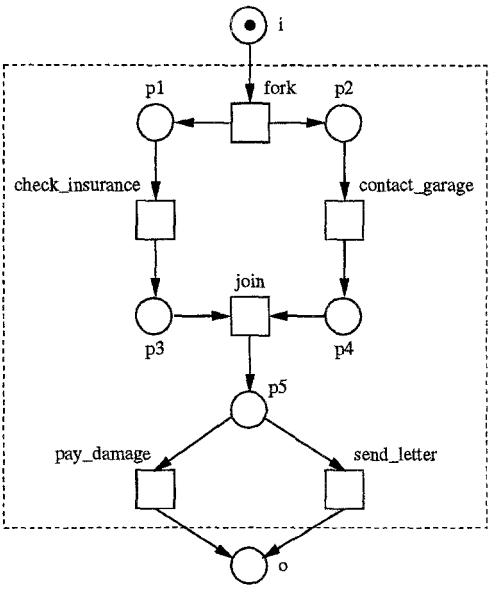

Fig. 3. The workflow procedure *process_claim*.

Cases are processed independently, i.e. a task executed for some case cannot influence a task executed for another case. Nevertheless, the throughput time of a case may increase if there are many other cases competing for the same resources. In this paper we abstract from resources; cases do not affect each other in any way. Therefore, it suffices to consider one case at a time. The token in place *i* in Figure 3 corresponds to one case. During the processing of a case there may be several tokens referring to the same case.

(If transition *fork* fires, then there are two tokens, one in *p1* and one in *p2*, referring to the same claim.) The processing of the case is completed if there is a token in place o and there are no other tokens also referring to the same case.

Petri nets which model workflow procedures have some typical properties. First of all, they always have two special places i and o, which correspond to the beginning and termination of the processing of a case respectively. Place i is a source place and o is a sink place. Secondly, for each transition t (place p) there should be directed path from place i to o via t (p). A Petri net which satisfies these two requirements is called a *Workflow net* (WF-net), see Figure 1.

Definition 6 (WF-net). A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net) if and only if:

(i) PN has two special places: i and o. Place i is a source place: $\bullet i = \emptyset$. Place o is a sink place: $o\bullet = \emptyset$.
(ii) If we add a transition t^* to PN which connects place o with i (i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.

A WF-net has one input place (i) and one output place (o) because any case handled by the procedure represented by the WF-net is created if it enters the WFMS and is deleted once it is completely handled by the WFMS, i.e., the WF-net specifies the life-cycle of a case. The second requirement in Definition 6 (the Petri net extended with t^* should be strongly connected), states that for each transition t (place p) there should be directed path from place i to o via t (p). This requirement has been added to avoid 'dangling tasks and/or conditions', i.e. tasks and conditions which do not contribute to the processing of cases.

It is easy to verify that the Petri net shown in Figure 3 is a WF-net.

3.4 Sound procedures

The two requirements stated in Definition 6 can be verified statically, i.e. they only relate to the structure of the Petri net. There is however a third requirement which should be satisfied:

> *For any case, the procedure will terminate eventually, and at the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there should be no dead tasks, i.e., it should be possible to execute an arbitrary task by following the appropriate route though the WF-net. These two additional constraints correspond to the so-called *soundness property*.

Definition 7 (Sound). A procedure modeled by a WF-net $PN = (P, T, F)$ is sound if and only if:

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state o. Formally:

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

(ii) State o is the only state reachable from state i with at least one token in place o. Formally:

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in (PN, i). Formally:

$$\forall_{t \in T} \exists_{M,M'} \; i \xrightarrow{*} M \xrightarrow{t} M'$$

Note that there is an overloading of notation: the symbol i is used to denote both the *place* i and the *state* with only one token in place i (see Section 2). The soundness property relates to the dynamics of a WF-net. The first requirement in Definition 7 states that starting from the initial state (state i), it is always possible to reach the state with one token in place o (state o). If we assume fairness (i.e. a transition that is enabled infinitely often will fire eventually), then the first requirement implies that eventually state o is reached. The fairness assumption is reasonable in the context of workflow management; all choices are made (implicitly or explicitly) by applications, humans or external actors. Clearly, they should not introduce an infinite loop. The second requirement states that the moment a token is put in place o, all the other places should be empty. Sometimes the term *proper termination* is used to describe the first two requirements [15]. The last requirement states that there are no dead transitions (tasks) in the initial state i.

For the WF-net shown in Figure 3 it is easy to see that it is sound. However, for complex workflow procedures it is far from trivial to check the soundness property.

4 Analysis of WF-nets

4.1 Introduction

In this section, we focus on analysis techniques that can be used to verify the soundness property. The soundness property is a property which relates to the dynamics of a WF-net. Therefore, the *coverability graph* (Peterson [21], Murata [20]) seems to be an obvious technique to check whether the WF-net is sound. Figure 4 shows the coverability graph which corresponds to the Petri net shown in Figure 3 (the initial state is i). There are only 7 reachable states, therefore it is easy to verify the three requirements stated in Definition 7.

In general the coverability graph can be used to decide whether a WF-net is sound. (In Section 4.2 we show that a sound WF-net is bounded. If the coverability graph has an unbounded state (an 'ω-state'), then the WF-net is not sound. Otherwise, we can use a simple algorithm to check the three requirements stated in Definition 7.) However, for complex procedures, the construction of the coverability graph may be very time consuming. The complexity of the algorithm to construct the coverability graph can be worse than primitive recursive space. Even for free-choice Petri nets the reachability problem is known to be EXPSPACE-hard (cf. Cheng, Esparza and Palsberg [9]). Therefore, any 'brute-force approach' to check soundness is bound to be intractable.

Fortunately, the problem of deciding whether a given WF-net is sound corresponds to standard Petri-net properties. In the remainder of this section, we present a technique to decide soundness. Along the way, we encounter some interesting properties of sound

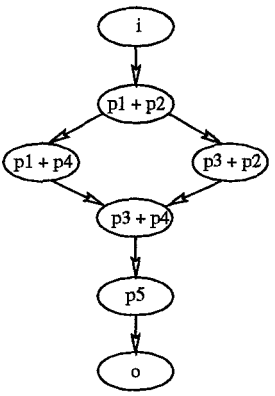

Fig. 4. The coverability graph of the Petri net shown in Figure 3.

WF-nets. Moreover, we will show that for most WF-net encountered in practice, the soundness property can be verified in polynomial time.

4.2 A necessary and sufficient condition for soundness

Given WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. For this purpose we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net that we obtain by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows:

$$\overline{P} = P$$
$$\overline{T} = T \cup \{t^*\}$$
$$\overline{F} = F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\}$$

Figure 5 illustrates the relation between PN and $\overline{PN}$.

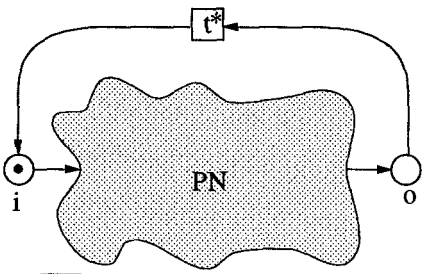

Fig. 5. $\overline{PN} = (P, T \cup \{t^*\}, F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\})$.

For an arbitrary WF-net PN and the corresponding extended Petri net $\overline{PN}$ we will prove the following result:

PN is sound if and only if $(\overline{PN}, i)$ is live and bounded.

First, we prove the 'if' direction.

Lemma 8. *If $(\overline{PN}, i)$ is live and bounded, then PN is a sound WF-net.*

Proof. $(\overline{PN}, i)$ is live, i.e., for every reachable state M there is a firing sequence which leads to a state in which t^* is enabled. Since o is the input place of t^*, we find that for any state M reachable from state i it is possible to reach a state with at least one token in place o. Consider an arbitrary reachable state $M' + o$, i.e. a state with at least one token in place o. In this state t^* is enabled. If t^* fires, then the state $M' + i$ is reached. Since $(\overline{PN}, i)$ is also bounded, M' should be equal to the empty state. Hence requirements (i) and (ii) hold and proper termination is guaranteed. Requirement (iii) follows directly from the fact that $(\overline{PN}, i)$ is live. Hence, PN is a sound WF-net. □

To prove the 'only if' direction, we first show that the extended net is bounded.

Lemma 9. *If PN is sound, then $(\overline{PN}, i)$ is bounded.*

Proof. Assume that PN is sound and (PN, i) not bounded. Since PN is not bounded there are two states M_i and M_j such that $i \xrightarrow{*} M_i$, $M_i \xrightarrow{*} M_j$ and $M_j > M_i$. (See for example the proof that the coverability tree is finite in Peterson [21] (Theorem 4.1).) However, since PN is sound we know that there is a firing sequence σ such that $M_i \xrightarrow{\sigma} o$. Therefore, there is a state M such that $M_j \xrightarrow{\sigma} M$ and $M > o$. Hence, it is not possible that PN is both sound and not bounded. So if PN is sound, then (PN, i) is bounded. From the fact that PN is sound and (PN, i) is bounded, we can deduce that $(\overline{PN}, i)$ is bounded. If transition t^* in $\overline{PN}$ fires, the net returns to the initial state i. □

Now we can prove that $(\overline{PN}, i)$ is live.

Lemma 10. *If PN is sound, then $(\overline{PN}, i)$ is live.*

Proof. Assume PN is sound. By Lemma 9 we know that $(\overline{PN}, i)$ is bounded. Because PN is sound we know that state i is a so-called home-marking of $\overline{PN}$, i.e., for every state M' reachable from $(\overline{PN}, i)$ it is possible to return to state i. In the original net (PN, i), it is possible to fire an arbitrary transition t (requirement (iii)). This is also the case in the modified net. Therefore, $(\overline{PN}, i)$ is live because for every state M' reachable from $(\overline{PN}, i)$ it is possible to reach a state which enables an arbitrary transition t. □

Theorem 11. *A WF-net PN is sound if and only if $(\overline{PN}, i)$ is live and bounded.*

Proof. It follows directly from Lemma 8, 9 and 10. □

Theorem 11 is an extension of the results presented in [2,24]. In [2] we restrict ourselves to free-choice WF-nets. Independently, Straub and Hurtado [24] found necessary and sufficient conditions for soundness of COPA nets. (COPA nets correspond to a subclass of free-choice Petri nets.)

Perhaps surprisingly, the verification of the soundness property boils down to checking whether the extended Petri net is live and bounded! This means that we can use standard Petri-net-based analysis tools to decide soundness. In Section 5 we will use Theorem 11 to prove that there is a comprehensive set of transformation rules which preserve soundness. However, first we consider an important subclass of WF-nets.

4.3 Free-choice WF-nets

Most of the WFMSs available at the moment, abstract from states between tasks, i.e., states are not represented explicitly. These WFMSs use building blocks such as the AND-split, AND-join, OR-split and OR-join to specify workflow procedures. The AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. Because these systems abstract from states, every choice is made *inside* an OR-split building block. If we model an OR-split in terms of a Petri net, the OR-split corresponds to a number of transitions sharing the same set of input places. This means that for these WFMSs, a workflow procedure corresponds to a free-choice Petri net (see Definition 5). We have evaluated many WFMSs (e.g. the ones mentioned in Section 3.1) and just one of these systems (COSA) allows for a construction which is comparable to a non-free choice WF-net. Therefore, it makes sense to consider free-choice Petri nets. Parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property (cf. Section 5). Another reason for restricting WF-nets to free-choice Petri nets is the following. If we allow non-free-choice Petri nets, then the choice between conflicting tasks may be influenced by the order in which the preceding tasks are executed. The routing of a case should be independent of the order in which tasks are executed. A situation where the free-choice property is violated is often a mixture of parallelism and choice. In our opinion parallelism itself should be separated from the choice between a parallel and a non-parallel execution of tasks.

Free-choice Petri nets have been studied extensively and are marked by strong theoretical results and efficient analysis techniques (cf. Best [8], Desel and Esparza [12,11,14], Hack [17]). As a result soundness can be determined in polynomial time for free-choice WF-nets.

Theorem 12. *For a free-choice WF-net it is possible to decide soundness in polynomial time.*

Proof. As a direct result of the Rank theorem ([12]), it is possible to decide liveness *and* boundedness in polynomial time. Therefore, the problem of checking whether a WF-net is sound can be solved in polynomial time using standard techniques. □

Clearly, the correctness of a free-choice WF-nets can be analyzed very efficiently. Moreover, it is possible to simplify Definition 7.

Proposition 13. *For free-choice WF-nets, the first two requirements in Definition 7 imply the third requirement.*

Proof. Let PN be a free-choice WF-net satisfying the first two requirements in Definition 7 ((i) and (ii)). These requirements state that the WF-net is proper terminating. It is easy to prove that the extended net $(\overline{PN}, i)$ is bounded without using the third requirement (see Lemma 9). Moreover, it is also easy to show that i is a so-called home-marking of $\overline{PN}$. Therefore $(\overline{PN}, i)$ is deadlock-free. Since $(\overline{PN}, i)$ is a deadlock-free, bounded, strongly connected, free-choice Petri net, we deduce that $(\overline{PN}, i)$ is live (see Theorem 4.31 in Desel and Esparza [12]). Since $(\overline{PN}, i)$ is live and bounded, PN is sound (Theorem 11) and the third requirement (iii) holds. □

Most of the workflow procedures we have seen in practice obey the free-choice property. This is a direct result of the fact that most of the workflow systems abstract from the explicit modeling of states. However, a Petri-net-based WFMS such as COSA (Software-Ley, Pullheim, Germany) allows for the construction of non-free-choice WF-nets. Fortunately, for non-free-choice WF-nets we can check a sufficient condition for soundness in polynomial time.

In the following theorem we assume that the reader is familiar with some advanced notions: rank, siphon and cluster. A cluster is a minimal set C of nodes satisfying the following conditions. If the set contains a place s, then $s\bullet \subseteq C$. If the set contains a place t, then $\bullet t \subseteq C$. For more information on these advanced topics, we refer to [12] and [20].

Theorem 14. *A WF-net PN is sound if the rank of $\overline{PN}$ is equal to the number of clusters minus 1 and every proper siphon contains at least one token.*

Proof. $\overline{PN}$ is strongly connected. Therefore we can apply Theorem 10.17 in [12]. This theorem states that a weakly connected Petri net with at least one place and one transition, a rank equal to the number of clusters minus 1 (i.e. $Rank(\overline{PN}) = |C_N| - 1$) and a token in every proper siphon, is live and bounded. By applying Theorem 11 we deduce that PN is sound. □

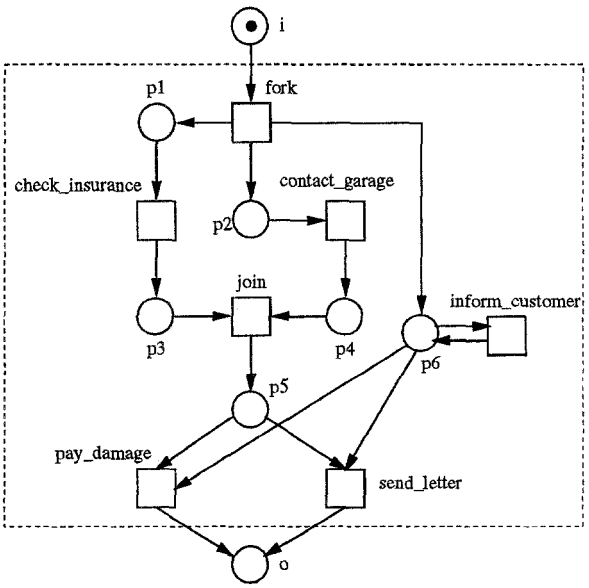

Fig. 6. The modified procedure *process_claim*.

The conditions stated in Theorem 14 are sufficient for soundness and can be checked in polynomial time. Figure 6 shows a WF-net which is not free-choice. The two transitions *pay_damage* and *send_letter* are in conflict with the new transition *inform_customer*

but the sets of input places differ. The rank of the extended WF-net is 5, the number of clusters is 6 and every proper siphon contains at least one token. Therefore, we can use Theorem 14 to prove that the WF-net is sound (in polynomial time).

Unfortunately, there are sound WF-nets which do not satisfy the conditions stated in Theorem 14. However, we can extend the class of free-choice WF-nets such that these conditions are necessary and sufficient for soundness. A Petri net that belongs to this class is called an *almost free-choice* WF-net.

Definition 15 (Almost free-choice). A Petri net is an *almost free-choice Petri net* iff, for every two transitions t_1 and t_2 either

- $(\bullet t_1 \cap \bullet t_2) = \emptyset$, or
- $\bullet t_1 = \bullet t_2$, or
- $\bullet t_1 = t_1\bullet$ and $\bullet t_1 \subseteq \bullet t_2$, or
- $\bullet t_2 = t_2\bullet$ and $\bullet t_2 \subseteq \bullet t_1$.

An almost free-choice Petri net is a free-choice Petri net extended with zero or more transitions which can not change the state of the net and preserve (non)liveness of the net. The additional transitions correspond to inquiry tasks, i.e. tasks which do not contribute to the processing of a case. These tasks are executed if someone requires information about a particular case. The WF-net in Figure 6 is almost free-choice; *inform_customer* corresponds to a so-called inquiry task.

Theorem 16. *For an almost free-choice WF-net it is possible to decide soundness in polynomial time.*

Proof. An almost free-choice WF-net can be transformed into a free-choice WF-net without changing liveness and boundedness properties. This can be done by simply removing the inquiry transitions. An inquiry transition is a transition t_1 such that (1) $\bullet t_1 = t_1\bullet$ and (2) there is another transition t_2 such that $\bullet t_1 \subseteq \bullet t_2$. Removing t_1 does not change boundedness, because t_1 cannot change the state. Liveness is also not affected because t_1 cannot change the state and t_1 is live if t_2 is live. By removing all inquiry transitions we obtain a free-choice WF-net. Then we can apply the Rank theorem to decide soundness. □

These results show that for a fairly large class of WF-nets we can decide soundness very efficiently. We have been involved in a number of workflow projects (Dutch Customs department, Dutch Justice department and a number of banks and insurance companies). In these projects we hardly ever experienced the need for WF-nets which are not almost free-choice.

5 Transformation rules

One of the major benefits of a WFMS is the ability to change business processes very easily, i.e., without a complete redesign of the information system. Moreover, workflow

mangement software is an essential enabler for management philosophies such as Business Process Reengineering (BPR) and Continuous Process Improvement (CPI). BPR and CPI are also a stimulus for the modification of business processes. As a result, the procedures supported by the WFMS are subject to frequent changes. These changes may introduce errors. In this paper we have shown that it is possible to check the soundness of a procedure very efficiently. However, we can use an alternative approach to make sure that the procedure remains sound. This approach is based on *soundness preserving transformation rules*.

In our opinion there are eight basic transformation rules ($T1a$, $T1b$, $T2a$, $T2b$, $T3a$, $T3b$, $T4a$ and $T4b$) which can be used to modify a sound workflow procedure. These transformation rules are shown in Figures 7, 8, 9 and 10 and elucidated in the sequel. The transformation rules correspond to the basic routing constructs identified by the Workflow Management Coalition ([26]). These rules should not be confused with the reduction rules presented in Petri-net literature (cf. Berthelot [7], Desel [10], Desel and Esparza [12] and Kovalyov [19]). The transformation rules presented in this section are not used for analysis purposes; they are used for the modification of WF-nets.

T1a Task $t1$ is replaced by two consecutive tasks $t2$ and $t3$. This transformation rule corresponds to the *division* of a task: a complex task is divided into two tasks which are less complicated. (See Figure 7.)

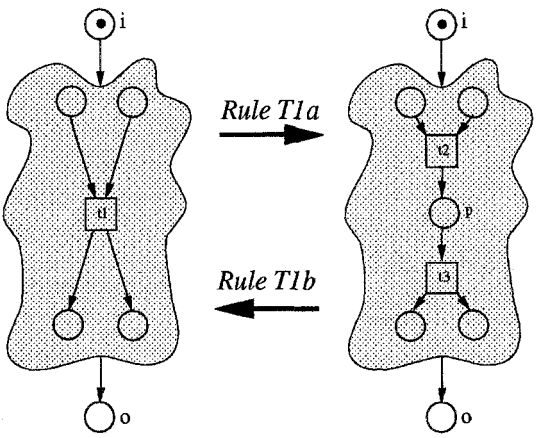

Fig. 7. Transformation rules: $T1a$ and $T1b$.

T1b Two consecutive tasks $t2$ and $t3$ are replaced by one task $t1$. This transformation rule is the opposite of $T1a$ and corresponds to the *aggregation* of tasks. Two tasks are combined into one task. (See Figure 7.)

T2a Task $t1$ is replaced by two conditional tasks $t2$ and $t3$. This transformation rule corresponds to the *specialization* of a task (e.g. *handle_order*) into two more specialized tasks (e.g. *handle_small_order* and *handle_large_order*).
(See Figure 8.)

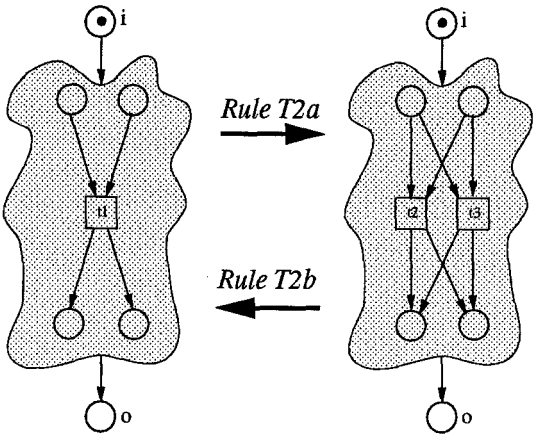

Fig. 8. Transformation rules: *T2a* and *T2b*.

T2b Two conditional tasks $t2$ and $t3$ are replaced by one task $t1$. This transformation rule is the opposite of *T2a* and corresponds to the *generalization* of tasks. Two rather specific tasks are replaced by one more generic task. (See Figure 8.)

T3a Task $t1$ is replaced by two parallel tasks $t2$ and $t3$. (See Figure 9.) The effect of the execution of $t2$ and $t3$ is identical to the effect of the execution of $t1$. The transitions $c1$ and $c2$ represent control activities to fork and join two parallel threads.

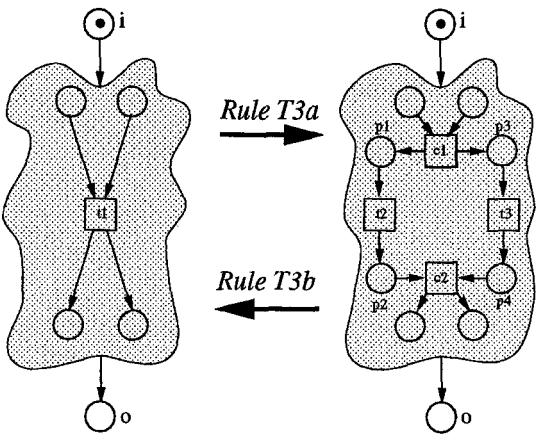

Fig. 9. Transformation rules: *T3a* and *T3b*.

T3b The opposite of transformation rule *T3a*: two parallel tasks $t2$ and $t3$ are replaced by one task $t1$. (See Figure 9.)

T4a Task $t1$ is replaced by an iteration of task $t2$. (See Figure 10.) The execution of task $t1$ (e.g. *type_letter*) corresponds to zero or more executions of task $t2$ (e.g. *type_sentence*). The transitions $c1$ and $c2$ represent control activities that mark the begin and

end of a sequence of '$t2$-tasks'. Typical examples of situations where iteration is required are quality control and communication.

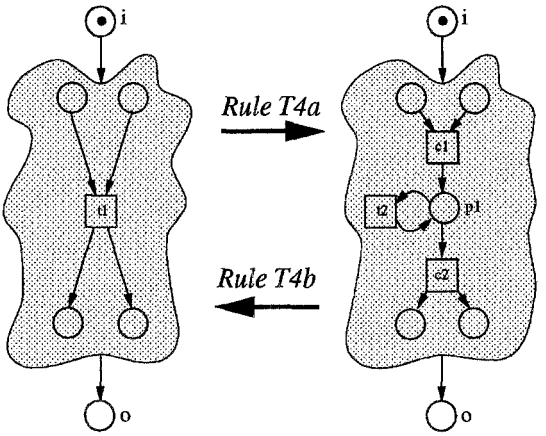

Fig. 10. Transformation rules: $T4a$ and $T4b$.

T4b The opposite of transformation rule $T4a$: the iteration of $t2$ is replaced by task $t1$. (See Figure 10.)

Formalization of these transformation rules is straightforward. To illustrate this we define $T1a$. (See Figure 7.)

Definition 17 (T1a). Let $PN = (P, T, F)$ be WF-net. This net can be transformed by rule T1a into a net $PN' = (P', T', F')$ iff there exist $t1 \in T, t2, t3 \in T'$ and $p \in P'$ such that $P' = P \cup \{p\}$, $p \notin P$, $T' = (T \setminus \{t1\}) \cup \{t2, t3\}$, $\{t2, t3\} \cap T = \emptyset$ and $F' = \{(x, y) \in F \mid (x \neq t1) \wedge (y \neq t1)\} \cup \{(x, t2) \in P \times T \mid (x, t1) \in F\} \cup \{(t3, y) \in T \times P \mid (t1, y) \in F\} \cup \{(t2, p), (p, t3)\}$.

It is easy to see that if we take a sound WF-net and we apply one of these transformation rules, then the resulting Petri net is still a WF-net. Moreover, the resulting WF-net is also sound.

Theorem 18. *The transformation rules $T1a$, $T1b$, $T2a$, $T2b$, $T3a$, $T3b$, $T4a$ and $T4b$ preserve soundness, i.e. if a WF-net is sound, then the WF-net transformed by one of these rules is also sound.*

Proof. Let PN be a sound WF-net. Let PN' be a net which is obtained by applying one of the transformation rules on PN. We have to prove that PN' is a sound WF-net. It is easy to see that PN' is a WF-net. There is still one input and one output place and the $\overline{PN'}$ is strongly connected. ($\overline{PN'}$ is the Petri net PN' with an extra transition t^* which connects place o and place i.) By Theorem 11 we know that, to prove soundness, it suffices to show $(\overline{PN'}, i)$ is live and bounded. $(\overline{PN}, i)$ is live and bounded. Therefore, we have to prove that each of the rules preserves liveness and boundedness. Because it is obvious that the rules preserve liveness and boundedness, we do not give a detailed proof.

It is for example possible to prove this by using the 6 liveness and boundedness preserving operations described in Murata [20]. □

The eight transformation rules shown in figures 7, 8, 9 and 10 preserve soundness. We can use these basic transformation rules to construct more complex transformation rules. Figure 11 shows two of these rules: *T5a* and *T5b*.

T5a Two consecutive tasks are replaced by two parallel tasks.
T5b Two parallel tasks are replaced by two consecutive tasks.

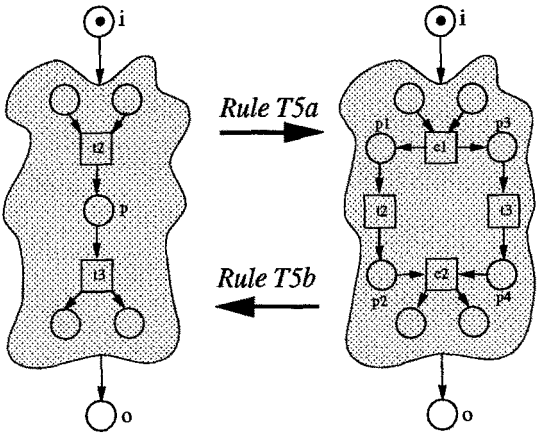

Fig. 11. Transformation rules: *T5a* and *T5b*.

The application of transformation rule *T5a* corresponds to the application of *T1b* followed by the application of *T3a*. Transformation rule *T5b* is a combination of *T3b* and *T1a*. Therefore, soundness is also preserved by the transformation rules *T5a* and *T5b*. We use the term 'sound transformation rule' to refer to a transformation rules which preserves soundness.

The WF-net which comprises only one task t is sound. We can use this net as a starting point for a sequence of sound transformations. By Theorem 18 we know that the resulting WF-net is sound.

Corollary 19. *If the Petri net* $PN = (\{i, o\}, \{t\}, \{\langle i, t\rangle, \langle t, o\rangle\})$ *is transformed into a Petri net* PN' *by applying a sequence of sound transformation rules (e.g. T1a, T1b, T2a, T2b, T3a, T3b, T4a, T4b, T5a and T5b), then* PN' *is sound.*

Consider for example the WF-net shown in Figure 3. We can construct this net by applying the transformation rules *T1a*, *T2a* and *T3a*, see Figure 12.

Note that the converse of corollary 19 is not true. There are sound WF-nets which cannot be constructed by the transformation rules defined in this section. Consider for example the WF-net shown in Figure 6; this net is sound but cannot be constructed by using the transformation rules. Nevertheless, the rules correspond to the basic routing primitives present in most WFMSs.

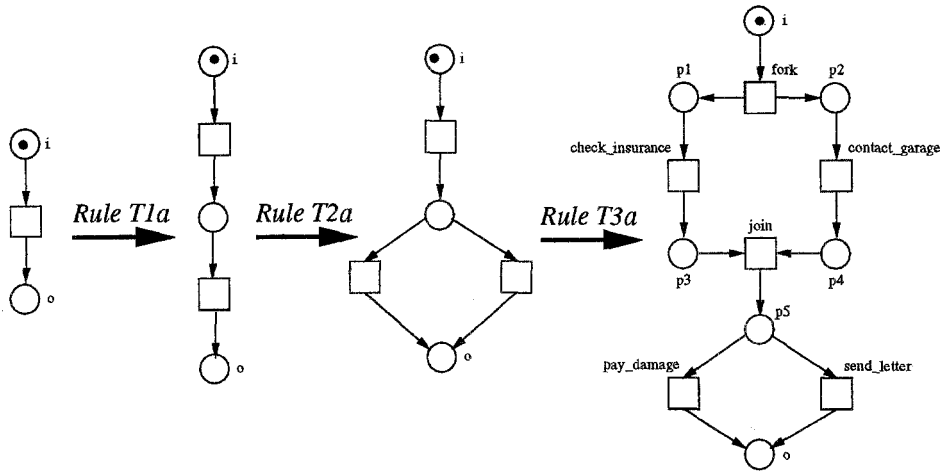

Fig. 12. Construction of the WF-net shown in Figure 3.

6 Related work

Many researchers have inversigated properties similar to the soundness property. Straub and Hurtado [24] have analyzed a similar property for COPA nets. The soundness property is also closely related to Valette's concept of a well-formed block [25] and Gostellow's concept of proper termination [15]. There is also a relation between reversibility (the possibility to return to the initial state, i.e., the initial state is a home-marking) and soundness. The results presented in this paper are marked by the fact that they are valid for any WF-net. Moreover, we have showed that for an important subclass (almost free-choice) the soundness property can be verified in polynomial time.

The use of Petri nets in the workflow domain is increasing. At the moment several Petri-net-based workflow tools are available. COSA (Software-Ley) is a Petri-net-based workflow management system. COSA is one of the leading workflow products in Europe. LEU (LION/Vebacom) is a WFMS based on FUNSOFT nets ([16]). INCOME (Promatis) and StructWare/BusinessSpecs (IvyTeam) are both Petri-net-based BPR tools which can be used to configure a WFMS. INCOME interfaces with Designer/2000 (Oracle), Plexus (BacTec) and CSE/Workflow (CSE systems). StructWare/BusinessSpecs interfaces with Staffware (Staffware), one of the worlds leading WFMSs. These tools show that workflow managemant is becoming an important application domain for Petri nets. However, these Petri-net-based workflow tools do not support advanced analysis techniques. Therefore, we have developed *WOFLAN* (WOrkFLow ANalyser). WOFLAN is a Petri-net-based analysis tool for the verfication of workflows. Amongst others it supports the analysis techniques described in this paper.

7 Conclusion

In this paper we have presented a class of Petri nets, the so-called WF-nets, suitable for the representation, validation and verification of workflow procedures. One of the mer-

its of this class is that we can verify the soundness property using standard techniques. Moreover, we have identified an important class of WF-nets that can be verified in polynomial time. Even though sound WF-nets have some nice properties from a theoretical point of view, they are powerful enough to model any workflow procedure. Moreover, we have shown that the plausible transformation rules encountered in practice preserve soundness.

In this paper we focused on the workflow procedures supported by a WFMS. To completely specify a workflow process we also have to specify the management of resources: given a task that needs to be executed for a specific case we have to specify the resource (person or machine) that is going to process the task (cf. Van der Aalst and Van Hee [5]). A direction for further research is to incorporate this dimension. We hope to find a necessary and sufficient condition for soundness given a WF-net extended with some mechanism to allocate resources to tasks.

References

1. W.M.P. van der Aalst. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.
2. W.M.P. van der Aalst. A class of Petri net for modeling and analyzing business processes. Computing Science Reports 95/26, Eindhoven University of Technology, Eindhoven, 1995.
3. W.M.P. van der Aalst. Petri-net-based Workflow Management Software. In A. Sheth, editor, *Proceedings of the NFS Workshop on Workflow and Process Automation in Information Systems*, pages 114–118, Athens, Georgia, May 1996.
4. W.M.P. van der Aalst. Three Good reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, Nov 1996.
5. W.M.P. van der Aalst and K.M. van Hee. Business Process Redesign: A Petri-net-based approach. *Computers in Industry*, 29(1-2):15–26, 1996.
6. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Modellen, Methoden en Systemen (in Dutch)*. Academic Service, Schoonhoven, 1997.
7. G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 360–376. Springer-Verlag, Berlin, 1987.
8. E. Best. Structure theory of Petri nets: the free choice hiatus. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 168–206. Springer-Verlag, Berlin, 1987.
9. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. In R.K. Shyamasundar, editor, *Foundations of software technology and theoretical computer science*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, Berlin, 1993.
10. J. Desel. Reduction and design of well-behaved concurrent systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 166–181. Springer-Verlag, Berlin, 1990.
11. J. Desel. A proof of the Rank theorem for extended free-choice nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 134–153. Springer-Verlag, Berlin, 1992.

12. J. Desel and J. Esparza. *Free choice Petri nets*, volume 40 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1995.
13. C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993.
14. J. Esparza. Synthesis rules for Petri nets, and how they can lead to new results. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 182–198. Springer-Verlag, Berlin, 1990.
15. K. Gostellow, V. Cerf, G. Estrin, and S. Volansky. Proper Termination of Flow-of-control in Programs Involving Concurrent Processes. *ACM Sigplan*, 7(11):15–27, 1972.
16. V. Gruhn. Validation and Verification of Software Process Models. In A. Endres and H. Weber, editors, *Software Development Environments and CASE Technology*, volume 509 of *Lecture Notes in Computer Science*, pages 271–286. Springer-Verlag, Berlin, 1991.
17. M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.
18. T.M. Koulopoulos. *The Workflow Imperative*. Van Nostrand Reinhold, New York, 1995.
19. A.V. Kovalyov. On complete reducability of some classes of Petri nets. In *Proceedings of the 11th International Conference on Applications and Theory of Petri Nets*, pages 352–366, Paris, June 1990.
20. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
21. J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, 1981.
22. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.
23. T. Schäl. *Workflow Management for Process Organisations*, volume 1096 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1996.
24. P.A. Straub and C. Hurtado. The Simple Control Property of Business Process Models. In *XV International Conference of the Chilean Computer Science Society*, 1995.
25. R. Valette. Analysis of Petri Nets by Stepwise Refinements. *Journal of Computer and System Sciences*, 18:35–46, 1979.
26. WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.