A class of Petri nets for modeling and analyzing business processes

W.M.P. van der Aalst
Department of Mathematics and Computing Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, telephone: -31 40 474295, e-mail: wsinwa@win.tue.nl

More and more firms are marching to the drumbeat of Business Process Reengineering (BPR) and Workflow Management (WFM). This trend exposes the need for techniques for the construction and analysis of business procedures. 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 in polynomial time. Based on this result we provide a comprehensive set of transformation rules which can be used to construct and modify correct procedures.

Keywords: Petri nets; free-choice Petri nets; Business Process Reengineering; Workflow Management; analysis of Petri nets.

1 Introduction

The flourish of trumpets surrounding the terms *Business Process Reengineering* (BPR) and *Workflow Management* (WFM) signifies the focus on *business processes*. Workflow management systems allow for the continuous improvement of the business processes at hand. Business Process Reengineering efforts are aimed at dramatic improvements by a radical redesign of the business processes. Today's competitive organizations are reshaping to the needs of their primary business processes. Therefore, it is important to furnish business processes with a theoretical basis, analysis techniques and tools. In this paper we focus on modeling and analyzing the procedures underlying these business processes.

Business processes are centered round *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 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. WFM and BPR are the keywords which herald a new era of frequent and/or radical changes of existing procedures.

In this paper we focus on the use of Petri nets ([17, 18, 19]) as a tool for the representation, validation and verification of business 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 *Business-Procedure nets* (BP-nets). A BP-net is a free-choice Petri net (Desel and Esparza [12]) 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.

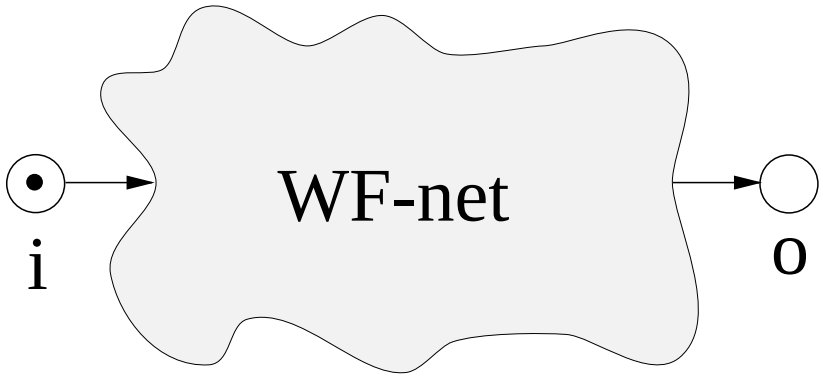

Figure 1: A procedure modeled by a BP-net.

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 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 in polynomial time. This technique is based on the rich theory developed for free-choice Petri nets (cf. Best [6], Desel and Esparza [12]).

BP-nets have some interesting properties. For example, it turns out that a BP-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. In the context of WFM and BPR, where procedures have to be modified frequently or radically, these transformation rules are useful.

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 BP-nets. In this section we also define the soundness property. In Section 4 we present a technique to verify the soundness property. Some new results for free-choice Petri nets are presented in Section 5. These results are used to prove that some extended soundness property holds for sound BP-nets. A set of transformation rules that preserve soundness is presented in Section 6.

2 Petri nets

Historically speaking, Petri nets originate from the early work of Carl Adam Petri ([19]). 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 [17].

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 triplet* (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.

Places may contain zero or more *tokens*, drawn as black dots. The *state*, often

referred to as marking, is the distribution of tokens over places. 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: $p_1 + 2p_2 + p_3$.

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. $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$
- $M_1 \xrightarrow{*} M_n$: there is a firing sequence which leads from M_1 to 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 $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$.

Let $\sigma = t_1 t_2 \ldots t_n$ be a firing sequence of length n. For k such that $1 \leq k \leq n$, we have the following notations:

- $\sigma(k) = t_k$
- $\sigma^k = t_1 t_2 \ldots t_k$

A state M is a *dead state* iff no transition is enabled in M. For a state M and a place p, we use $M(p)$ to denote the number of tokens in p in state M. For two states M and N, $M \leq N$ iff for each place p: $M(p) \leq N(p)$.

Let us define some properties.

Definition 2 (Conservative) *A Petri net PN is conservative iff there is a positive integer $w(p)$ for every place p such that, given an arbitrary initial state M, the weighted sum of tokens is constant for every reachable state M'.*

Definition 3 (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 4 (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 5 (Strongly connected) *A Petri net is strongly connected iff, for every two places (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 business procedures. As we will see in Section 3, it suffices to consider Petri nets satisfying the so-called *free-choice property*.

Definition 6 (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 [6], Desel and Esparza [12, 11, 13], Hack [14]) 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.

For reasons of simplicity we only consider classical Petri nets. (As a matter of fact only free-choice Petri nets.) However, the results in this paper can be extended to *high-level Petri nets*, i.e. Petri nets extended with (i) 'color' (tokens have a value), (ii) 'time' (it is possible to model durations) and (iii) 'hierarchy' (a net may be composed of subnets). In fact we are planning to incorporate the techniques presented in this paper in the software package *ExSpect* ([10]). ExSpect is a tool based on high-level Petri nets which has been used to model and analyze many industrial systems ([3]). For more details about the model ExSpect is based on the reader is referred to [1, 2, 15]. As a matter of fact, it is a model quite similar to the CPN-model by Jensen (cf. [16]).

3 BP-nets

3.1 What is a procedure?

A common feature of Workflow Management and Business Process Reengineering is the focus on *business processes*. Workflow management systems are centered round the definition of a business process, often referred to as workflow.

Business Process Reengineering involves the explicit reconsideration and redesign of business processes.
The objective of a business process is the processing of *cases* (e.g. claims, orders, travel expenses). To completely define a business process we have to specify two things ([5, 4]):

(i) A *procedure*: a partially ordered set of *tasks*.

(ii) An *allocation* of *resources* to tasks.

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.
In this paper we concentrate on modeling (business) procedures, i.e. we abstract from the resources required to execute these procedures.
To illustrate the term (business) procedure we will use the following example. Consider an automobile insurance company. The business 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 business 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*).

3.2 Modeling a procedure

We use Petri nets for modeling and analyzing business 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 business procedure *process_claim*, see figure 2. 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.

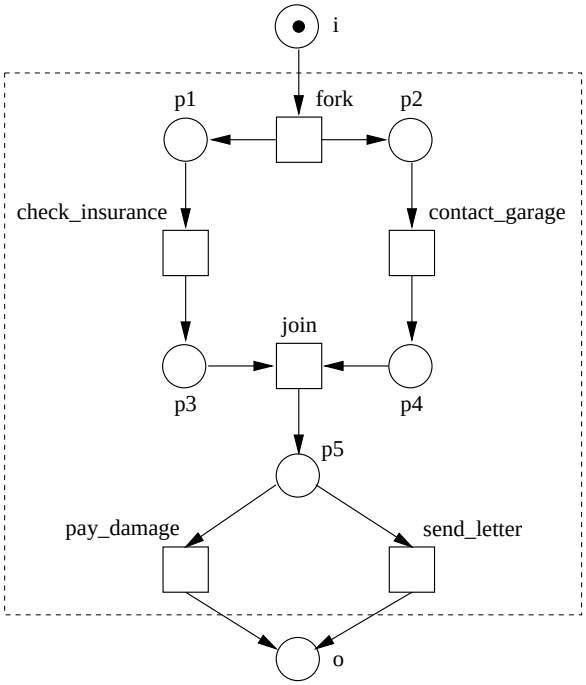

Figure 2: The business procedure *process_claim*.

Cases are processed independently, i.e. a task executed for some case cannot influence a task executed for another task. 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 (cf. Section 5). The token in place *i* in figure 2 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 business 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, a Petri net which represents a business procedure is always a free-choice Petri net. Thirdly, for each transition *t* there should be directed path from place *i* to *o* via *t*. A Petri net which satisfies these three requirements is called a *Business-Procedure net* (BP-net), see figure 1.

Definition 7 (BP-net) *A Petri net $PN = (P, T, F)$ is a BP-net (Business-Procedure 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) PN is a free-choice Petri net.

(iii) 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.

The reason for restricting BP-nets to free-choice Petri nets is pragmatic: we simply cannot think of a sensible business procedure which violates the free-choice property (see definition 6). We can model parallelism, sequential routing, conditional routing and iteration without violating the free-choice property (cf. Section 6). The third requirement (the Petri net extended with t^* should be strongly connected), states that for each transition t there should be directed path from place i to o via t. This requirement has been added to avoid 'dangling tasks', i.e. tasks which do not contribute to the processing of cases.

It is easy to verify that the Petri net shown in figure 2 is a BP-net.

3.3 Sound procedures

The three requirements stated in definition 7 can be verified statically, i.e. they only relate to the structure of the Petri net. There is however a fourth property which should be satisfied:

> *For any case, the procedure will terminate eventually and 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*.

Definition 8 (Sound) *A procedure modeled by a BP-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:

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

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

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

Note that the soundness property relates to the dynamics of a BP-net. The first requirement in definition 8 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*). (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).) 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 second requirement states that the moment a token is put in place *o*, all the other places should be empty.

For the BP-net shown in figure 2 it is easy to see that it is sound. However, for complex business procedures it is far from trivial to check the soundness property.

4 Analysis of BP-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 BP-net. Therefore, the *coverability graph* (Peterson [18], Murata [17]) seems to be an obvious technique to check whether the BP-net is sound. Figure 3 shows the coverability graph which corresponds to the Petri net shown in figure 2 (the initial state is *i*). There are only 6 reachable states, therefore it is easy to verify the two requirements stated in definition 8.
In general the coverability graph can be used to decide whether a BP-net is sound.[1] 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.

[1] In Section 4.2 we show that a sound BP-net is bounded. If the coverability graph has an unbounded state (an 'ω-state'), then the BP-net is not sound. Otherwise, we can use a simple algorithm to check the two requirements stated in definition 8.

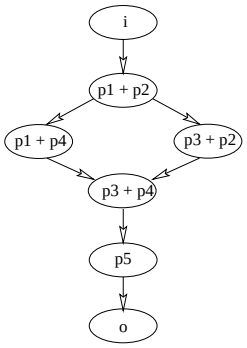

Figure 3: The coverability graph of the Petri net shown in figure 2.

Fortunately, the problem of deciding whether a given BP-net is sound is tractable. In the remainder of this section, we present a technique to decide soundness in polynomial time. Along the way, we encounter some interesting properties of sound BP-nets.

4.2 A necessary and sufficient condition for soundness

Given BP-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. This 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 4 illustrates the relation between PN and $\overline{PN}$.

For an arbitrary BP-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 conservative.

First, we prove the 'if' direction.

Lemma 1 *If $(\overline{PN}, i)$ is live and conservative, then PN is a sound BP-net.*

Proof.
$(\overline{PN}, i)$ is live, i.e. for every reachable state M there is a firing sequence which

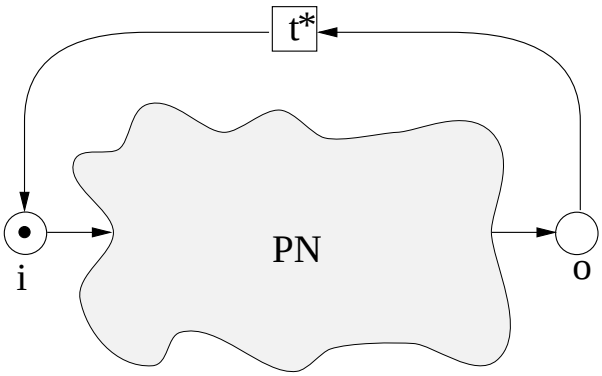

Figure 4: $\overline{PN} = (P, T \cup \{t^*\}, F \cup \{\langle o, t^*\rangle, \langle t^*, i\rangle\})$.

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. $\overline{PN}$ is conservative, therefore there is a semi-positive place invariant with a support equal to P. The places i and o have the same positive weight because t^* may move a token from o to i. The only state with at least one token in place o and reachable from state i is the state o.
So if $(\overline{PN}, i)$ is live and conservative, then PN satisfies the following properties: (i) for every state M reachable from state i, there exists a firing sequence leading from state M to state o and (ii) state o is the only state reachable from state i with at least one token in place o. Hence, PN is a sound BP-net. $\triangle$

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

Lemma 2 *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 $i \stackrel{*}{\rightarrow} M_i$, $M_i \stackrel{*}{\rightarrow} M_j$ and $M_j > M_i$. (See for example the proof that the coverability tree is finite in Peterson [18] (theorem 4.1).) However, since PN is sound we know that there is a firing sequence σ such that $M_i \stackrel{\sigma}{\rightarrow} o$. Therefore, there is a state M such that $M_j \stackrel{\sigma}{\rightarrow} M$ 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. $\triangle$

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

Lemma 3 *If PN is sound, then $(\overline{PN}, i)$ is live and conservative.*

Proof.
Assume PN is sound. By lemma 2 we know that $(\overline{PN}, i)$ is bounded. Because PN is sound we know that state i is a so-called home-marking of $\overline{PN}$. 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]). $\overline{PN}$ is a so-called well-formed net. Since every well-formed free-choice Petri net has a positive place-invariant, we deduce that $\overline{PN}$ is conservative. △

Theorem 1 *A BP-net PN is sound if and only if $(\overline{PN}, i)$ is live and conservative.*

Proof.
It follows directly from lemma 1 and lemma 3. △

Since boundedness and 'conservativeness' coincide for live free-choice Petri nets, we formulate the following corollary.

Corollary 1 *A BP-net PN is sound if and only if $(\overline{PN}, i)$ is live and bounded.*

Proof.
A live free-choice Petri net is bounded iff it is conservative (cf. Desel and Esparza [12]). △

Perhaps surprisingly, the verification of the soundness property boils down to checking whether the extended Petri net is live and bounded! As a direct result of the Rank theorem ([8, 12]), it is possible to decide liveness *and* boundedness in polynomial time. Therefore, the problem of checking whether a BP-net is sound can be solved in polynomial time using standard techniques.

In Section 6 we will use theorem 1 to prove that there is a comprehensive set of transformation rules which preserve soundness. However, first we consider the situation where we start with *n* tokens in place *i* of a sound BP-net.

5 Multiple cases

In Section 3 we stated that individual cases do not affect each other, since we abstract from resources. Therefore, it suffices to consider one case at a time to verify the correctness of a procedure. However, if we want to model a procedure that is used to process multiple cases at the same time, we need to resort to a high-level Petri net. This high-level Petri net is organized as follows. Each token has

a value which refers to the case it belongs to and transitions can only consume tokens which belong to the same case. It is easy to see that in this high-level Petri net individual cases do not affect each other. Nevertheless, it is interesting to see what happens if we abstract from color, i.e. we allow multiple indistinguishable cases. In this section we will show that we can extend the soundness property for the situation where there are an arbitrary number of cases. As it turns out this extended soundness property coincides with the soundness property defined in Section 3.3.

First we prove some preliminary results which hold for any free-choice Petri net.

5.1 Substate-ordering Lemma

One of the fundamental properties of a free-choice Petri net is the fact that it can be partitioned into *clusters*.

Definition 9 (Cluster) *Let t be a transition in a free-choice Petri net. The cluster of t, denoted by $[t]$, is the set $\bullet t \cup \{t' \in T \mid \bullet t' = \bullet t\}$. The cluster of a place p, also denoted by $[p]$, is the set $p\bullet \cup \{p' \in P \mid (p'\bullet \cap p\bullet) \neq \emptyset\}$.*

Note that a place p and a transition t belong to the same cluster (i.e. $[p] = [t]$) iff $p \in \bullet t$. For free-choice Petri nets, we have the following property. If transition t is enabled, then any transition in $[t]$ is enabled. A cluster c is called *enabled* iff the transitions in c are enabled.

The first result we present is the *advance lemma*. This lemma shows that given a firing sequence it is possible to advance the firing of certain transitions.

Lemma 4 (Advance lemma) *Let $\sigma = t_1 t_2 \ldots t_k$ be a firing sequence of a free-choice Petri net such that σ leads from state M to state M', i.e. $M \xrightarrow{\sigma} M'$. If a cluster c is enabled in state M and t_i is the first transition in σ such that $t_i \in c$, then $M \xrightarrow{\sigma'} M'$ with $\sigma' = t_i t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_k$.*

Proof.
In state M each of the transitions in c is enabled, i.e. transition t_i is enabled in state M. The transitions t_j with $1 \leq j < i$ are not disabled by the advanced firing of t_i, because they belong to different clusters. Therefore, the firing sequence $\sigma' = t_i t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_k$ is possible. Since σ' is a permutation of σ, we deduce that $M \xrightarrow{\sigma'} M'$. $\triangle$

We use the advance lemma to prove the *substate-ordering lemma*. The substate-ordering lemma is illustrated in figure 5.

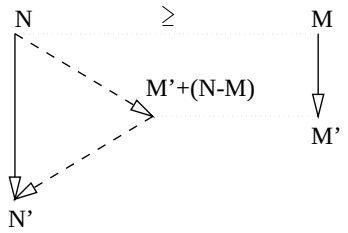

Figure 5: The substate-ordering lemma.

Lemma 5 (Substate-ordering lemma) *Let PN be a free-choice Petri net and N and N' states of PN such that $N \xrightarrow{*} N'$ and N' is dead. For any substate M of N (i.e. $M \leq N$), there is a dead state M' such that $M \xrightarrow{*} M'$ and $M' + (N - M) \xrightarrow{*} N'$.*

Proof.
Let $\sigma = t_1 t_2 \ldots t_k$ be an arbitrary firing sequence leading from N to N' ($N \xrightarrow{\sigma} N'$). We use induction upon the length k of σ.
If $k = 0$, then $N = N'$. Since N is dead ($N = N'$) and $M \leq N$, M is also dead. Hence, $M' = M$ is a dead state such that $M \xrightarrow{*} M'$ and $M' + (N - M) \xrightarrow{*} N'$.
Assume $k > 0$. If M is dead, then for $M' = M$ the lemma holds. Therefore, we may assume that M is not dead. Let t_i be the first transition in σ which is enabled in M, i.e. t_i is enabled in M and for all $1 \leq j < i$: t_j is not enabled in M. Note that such a transition exists, because $M \leq N$, M is not dead and N' is dead. The cluster $[t_i]$ is enabled in N and t_i is the first transition in σ which belongs to $[t_i]$. We can use lemma 4 to prove that $N \xrightarrow{\sigma'} N'$ with $\sigma' = t_i t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_k$. Let N_1 and M_1 be states such that $N \xrightarrow{t_i} N_1$ and $M \xrightarrow{t_i} M_1$. By the induction hypothesis we can show that there is a dead state M' such that $M_1 \xrightarrow{*} M'$ and $M' + (N_1 - M_1) \xrightarrow{*} N'$. By the definition of N_1 and M_1 we conclude that $M \xrightarrow{*} M'$ and $M' + (N - M) \xrightarrow{*} N'$. △

Note that these results hold for any free-choice Petri net. The substate-ordering lemma will be used to prove theorem 2.

5.2 Sound BP-nets which handle multiple cases

Consider the BP-net shown in figure 6. If we add a transition t^* which connects o and i, then the resulting net is live and bounded. Therefore, the BP-net shown in figure 6 is sound. If we put one token in place i, then eventually there will be one token in o and at the same time all the other places will be empty. What happens if we put 10 tokens in place i? Even for the small net shown in figure 6 it is not easy

to see whether some extended soundness property holds. In the following theorem we demonstrate what happens if we take a sound BP-net and put n tokens in place i.

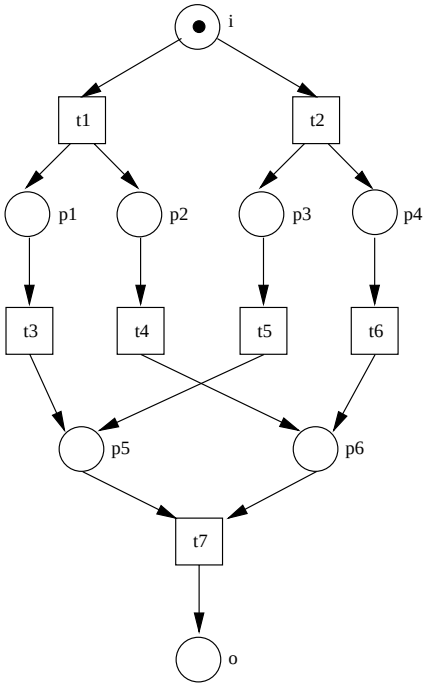

Figure 6: A sound BP-net.

Theorem 2 *If PN is sound, then for every $n \in \mathbb{N}$:*

(i) For every state M reachable from state ni,[2] there exists a firing sequence leading from state M to state no. Formally:

$$\forall_M (ni \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} no)$$

(ii) State no is the only state reachable from state ni with at least n tokens in place o. Formally:

$$\forall_M (ni \xrightarrow{*} M \ \wedge \ M \geq no) \Rightarrow (M = no)$$

Proof.
Assume PN is sound. By theorem 1, we know that $(\overline{PN}, i)$ is live and conservative. Therefore, $\overline{PN}$ has a positive place invariant which assigns identical weights to the places i and o. This invariant also holds for PN. Hence, the only reachable state with at least n tokens in place o is the state no, i.e. (ii) holds.

Before we prove that (i) holds we prove that for any state M reachable from state ni (i.e. $ni \xrightarrow{*} M$), it is possible to reach a dead state N', i.e. (PN, M) is not deadlock-free. Suppose that (PN, M) is deadlock-free. Since (PN, M) is bounded, there is some *recurrent* state X such that $M \xrightarrow{*} X$ and any infinite firing sequence starting from X will visit X infinitely often. Consider all the firing sequences σ such that $X \xrightarrow{\sigma} X$. Let P_X be the set of places "affected" by at least one of these firing sequences. Since PN is a free-choice Petri net, it is easy to verify that P_X is a trap. Clearly, the places i and o are not in P_X. Therefore, P_X is also a trap of $\overline{PN}$. In state ni there are no tokens in trap P_X. By using the Home marking theorem (cf. Best, Desel and Esparza [7]), we deduce that ni is not a home marking of $(\overline{PN}, ni)$. However, state i is a home marking of $(\overline{PN}, i)$ and ni is also a home marking of $(\overline{PN}, ni)$. Based on this contradiction, we deduce that (PN, M) is not deadlock-free.

Remains to prove that for any state M reachable from state ni, there is a firing sequence leading from state M to state no (see (i)). We have just deduced that (PN, M) is not deadlock-free, i.e. given a state M reachable from state ni it is possible to reach some dead state N'.
It suffices to prove that state N' is equal no. We use induction to prove this.

[2] Note that ni is used to denote the state with n tokens in place i; no is used to denote the state with n tokens in place o.

- If $n = 0$ or $n = 1$, this holds by definition. If $n = 0$ the only reachable state is the state without tokens. (This state can be denoted by $0o$.) If $n = 1$, the only reachable dead state is $1o$ (see definition 8).
- Assume $n > 1$. By applying lemma 5 we find that there is a dead state M' such that $i \xrightarrow{*} M'$ and $M' + (ni - i) \xrightarrow{*} N'$. Since PN is sound we know that the only state M' such that $i \xrightarrow{*} M'$ is the state o, i.e. $M' = o$. Hence, $o + (n-1)i \xrightarrow{*} N'$. Since o is a sink place ($o\bullet = \emptyset$), $(n-1)i \xrightarrow{*} N' - o$. The state $N' - o$ is also dead. By the induction hypothesis we conclude that state N' is equal to no.

Hence, (i) also holds. $\triangle$

This theorem shows that if we extend the soundness property to the situation where there are an arbitrary number of tokens in i (in a straightforward manner), then this extended soundness property coincides with the soundness property defined in Section 3.3.

6 Transformation rules

Workflow Management and Business Process Reengineering are marked by the awareness that procedures should be subject to change. Therefore, it is interesting to investigate which changes preserve soundness.

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 business procedure. These transformation rules are shown in figures 7, 8, 9 and 10 and elucidated in the sequel.

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.)

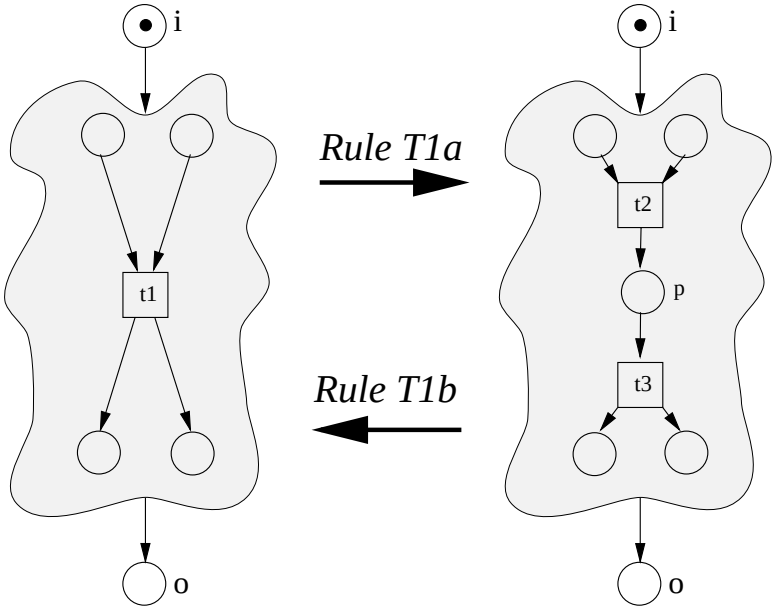

Figure 7: Transformation rules: $T1a$ and $T1b$.

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.)

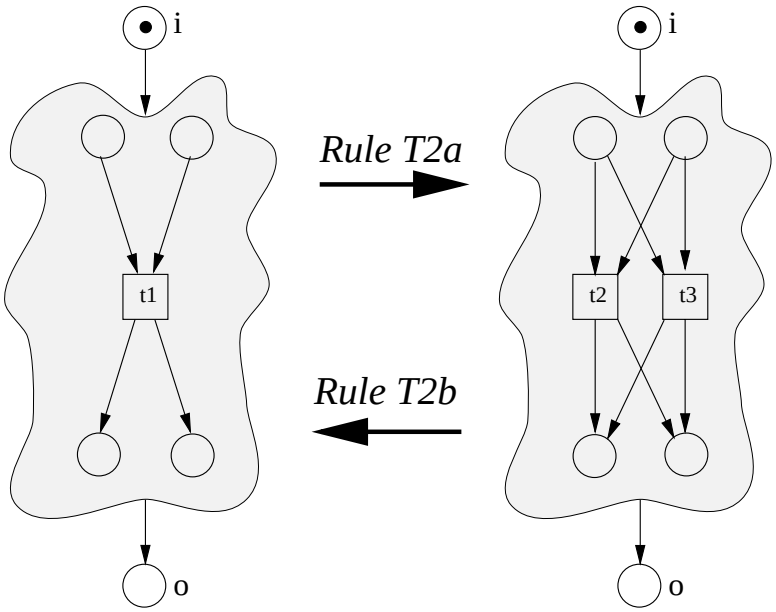

Figure 8: Transformation rules: $T2a$ and $T2b$.

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.

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.

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

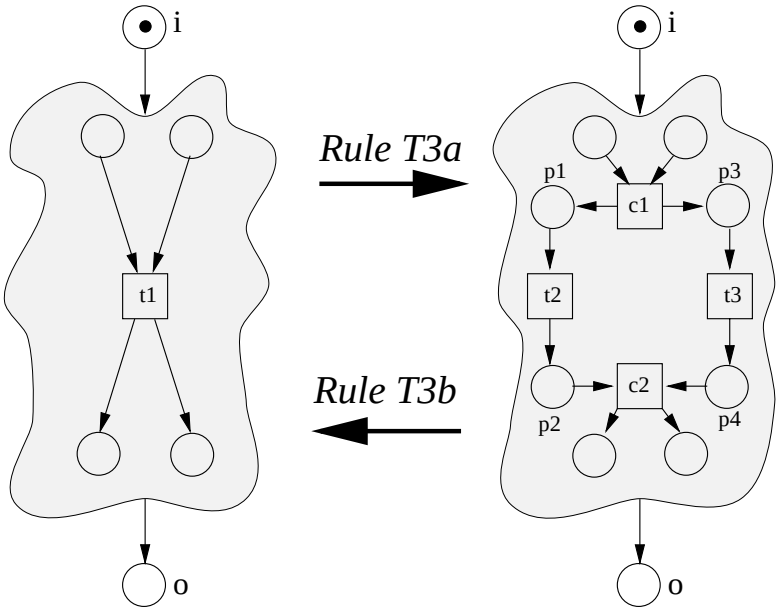

Figure 9: Transformation rules: *T*3*a* and *T*3*b*.

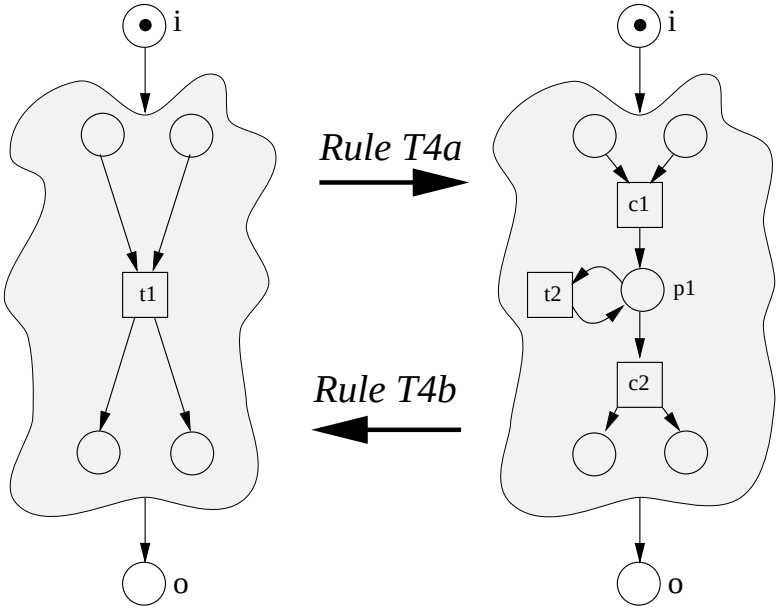

Figure 10: Transformation rules: *T*4*a* and *T*4*b*.

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

Theorem 3 *The transformation rules $T1a$, $T1b$, $T2a$, $T2b$, $T3a$, $T3b$, $T4a$ and $T4b$ preserve soundness, i.e. if a BP-net is sound, then the BP-net transformed by one of these rules is also sound.*

Proof.
We use theorem 1 to prove that the transformation rules preserve soundness. Assume that the net PN is sound. By theorem 1 we know that $(\overline{PN}, i)$ is live and conservative. The transformation rule transforms PN into PN'. $\overline{PN'}$ is the Petri net PN' with an extra transition t^* which connects place o and place i. By theorem 1 we also know that PN' is sound if and only if $(\overline{PN'}, i)$ is live and conservative.
(i) $(\overline{PN'}, i)$ is live
Each of the transformation rules $T1a$, $T1b$, $T2a$, $T2b$, $T3a$, $T3b$, $T4a$ and $T4b$ preserves liveness. It is easy to verify this for each transformation rule. Consider the transformation rules shown in figure 7 and 8. Transition $t1$ is live if and only if $t2$ and $t3$ are live (i.e. $T1a$, $T1b$, $T2a$ and $T2b$ preserve liveness). The transformation rules $T3a$ and $T3b$ (see figure 9) also preserve liveness: $t1$ is live if and only if $c1$, $c2$, $t2$ and $t3$ are live. The transformation rules shown in figure 10 (i.e. $T4a$ and $T4b$) also preserve liveness: $t1$ is live if and only if $c1$, $c2$, and $t2$ are live.
(ii) $\overline{PN'}$ is conservative
$\overline{PN}$ has a positive place-invariant. It is easy to see that this place-invariant can be modified such that it is an invariant of $\overline{PN'}$.
Hence, PN' is sound. $\triangle$

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.

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 BP-net which comprises only one task t is sound. We can use this net as a

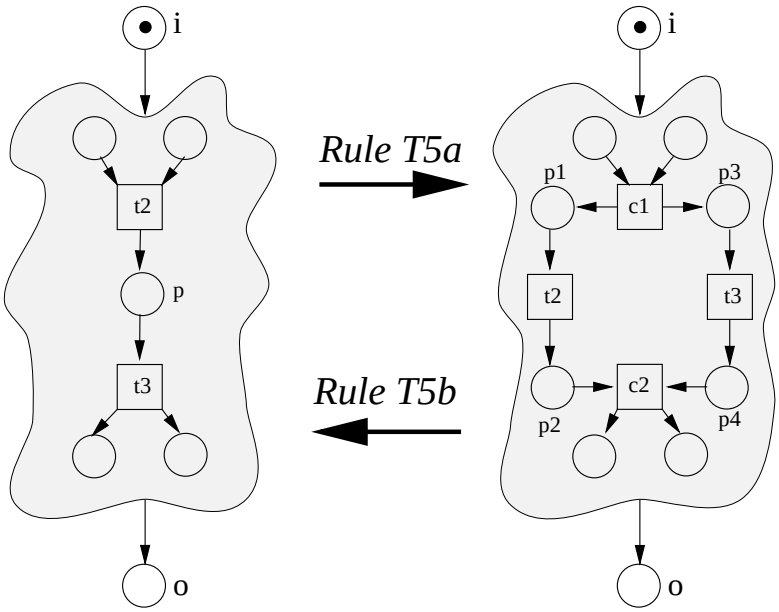

Figure 11: Transformation rules: $T5a$ and $T5b$.

starting point for a sequence of sound transformations. By theorem 3 we know that the resulting BP-net is sound.

Corollary 2 *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 BP-net shown in figure 2. We can construct this net by applying the transformation rules $T1a$, $T2a$ and $T3a$, see figure 12.

Note that the converse of corollary 2 is not true. There are sound BP-nets which cannot be constructed by the transformation rules defined in this section. Consider for example the BP-net shown in figure 6: this net is sound but cannot be constructed by using the transformation rules.

7 Conclusion

In this paper we have presented a class of Petri nets, the so-called BP-nets, suitable for the representation, validation and verification of procedures. One of the merits of this class is that we can verify the soundness property in polynomial time. Even though sound BP-nets have some nice properties from a theoretical point of view, they are powerful enough to model any business procedure. Moreover, we have

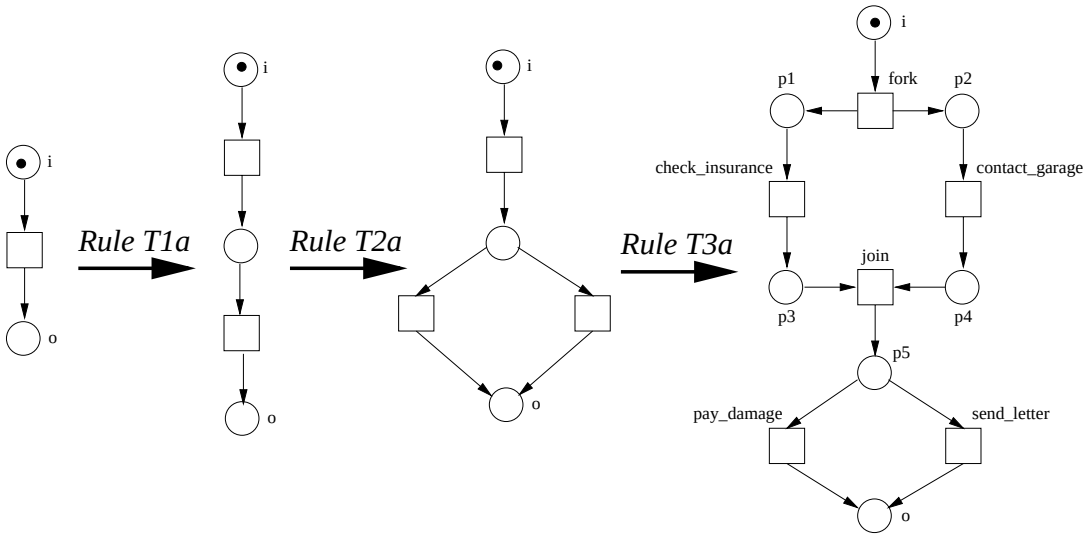

Figure 12: Construction of the BP-net shown in figure 2.

shown that the plausible transformation rules encountered when reengineering a business procedure preserve soundness.

In this paper we focused on the procedure underlying a business process. To completely specify a business 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 of machine) that is going to process the task (cf. Van der Aalst and Van Hee [5, 4]). A direction for further research is to incorporate this dimension. We hope to find a necessary and sufficient condition for soundness given a BP-net extended with some mechanism to allocate resources to tasks.

Acknowledgements

The author would like to thank Dr. M. Voorhoeve for his valuable contribution to Section 5.1 and Ir. A.A. Basten for his useful suggestions.

References

[1] W.M.P. van der Aalst. *Timed coloured Petri nets and their application to logistics*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.

[2] W.M.P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets*

1993, volume 691 of *Lecture Notes in Computer Science*, pages 453–472. Springer-Verlag, Berlin, 1993.

[3] W.M.P. van der Aalst. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.

[4] W.M.P. van der Aalst and K.M. van Hee. Framework for Business Process Redesign. In J.R. Callahan, editor, *Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE 95)*, pages 36–45, Berkeley Springs, April 1995. IEEE Computer Society Press.

[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] 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.

[7] E. Best, J. Desel, and J. Esparza. Traps characterize home states in free-choice systems. *Theoretical Computer Science*, 101:161–176, 1992.

[8] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1381, 1991.

[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] Bakkenist Management Consultants. *ExSpect 4.2 User Manual*, 1994.

[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, UK, 1995.

[13] 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.

[14] M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.

[15] K.M. van Hee. *Information System Engineering: a Formal Approach*. Cambridge University Press, 1994.

[16] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992.

[17] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[18] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, 1981.

[19] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.