Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques

W.M.P. van der Aalst

Eindhoven University of Technology, Faculty of Technology and Management, Department of Information and Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
`w.m.p.v.d.aalst@tm.tue.nl`

Abstract. Workflow management systems facilitate the everyday operation of business processes by taking care of the logistic control of work. In contrast to traditional information systems, they attempt to support frequent changes of the workflows at hand. Therefore, the need for analysis methods to verify the correctness of workflows is becoming more prominent. In this chapter we present a method based on Petri nets. This analysis method exploits the structure of the Petri net to find potential errors in the design of the workflow. Moreover, the analysis method allows for the compositional verification of workflows.

1 Introduction

Workflow management systems (WFMS) are used for the modeling, analysis, enactment, and coordination of structured business processes by groups of people. Business processes supported by a WFMS are *case-driven*, i.e., tasks are executed for specific cases. Approving loans, processing insurance claims, billing, processing tax declarations, handling traffic violations and mortgaging, are typical case-driven processes which are often supported by a WFMS. These case-driven processes, also called *workflows*, are marked by three dimensions: (1) the control-flow dimension, (2) the resource dimension, and (3) the case dimension (see Figure 1). The control-flow dimension is concerned with the partial ordering of tasks, i.e., the workflow *process*. The tasks which need to be executed are identified and the routing of cases along these tasks is determined. Conditional, sequential, parallel and iterative routing are typical structures specified in the control-flow dimension. Tasks are executed by resources. Resources are human (e.g., employee) and/or non-human (e.g., device, software, hardware). In the resource dimension these resources are classified by identifying roles (resource classes based on functional characteristics) and organizational units (groups, teams or departments). Both the control-flow dimension and the resource dimension are generic, i.e., they are not tailored towards a specific case. The third dimension of a workflow is concerned with individual cases which are executed according to the process definition (first dimension) by the proper resources (second dimension).

Managing workflows is not a new idea. Workflow control techniques have existed for decades and many management concepts originating from production and logistics are also applicable in a workflow context. However, just recently, commercially available generic WFMS's have become a reality. Although these systems have been

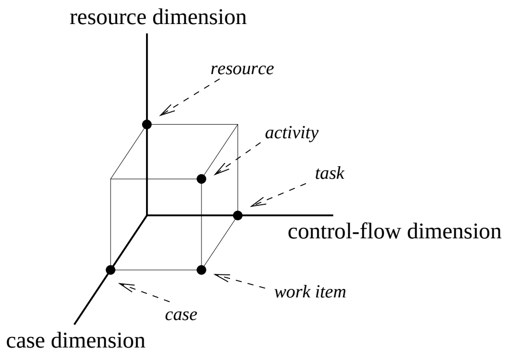

Fig. 1. The three dimensions of workflow.

applied successfully, contemporary WFMS's have at least two important drawbacks. First of all, today's systems do not scale well, have limited fault tolerance and are inflexible. Secondly, a solid theoretical foundation is missing. Most of the more than 250 commercially available WFMS's use a vendor-specific ad-hoc modeling technique to design workflows. In spite of the efforts of the Workflow Management Coalition [25], real standards are missing. The absence of formalized standards hinders the development of tool-independent analysis techniques. As a result, contemporary WFMS's do not facilitate advanced analysis methods to determine the correctness of a workflow.

As many researchers have indicated [11, 18, 26], Petri nets constitute a good starting point for a solid theoretical foundation of workflow management. In this chapter we focus on the control-flow dimension. We use Petri nets to specify the partial ordering of tasks. Based on a Petri-net-based representation of the workflow, we tackle the problem of verification. We will provide techniques to verify the so-called *soundness property* introduced in [2]. A workflow is sound if and only if, for any case, the process terminates properly, i.e., termination is guaranteed, there are no dangling references, and deadlock and livelock are absent.

This chapter extends the results presented in [2]. We will show that in most of the situations encountered in practice, the soundness property can be checked in polynomial time. Moreover, we identify suspicious constructs which may endanger the correctness of a workflow. We will also show that the approach presented in this chapter allows for the compositional verification of workflows, i.e., the correctness of a process can be decided by partitioning it into sound subprocesses. To support the application of the results presented in this chapter, we have developed a Petri-net-based workflow analyzer called *Woflan* [4, 5, 23, 24]. Woflan is a workflow management system independent analysis tool which interfaces with some of the leading products at the Dutch workflow market.

2 Workflow Perspectives

This chapter uses the soundness property as the criterion for correctness. It is clear that this property does not capture all possible errors because it primarily focuses on the control flow. Before we focus on techniques to verify soundness, we discuss the usefulness of a control-flow-based criterion for correctness.

The primary task of a workflow management system is to enact case-driven business processes by joining several perspectives. The following perspectives are relevant for workflow modeling and workflow execution: (1) *control flow* (or process) perspective, (2) *resource* (or organization) perspective, (3) *data* (or information) perspective, (4) *task* (or function) perspective, (5) *operation* (or application) perspective. These perspectives are similar to the perspectives given in [16] and the control flow and resource perspectives correspond to the first two dimensions shown in Figure 1. The third dimension reflects the fact that workflows are case-driven.

In the control-flow perspective, *workflow process definitions* (workflow schemas) are defined to specify which *tasks* need to be executed and in what order (i.e., the routing or control flow). A task is an atomic piece of work. Workflow process definitions are instantiated for specific *cases* (i.e., workflow instances). Since a case is an instantiation of a process definition, it corresponds to the execution of concrete work according to the specified routing. In the *resource* perspective, the organizational structure and the population are specified. The organizational structure describes relations between roles (resource classes based on functional aspects) and groups (resource classes based on organizational aspects). Thus clarifying organizational issues such as responsibility, availability, and authorization. Resources, ranging from humans to devices, form the organizational population and are allocated to roles and groups. The data perspective deals with *control* and *production data*. Control data are data introduced solely for workflow management purposes, e.g., variables introduced for routing purposes. Production data are information objects (e.g., documents, forms, and tables) whose existence does not depend on workflow management. The task perspective describes the elementary operations performed by resources while executing a task for a specific case. In the operational perspective the elementary actions are described. These actions are often executed using applications ranging from a text editor to custom build applications to perform complex calculations. Typically, these applications create, read, or modify control and production data in the information perspective.

This chapter addresses the problem of workflow verification. Although each of the perspectives is relevant, we focus on the control flow perspective. In fact, we focus on the life cycle of one case in isolation. In the remainder of this section, we will motivate why it is reasonable to abstract from the other perspectives when verifying a workflow.

We abstract from the resource perspective because, given today's workflow technology, at any time there is only one resource working on a task which is being executed for a specific case. In today's workflow management systems it is not possible to specify that several resources are collaborating in executing a task. Note that even if multiple persons are executing one task, e.g., writing a report, only one person is allocated to that task from the perspective of the workflow management system: This is the person that selected the work item from the in-basket (i.e., the electronic worktray). Since a person is working on one task at a time and each task is eventually executed by one person (although it may be allocated to a group a people), it is sufficient to check whether all resources classes have at least one resource. In contrast to many other application domains such a flexible manufacturing systems, anomalies such as a deadlock resulting from locking problems are not possible. Therefore, from the viewpoint of verification, i.e., analyzing the logical correctness of a workflow, it is reasonable to abstract from resources. However, if in the future collaborative features are explicitly supported by the workflow management system (i.e., a tight integration of groupware and workflow technology), then the resource perspective should be taken into account.

We partly abstract from the data perspective. The reason we abstract from production data is that these are outside the scope of the workflow management system. These data can be changed at any time without notifying the workflow management system. In fact their existence does not even depend upon the workflow application and they may be shared among different workflows, e.g., the bill-of-material in manufacturing is shared by production, procurement, sales, and quality control processes. The control data used by the workflow management system to route cases are managed by the workflow management system. However, some of these data are set or updated by humans or applications. For example, a decision is made by a manager based on intuition or a case is classified based on a complex calculation involving production data. Clearly, the behavior of a human or a complex application cannot be modeled completely. Therefore, some abstraction is needed to incorporate the data perspective when verifying a given workflow. The abstraction used in this chapter is the following. Since control data (i.e., workflow attributes such as the age of a customer, the department responsible, or the registration date) are only used for the routing of a case, we incorporate the routing decisions but not the actual data. For example, the decision to accept or to reject an insurance claim is taken into account, but not the actual data where this decision is based on. Therefore, we consider each choice to be a non-deterministic one. There are other reasons for abstracting from the workflow attributes. If we are able to prove soundness (i.e., the correctness criterion used in this chapter) for the situation without workflow attributes, it will also hold for the situation with workflow attributes (assuming certain fairness properties). Last but not least, we abstract from triggers and workflow attributes because it allows us to use ordinary Petri nets (i.e., P/T nets) rather than high-level Petri nets. From an analysis point of view, this is preferable because of the availability of efficient algorithms and powerful analysis tools.

For similar reasons we (partly) abstract from the task and operation perspectives. We consider tasks to be atomic and abstract from the execution of operations inside tasks. The workflow management system can only launch applications or trigger people and monitor the results. It cannot control the actual execution of the task. Therefore, from the viewpoint of verification, it is reasonable to focus on the control-flow perspective. In fact, it suffices to consider the life cycle of one case in isolation. The only way cases interact directly is the competition for resources and the sharing of production data. (Note that control data are strictly separated.) Therefore, if we abstract from resources and data, it suffices to consider one case in isolation. The competition between cases for resources is only relevant for performance analysis.

3 Petri Nets

This section introduces the basic Petri net terminology and notations. Readers familiar with Petri nets can skip this section.[1]

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 do not consider multiple arcs from one node to another. 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., $M \in P \to \mathbb{N}$. 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: $p_1 + 2p_2 + p_3$. To compare states we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$

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 a 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 via a (possibly empty) set of intermediate states $M_2, ...M_{n-1}$, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} ... \xrightarrow{t_{n-1}} M_n$

[1] Note that states are represented by weighted sums and note the definition of (elementary) (conflict-free) paths.

A state M_n is called *reachable* from M_1 (notation $M_1 \overset{*}{\rightarrow} M_n$) iff there is a firing sequence σ such that $M_1 \overset{\sigma}{\rightarrow} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \overset{*}{\rightarrow} M_1$.

We use (PN, M) to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M \overset{*}{\rightarrow} M'$.

Let us define some standard properties for Petri nets. First, we define properties related to the dynamics of a Petri net, then we give some structural properties.

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.*

A Petri net is *structurally live* if there exists an initial state such that the net is live.

Definition 3 (Bounded, safe). *A Petri net (PN, M) is bounded iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less than n. The net is safe iff for each place the maximum number of tokens does not exceed 1.*

A Petri net is *structurally bounded* if the net is bounded for any initially state.

Definition 4 (Well-formed). *A Petri net PN is well-formed iff there is a state M such that (PN, M) is live and bounded.*

Paths connect nodes by a sequence of arcs.

Definition 5 (Path, Elementary, Conflict-free). *Let PN be a Petri net. A path C from a node n_1 to a node n_k is a sequence $\langle n_1, n_2, \dots, n_k \rangle$ such that $\langle n_i, n_{i+1} \rangle \in F$ for $1 \leq i \leq k-1$. C is elementary iff, for any two nodes n_i and n_j on C, $i \neq j \;\Rightarrow\; n_i \neq n_j$. C is conflict-free iff, for any place n_j on C and any transition n_i on C, $j \neq i-1 \;\Rightarrow\; n_j \notin \bullet n_i$.*

For convenience, we introduce the alphabet operator α on paths. If $C = \langle n_1, n_2, \dots, n_k \rangle$, then $\alpha(C) = \{n_1, n_2, \dots, n_k\}$.

Definition 6 (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 path leading from x to y.*

Definition 7 (Free-choice). *A Petri net is a free-choice Petri net iff, for every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.*

Definition 8 (State machine). *A Petri net is state machine iff each transition has exactly one input and one output place.*

Definition 9 (S-component). *A subnet $PN_s = (P_s, T_s, F_s)$ is called an S-component of a Petri net $PN = (P, T, F)$ if $P_s \subseteq P$, $T_s \subseteq T$, $F_s \subseteq F$, PN_s is strongly connected, PN_s is a state machine, and for every $q \in P_s$ and $t \in T$: $(q, t) \in F \Rightarrow (q, t) \in F_s$ and $(t, q) \in F \Rightarrow (t, q) \in F_s$.*

Definition 10 (S-coverable). *A Petri net is S-coverable iff for any node there exist an S-component which contains this node.*

See [10, 20] for a more elaborate introduction to these standard notions.

4 WF-Nets

In Figure 1 we indicated that a workflow has (at least) three dimensions. The control-flow dimension is the most prominent one, because the core of any workflow system is formed by the processes it supports. In the control-flow dimension building blocks such as the AND-split, AND-join, OR-split, and OR-join are used to model sequential, conditional, parallel and iterative routing (WFMC [25]). Clearly, a Petri net can be used to specify the routing of cases. *Tasks* are modeled by transitions and causal dependencies are modeled by places and arcs. In fact, a place corresponds to a *condition* which can be used as pre- and/or post-condition for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs. Moreover, in [1] it is shown that the Petri net approach also allows for useful routing constructs absent in many WFMS's.

A Petri net which models the control-flow dimension of a workflow, is called a *WorkFlow net* (WF-net). It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation.

Definition 11 (WF-net). *A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net) if and only if:*

(i) There is one source place $i \in P$ such that $\bullet i = \emptyset$.
(ii) There is one sink place $o \in P$ such that $o\bullet = \emptyset$.
(iii) Every node $x \in P \cup T$ is on a path from i to o.

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 when 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 third requirement in Definition 11 has been added to avoid 'dangling tasks and/or conditions', i.e., tasks and conditions which do not contribute to the processing of cases.

Given the definition of a WF-net it is easy derive the following properties.

Proposition 1 (Properties of WF-nets). *Let $PN = (P, T, F)$ be Petri net.*

- *If PN is WF-net with source place i, then for any place $p \in P$: $\bullet p \neq \emptyset$ or $p = i$, i.e., i is the only source place.*
- *If PN is WF-net with sink place o, then for any place $p \in P$: $p\bullet \neq \emptyset$ or $p = o$, i.e., o is the only sink place.*
- *If PN is a WF-net and we add a transition t^* to PN which connects sink place o with source place i (i.e., $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.*
- *If PN has a source place i and a sink place o and adding a transition t^* which connects sink place o with source place i yields a strongly connected net, then every node $x \in P \cup T$ is on a path from i to o in PN and PN is a WF-net.*

Figure 2 shows a WF-net which models the processing of complaints. First the complaint is registered (task *register*), then in parallel a questionnaire is sent to the complainant (task *send_questionnaire*) and the complaint is evaluated (task *evaluate*). If the

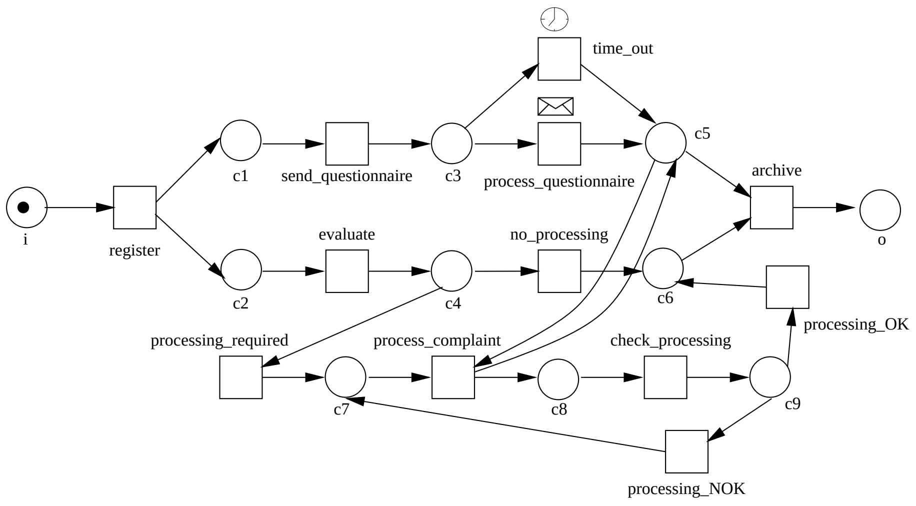

Fig. 2. A WF-net for the processing of complaints.

complainant returns the questionnaire within two weeks, the task *process_questionnaire* is executed. If the questionnaire is not returned within two weeks, the result of the questionnaire is discarded (task *time_out*). Based on the result of the evaluation, the complaint is processed or not. The actual processing of the complaint (task *process_complaint*) is delayed until condition *c5* is satisfied, i.e., the questionnaire is processed or a time-out has occurred. The processing of the complaint is checked via task *check_processing*. Finally, task *archive* is executed. Note that sequential, conditional, parallel and iterative routing are present in this example.

The WF-net shown in Figure 2 clearly illustrates that we focus on the control-flow dimension. We abstract from resources, applications, and technical platforms. Moreover, we also abstract from *case variables* and *triggers*. Case variables are used to resolve choices (OR-split), i.e., the choice between *processing_required* and *no_processing* is (partially) based on case variables set during the execution of task *evaluate*. The choice between *processing_OK* and *processing_NOK* is resolved by testing case variables set by *check_processing*. In the WF-net we abstract from case variables by introducing non-deterministic choices in the Petri-net. If we don't abstract from this information, we would have to model the (unknown) behavior of the applications used in each of the tasks and analysis would become intractable. In Figure 2 we have indicated that *time_out* and *process_questionnaire* require triggers. The clock symbol denotes a time trigger and the envelope symbol denotes an external trigger. Task *time_out* requires a time trigger ('two weeks have passed') and *process_questionnaire* requires a message trigger ('the questionnaire has been returned'). A trigger can be seen as an additional condition which needs to be satisfied. In the remainder of this chapter we abstract from these trigger conditions. We assume that the environment behaves fairly, i.e., the liveness of a transition is not hindered by the continuous absence of a specific trigger. As a result, every trigger condition will be satisfied eventually.

5 Soundness

In this section we summarize some of the basic results for WF-nets presented in [2]. The remainder of this chapter will build on these results.

The three requirements stated in Definition 11 can be verified statically, i.e., they only relate to the structure of the Petri net. However, there is another requirement 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.*

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 requirements correspond to the so-called *soundness property*.

Definition 12 (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:[2]

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

(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 the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 12 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 a strong notion of fairness, then the first requirement implies that eventually state o is reached. Strong fairness means in every infinite firing sequence, each transition fires infinitely often. 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. Note that the traditional notions of fairness (i.e., weaker forms of fairness with just local conditions, e.g., if a transition is enabled infinitely often, it will fire eventually) are not sufficient. See [3, 17] for more details. 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 [14]. The last requirement states that there are no dead transitions (tasks) in the initial state *i*.

[2] 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 3).

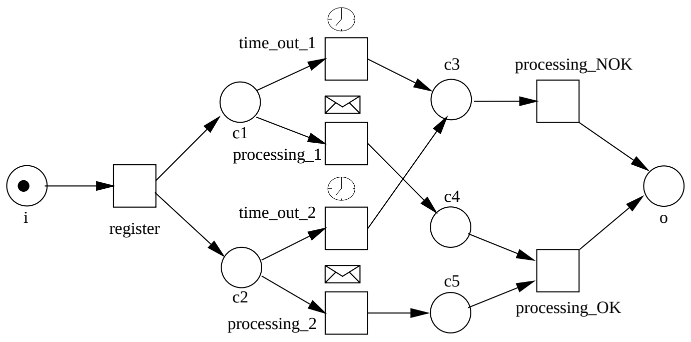

Fig. 3. Another WF-net for the processing of complaints.

Figure 3 shows a WF-net which is not sound. There are several deficiencies. If *time_out_1* and *processing_2* fire or *time_out_2* and *processing_1* fire, the WF-net will not terminate properly because a token gets stuck in *c4* or *c5*. If *time_out_1* and *time_out_2* fire, then the task *processing_NOK* will be executed twice and because of the presence of two tokens in *o* the moment of termination is not clear.

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. In [2] we have shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net obtained 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: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\}$. In the remainder we will call such an extended net the *short-circuited* net of PN. The short-circuited net allows for the formulation of the following theorem.

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

Proof. See [2]. □

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness.

6 Structural Characterization of Soundness

Theorem 1 gives a useful characterization of the quality of a workflow process definition. However, there are a number of problems:

- For a complex WF-net it may be intractable to decide soundness. (For arbitrary WF-nets liveness and boundedness are decidable but also EXPSPACE-hard, cf. Cheng, Esparza and Palsberg [8].)
- Soundness is a minimal requirement. Readability and maintainability issues are not addressed by Theorem 1.
- Theorem 1 does not show how a non-sound WF-net should be modified, i.e., it does not identify constructs which invalidate the soundness property.

These problems stem from the fact that the definition of soundness relates to the dynamics of a WF-net while the workflow designer is concerned with the static structure of the WF-net. Therefore, it is interesting to investigate structural characterizations of sound WF-nets. For this purpose we introduce three interesting subclasses of WF-nets: free-choice WF-nets, well-structured WF-nets, and S-coverable WF-nets.

6.1 Free-Choice WF-Nets

Most of the WFMS's available at the moment, abstract from states between tasks, i.e., states are not represented explicitly. These WFMS's 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 WFMS's, a workflow procedure corresponds to a free-choice Petri net (cf. Definition 7).

It is easy to see that a process definition composed of AND-splits, AND-joins, OR-splits and OR-joins is free-choice. If two transitions t_1 and t_2 share an input place ($\bullet t_1 \cap \bullet t_2 \neq \emptyset$), then they are part of an OR-split, i.e., a 'free choice' between a number of alternatives. Therefore, the sets of input places of t_1 and t_2 should match ($\bullet t_1 = \bullet t_2$). Figure 3 shows a free-choice WF-net. The WF-net shown in Figure 2 is not free-choice; *archive* and *process_complaint* share an input place but the two corresponding input sets differ.

We have evaluated many WFMS's and just one of these systems (COSA [21]) allows for a construct which is comparable to a non-free choice WF-net. Therefore, it makes sense to consider free-choice Petri nets in more detail. Clearly, parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property. 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. Figure 4 shows such a situation. Firing transition *t1* introduces parallelism. Although there is no real choice between *t2* and *t5* (*t5* is not enabled), the parallel execution of *t2* and *t3* results in a situation where *t5* is not allowed to occur. However, if the execution of *t2* is delayed until *t3* has been executed, then there is a real choice between *t2* and *t5*. In our opinion parallelism itself should be separated from the choice between two or more alternatives. Therefore, we consider the non-free-choice construct shown in Figure 4 to be improper. In literature, the term *confusion* is often used to refer to the situation shown in Figure 4.

Free-choice Petri nets have been studied extensively (cf. Best [7], Desel and Esparza [10, 9, 12], Hack [15]) 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 example, the well-known Rank Theorem (Desel and Esparza [10]) enables us to formulate the following corollary.

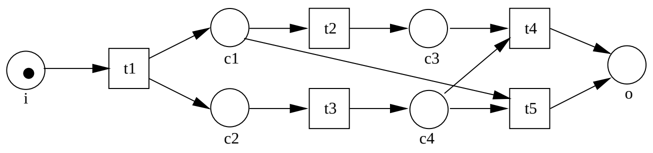

Fig. 4. A non-free-choice WF-net containing a mixture of parallelism and choice.

Corollary 1. *The following problem can be solved in polynomial time.*
Given a free-choice WF-net, to decide if it is sound.

Proof. Let PN be a free-choice WF-net. The short-circuited net $\overline{PN}$ is also free-choice. Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time (Rank Theorem [10]). By Theorem 1, this corresponds to soundness. □

Corollary 1 shows that, for free-choice nets, there are efficient algorithms to decide soundness. Moreover, a sound free-choice WF-net is guaranteed to be safe (given an initial state with just one token in i).

Lemma 1. *A sound free-choice WF-net is safe.*

Proof. Let PN be a sound free-choice WF-net. $\overline{PN}$ is the Petri net PN extended with a transition connecting o and i. $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable [10], i.e., each place is part of an embedded strongly connected state-machine component. Since initially there is just one token $(\overline{PN}, i)$ is safe and so is (PN, i). □

Safeness is a desirable property, because it makes no sense to have multiple tokens in a place representing a condition. A condition is either true (1 token) or false (no tokens).

Although most WFMS's only allow for free-choice workflows, free-choice WF-nets are not a completely satisfactory structural characterization of 'good' workflows. On the one hand, there are non-free-choice WF-nets which correspond to sensible workflows (cf. Figure 2). On the other hand there are sound free-choice WF-nets which make no sense. Nevertheless, the free-choice property is a desirable property. If a workflow can be modeled as a free-choice WF-net, one should do so. A workflow specification based on a free-choice WF-net can be enacted by most workflow systems. Moreover, a free-choice WF-net allows for efficient analysis techniques and is easier to understand. Non-free-choice constructs such as the construct shown in Figure 4 are a potential source of anomalous behavior (e.g., deadlock) which is difficult to trace.

6.2 Well-Structured WF-Nets

Another approach to obtain a structural characterization of 'good' workflows, is to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. As shown in Figure 5, an AND-split should be complemented by an AND-join and an OR-split should be complemented by an OR-join.

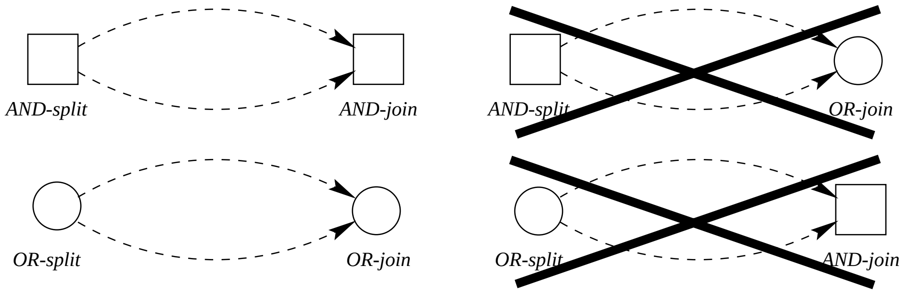

Fig. 5. Good and bad constructions.

One of the deficiencies of the WF-net shown in Figure 3 is the fact that the AND-split *register* is complemented by the OR-join *c3* or the OR-join *o*. To formalize the concept illustrated in Figure 5 we give the following definition.

Definition 13 (Well-handled). *A Petri net PN is well-handled iff, for any pair of nodes x and y such that one of the nodes is a place and the other a transition and for any pair of elementary paths C_1 and C_2 leading from x to y, $\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$.*

Note that the WF-net shown in Figure 3 is not well-handled. Well-handledness can be decided in polynomial time by applying a modified version of the max-flow min-cut technique described in [5]. A Petri net which is well-handled has a number of nice properties, e.g., strong connectedness and well-formedness coincide.

Lemma 2. *A strongly connected well-handled Petri net is well-formed.*

Proof. Let PN be a strongly connected well-handled Petri net. Clearly, there are no circuits that have PT-handles nor TP-handles [13]. Therefore, the net is structurally bounded (See Theorem 3.1 in [13]) and structurally live (See Theorem 3.2 in [13]). Hence, PN is well-formed. □

Clearly, well-handledness is a desirable property for any WF-net PN. Moreover, we also require the short-circuited $\overline{PN}$ to be well-handled. We impose this additional requirement for the following reason. Suppose we want to use PN as a part of a larger WF-net PN'. PN' is the original WF-net extended with an 'undo-task'. See Figure 6. Transition $undo$ corresponds to the undo-task, transitions $t1$ and $t2$ have been added to make PN' a WF-net. It is undesirable that transition $undo$ violates the well-handledness property of the original net. However, PN' is well-handled iff $\overline{PN}$ is well-handled. Therefore, we require $\overline{PN}$ to be well-handled. We use the term *well-structured* to refer to WF-nets whose extension is well-handled.

Definition 14 (Well-structured). *A WF-net PN is well-structured iff $\overline{PN}$ is well-handled.*

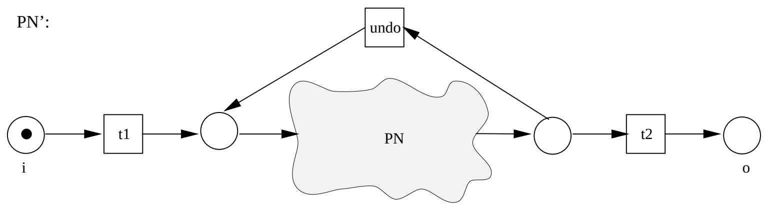

Fig. 6. The WF-net PN' is well-handled iff $\overline{PN}$ is well-handled.

Well-structured WF-nets have a number of desirable properties. Soundness can be verified in polynomial time and a sound well-structured WF-net is safe. To prove these properties we use some of the results obtained for *elementary extended non-self controlling nets*.

Definition 15 (Elementary extended non-self controlling). *A Petri net PN is elementary extended non-self controlling (ENSC) iff, for every pair of transitions t_1 and t_2 such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, there does not exist an elementary path C leading from t_1 to t_2 such that $\bullet t_1 \cap \alpha(C) = \emptyset$.*

Theorem 2. *Let PN be a WF-net. If PN is well-structured, then $\overline{PN}$ is elementary extended non-self controlling.*

Proof. Assume that $\overline{PN}$ is not elementary extended non-self controlling. This means that there is a pair of transitions t_1 and t_k such that $\bullet t_1 \cap \bullet t_k \neq \emptyset$ and there exist an elementary path $C = \langle t_1, p_2, t_2, \ldots, p_k, t_k \rangle$ leading from t_1 to t_k and $\bullet t_1 \cap \alpha(C) = \emptyset$. Let $p_1 \in \bullet t_1 \cap \bullet t_k$. $C_1 = \langle p_1, t_k \rangle$ and $C_2 = \langle p_1, t_1, p_2, t_2, \ldots, p_k, t_k \rangle$ are paths leading from p_1 to t_k. (Note that C_2 is the concatenation of $\langle p_1 \rangle$ and C.) Clearly, C_1 is elementary. We will also show that C_2 is elementary. C is elementary, and $p_1 \notin \alpha(C)$ because $p_1 \in \bullet t_1$. Hence, C_2 is also elementary. Since C_1 and C_2 are both elementary paths, $C_1 \neq C_2$ and $\alpha(C_1) \cap \alpha(C_2) = \{p_1, t_k\}$, we conclude that $\overline{PN}$ is not well-handled. □

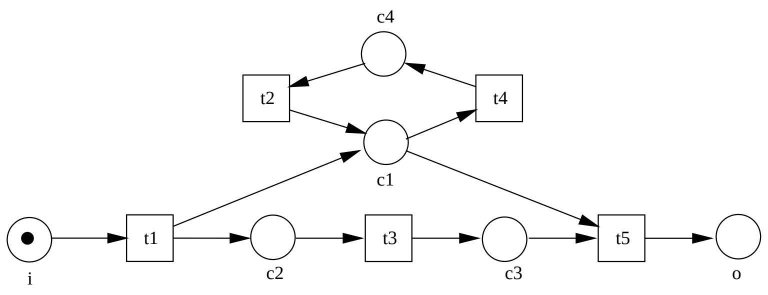

Fig. 7. A well-structured WF-net.

Consider for example the WF-net shown in Figure 7. The WF-net is well-structured and, therefore, also elementary extended non-self controlling. However, the net is not free-choice. Nevertheless, it is possible to verify soundness for such a WF-net very efficiently.

Corollary 2. *The following problem can be solved in polynomial time. Given a well-structured WF-net, to decide if it is sound.*

Proof. Let PN be a well-structured WF-net. The short-circuited net $\overline{PN}$ is elementary extended non-self controlling (Theorem 2) and structurally bounded (see proof of Lemma 2). For bounded elementary extended non-self controlling nets the problem of deciding whether a given marking is live, can be solved in polynomial time (See [6]). Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time. By Theorem 1, this corresponds to soundness. $\square$

Lemma 3. *A sound well-structured WF-net is safe.*

Proof. Let $\overline{PN}$ be the net PN extended with a transition connecting o and i. $\overline{PN}$ is extended non-self controlling. $\overline{PN}$ is covered by state-machines (S-components), see Corollary 5.3 in [6]. Hence, $\overline{PN}$ is safe and so is PN (see proof of Lemma 1). $\square$

Well-structured WF-nets and free-choice WF-nets have similar properties. In both cases soundness can be verified very efficiently and soundness implies safeness. In spite of these similarities, there are sound well-structured WF-nets which are not free-choice (Figure 7) and there are sound free-choice WF-nets which are not well-structured. In fact, it is possible to have a sound WF-net which is neither free-choice nor well-structured (Figures 2 and 4).

6.3 S-Coverable WF-Nets

What about the sound WF-nets shown in Figure 2 and Figure 4? The WF-net shown in Figure 4 can be transformed into a free-choice well-structured WF-net by separating choice and parallelism. The WF-net shown in Figure 2 cannot be transformed into a free-choice or well-structured WF-net without yielding a much more complex WF-net. Place *c5* acts as some kind of milestone which is tested by the task *process_complaint*. Traditional workflow management systems which do not make the state of the case explicit, are not able to handle the workflow specified by Figure 2. Only workflow management systems such as COSA [21] have the capability to enact such a state-based workflow. Nevertheless, it is interesting to consider generalizations of free-choice and well-structured WF-nets: *S-coverable WF-nets* can be seen as such a generalization.

Definition 16 (S-coverable). *A WF-net PN is S-coverable if the short-circuited net $\overline{PN}$ is S-coverable.*

The WF-nets shown in Figure 2 and Figure 4 are S-coverable. The WF-net shown in Figure 3 is not S-coverable. The following two corollaries show that S-coverability is a generalization of the free-choice property and well-structuredness.

Corollary 3. *A sound free-choice WF-net is S-coverable.*

Proof. The short-circuited net $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable (cf. [10]). □

Corollary 4. *A sound well-structured WF-net is S-coverable.*

Proof. $\overline{PN}$ is extended non-self controlling (Theorem 2). Hence, $\overline{PN}$ is S-coverable (cf. Corollary 5.3 in [6]). □

All the sound WF-nets presented in this chapter are S-coverable. Every S-coverable WF-net is safe. The only WF-net which is not sound, i.e., the WF-net shown in Figure 3, is not S-coverable. These and other examples indicate that there is a high correlation between S-coverability and soundness. It seems that S-coverability is one of the basic requirements any workflow process definition should satisfy. From a formal point of view, it is possible to construct WF-nets which are sound but not S-coverable. Typically, these nets contain places which do not restrict the firing of a transition, but which are not in any S-component. (See for example Figure 65 in [19].) From a practical point of view, these WF-nets are to be avoided. WF-nets which are not S-coverable are difficult to interpret because the structural and dynamical properties do not match. For example, these nets can be live and bounded but not structurally bounded. There seems to be no practical need for using constructs which violate the S-coverability property. Therefore, we consider S-coverability to be a basic requirement any WF-net should satisfy.

Another way of looking at S-coverability is the following interpretation: S-components corresponds to *document flows*. To handle a workflow several pieces of information are created, used, and updated. One can think of these pieces of information as physical documents, i.e., at any point in time the document is in one place in the WF-net. Naturally, the information in one document can be copied to another document while executing a task (i.e., transition) processing both documents. Initially, all documents are present but a document can be empty (i.e., corresponds to a blank piece paper). It is easy to see that the flow of one such document corresponds a state machine (assuming the existence of a transition t^*). These document flows synchronize via joint tasks. Therefore, the composition of these flows yields an S-coverable WF-net. One can think of the document flows as threads. Consider for example the short-circuited net of the WF-net shown in Figure 2. This net can be composed out of the following two threads: (1) a thread corresponding to the processing of the form (places i, $c1$, $c3$, $c5$ and o) and (2) a thread corresponding to the actual processing of the complaint (places i, $c2$, $c4$, $c5$, $c6$, $c7$, $c8$, and $c9$). Note that the tasks *register* and *archive* are used in both threads.

Although a WF-net can, in principle, have exponentially many S-components, they are quite easy to compute for workflows encountered in practice (see also the above interpretation of S-component as document flows or threads). Note that S-coverability only depends on the structure and the degree of connectedness is generally low (i.e., the incidence matrix of a WF-net typically has few non-zero entries [5]). Unfortunately, in general, it is not possible to verify soundness of an S-coverable WF-net in polynomial time. The problem of deciding soundness for an S-coverable WF-net is PSPACE-complete. For most applications this is not a real problem. In most cases the number of tasks in one workflow process definition is less than 100 and the number of states is less than 200,000. Tools using standard techniques such as the construction of the coverability graph have no problems in coping with these workflow process definitions.

6.4 Summary

The three structural characterizations (free-choice, well-structured and S-coverable) turn out to be very useful for the analysis of workflow process definitions. Based on our experience, we have good reasons to believe that S-coverability is a desirable property any workflow definition should satisfy. Constructs violating S-coverability can be detected easily and tools can be build to help the designer to construct an S-coverable WF-net. S-coverability is a generalization of well-structuredness and the free-choice property (Corollary 3 and 4). Both well-structuredness and the free-choice property also correspond to desirable properties of a workflow. A WF-net satisfying at least one one of these two properties can be analyzed very efficiently. However, we have shown that there are workflows that are not free-choice and not well-structured. Consider for example Figure 2. The fact that task *process_complaint* tests whether there is a token in *c5*, prevents the WF-net from being free-choice or well-structured. Although this is a very sensible workflow, most workflow management systems do not support such an advanced routing construct. Even if one is able to use state-based workflows (e.g., COSA) allowing for constructs which violate well-structuredness and the free-choice property, then the structural characterizations are still useful. If a WF-net is not free-choice or not well-structured, one should locate the source which violates one of these properties and check whether it is really necessary to use a non-free-choice or a non-well-structured construct. If the non-free-choice or non-well-structured construct is really necessary, then the correctness of the construct should be double-checked, because it is a potential source of errors. This way the readability and maintainability of a workflow process definition can be improved.

7 Composition of WF-Nets

The WF-nets in this chapter are very simple compared to the workflows encountered in practise. For example, in the Dutch Customs Department there are workflows consisting of more than 80 tasks with a very complex interaction structure (cf. [1]). For the designer of such a workflow the complexity is overwhelming and communication with end-users using one huge diagram is difficult. In most cases hierarchical (de)composition is used to tackle this problem. A complex workflow is decomposed into subflows and each of the subflows is decomposed into smaller subflows until the desired level of detail is reached. Many WFMS's allow for such a hierarchical decomposition. In addition, this mechanism can be utilized for the reuse of existing workflows. Consider for example multiple workflows sharing a generic subflow. Some WFMS-vendors also supply reference models which correspond to typical workflows in insurance, banking, finance, marketing, purchase, procurement, logistics, and manufacturing.

Reference models, reuse and the structuring of complex workflows require a hierarchy concept. The most common hierarchy concept supported by many WFMS's is *task*

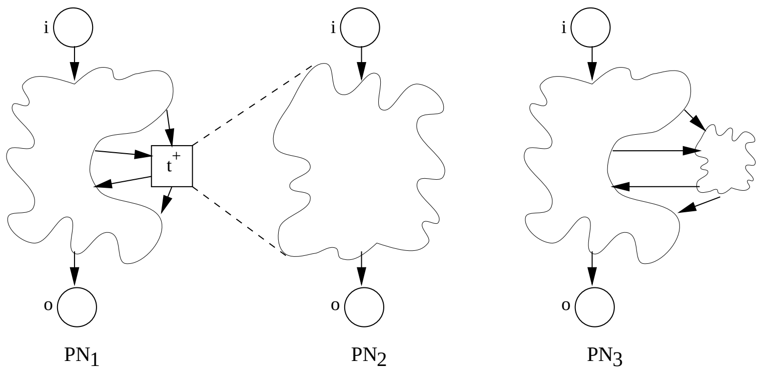

Fig. 8. Task refinement: WF-net PN_3 is composed of PN_1 and PN_2.

refinement, i.e., a task can be refined into a subflow. This concept is illustrated in Figure 8. The WF-net PN_1 contains a task t^+ which is refined by another WF-net PN_2, i.e., t^+ is no longer a task but a reference to a subflow. A WF-net which represents a subflow should satisfy the same requirements as an ordinary WF-net (see Definition 11). The semantics of the hierarchy concept are straightforward; simply replace the refined transition by the corresponding subnet. Figure 8 shows that the refinement of t^+ in PN_1 by PN_2 yields a WF-net PN_3.

The hierarchy concept can be exploited to establish the correctness of a workflow. Given a complex hierarchical workflow model, it is possible to verify soundness by analyzing each of the subflows separately. The following observation is important for compositionality.

Lemma 4. *Let $PN = (P, T, F)$ be a sound WF-net. For any $t \in T$, (i) if $t \in i\bullet$, then $\bullet t = \{i\}$, and (ii) if $t \in \bullet o$, then $t\bullet = \{o\}$.*

Proof. We prove (i) by contradiction. If $t \in i\bullet$ and $\bullet t \neq \{i\}$, then there exists a $p \in (\bullet t) \setminus \{i\}$. Clearly, t is dead because i and p cannot be marked at the same time. The proof of (ii) is similar. $\square$

The following theorem shows that the soundness property defined in this chapter allows for modular analysis.

Theorem 3 (Compositionality). *Let $PN_1 = (P_1, T_1, F_1)$ and $PN_2 = (P_2, T_2, F_2)$ be two WF-nets such that $T_1 \cap T_2 = \emptyset$, $P_1 \cap P_2 = \{i, o\}$ and $t^+ \in T_1$. $PN_3 = (P_3, T_3, F_3)$ is the WF-net obtained by replacing transition t^+ in PN_1 by PN_2, i.e., $P_3 = P_1 \cup P_2$, $T_3 = (T_1 \setminus \{t^+\}) \cup T_2$ and*

$$\begin{aligned} F_3 = {} & \{(x, y) \in F_1 \mid x \neq t^+ \ \wedge \ y \neq t^+\} \ \cup \ \{(x, y) \in F_2 \mid \{x, y\} \cap \{i, o\} = \emptyset\} \ \cup \\ & \{(x, y) \in P_1 \times T_2 \mid (x, t^+) \in F_1 \ \wedge \ (i, y) \in F_2\} \cup \\ & \{(x, y) \in T_2 \times P_1 \mid (t^+, y) \in F_1 \ \wedge \ (x, o) \in F_2\}. \end{aligned}$$

For PN_1, PN_2 and PN_3 the following statements hold:

1. *If PN_3 is free-choice, then PN_1 and PN_2 are free-choice.*
2. *If PN_3 is well-structured, then PN_1 and PN_2 are well-structured.*
3. *If (PN_1, i) is safe and PN_1 and PN_2 are sound, then PN_3 is sound.*
4. *(PN_1, i) and (PN_2, i) are safe and sound iff (PN_3, i) is safe and sound.*
5. *PN_1 and PN_2 are free-choice and sound iff PN_3 is free-choice and sound.*
6. *If PN_3 is well-structured and sound, then PN_1 and PN_2 are well-structured and sound.*
7. *If $\bullet t^+$ and $t^+\bullet$ are both singletons, then PN_1 and PN_2 are well-structured and sound iff PN_3 is well-structured and sound.*

Proof.

1. The only transitions that may violate the free-choice property are t^+ (in PN_1) and $\{t \in T_2 \mid (i,t) \in F_2\}$ (in PN_2). Transition t^+ has the same input set as any of the transitions $\{t \in T_2 \mid (i,t) \in F_2\}$ in PN_3 if we only consider the places in $P_3 \cap P_1$. Hence, t^+ does not violate the free-choice property in PN_1. All transitions t in PN_2 such that $(i,t) \in F_2$ respect the free-choice property; the input places in $P_3 \setminus P_2$ are replaced by i.
2. $\overline{PN_1}$ ($\overline{PN_2}$) is well-handled because any elementary path in $\overline{PN_1}$ ($\overline{PN_2}$) corresponds to a path in $\overline{PN_3}$.
3. Let (PN_1, i) be safe and let PN_1 and PN_2 be sound. We need to prove that $(\overline{PN_3}, i)$ is live and bounded. The subnet in $\overline{PN_3}$ which corresponds to t^+ behaves like a transition which may postpone the production of tokens for $t^+\bullet$. It is essential that the input places of t^+ in $(\overline{PN_3}, i)$ are safe. This way it is guaranteed that the states of the subnet correspond to the states of $(\overline{PN_2}, i)$. Hence, the transitions in $T_3 \cap T_2$ are live (t^+ is live) and the places in $P_3 \setminus P_1$ are bounded. Since the subnet behaves like t^+, the transitions in $T_3 \cap (T_1 \setminus \{t^+\})$ are live and the places in $P_3 \cap P_1$ are bounded. Hence, PN_3 is sound.
4. Let (PN_1, i) and (PN_2, i) be safe and sound. Clearly, PN_3 is sound (see proof of 3.). (PN_3, i) is also safe because every reachable state corresponds to a combination of a safe state of (PN_1, i) and a safe state of (PN_2, i).
   Let (PN_3, i) be safe and sound. Consider the subnet in PN_3 which corresponds to t^+. X is the set of transitions in $T_3 \cap T_2$ consuming from $\bullet t^+$ and Y is the set of transitions in $T_3 \cap T_2$ producing tokens for $t^+\bullet$. If a transition in X fires, then it should be possible to fire a transition in Y because of the liveness of the original net. If a transition in Y fires, the subnet should become empty. If the subnet is not empty after firing a transition in Y, then there are two possibilities: (1) it is possible to move the subnet to a state such that a transition in Y can fire (without firing transitions in $T_3 \cap T_1$) or (2) it is not possible to move to such a state. In the first case, the places $t^+\bullet$ in PN_3 are not safe. In the second case, a token is trapped in the subnet or the subnet is not safe the moment a transition in X fires. (PN_2, i) corresponds to the subnet bordered by X and Y and is, as we have just shown, sound and safe. It remains to prove that (PN_1, i) is safe and sound. Since the subnet which corresponds to t^+ behaves like a transition which may postpone the production of tokens, we can replace the subnet by t^+ without changing dynamic properties such as safeness and soundness.

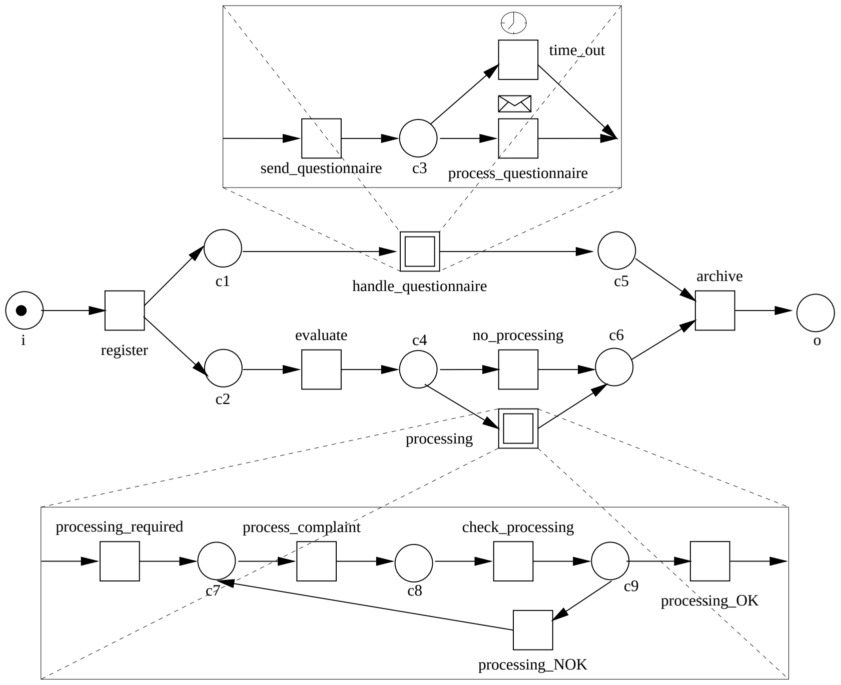

Fig. 9. A hierarchical WF-net for the processing of complaints.

5. Let PN_1 and PN_2 be free-choice and sound. Since $(\overline{PN_1}, i)$ is safe (see Lemma 1), PN_3 is sound (see proof of 3.). It remains to prove that PN_3 is free-choice. The only transitions in PN_3 which may violate the free-choice property are the transitions in $T_3 \cap T_2$ consuming tokens from $\bullet t^+$. Because PN_2 is sound, these transitions need to have an input set identical to $\bullet t^+$ in PN_1 (cf. Lemma 4). Since PN_1 is free-choice, PN_3 is also free-choice.
   Let PN_3 be free-choice and sound. PN_1 and PN_2 are also free-choice (see proof of 1.). Since (PN_3, i) is safe (see Lemma 1), PN_1 and PN_2 are sound (see proof of 4.).
6. Let PN_3 be well-structured and sound. PN_1 and PN_2 are also well-structured (see proof of 2.). Since (PN_3, i) is safe (see Lemma 3), PN_1 and PN_2 are sound (see proof of 4.).
7. It remains to prove that if PN_1 and PN_2 are well-structured, then PN_3 is also well-structured. Suppose that PN_3 is not well-structured. In this case, there is a pair of nodes x and y such that one of the nodes is a place and the other a transition and such that there are two disjoint elementary paths leading from x to y in $\overline{PN_3}$ (cf. Definitions 13 and 14). Since PN_1 is well-structured, at least one of these paths runs via the refinement of t^+. However, because t^+ has precisely one input and one output place and PN_2 is also well-structured, this is not possible.

$\square$

Theorem 3 is a generalization of Theorem 3 in [22]. It extends the concept of a block with multiple entry and exit transitions and gives stronger results for specific subclasses.

Figure 9 shows a hierarchical WF-net. Both of the subflows (*handle_questionnaire* and *processing*) and the main flow are safe and sound. Therefore, the overall workflow represented by the hierarchical WF-net is also safe and sound. Moreover, the free-choice property and well-structuredness are also preserved by the hierarchical composition. Theorem 3 is of particular importance for the reuse of subflows. For the analysis of a complex workflow, every safe and sound subflow can be considered to be a single task. This allows for an efficient modular analysis of the soundness property. Moreover, the statements embedded in Theorem 3 can help a workflow designer to construct correct workflow process definitions.

8 Conclusion

In this chapter we have investigated a basic property that any workflow process definition should satisfy: the soundness property. For WF-nets, this property coincides with liveness and boundedness. In our quest for a structural characterization of WF-nets satisfying the soundness property, we have identified three important subclasses: free-choice, well-structured, and S-coverable WF-nets. The identification of these subclasses is useful for the detection of design errors.

If a workflow is specified by a hierarchical WF-net, then modular analysis of the soundness property is often possible. A workflow composed of correct subflows can be verified without incorporating the specification of each subflow.

The results presented in this chapter give workflow designers a handle to construct correct workflows. Although it is possible to use standard Petri-net-based analysis tools, we have developed a workflow analyzer which can be used by people not familiar with Petri-net theory [4, 5, 23, 24]. This workflow analyzer interfaces with existing workflow products such as Staffware, COSA, METEOR, and Protos.

Acknowledgements
The author would like to thank Marc Voorhoeve and Twan Basten for their valuable suggestions and all the other people involved in the development of Woflan, in particular Eric Verbeek and Dirk Hauschildt.

References

1. 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.
2. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
3. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

4. W.M.P. van der Aalst. Woflan: A Petri-net-based Workflow Analyzer. *Systems Analysis - Modelling - Simulation*, 35(3):345–357, 1999.
5. W.M.P. van der Aalst, D. Hauschildt, and H.M.W. Verbeek. A Petri-net-based Tool to Analyze Workflows. In B. Farwer, D. Moldt, and M.O. Stehr, editors, *Proceedings of Petri Nets in System Engineering (PNSE'97)*, pages 78–90, Hamburg, Germany, September 1997. University of Hamburg (FBI-HH-B-205/97).
6. K. Barkaoui, J.M. Couvreur, and C. Dutheillet. On liveness in Extended Non Self-Controlling Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag, Berlin, 1995.
7. 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.
8. 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.
9. 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.
10. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
11. 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.
12. 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.
13. J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer-Verlag, Berlin, 1990.
14. 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.
15. M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.
16. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
17. E. Kindler and W.M.P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 1999 (to appear).
18. G. De Michelis, C. Ellis, and G. Memmi, editors. *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, Zaragoza, Spain, June 1994.
19. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer-Verlag, Berlin, 1985.
20. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
21. Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1998.
22. R. Valette. Analysis of Petri Nets by Stepwise Refinements. *Journal of Computer and System Sciences*, 18:35–46, 1979.
23. E. Verbeek and W.M.P. van der Aalst. Woflan Home Page. http://www.win.tue.nl/˜woflan.

24. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. Computing Science Report 99/02, Eindhoven University of Technology, Eindhoven, 1999.
25. WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.
26. M. Wolf and U. Reimer, editors. *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow*, Basel, Switzerland, Oct 1996.