An Alternative Way to Analyze Workflow Graphs

W. M. P. van der Aalst, A. Hirnschall, and H. M. W. Verbeek

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

Abstract. At the CAiSE conference in Heidelberg in 1999, Wasim Sadiq and Maria Orlowska presented an algorithm to verify workflow graphs [19]. The algorithm uses a set of reduction rules to detect structural conflicts. This paper shows that the set of reduction rules presented in [19] is not complete and proposes an alternative algorithm. The algorithm translates workflow graphs into so-called WF-nets. WF-nets are a class of Petri nets tailored towards workflow analysis. As a result, Petri-net theory and tools can be used to verify workflow graphs. In particular, our workflow verification tool Woflan [21] can be used to detect design errors. It is shown that the absence of structural conflicts, i.e., deadlocks and lack of synchronization, conforms to soundness of the corresponding WF-net [2]. In contrast to the algorithm presented in [19], the algorithm presented in this paper is complete. Moreover, the complexity of this alternative algorithm is given.

1 Introduction

Business processes can be formally defined by process models that need to be correct in order to not directly affect business objectives negatively. Proper definition, analysis, verification, and refinement of these models is indispensable before enacting the process model using a workflow management system. There are several aspects of a process model including process structure, data flow, roles, application interface, temporal constraints, and others. The techniques used in this paper, i.e., workflow graphs [19,20] and workflow nets [2], focus on the process structure. The structure of a workflow defines the way of execution, scheduling, and coordination of workflow tasks.

Various approaches to workflow modeling can be found in literature [2,4,7,9,12,13,16,18,19]. Most workflow management systems use a proprietary workflow language. Despite the standardization efforts of the Workflow Management Coalition [13] a "lingua franca" is still missing. The specification of Interface 1/WPDL is ambiguous (no formal semantics is given) and its expressive power is limited. Moreover, the languages of many existing tools and Interface 1/WPDL do not provide starting point for workflow analysis. Therefore,

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 535–552, 2002.

techniques such as workflow graphs [19,20] and workflow nets [2] have been proposed. Workflow nets are based on Petri nets and the application of these nets has been explored by many authors [1,7]. Workflow graphs have been introduced by Wasim Sadiq and Maria Orlowska as a more direct way of modeling workflow processes [19,20].

The design of large workflow specifications can result in hidden errors, which may lead to undesirable execution of some or all possible instances of a workflow. These problems should be corrected during the design phase rather than after deploying the workflow application. Only limited work in literature covers workflow verification. Some issues of workflow structure verification have been examined in [9] together with complexity evaluations. In [18] the issue of correctness in workflow modeling has been identified.

This paper shows that the set of reduction rules for the detection of structural conflicts presented by Wasim Sadiq and Maria Orlowska in [19] is not complete. Instead an alternative algorithm is presented that translates workflow graphs into workflow nets. Workflow nets are a subclass of Petri nets tailored toward workflow analysis [2]. Through this translation it is possible to verify workflow graphs using Petri-net-based analysis tools such as Woflan [21]. In contrast to the technique described in [19], the algorithm presented in this paper is complete. Moreover, the computational complexity of our approach is at least as good as other analysis techniques specifically tailored towards workflow graphs [14].

In this paper we first present the definition of a workflow graph together with its consistency and correctness criteria. A counter example showing that the reduction rules in [19] are not complete and an alternative algorithm and its complexity [14], are presented in Section 3. Section 4 defines Petri nets, workflow nets, and verification criteria. Section 5 outlines an algorithm for mapping workflow graphs onto workflow nets. Woflan, a tool for analyzing workflow process definitions specified in terms of Petri nets is described in Section 6. Section 7 draws the conclusion that the algorithm presented in this paper is complete, efficient, and allows for more advanced constructs such as arbitrary cycles.

2 Workflow Graphs

Figure 1 shows process modeling objects that may be *nodes* or *edges*. The *control flow relation* links two nodes in a graph and shows the execution order. A *node* can either be a *task* or a *choice/merge coordinator*. A *task* stands for work required to reach an objective and is used to build *forks* and *synchronizers*. *Choice/merge coordinators* are represented by a circle. In a workflow graph two nodes are linked together by a control flow relation represented by a directed edge. It shows the execution order between *start tasks* and *end tasks* of a workflow graph.

A *sequence* consists of a node that has an incoming and an outgoing arc.

A *fork* node allows independent execution between concurrent paths within a workflow graph and is modeled by connecting two or more outgoing control flow relations to a task.

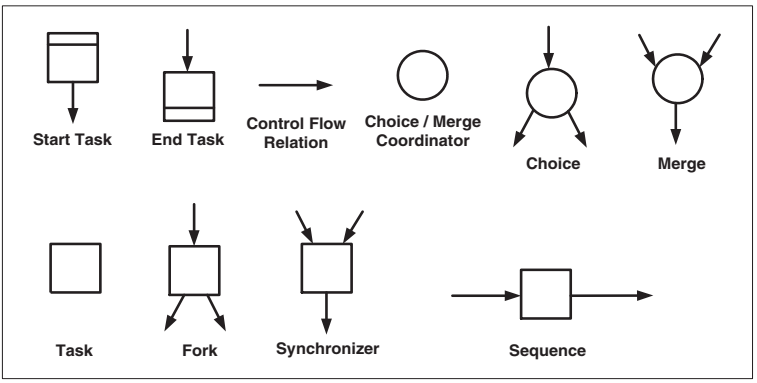

Fig. 1. Process modeling objects

A *synchronizer* node with more than one incoming control flow relation is applied to synchronize such concurrent paths. A synchronizer waits until all incoming control flow relations have lead into the task.

A *choice* node has two or more outgoing control flow relations resulting in mutually exclusive alternative paths. This ensures that only one alternative outgoing control flow relation is selected at run-time.

A *merge* node is the counterpart of the choice node and has two or more incoming control flow relations. It joins mutually exclusive alternative paths into one path.

Definition 1 (Workflow graph). *A workflow graph is a tuple* $WG = (N, T, T^S, T^E, C, F)$*:*

- N *is a finite set of nodes,*
- $T \subseteq N$ *is a finite set of tasks,*
- $T^S \subseteq T$ *is a finite set of start tasks,*
- $T^E \subseteq T$ *is a finite set of end tasks,*
- $C \subseteq N$ *is a finite set of choice/merge coordinators,*
- $N = T \cup C$*, and*
- $F \subseteq N \times N$ *is the control flow relation.*

The relation F defines a directed graph with nodes N and arcs F. In this directed graph, we can define the input nodes and the output nodes of a given node. $\bullet x = \{y \in N \quad Fx\}$ is the set of input nodes of $x \in N$ and $x\bullet = \{y \in N \quad Fy\}$ is the set of output nodes of x.

Figure 2 shows a workflow graph in the left column. The nodes are represented by rectangles and circles where the first stand for tasks and the latter for choice/merge coordinators. *C1* and *C2* are choice coordinators. *C3* is a merge coordinator. The start and end tasks of the workflow graph are marked as *T1* and *T9* respectively. Control flow relations are modeled as arcs between the nodes.

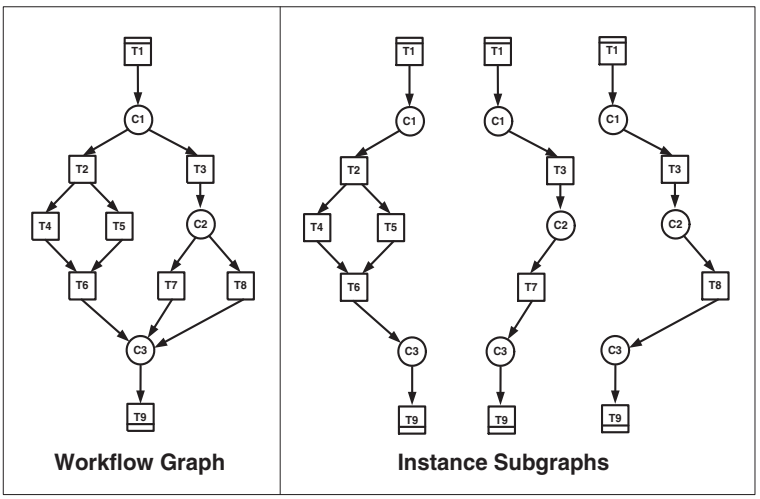

Fig. 2. A workflow graph (left) and its three instance subgraphs (right)

Task *T2* serves as an input node for *T4* while the latter task represents an output node of *T2*.

Definition 2 allows for graphs which are unconnected, without start/end tasks, tasks without any input and output, etc. Therefore we need to restrict the definition to consistent workflow graphs.

Definition 2 (Consistent). *A workflow graph* $WG = (N, T, T^S, T^E, C, F)$ *is consistent if:*

- *for all* $t \in T$: $\bullet t = \emptyset$ *if and only if* $t \in T^S$,
- *for all* $t \in T$: $t\bullet = \emptyset$ *if and only if* $t \in T^E$,
- (N, F) *is a directed acyclic graph, and*
- *every node is on a path from some start task to some end task, i.e., for all* $n \in N$: *there is a* $t^s \in T^S$ *and a* $t^e \in T^E$ *such that* $t^s F^* n$ *and* $nF^* t^e$.

In the remainder we only consider consistent workflow graphs. Moreover, without loosing generality we assume that both T^S and T^E are singletons, i.e., $T^S = \{t^s\}$ and $T^E = \{t^e\}$.

We need to define the concept of instance subgraphs before presenting the correctness criteria for workflow graphs. The right column of Figure 2 shows which possible paths the execution of the workflow graph in the left column might take. Choice coordinator *C1* can lead a token to the fork *T2* or to task *T3*. In the latter case the choice coordinator *C2* leads to the creation of two possible paths of workflow instances. Thus, each of these instance subgraphs represents a subset of workflow tasks that may be executed for a particular instance of a workflow. They can be generated by visiting a workflow graph's

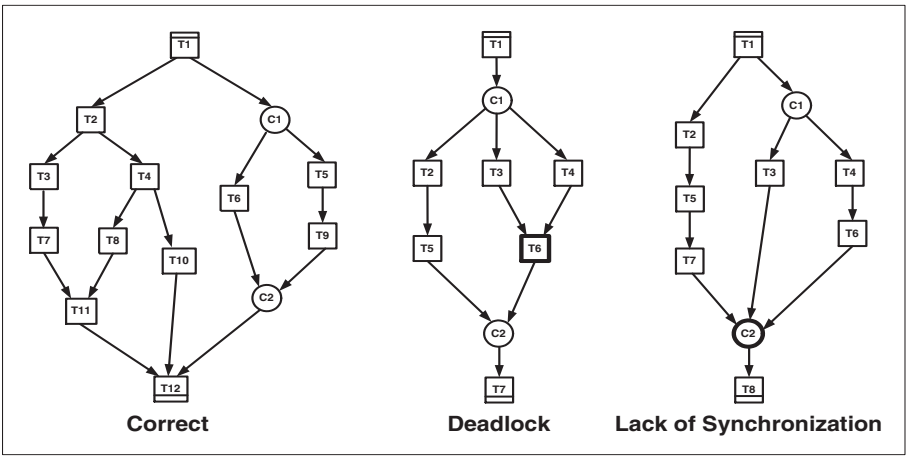

Fig. 3. A correct workflow graph and two incorrect ones exhibiting deadlock and lack of synchronization

nodes on the semantic basis of underlying modeling structures. The subgraph representing the visited nodes and flows forms an instance subgraph.

The semantics of a workflow graph are given by the set of instance subgraphs. Note that instance subgraphs correspond to the concept of runs/occurrence graphs of Petri nets [17]. The concept of instance subgraphs allows us to define the following notion of correctness.

Definition 3 (Correctness criteria). *A workflow graph is correct if and only if there are no structural conflicts:*

- *Correctness criterion 1*
  Deadlock free workflow graphs: A workflow graph is free of deadlock structural conflicts if it does not generate an instance subgraph that contains only a proper subset of the incoming nodes of an and-join node (i.e., synchronizer).
- *Correctness criterion 2*
  Lack of synchronization free workflow graphs: A workflow graph is free of lack of synchronization structural conflicts if it does not generate an instance subgraph that contains more than one incoming node of an or-join node (e.g., a merge).

It has been mentioned that all split structures introduced after a start task must be closed through a join structure before reaching the final structure. Thus, a synchronizer is used for joining fork-split paths and a merge for choice coordinator-split paths. Figure 3 shows examples for a *deadlock error* and *lack of synchronization*. Joining the choice coordinator *C1* with the synchronizer *T6* leads to a deadlock. Similarly, joining the multiple paths leaving start task *T1* with the merge coordinator *C2* introduces a lack of synchronization conflict. It

means that the merge coordinator results in unintentional multiple activation of nodes that follow the merge coordinator.

3 An Algorithm and a Counter Example

As mentioned in Section 1, in [19] a set of reduction rules is presented. The authors claim that, using these rules, a correct workflow graph can be reduced to an empty workflow graph, whereas an incorrect workflow graph cannot be reduced to that extent. In [14], a counter example is presented showing that some correct workflow graphs cannot be reduced to the empty workflow graphs. This section briefly discusses the set of reduction rules and presents another counter example.

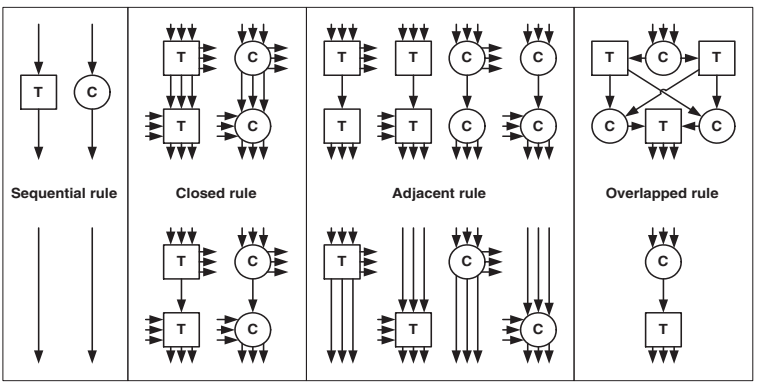

Fig. 4. Reduction rules of [19]

The set of reduction rules as presented in [19] consist of four rules: the *sequential* rule, the *adjacent* rule, the *closed* rule, and the *overlapped* rule. [20] claims that the complexity of applying these four rules is $O(n^2)$, where $n = |N| + |F|$.

- The sequential rule reduces sequential nodes, that is, nodes that have exactly one input node and one output node. A sequential node is reduced by removing it from the graph and adding an arc from its input node to its output node.
- The closed rule collapses multiple arcs between nodes of the same type to a single arc. Note that this rule is slightly out of the ordinary, because in a workflow graph (which is basically a directed acyclic graph [19]) multiple arcs cannot exist. Evidently, in a *reduced* workflow graph multiple arcs are allowed to exist.
- The adjacent rule reduces adjacent nodes, that is, nodes that have exactly one input node or one output node, and where the input node or output

node is of the same type. An adjacent node is reduced by removing it from the graph and adding arcs connecting all its input nodes to all its output nodes.

– The overlapped rule reduces a subgraph in between a coordinator and a task, provided that the coordinator has only tasks as output nodes, the task has only coordinators as input nodes, every input node of the task is an output node for every output node of the coordinator, and every output node of the coordinator is an input node of the task. This subgraph is reduced by removing all output nodes of the coordinator and all input nodes of the task from the graph and adding an arc from the coordinator to the task.

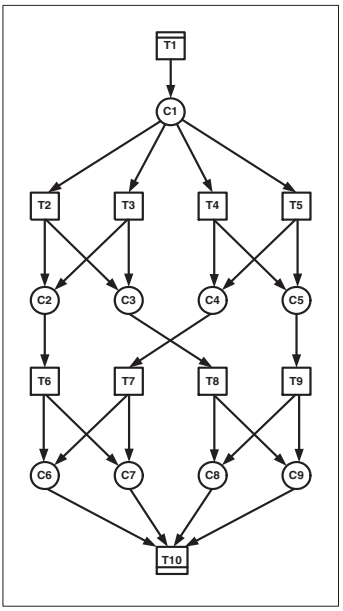

Fig. 5. Counter Example

Figure 4 visualizes the reduction rules proposed by [19]. The upper row shows workflow graph constructs before the application of a particular reduction rule. The lower row displays the results while the columns separate the different rules.

Although [19] claims otherwise, these rules are not complete. Figure 5 shows a correct workflow graph that cannot be reduced by the rules. This incompleteness was already signaled in [14], where another counter example is presented. [14] introduces three additional rules for replacing the overlapped rule, and claims that using the six remaining rules (i) the set of reduction rules is complete, and (ii) the complexity is $O(n^2.m^2)$, where $n = |N| + |F|$ and $m = |N|$. Whereas the counter example shown in [14] only needs two of the three replacement rules

(leaving the third rule a bit as a surprise), our counter example needs all three replacement rules to reduce the workflow graph to the empty graph.

4 Workflow Nets

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 4 (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.

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. 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: $p_1 + 2p_2 + p_3$. A Petri net PN and its initial marking M are denoted by (PN, M)

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.

The firing rule specifies how a Petri net can move from one state to the next one. If at any time multiple transitions are enabled, a non-deterministic choice is made. A firing sequence $\sigma = t_1 t_2 \ldots t_n$ is enabled if, starting from the initial marking, it is possible to subsequently fire $t_1, t_2, \ldots t_n$. A marking M is reachable from the initial marking if there exists a enabled firing sequence resulting in M. Using these notions we define some standard properties for Petri nets.

Definition 5 (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 6 (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.*

Definition 7 (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.*

Free-choice nets from an important subclass of Petri nets for which strong theoretical results exist. In a free-choice net choice and synchronization are separated.

Definition 8 (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 $\bullet t_1 = \bullet t_2$.*

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 9 (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 workflow management system and is deleted once it is completely handled by the workflow management system, i.e., the WF-net specifies the life-cycle of a case. The third requirement in Definition 9 has been added to avoid 'dangling tasks and/or conditions', i.e., tasks and conditions which do not contribute to the processing of cases.

The three requirements stated in Definition 9 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 10 (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.

(ii) State o is the only state reachable from state i with at least one token in place o.
(iii) There are no dead transitions in (PN, i).

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 10 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 [2,11] for more details. The second requirement states that the moment a token is put in place o, all the other places should be empty. The third requirement rules out dead parts.

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. In [1] 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. Note that $\overline{PN}$ is strongly connected.

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

Proof. See [1]. □

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

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 [5].)

Free-choice Petri nets have been studied extensively (cf. Best [3], Desel and Esparza [6], Hack [8]) because they seem to be a good compromise between expressive power and analyzability (cf. Definition 8). 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 [6]) enables us to formulate the following corollary.

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 [6]). By Theorem 1, this corresponds to soundness. □

Corollary 1 shows that, for free-choice nets, there are efficient algorithms to decide soundness.

5 Mapping Workflow Graphs onto WF-Nets

In this section we introduce an approach that maps workflow graphs onto WF-nets. This way Petri-net-based analysis techniques can be used to verify workflow graphs. Figure 6 visualizes the algorithm for mapping workflow graphs to Petri nets. Tasks are mapped onto transitions and choice/merge coordinators are mapped onto places. In row *a*) of Figure 6 the easiest case of mapping a workflow net to a Petri net can be seen. Whenever a task is directly followed by a choice/merge coordinator then no mapping adjustments are required. In Row *b*) a place has to be put between two directly connected tasks. It is marked with *p* and the task labels in brackets.

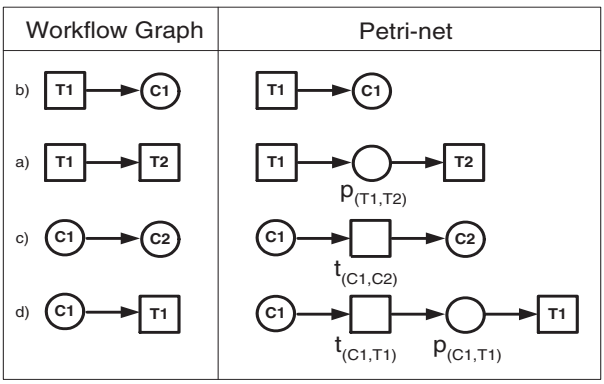

Fig. 6. Mapping workflow graphs to Petri nets

If two choice/merge coordinators are connected to each other as in row *c*) of Figure 6 then a transition must be put between the corresponding places. The place labels refer to the names of both coordinators. Row *d*) of the workflow graph column shows a coordinator connected to a task. In order to achieve Petri net mapping, an additional transition and place have to be added. Since the choice is made in the coordinator, a silent transition needs to be introduced. The following definition formatices the mapping of workflow graphs onto Petri nets.

Definition 11 (Petrify). *Let* $WG = (N, T, T^S, T^E, C, F)$ *be a consistent workflow graph with a unique source and sink node. The function petrify maps a workflow graph onto a Petri net* $PN = (P', T', F')$ *where:*

- $P' = C \cup \{i, o\} \cup \{p_{(x,y)} \quad Fy\ wedge \quad \in T\}$,
- $T' = T \cup \{t_{(x,y)} \quad Fy\ wedge \quad \in C\}$,
- $F' = \{(i,t) \quad \in T^S\}\ cup\ \{(t,o) \quad \in T^E\}\ cup\ \{(t,c) \quad Fc\ wedge \quad \in T\ wedge \quad \in C\}\ cup$
  $\bigcup\{\{(c, t_{(c,t)}), (t_{(c,t)}, p_{(c,t)}), (p_{(c,t)}, t)\} \quad Ft\ wedge \quad \in C\ wedge \quad \in T\}\ cup$
  $\bigcup\{\{(c, t_{(c,c')}), (t_{(c,c')}, c')\} \quad Fc'\ wedge \quad \in C\ wedge\ ' \in C\}\ cup$
  $\bigcup\{\{(t, p_{(t,t')}), (p_{(t,t')}, t')\} \quad Ft'\ wedge \quad \in T\ wedge\ ' \in T\}$

Function *petrify* results in a Petri net satisfying a number of properties as mentioned by the following theorem.

Theorem 2. *Let WG be a consistent workflow graph and petrify*$(WG) = PN$.

- *PN is a WF-net,*
- *PN is free-choice,*
- *PN is sound if and only if WG has no structural conflicts.*

Proof. It is easy to show that *PN* is a WF-net. There is one source place i and one sink place o. These places are explicitly added by the function *petrify*. Moreover, every node is on a path from i to o since in the corresponding workflow graph all nodes are on a path from start to end and all connections are preserved by the mapping given in Definition 11.

To show that *PN* is free-choice, we consider all places with multiple output arcs. These places all correspond to choice/merge coordinators. All additional places added by function *petrify* have only output arc (except o which has none). All outgoing arcs of a choice/merge coordinators are mapped onto a transition with only one input place. Therefore, *PN* is free-choice.

Consider definitions 3 and 10. Clearly, the two requirements stated in Definition 3 correspond to the first two requirements of Definition 10. Remains to prove that the absence of structural conflicts in *WG* implies that there are no dead transitions in *PN*. This is a direct result of Proposition 13 in [1] which demonstrates that for free-choice nets the first two requirements imply the third one. □

Corollary 2. *The following problem can be solved in polynomial time. Given a consistent workflow graph, to decide if it is correct.*

Proof. For free-choice WF-nets, soundness can be checked in polynomial time (Corollary 1). The mapping given in Definition 11 can also be done in polynomial time. Therefore, correctness of a consistent workflow graph can be verified in polynomial time. □

The complexity of the algorithm presented by Sadiq and Orlowska is $O(n^2)$ where $n = |N| + |F|$ [20]. However, this algorithm does not reduce all workflow graphs without structural conflicts as indicated in Section 3. Lin, Zhao, Li, and Chen [14] claim to have solved this problem. The complexity of the algorithm presented in [14] is $O(n^2.m^2)$ where $n = |N| + |F|$ and $m = |N|$.

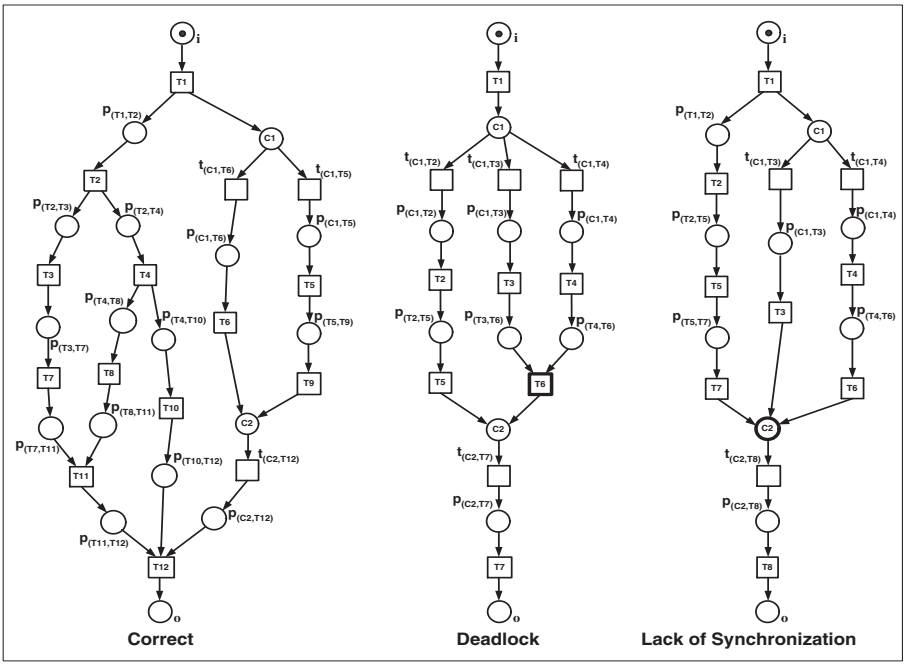

Fig. 7. Three WF-nets corresponding to the three workflow graphs shown in Figure 3

In [10] an algorithm is given which decides whether a strongly-connected free-choice net is live and bounded. The complexity of this algorithm is $O(p^2.t)$ where p is the number of places and t the number of transitions. This algorithm is an improvement of the approach based on the Rank theorem and uses state-machine decomposability by computing a sufficient set of minimal deadlocks to cover the net. This result can be used to analyze workflow graphs efficiently as indicated by the following theorem.

Theorem 3. *Let $WG = (N, T, T^S, T^E, C, F)$ be a consistent workflow graph. An upper bound for the complexity of checking whether WG has no structural conflicts through the construction of the corresponding WF-net and verifying whether the short-circuited net is live and bounded is $O(k^2.l)$ where $k = |C| + |F|$ and $l = |T| + |F|$.*

Proof. petrify$(WG) = PN = (P', T', F')$. The number of places in the short-circuited net is $|P'| < |C| + 2 + |F|$. The number of transitions is $|T'| < |T| + |F| + 1$. The complexity of the algorithm presented in [10] is $O(p^2.t)$ where p is the number of places and t the number of transitions. Hence, deciding whether PN is sound has a complexity of $O(k^2.l)$ where $k = |C| + |F|$ and $l = |T| + |F|$.

The complexity of transforming *WG* into *PN* is smaller. Therefore, the overall complexity is $O(k^2.l)$. □

This result shows that our approach is at least as good as the algorithm presented in [14]. The complexity of the algorithm presented in [14] is $O(n^2.m^2)$ where $n = |N| + |F|$ and $m = |N|$. If we assume that the number of arcs in a workflow graph (i.e., $|F|$) is of the same order of magnitude as the number of nodes (i.e., $|N|$), then the complexity of the algorithm presented in [14] is $O(n^2.m^2) = O(x^4)$ and the complexity of the algorithm presented in this paper is $O(k^2.l) = O(x^3)$ where $x = |N|$. If we assume that the number of arcs is quadratic in terms of the number of nodes, then the complexity of the algorithm presented in [14] is $O(n^2.m^2) = O(x^6)$ and the complexity of the algorithm presented in this paper is $O(k^2.l) = O(x^6)$ where $x = |N|$. This means that only in a worst-case scenario where the graph is dense, the complexities of both algorithms are comparable. If the graph is not dense, the complexity of our algorithm is significantly better.

6 Diagnostics and Petri-Net-Based Reduction Rules

Theorem 3 shows that Petri-net can be used to analyze workflow graphs efficiently. However, one of the features of the reduction rules given in [14,19,20] is the fact that useful error diagnostics are given in the form of an irreducible graph. In this section, we briefly discuss the diagnostics provided by our Petri-net-based verification tool Woflan. Moreover, we also provide pointers to Petri-net-based reduction rules. These reduction rules are more powerful than the rules given in [14,19,20].

Woflan (WOrkFLow ANalyzer) has been designed to verify process definitions which are downloaded from a workflow management system [21]. At the moment there are several workflow tools that can interface with Woflan, among which Staffware (Staffware plc., Berkshire, UK) and COSA (COSA Solutions/ Software-Ley, Pullheim, Germany) are the most prominent ones. The BPR tool Protos (Pallas Athena, Plasmolen, The Netherlands) can also interface with Woflan. If the workflow process definition is not sound, Woflan guides the user in finding and correcting the error. Since a detailed description of the functionality of Woflan is beyond the scope of this paper, we will use the example WF-nets shown in Figure 7 and the WF-net shown in Figure 8, which corresponds to the counter example shown in Figure 5, to illustrate the features of Woflan. For the *Deadlock* WF-net, Woflan gives the following diagnostics:

- The net is a WF-net, but is not coverable by so called S-components [6]. Because we know that the WF-net is (by construction) free-choice, we deduce (see [21]) that the WF-net is not sound, and thus that the corresponding workflow graph (see Figure 3) is not correct.
- Woflan points out the fact that a PT-handle exists in the WF-net: Starting from place *C1* there exist two mutual disjoint paths to transition *T6*. This clearly indicates the source of the error.

The *Lack of Synchronization* WF-net is diagnosed by Woflan as follows:

– This net is also a WF-net, and like the *Deadlock* WF-net, it cannot be covered by S-component. Hence, this WF-net is also not sound.
– In this net, a TP-handle exists: Starting from transition *T1* there exists mutual disjoint paths to the place *C2*. Once more, this clearly indicates the source of the error.

Finally, both the correct WF-net shown in Figure 7 and the WF-net shown in Figure 8 are diagnosed as follows:

– These nets are WF-nets, *and* they can be covered by S-components. As a result, no unbounded places exist and these WF-nets can still be sound.
– All transitions are live, hence the WF-nets are sound.

Note that the WF-net shown in Figure 8 corresponds to the workflow graph shown in Figure 5, i.e., the counter example. This graph can not be reduced by the technique presented in [19,20]. However, it can be analyzed by Woflan.

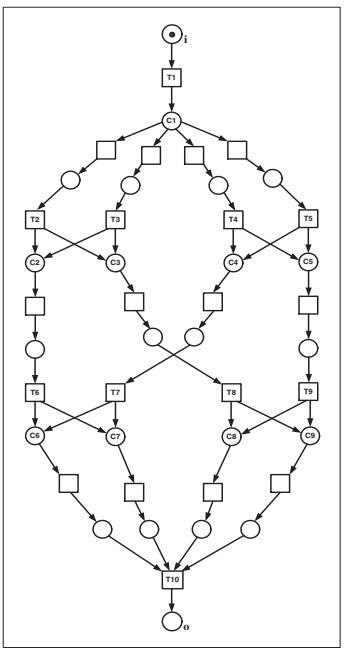

Fig. 8. Counter example mapped to workflow net

Woflan also supports a set of reduction rules. Before the analysis of soundness starts, the reduction rules presented in [15] can be used to reduce the size of the WF-net. These rules have be added to improve the analysis of large models. Note that the set of rules described in [15] is not complete. Therefore,

there are sound WF-nets that cannot be reduced to the "empty net". However, for live and bounded free-choice net there is a complete set of reductions rules $\{\phi_A, \phi_S, \phi_T\}$, cf. citedeselesparza. Rule ϕ_A is an abstraction rule which replaces place/transition pairs by arcs. Rules ϕ_S and ϕ_T are linear dependency rules which remove redundant places respectively transitions. As is shown in [6] these rules can be used to reduce any live and bounded free-choice net into a net consisting of one place and one transition. This means that the short-circuited Petri-net representation of any correct workflow graph can be reduced into a net $PN = (\{p\}, \{t\}, \{\langle p, t\rangle, \langle t, p\rangle\})$ in polynomial time. If the workflow graph is not correct, the reduction will stop before reaching the net consisting of one place and one transition. This will provide similar diagnostics as in [14,19,20]. However, (1) only three reduction rules are needed (instead of seven), (2) the reduction applies to a larger class of workflow processes (e.g., having loops), and (3) the rules are more compact and their correctness can be verified using standard Petri-net theory. Currently we are investigating if we can map the seven rules of [14] onto $\{\phi_A, \phi_S, \phi_T\}$.

7 Conclusion

In this paper, we presented an alternative analysis technique for the verification of workflow graphs as introduced by Wasim Sadiq and Maria Orlowska [19,20]. We presented a counter example showing that the reduction rules given in [19,20] cannot be applied. Moreover, we provided an alternative approach for identifying structural conflicts with an algorithm who's complexity is $O(k^2.l)$ where $k = |C| + |F|$ and $l = |T| + |F|$. This algorithm outperforms the algorithm presented in [14] if the workflow graph is not dense. This is remarkable since the techniques presented in [19,20,14] are tailored towards workflow graphs while our approach is based on standard Petri-net-based techniques.

Within a complexity range that is at least as good as the approach presented in [14], the algorithm in this paper can handle workflow graphs with cycles and more advanced synchronization constructs (as long as they correspond to free-choice nets). The mapping presented in this paper, allows for the verification using our analysis tool Woflan. Woflan provides high-quality diagnostics in case of an error and allows for a smooth transition to more expressive models, e.g., workflow languages having non-free choice constructs.

References

1. 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. 536, 544, 546
2. 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. 535, 536, 544
3. 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. 544

4. F. Casati, F. S. Ceri B. Pernici, and G. Pozzi. Conceptual Modeling of Workflows. In M.P. Papazoglou, editor, *Proceedings of the 14th International Object-Oriented and Entity-Relationship Modeling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 341–354. Springer-Verlag, Berlin, 1998. 535
5. 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. 544
6. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995. 544, 545, 548, 550
7. 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. 535, 536
8. M. H. T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972. 544
9. A. H. M. ter Hofstede, M. E. Orlowska, and J. Rajapakse. Verification Problems in Conceptual Workflow Specifications. *Data and Knowledge Engineering*, 24(3):239–256, 1998. 535, 536
10. P. Kemper. Linear Time Algorithm to Find a Minimal Deadlock in a Strongly Connected Free-Choice Net. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 319–338. Springer-Verlag, Berlin, 1993. 547
11. E. Kindler and W. M. P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 70(6):269–274, June 1999. 544
12. D. Kuo, M. Lawley, C. Liu, and M. E. Orlowska. A General Model for Nested Transactional Workflows. In *Proceedings of the International Workshop on Advanced Transaction Models and Architecture (ATMA '96)*, pages 18–35, Bombay, India, 1996. 535
13. P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997. 535
14. H. Lin, Z. Zhao, H. Li, and Z. Chen. A Novel Graph Reduction Algorithm to Identify Structural Conflicts. In *Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-35)*. IEEE Computer Society Press, 2002. 536, 540, 541, 546, 548, 550
15. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989. 549
16. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998. 535
17. 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. 539
18. W. Sadiq and M. E. Orlowska. On Correctness Issues in Conceptual Modeling of Workflows. In *Proceedings of the 5th European Conference on Information Systems (ECIS '97)*, pages 19–21, Cork, Ireland, 1997. 535, 536

19. W. Sadiq and M. E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In M. Jarke and A. Oberweis, editors, *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99)*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, Berlin, 1999. 535, 536, 540, 541, 548, 549, 550
20. W. Sadiq and M. E. Orlowska. Analyzing Process Models using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000. 535, 536, 540, 546, 548, 549, 550
21. H. M. W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001. 535, 536, 548