Diagnosing Workflow Processes using Woflan

H.M.W. VERBEEK[1], T. BASTEN[2] AND W.M.P. VAN DER AALST[1]

[1] *Faculty of Technology Management, Eindhoven University of Technology, the Netherlands*
[2] *Faculty of Electrical Engineering, Eindhoven University of Technology, the Netherlands*
Email: h.m.w.verbeek@tue.nl

Workflow management technology promises a flexible solution for business-process support facilitating the easy creation of new business processes and modification of existing processes. Unfortunately, today's workflow products have no support for workflow verification. Errors made at design-time are not detected and result in very costly failures at run-time. This paper presents the verification tool Woflan. Woflan analyzes workflow process definitions downloaded from commercial workflow products using state-of-the-art Petri-net-based analysis techniques. This paper describes the functionality of Woflan emphasizing diagnostics to locate the source of a design error. Woflan is evaluated via two case studies, one involving twenty groups of students designing a complex workflow process and one involving an industrial workflow process designed by Staffware Benelux. The results are encouraging and show that Woflan guides the user in finding and correcting errors in the design of workflows.

Keywords: Workflow management, Petri nets, Verification, Woflan, Case study.

1. INTRODUCTION

Workflow management systems take care of the automated support and coordination of business processes to reduce costs and flow times and to increase quality of service and productivity [23, 26, 28, 29, 38]. A critical challenge for workflow management systems is their ability to respond effectively to changes in business processes [6, 7, 13, 27, 32, 48]. Changes may range from simple modifications of a workflow process such as adding a task to a complete restructuring of the workflow process to improve efficiency. Changes may also involve the creation of new processes. Today's workflow management systems are ill suited to dealing with frequent changes, because there are hardly any checks to assure some minimal level of correctness. Creating or modifying a complex process that combines parallel and conditional routing is an activity subject to errors. Even a simple change as adding a task can cause a deadlock or livelock. A deadlock occurs if at some *unexpected* point in the workflow process it is no longer possible to make any progress for a certain case (workflow instance) that is being handled. Note that the *expected* termination of progress is something desirable, because it corresponds to the successful completion of a case. A livelock occurs if it is possible to make continuous progress for a certain case, however, without progressing towards successful completion and without ending in a deadlock (e.g., an endless loop). Contemporary workflow management systems do not support advanced techniques to verify the correctness of workflow process definitions [2, 24]. These systems typically restrict themselves to a number of (trivial) syntactical checks. Therefore, serious errors such as deadlocks and livelocks may remain undetected. This means that an erroneous workflow may go into production, thus causing dramatic problems for the organization. An erroneous workflow may lead to extra work, legal problems, dissatisfied customers, managerial problems, and depressed employees. Therefore, it is important to verify the correctness of a workflow process definition *before* it becomes operational. The role of verification becomes even more important as many enterprises are making Total Quality Management (TQM) one of their focal points. For example, an ISO 9000 certification and compliance forces companies to document business processes and to meet self-imposed quality goals [25]. Clearly, rigorous verification of workflow processes can be used as a tool to ensure certain levels of quality.

The development of *Woflan*[1] started at the end of 1996. The goal was to build a verification tool specifically designed for workflow analysis. Right from the start, there have been three important requirements for Woflan:

1. Woflan should be *product independent*, i.e., it should be possible to analyze processes designed with various workflow products of different vendors.
2. Woflan should be able to handle *complex workflows* with up to hundreds of tasks.
3. Woflan should give to-the-point *diagnostic information* for repairing detected errors.

Based on these requirements, we decided to base Woflan on Petri nets. Petri nets are a universal modeling language

[1] see http://www.tm.tue.nl/it/woflan

with a solid mathematical foundation. Yet, Petri nets are close to the diagramming techniques used in today's workflow management systems. The efficient analysis techniques developed for Petri nets allow for the analysis of complex workflows. The graphical representation of Petri nets and the available analysis techniques are particularly useful for generating meaningful diagnostic information. Since the release of version 1.0 of the tool in 1997, we have been continuously improving Woflan. Both new theoretical results and practical experiences stimulated several enhancements. Pivotal to Woflan is the notion of **soundness** of a workflow process [1, 2, 4]. This notion expresses the minimal requirements any workflow should satisfy. Informally, a workflow process is sound if it satisfies the following conditions.

(option to complete) It should always be possible to complete a case (workflow instance) that is handled according to the process. This condition guarantees the absence of deadlocks and livelocks.

(proper completion) It should not be possible that the workflow process signals completion of a case while there is still work in progress for that case.

(no dead tasks) For every task, there should be an execution of the workflow process that executes it. This restriction means that every task has a meaningful role in the workflow process.

The current version 2.1 of Woflan can analyze workflows designed with the workflow products *COSA*, *Staffware*, *METEOR*, and *Protos*. COSA (COSA Solutions/Software Ley, [42]) is one of the leading workflow management systems on the Dutch workflow market. COSA allows for the modeling and enactment of complex workflow processes which use advanced routing constructs. The modeling language of COSA is based on Petri nets. However, COSA does not support verification. Woflan can analyze any workflow process definition constructed by using CONE (COSA Network Editor), the design tool of the COSA system. Woflan can also import workflow process definitions from Staffware (Staffware Plc, [43]). Staffware is one of the most widespread workflow management systems in the world. It uses a proprietary graphical input language for defining workflow process definitions. Nevertheless, Woflan can analyze some useful properties of workflow process definitions made with Staffware. Woflan can also be used to analyze process definitions made with METEOR and Protos. METEOR (LSDIS, [40]) is a workflow management system based on CORBA and supports transactional workflows ([22]). Protos (Pallas Athena, [31]) is a Business-Process-Reengineering tool which can be used to (re)design and document workflow processes.

This paper focuses on version 2.1 of Woflan and, in particular, on the diagnosis process that it supports. This process has been developed based on experiences with earlier versions of Woflan. It implements several well-known Petri-net analysis techniques that are relevant in the context of workflow management. However, it also implements a new technique; Woflan can generate so-called *behavioral error sequences*. One can think of such a behavioral error sequence as a doomsday scenario: It gives a minimal sequence of tasks whose execution unavoidably leads to an error. Thus, it clearly shows the roots of the error in a workflow. These sequences can be used for diagnosing errors that are not easy to detect with standard analysis techniques available in earlier versions of Woflan. The functionality of Woflan 2.1 has been evaluated via two case studies. The first case study uses workflow process definitions developed by students of the course *Workflow Management & Groupware* (1R420), attended by 42 students of the Eindhoven University of Technology, and the course *Workflow Management: Models, Methods, and Tools* (25756), attended by 15 students of the University of Karlsruhe. These students formed 20 groups which independently designed the workflow in a travel agency consisting of about 60 tasks and other building blocks. These workflows were designed with Protos. We collected the workflows and analyzed them with Woflan 2.1. Most of the designed workflows contained several errors that were repaired using the diagnostics provided by Woflan. This case study proved to be very useful for testing the diagnosis process of Woflan. The second case study involves the analysis of an industrial workflow process definition developed by Staffware Benelux and containing more than 100 tasks and other building blocks. In the experiment, a workflow designer of Staffware Benelux introduced several (non-trivial) errors in a version of the workflow that was known to be correct. We analyzed the resulting process definition in Woflan. The exact number of errors and the type of errors were not known to us. We succeeded in finding six out of seven errors in the workflow process definition; also, the corrections we made based on the diagnostics of Woflan turned out to be the appropriate ones. This second case study complements the first one; it strengthens our belief that our approach of workflow-product-independent verification support is feasible.

The remainder of this paper is organized as follows. Section 2 introduces a class of Petri nets called P/T nets and summarizes some well-known results and analysis techniques. Section 3 introduces the area of workflow management and our approach to verification of workflows. In Section 4, we present a subclass of P/T nets for modeling workflows called WF nets and we formalize the soundness property on these WF nets. The section also introduces some specific techniques for analyzing WF nets, including the above mentioned technique of behavioral error sequences. Together with the standard analysis techniques of Section 2, these techniques form the (mathematical) basis for Woflan. The inclusion of the material in Sections 2 and 4 makes the paper self-contained and it allows the interested reader to study the Petri-net foundation of Woflan. Section 5 discusses the tool Woflan and the diagnosis process that it supports to decide whether or not a WF net satisfies the soundness property. The two case studies used for evaluating Woflan are presented in Section 6. Section 7 discusses related work. Finally, Section 8 presents conclusions and topics for future work.

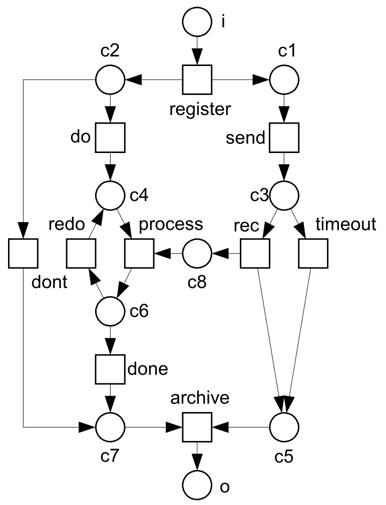

FIGURE 1. The example P/T net N

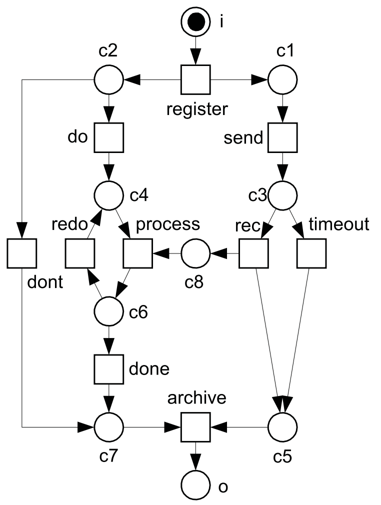

FIGURE 2. An example system S for net N

2. P/T NETS

Woflan is based on Petri nets. As indicated in the introduction, there are several reasons for using Petri nets for the verification of workflow process definitions. The interested reader is referred to [2, 10, 18] for a more elaborate discussion on the use of Petri nets in the workflow domain. In this section, we introduce a standard class of Petri nets called P/T nets. First, we introduce some basic definitions and useful properties. Second, we introduce some analysis techniques on P/T nets. Readers familiar with Petri nets can browse through this section to become familiar with the notations used. An extensive treatment of Petri nets can be found in [16, 33, 34, 35]. In this section, we restrict ourselves to the material that is needed to understand the foundations of Woflan.

2.1. Basic definitions

2.1.1. P/T nets
A P/T net is a directed graph with two kinds of nodes: *transitions* and *places*. Arcs in the graph always connect a node of one kind to a node of the other kind.

DEFINITION 2.1. (*P/T net*) The triple $N = (P, T, F)$ is a P/T net iff:

(i). P is a finite, non-empty set of places.
(ii). T is a finite, non-empty set of transitions such that $P \cap T = \emptyset$.
(iii). $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation.

It is common practice to draw places by circles and transitions by squares. An example of a P/T net can be seen in Figure 1. A P/T net models the *structure* of a process. The class of Petri nets introduced in Definition 2.1 is sometimes referred to as the class of *ordinary* P/T nets to distinguish it from the class of Petri nets that allows more than one arc between a pair of nodes. For the sake of simplicity, we allow in this paper at most one arc between any two nodes. However, most results extend in a relatively straightforward way to nets that may have multiple arcs between pairs of nodes.

DEFINITION 2.2. (*Preset, postset*) Let $N = (P, T, F)$ be a P/T net. For $n \in P \cup T$, the preset of n, $\bullet n$, equals $\{n_0 \in P \cup T | (n_0, n) \in F\}$; the postset of n, $n\bullet$, equals $\{n_0 \in P \cup T | (n, n_0) \in F\}$.

For a node (a place or a transition) n, its preset corresponds to the set of nodes (called *input nodes*) from which there is an arc (called an *input arc*) to n; its postset corresponds to the set of nodes (called *output nodes*) to which there is an arc (called an *output arc*) from n.

2.1.2. Systems
Places in a P/T net may contain so-called *tokens*. The distribution of tokens over the places determines the *state* of the P/T net, also called the *marking* of the P/T net. Graphically, tokens are typically represented by black dots. For example, if we add the marking consisting of a token in the place labeled `i` to our example P/T net N of Figure 1, we get the marked P/T net (or system) as shown in Figure 2. Since a place may contain multiple tokens, a marking can be represented as a bag or finite multi-set.

NOTATION 2.1. (*Bags*) A bag over some alphabet A is a function from A to the natural numbers that assigns only a finite number of elements from A a positive value. For a bag X over alphabet A and $a \in A$, $X(a)$ denotes the number of occurrences of a in X, often called the cardinality of a in X. Note that a finite *set* of elements from A is also a bag over A, namely the function yielding 1 for every element in the set and 0 otherwise. The set of all bags over A is denoted $B(A)$. We use brackets to explicitly enumerate a bag and superscripts to denote cardinalities. For example, $[a^2, b^3, c]$ is the bag with two a's, three b's, and one c; the bag $[a^2|P(a)]$, where P is a predicate on A, contains two elements a for every a such that $P(a)$ holds. The sum of two bags X and Y, denoted $X + Y$, is defined as $[a^n | a \in A \wedge n = X(a) + Y(a)]$. The difference of X and Y, denoted $X - Y$, is defined as $[a^n | a \in A \wedge n = (X(a) - Y(a)) \max 0]$. Bag X is a subbag of Y, denoted $X \leq Y$, iff, for all $a \in A$, $X(a) \leq Y(a)$.

DEFINITION 2.3. (*System*) A bag $M \in B(P)$ is called a marking of a P/T net (P, T, F). The pair $S = (N, M)$ is called a *system* with initial marking M.

2.1.3. *Behavior of systems*

Using a system, we can model a process structure as well as the current state of the process. However, we do not know yet how the process gets from one state to another. For this reason, we define the so-called firing rule.

DEFINITION 2.4. (*Firing rule*) Let $N = (P, T, F)$ be a P/T net. Marking M of N enables transition t in T iff $\bullet t \leq M$. Marking M_1 is reached from M by firing t, denoted $M \xrightarrow{t} M_1$, iff $\bullet t \leq M$ and $M_1 = M - \bullet t + t\bullet$.

So, a transition is *enabled* iff its preset is a subbag of the actual marking, implying that there is a token in every input place of the transition. Note that we use the fact that the preset is a set and hence a bag. When a transition is enabled, we can reach a new marking by *firing* this transition. This new marking can be constructed by removing the transition's preset from the original marking and adding the transition's postset. For example, in our system of Figure 2, only the `register` transition is enabled. When `register` fires, the new marking becomes [`c1`, `c2`]: The token from place `i` is removed and new tokens are added to places `c1` and `c2`.

2.2. Analysis of nets

Petri nets are known for the availability of many analysis techniques. Clearly, this is a great asset in favor of the use of Petri nets for workflow modeling. The analysis techniques can be used to prove qualitative properties (safety properties, invariance properties, deadlock, etc.) and to calculate performance measures (response times, waiting times, occupation rates, etc.). In this paper, the primary focus is on qualitative verification.

2.2.1. *Structural analysis*

A structural property of a P/T net is a property that does not depend on the marking of the net. Therefore, it can be defined on P/T nets rather than on systems. In process modeling, the simple combination of places and transitions can be used to devise various routing constructs ranging from a simple sequence to a delicate mixture of choice and synchronization. In the context of workflow design, certain, more advanced, constructs are considered to be suspicious and a potential source of errors. Therefore, we review the standard structural properties for P/T nets. A strong point of structural properties is that most of them can be computed efficiently.

As in all directed-graph structures, we can distinguish directed and undirected paths in P/T nets.

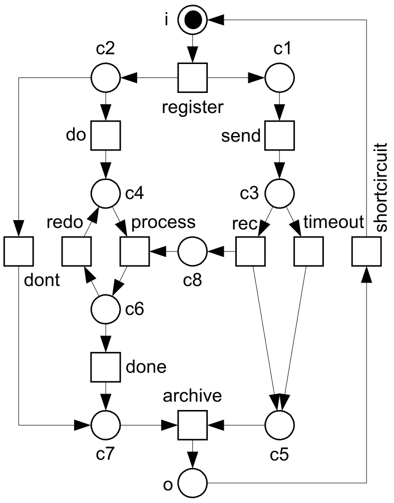

FIGURE 3. The short-circuited system $\underline{S} = (\underline{N}, [\texttt{i}])$

DEFINITION 2.5. (*(Strongly) connected P/T net*) A P/T net is called *connected* iff there exists a(n undirected) path between every two nodes. It is *strongly* connected iff there exists a directed path between every two nodes.

The P/T net `N` of Figure 1 is connected, but not strongly connected: For instance, there is no directed path from `o` to `i`. If we short-circuit net `N` of Figure 1 with the `shortcircuit` transition from `o` to `i`, we get a net that is strongly connected. Figure 3 shows the resulting net $\underline{\texttt{N}}$. (Actually, it shows a system based on $\underline{\texttt{N}}$ but, at this point, the marking is not relevant.)

A (directed or undirected) path is called *elementary* iff all nodes in the path are different.

DEFINITION 2.6. (*PT-handle, TP-handle [20]*) Let $N = (P, T, F)$ be a P/T net. A place-transition pair $(p, t) \in P \times T$ is called a PT-handle iff there exist two elementary directed paths from p to t sharing only the two nodes p and t; a transition-place pair $(t, p) \in T \times P$ is called a TP-handle iff there exist two elementary directed paths from t to p sharing only nodes p and t.

Since PT-handles and TP-handles can easily introduce design flaws in (workflow) process definitions (see Section 5.1.4), we name nets without these potentially correctness-threatening constructs well-handled.

DEFINITION 2.7. (*Well-handled P/T net*) A P/T net is *well-handled* iff it has no PT-handles and no TP-handles.

Net `N` of Figure 1 is not well-handled: It contains one PT-handle (see Figure 4) and two TP-handles (see Figure 5).

A P/T net is called *free-choice* iff every two transitions sharing at least one input place have identical presets. Net `N` of Figure 1 is free-choice.

DEFINITION 2.8. (*Free-choice P/T net*) A P/T net (P, T, F) is free-choice iff $\forall t_0, t_1 \in T : \bullet t_0 \cap \bullet t_1 = \emptyset \vee \bullet t_0 = \bullet t_1$.

A net is called a state machine iff all transitions have exactly

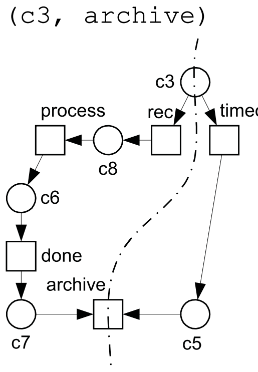

FIGURE 4. The only PT-handle in net N

one input and one output place.

DEFINITION 2.9. (*State machine*) A P/T net (P, T, F) is a state machine iff $\forall t \in T : |\bullet t| = |t\bullet| = 1$.

DEFINITION 2.10. (*Subnet*) Let $N = (P, T, F)$ and $N_0 = (P_0, T_0, F_0)$ be P/T nets. Net N_0 is a subnet of net N iff $P_0 \subseteq P$, $T_0 \subseteq T$, and $F_0 = F \cap ((P_0 \times T_0) \cup (T_0 \times P_0))$.

DEFINITION 2.11. (*S-component*) Let $N = (P, T, F)$ be a P/T net and $N_0 = (P_0, T_0, F_0)$ a subnet of N; let $\bullet$ denote the preset and postset functions of N. Subnet N_0 is an S-component of N iff N_0 is a strongly connected state machine such that $\forall p \in P_0 : \bullet p \cup p\bullet \subseteq T_0$.

If a P/T net corresponds to a set of S-components, it is S-coverable. Net N of Figure 1 has no S-components. P/T net $\underline{\text{N}}$ of Figure 3 has two S-components (see Figure 6) but is not S-coverable: Place c8 is not contained in any of these S-components.

DEFINITION 2.12. (*S-coverability*) A P/T net (P, T, F) is S-coverable iff for each place $p \in P$ there is an S-component (P_0, T_0, F_0) of N such that $p \in P_0$.

A place-invariant is a weighted sum over the places that is invariant under each possible transition firing.

DEFINITION 2.13. (*Place-invariant*) Let $N = (P, T, F)$ be a P/T net and w a weight function from P to the integer numbers. Function w is a place-invariant of N iff $\forall t \in T$: $(\sum p \in \bullet t : w(p)) = (\sum p \in t\bullet : w(p))$.

Note that despite the fact that the above explanation of a place-invariant is in terms of transition firings, a place-invariant is a structural property: It is independent of the marking of the net. For example, a place-invariant of net N of Figure 1 is the function that assigns the weight 1 to the places i, c1, c3, c5, and o and 0 to the other places. A convenient way to represent this function is i+c1+c3+c5+o.

It is not difficult to see that if w_0 and w_1 are place-invariants, the elementwise sum $w_0 + w_1$ and the elementwise difference $w_0 - w_1$ are place-invariants too. As a result, a net has only the place-invariant containing only weights 0 or it has infinitely many place-invariants.

Exchanging the roles of places and transitions in the notion of a place-invariant yields the concept of a so-called transition-invariant. However, transition-invariants do not play a role in this paper.

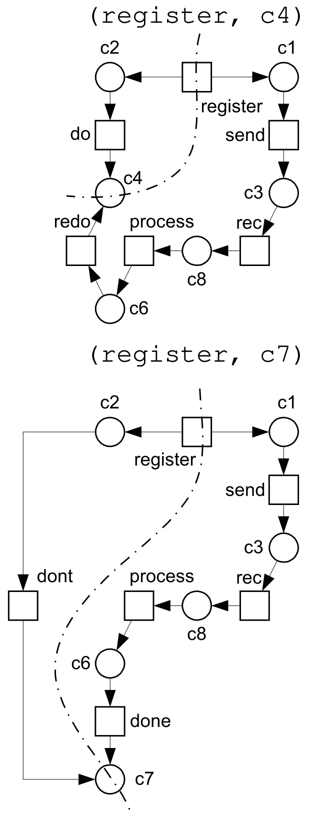

FIGURE 5. TP-handles in net N

2.2.2. *Occurrence sequences*

Behavioral analysis techniques are those techniques that use the initial marking of a P/T net. Therefore, these techniques use systems instead of P/T nets. An elementary behavioral technique is the analysis of the so-called *occurrence sequences* of a system. An occurrence sequence is simply a chain of transition firings.

DEFINITION 2.14. (*Occurrence sequence*) Let $S = (N, M_0)$ be a system, let $M_1, \ldots, M_n$, for some natural number n, be markings of $N = (P, T, F)$, and let $t_0, t_1, \ldots, t_{n-1}$ be transitions in T. Sequence $s = M_0 t_0 M_1 \ldots t_{n-1} M_n$ is an occurrence sequence of S iff $\forall i, 0 \leq i < n : M_i \xrightarrow{t_i} M_{i+1}$.

An occurrence sequence of a system projected onto transitions yields a so-called *firing sequence*.

Consider again P/T net N of Figure 1. Assuming initial marking [c4, c5, c8], the set of firing sequences equals {process, process redo, process done, process done archive}. Note that the sets of firing and occurrence sequences are prefix-closed, i.e., every prefix of a firing (occurrence) sequence is also a firing (occurrence) sequence.

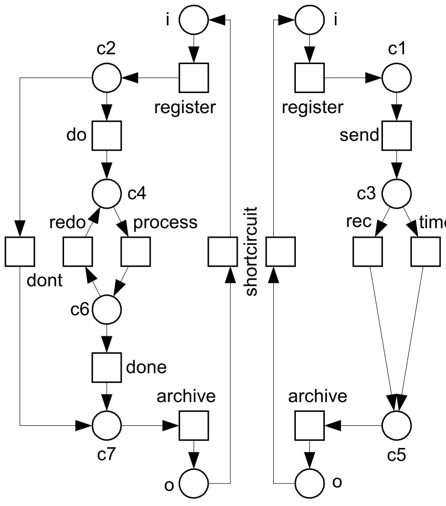

FIGURE 6. S-components of net $\underline{\mathtt{N}}$

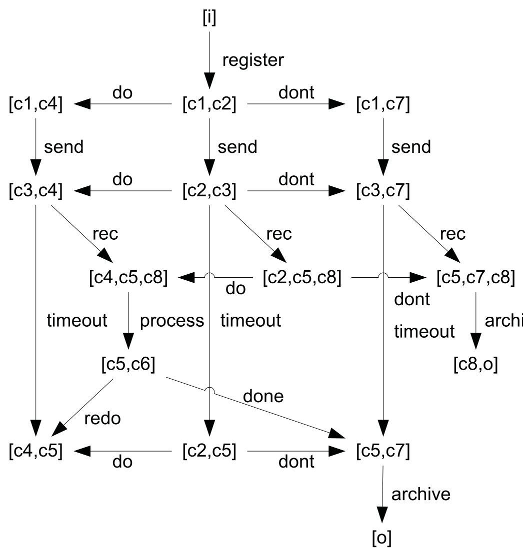

FIGURE 7. The OG of system S

2.2.3. Occurrence graph

The set of occurrence sequences of a system can be embedded into a graph. Every occurrence sequence corresponds to some path in that graph and vice versa.

NOTATION 2.2. (*Reachability*) Let $N = (P, T, F)$ be a P/T net. Marking M_1 is reachable from marking M_0, denoted $M_0 \longrightarrow M_1$, iff system (N, M_0) has an occurrence sequence ending in M_1.

In system S of Figure 2, marking [c4, c5, c8] is reachable from the initial marking [i], while from [c4, c5, c8] both [c4, c5] and [o] are reachable.

DEFINITION 2.15. (*Occurrence graph*) Let $S = ((P, T, F), M_0)$ be a system; let $H \subseteq B(P)$ be a set of markings, let $A \subseteq (H \times T \times H)$ be a set of T-labeled arcs, and let $G = (H, A)$ be a graph which satisfies the following requirements:

(i). $H = \{M \in B(P) | M_0 \longrightarrow M\}$;
(ii). $A = \{(M, t, M_1) \in (H \times T \times H) | M \xrightarrow{t} M_1\}$.

Graph G is called the occurrence (or reachability) graph (OG) of S.

The OG of system S of Figure 2 is given in Figure 7.

The OG embeds precisely all occurrence sequences of the system. The construction of this graph is straightforward, although termination is not guaranteed, because it might be infinite. For example, the OG of system $\underline{\mathtt{S}}$ of Figure 3 has infinitely many nodes. In this system, firing the transitions `register send rec dont archive shortcircuit` over and over again, leads to infinitely many markings $[\mathtt{i}, \mathtt{c8}^n]$, for arbitrary $n > 0$. After one firing of these transitions, there is one token in c8, after two firings there are two, and so on. There is no limit to the number of tokens in c8. Place c8 is said to be *unbounded*. As a result, the number of markings in the OG is infinite.

2.2.4. Coverability graph

A solution to cope with unbounded places is the notion of a so-called coverability graph. A coverability graph is a finite variant of an OG. However, we have to pay a price: First, we must allow markings to be infinite to deal with unbounded behavior. Second, a P/T system may have a number of possible coverability graphs, whereas it always has one unique OG.

An extended bag over some alphabet A is a function from A to the natural numbers plus ω (denoting infinity). The set of all extended bags over A is denoted $B^{\omega}(A)$. All operations on bags can be defined for extended bags in a straightforward way. An extended bag $M \in B^{\omega}(P)$ is called an *extended* marking of a P/T net (P, T, F). The set of extended markings can be partitioned into a set of *finite* markings $B(P)$ and a set of *infinite* markings $B^{\omega}(P) \setminus B(P)$.

A coverability graph of a system is a variant of the OG, where paths in the OG with infinitely many different (finite) markings are represented by a finite number of infinite markings. An infinite marking is introduced in a coverability graph if we encounter a marking M_1 on an occurrence sequence that has a smaller marking M_0 as one of its predecessors: The places in $M_1 - M_0$ are unbounded and are marked with ω. It is known that a coverability graph is always finite ([33], p. 70).

DEFINITION 2.16. (*Coverability graph*) Let $S = ((P, T, F), M_0)$ be a system, let $H \subseteq B^{\omega}(P)$ be a set of extended markings, let $A \subseteq (H \times T \times H)$ be a set of T-labeled arcs, and let $G = (H, A)$ be a graph which can be constructed as follows:

(i). Initially, $H = \{M_0\}$ and $A = \emptyset$.
(ii). Take an M from H and a t from T such that M enables t and such that no M_1 exists with $(M, t, M_1) \in A$. Let $M_2 = M - \bullet t + t \bullet$. Add M_3 to H and (M, t, M_3) to A, where for every $p \in P$:

(a) $M_3(p) = \omega$, if there is a node M_1 in H such that $M_1 \leq M_2$, $M_1(p) < M_2(p)$, and there is a (directed) path from M_1 to M in G;
(b) $M_3(p) = M_2(p)$, otherwise.

Repeat this step until no new arcs can be added.

G is called a coverability graph (CG) of S.

The result of this algorithm may vary depending on the order in which markings are considered in the second step (see [33] for more details). Nevertheless, a CG of a system can be used to analyze the behavior of the system. The short-circuited net $\underline{S}$ of Figure 3 has a unique CG which is shown in Figure 8.

Given a system and a CG of this system, every occurrence sequence of the system corresponds to a path in the CG. The converse is not necessarily true: There may be paths in the CG that do not correspond to any occurrence sequence. However, a path containing only finite markings does correspond to some occurrence sequence. This conforms to the fact that the CG is identical to the OG if the former has no infinite markings. The theoretical worst-case complexity of generating a CG is non-primitive recursive space, although for small to medium sized systems (up to 100 transitions) generating a CG is often feasible.

In [21], Finkel introduces the notion of a *minimal* CG (MCG) of a P/T system. An MCG of a system with infinite OG is usually much smaller than a CG of the system. Another advantage is that the MCG of a system is unique. However, the MCG of a system with finite OG may differ from that OG. It is beyond the scope of this paper to go into more detail.

2.2.5. *Behavioral properties*

Behavioral properties of a P/T net are those properties that depend on the marking of the net. Thus, these properties are defined on systems. In the remainder, we do not go into detail about the precise complexities of the algorithms to determine behavioral properties (see [19] for more information). For our purposes, it suffices to know that the theoretical complexity of computing behavioral properties is often much worse than the complexity of computing structural properties.

DEFINITION 2.17. (*Dead transition*) A transition $t \in T$ of a system $((P, T, F), M_0)$ is *dead* iff there is no marking reachable from M_0 enabling t.

A transition is live iff it can always fire again.

DEFINITION 2.18. (*Liveness*) A transition $t \in T$ of a system $S = ((P, T, F), M_0)$ is live iff $\forall M \in B(P), M_0 \longrightarrow M : \exists M_1 \in B(P), M \longrightarrow M_1 : M_1$ enables t. System S is live iff all transitions are live.

System `S` of Figure 2 is not live: For instance, no transition firings are possible in reachable marking [`o`] (see Figure 7). The short-circuited system $\underline{S}$ of Figure 3 is also not live: No transition firings are possible in reachable marking [`c4`, `c5`] (see Figure 8).

A system is *bounded* iff it has no unbounded places. An equivalent definition for boundedness is to require that the number of reachable markings, or the system's OG, is finite. A system is called safe iff all places in any reachable marking contain at most one token.

DEFINITION 2.19. (*Boundedness, safeness*) A system $((P, T, F), M_0)$ is bounded iff $\forall M \in B(P), M_0 \longrightarrow M : \forall M_1 \in B(P), M \longrightarrow M_1 : \neg(M < M_1)$. A system $((P, T, F), M_0)$ is safe iff $\forall M \in B(P), M_0 \longrightarrow M : \forall p \in P : M(p) \leq 1$.

Note that, for a bounded system, the CG-generation algorithm of Definition 2.16 yields the OG of the system.

System `S` of Figure 2 is bounded and safe. The latter is straightforward to see in its OG: In each marking, every place occurs at most once. However, the short-circuited system $\underline{S}$ of Figure 3 is unbounded, which follows directly from the fact that there are infinite markings in the CG of Figure 8.

3. WORKFLOW MANAGEMENT

In the last decade, *workflow management systems* have become a popular tool to support the logistics of business processes in banks, insurance companies, and governmental institutions [2, 23, 26, 28, 29, 38, 39]. Before, there were no generic tools to support workflow management. As a result, parts of the business process were hard-coded in the applications. For example, an application to support task X triggers another application to support task Y. This means that one application knows about the existence of another application. This is undesirable, because every time the underlying business process is changed, applications need to be modified. Moreover, similar constructs need to be implemented in several applications and it is not possible to monitor and control the entire workflow. Therefore, several software vendors recognized the need for workflow management systems. A workflow management system is a generic software tool that allows for the definition, execution, registration, and control of business processes or *workflows*. At the moment, many vendors are offering a workflow management system. This shows that the software industry recognizes the potential of workflow management tools.

As indicated in the introduction (see also [2, 10, 18]), P/T nets are a good starting point for a solid foundation of workflow management. We use P/T nets to specify the partial ordering of tasks in a workflow. Based on a P/T-net representation of the workflow process, we tackle the problem of verification.

3.1. Workflow processes

The fundamental property of a workflow process is that it is *case-based*. This means that every piece of work is executed for a specific *case*, also called a *workflow instance*. Examples of cases are an insurance claim, a tax declaration, a customer complaint, a mortgage, an order, or a request for information. Thus, handling an insurance claim, a tax declaration, or a customer complaint are typical examples of

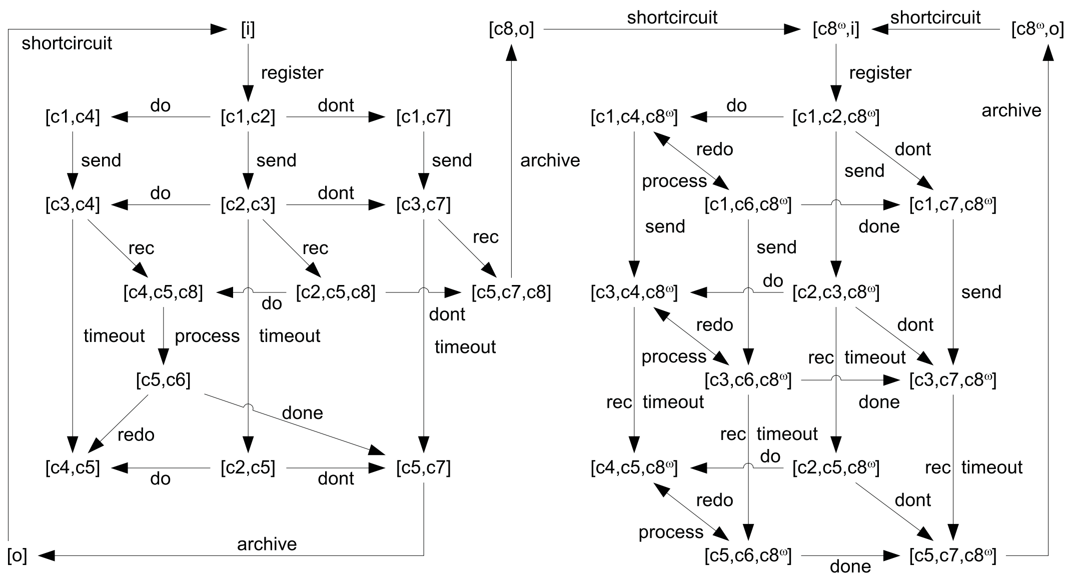

FIGURE 8. The CG for the short-circuited system $\underline{S}$

workflow processes. Cases are often generated by an external customer. However, it is also possible that a case is generated by another department within the same organization (internal customer). A typical example of a process that is not case-based, and hence not a workflow process, is a production process such as the assembly of bicycles. The task of putting a tire on a wheel is (generally) independent of the specific bicycle for which the wheel will be used. Note that the production of bicycles to order, i.e., procurement, production, and assembly are driven by individual orders, can be considered as a workflow process.

The goal of workflow management is to handle cases as efficient and effective as possible. A workflow process is designed to handle large numbers of similar cases. Handling one customer complaint usually does not differ much from handling another customer complaint. The basis of a workflow process is the *workflow process definition*. This process definition specifies which *tasks* need to be executed in what *order*. Alternative terms for workflow process definition are: ‘procedure’, ‘workflow schema’, ‘flow diagram’, and ‘routing definition’. Tasks are ordered by specifying for each task the *conditions* that need to be fulfilled before it may be executed. In addition, it is specified which conditions are fulfilled by executing a specific task. Thus, a partial ordering of tasks is obtained. In a workflow process definition, standard routing elements are used to describe sequential, alternative, parallel, and iterative routing thus specifying the appropriate route of a case. The workflow management coalition (WfMC) has standardized a few basic building blocks for constructing workflow process definitions [29]. A so-called *OR-split* is used to specify a choice between several alternatives; an *OR-join* specifies that several alternatives in the workflow process definition come together. An *AND-split* and an *AND-join* can be used to specify the beginning and the end of parallel branches in the workflow process definition. The routing decisions in OR-splits are often based on data such as the age of a customer, the department responsible, or the contents of a letter from the customer.

Many cases can be handled by following the same workflow process definition. As a result, the same task has to be executed for many cases. A task that needs to be executed for a specific case is called a *work item*. An example of a work item is the order to execute task ‘send refund form to customer’ for case ‘complaint of customer Baker’. Most work items need a *resource* in order to be executed. A resource is either a machine (e.g., a printer or a fax) or a person (participant, worker, or employee). Besides a resource, a work item often needs a *trigger*. A trigger specifies who or what initiates the execution of a work item. Often, the trigger for a work item is the resource that must execute the work item. Other common triggers are external triggers and time triggers. An example of an external trigger is an incoming phone call of a customer; an example of a time trigger is the expiration of a deadline. A work item that is being executed is called an *activity*. If we take a photograph of the state of a workflow, we see cases, work items, and activities. Work items link cases and tasks. Activities link cases, tasks, triggers, and resources.

A thorough investigation of the business processes in a company that results in a complete set of efficient and effective workflow processes is the basis of the successful introduction of a workflow system. Formal (qualitative and quantitative) verification can be a useful aid in obtaining the desired effectiveness and efficiency.

3.2. Workflow perspectives and abstraction

In the previous subsection, we introduced the workflow concepts used in the remainder of this paper. Workflow management has many aspects and typically involves many disciplines. The verification tool presented in this paper focuses on the control-flow perspective (i.e., workflow process definitions) and abstracts from other perspectives. This subsection motivates why it is reasonable to restrict the analysis focus to a single perspective. Therefore, we start by introducing the perspectives commonly identified in workflow literature [26].

3.2.1. *Perspectives*

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 ones given in [26].)

In the control-flow perspective, *workflow process definitions* are defined to specify which tasks need to be executed and in what order (i.e., the routing or control flow). The concepts relevant for this perspective (task, condition, and AND/OR-split/join) have been introduced in Section 3.1.

In the resource perspective, the organizational structure and the population are specified. Resources, ranging from humans to devices, form the organizational population and are mapped onto resource classes. In office environments, where workflow management systems are typically used, the resources are mainly human. However, because workflow management is not restricted to offices, we prefer the term resource. To facilitate the allocation of work items to resources, resources are grouped into classes. A *resource class* is a group of resources with similar characteristics. There may be many resources in the same class and a resource may be a member of multiple resource classes. If a resource class is based on the capabilities (i.e., functional requirements) of its members, it is called a *role*. If the classification is based on the structure of the organization, such a resource class is called an *organizational unit* (e.g., team, branch, or department). The resource classification describes the structure of the organization.

The data perspective deals with *control* and *production data*. Control data are data introduced solely for workflow management purposes. Control data are often used for routing decisions in OR-splits. Production data are information objects (e.g., documents, forms, and tables) whose existence does not depend on workflow management.

The task perspective describes the content of the process steps, i.e., it describes the characteristics of each task. A task is a logical unit of work with characteristics such as the set of operations that need to be performed, description, expected duration, due-date, priority, trigger (i.e., time, resource, or external trigger), and required resources classes (i.e., roles and organizational units).

In the operational perspective, the elementary actions are described. Note that one task may involve several operations. These operations are often executed using applications ranging from a text editor to custom-built applications for performing complex calculations. Typically, these applications create, read, or modify control and production data in the data perspective.

This paper addresses the problem of qualitative workflow verification. That is, we focus on properties of a logical nature (i.e., the soundness property introduced in Section 1) and not on performance issues (quantitative verification). For the purpose of qualitative verification, we only consider the control-flow perspective of a workflow. In the remainder of this subsection, we discuss a number of abstractions motivating why this simplification is reasonable.

3.2.2. *Abstraction from resources*

Detailed knowledge of the allocation of resources to work items, the duration of activities, and the timing characteristics of triggers are a crucial factor when analyzing the performance of a workflow. However, for qualitative verification, it is only relevant whether certain execution paths are possible or not. It is important to note that the allocation of resources can only restrict the routing of cases, i.e., it does not enable execution paths that are excluded in the control-flow perspective. Since resource allocation can only exclude execution paths, for qualitative verification, it suffices to focus on potential deadlocks resulting from the unavailability of resources. In the next few paragraphs, we argue that deadlocks resulting from restrictions imposed by resource allocation are generally absent, thus motivating why it is reasonable to abstract from resources.

A potential, resource-inflicted deadlock could arise (1) when multiple tasks try to allocate multiple resources at the same time, or (2) when there are tasks imposing such demanding constraints that no resource qualifies.

The first type of deadlock often occurs in flexible manufacturing systems where both space and tools are needed to complete operations thus potentially resulting in *locking* problems [41]. However, given today's workflow technology, such deadlocks cannot occur in a workflow management system: 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 contributing to the execution of one activity, e.g., writing a report for a given case, only one person is assigned to that activity from the perspective of the workflow management system: This is the person that selected the corresponding work item from the in-basket (i.e., the electronic worktray). Therefore, from the viewpoint of qualitative verification, it is reasonable to abstract from these locking problems. (Nevertheless, if in the future collaborative features are explicitly supported by workflow management systems, then these problems should be taken into account.)

The second type of deadlock occurs when there is no suitable resource to execute a task for a given case, e.g., there is not a single resource within a resource class. Generally, such problems can be avoided quite easily by checking whether all resource allocations yield non-empty sets of qualified resources. However, there may be some subtle errors resulting from case management (a subset of tasks for a given case is required to be executed by the same resource) and function separation (two tasks are not to be executed by the same resource to avoid security violations). For example, task 1 should be executed by the same person as task 2 and task 2 should be executed by the same person as task 3. However, task 3 should not be executed by the person who executed task 1. Clearly, there is no person qualified to execute task 3. Such problems highly depend on the workflow management system being used and are fairly independent of the routing structure. Therefore, in our approach of workflow-product-independent verification we abstract from this type of resource-driven deadlocks.

3.2.3. Abstraction from data and triggers

Recall that the data perspective deals with both control and production data. We abstract from production data because 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 workflow processes, e.g., the bill-of-material in manufacturing is shared by production, procurement, sales, and quality-control processes.

We partly abstract from control data. In contrast to production data, the control data used by the workflow management system for routing 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 when verifying a given workflow. The abstraction used in this paper is the following. Since control data 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. Moreover, we assume a fair behavior with respect to these choices and exclude conspiracies [12].

We also abstract from triggers, because a workflow management system cannot control the occurrence of triggers. As for choices, we only assume fairness with respect to the occurrence of triggers: An enabled task cannot be blocked forever (or infinitely often) because the corresponding trigger is never received.

The fairness assumptions on choices and triggers are reasonable: Without these assumptions any iteration or trigger would create a potential livelock or deadlock.

There are other reasons for abstracting from data and triggers. If we are able to prove soundness (i.e., the correctness criterion introduced in Section 1) for the process definition after abstraction, it will also hold for the situation where the routing of cases is based on control data or the occurrence of triggers (under the fairness assumptions mentioned before). If the logical correctness of the workflow depends on mutual dependencies between control data, the invariance of certain control data, or the occurrence of a specific trigger, it is not possible to prove soundness. However, one might argue that such a workflow is poorly designed. Last but not least, we abstract from data and triggers because it allows us to use classical 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.

3.2.4. Abstraction from task content and operations

As a final abstraction, we consider tasks to be atomic abstracting from the duration of tasks and 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 qualitative verification, it is reasonable to consider tasks as atomic entities.

Note that we do not explicitly consider transactional workflows [22]. There are several reasons for this. First of all, most workflow management systems (in particular the commercial ones) do not support transactional features in the workflow modeling language. Second, as is shown in [10], the various transactional dependencies can easily be modeled in terms of Petri nets. Therefore, we can straightforwardly extend the approach in this paper to transactional workflows.

3.3. Verification approach

In the previous subsection, it has been shown that for the purpose of qualitative verification it is reasonable to abstract from resources, data, triggers, the content of tasks, and operations and to focus on the control-flow perspective. In fact, it suffices to consider the control flow of one case in isolation. The only way cases interact directly, is via the competition for resources and the sharing of production data. (Note that control data are strictly separated.) Therefore, if we abstract from resources and production data, it suffices to consider one case in isolation. The competition between cases for resources is only relevant for performance analysis.

The principal goal of the approach presented in this paper is to verify the correctness of a workflow specified in *some* workflow management system, i.e., the approach is *not* tailored towards a *specific* workflow management system. Despite the efforts of the Workflow Management Coalition (WfMC, [29]), there is no consensus on the language for specifying workflows. The format proposed by the WfMC for exchanging workflow process definitions, i.e., Interface 1: Workflow Process Definition Language (WPDL), is only partially supported by the existing systems. (Typically, workflow management systems are unable to import and handle all constructs.) Moreover, WPDL has no formal semantics which means that it is impossible to reason about the correctness of a given workflow process definition. Therefore, we propose to directly translate a workflow process definition specified in some workflow management system to a Petri net, applying the abstraction discussed in Section 3.2. The resulting P/T net should of course be consistent with the (formal or informal) semantics of the workflow process definition as defined by the workflow management system being used.

The P/T net in Figure 1 models a typical workflow process, namely the processing of complaints. Assume that the initial marking is [i], thus obtaining the system of Figure 2. Marking [i] corresponds to the fact that a new complaint has been received. First, this complaint is registered (register). Task register is an example of an AND-split. Upon completion of this task, in parallel, a form is sent (send) to the complainant and the complaint is evaluated to determine whether it needs to be processed (do) or not (dont). The two transitions do and dont together form an OR-split. The two transitions model a single task in the real workflow which might be called something like ‘evaluate’. If the form that is sent to the complainant is received in time (rec), the complaint can be processed. If it is not received in time (timeout), the form cannot be used for the processing of the complaint. After the complaint has been processed (process), a check is made to determine whether it has been processed correctly (done) or not (redo) (another OR-split). If not, it needs to be processed again. Place c7 is an example of an OR-join: Two alternative process branches are joined. In the end, the complaint is archived (archive). Transition archive is an example of an AND-join.

We see that the P/T-net representation of a workflow process definition is straightforward: Tasks are represented by *transitions* and conditions by *places*. Two special places are added, one to indicate that a new case has been created, place i, and another to indicate that a case has been completed, place o. It is clear that standard building blocks such as the AND-split, AND-join, OR-split, and OR-join (see [29, 47]) can be modeled by P/T nets.

To illustrate the spectrum of languages used to specify workflow processes and their mapping onto P/T nets, we present two workflow process definitions (one using COSA and one using Staffware) corresponding to the P/T net shown in Figure 1.

Figure 9 shows the workflow process designed using CONE (COSA Network Editor). CONE is the design tool of the workflow management system COSA [42]. Since COSA is based on Petri nets, it is easy to see that the workflow specification corresponds to the P/T net shown in Figure 1. Note that the transitions do and dont in Figure 1 correspond to one task called evaluate in Figure 9, as explained above. This task is an OR-split which sets a variable named do. Based on this variable, either the arc from evaluate to c4 is activated or the arc from evaluate to c7 is activated. The arc conditions shown in Figure 9 are evaluated at run-time and determine whether a token is produced for c4 or c7. Similar remarks hold for the task named check. By using a set of simple translation rules, any workflow process definition designed using COSA can be translated to a P/T net. Note that during the translation one abstracts from data, i.e., the four arc conditions shown in Figure 9 are translated

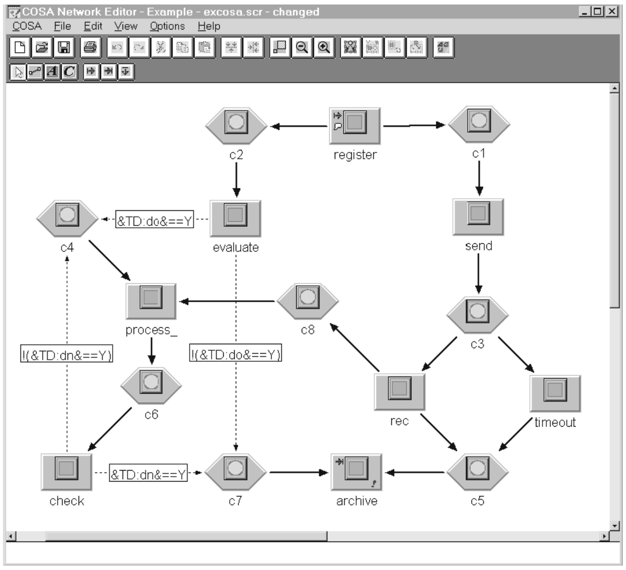

FIGURE 9. The COSA specification of the process of Figure 1

to two non-deterministic choices (as in Figure 1).

Figure 10 shows the same workflow process specified using the Graphical Workflow Designer (GWD) of Staffware [43]. The behavior of the specification shown in Figure 10 corresponds to the P/T net shown in Figure 1. Nevertheless, the diagram is quite different. Staffware tasks, called steps in Staffware, have OR-join/AND-split semantics. Therefore, explicit building blocks need to be added to synchronize (AND-join) and select (OR-split). A wait step, which is represented by a sand timer, is used to synchronize parallel flows. Conditions, represented by diamonds, correspond to binary choices. Moreover, Staffware does not have the concept of places. In the example of Figure 1, places are, among other things, used for OR-joins. To emulate OR-joins in the Staffware model corresponding to the P/T net of Figure 1, three so-called complex routers (which can be interpreted as automatic steps) have been added: join, done, and do. These three routers need to be added to join alternative flows. The traffic light in Figure 10 shows the beginning of the workflow process and the stop sign shows the end. Note that the timeout is modeled explicitly in Figure 10 and is attached to task rec. If rec is not executed within a given period, then task timeout is triggered. Using the translation described in [9], one can automatically translate a Staffware process definition to a Petri net. It should be noted that the translation of [9] applied to the workflow process shown in Figure 10 results in a P/T net that is different from the one shown in Figure 1: The resulting P/T net is considerably larger because the translation is generic. For example, the automatic steps join, done, and do shown in Figure 10 are not present in Figure 1 but will be present as transitions in the result of the translation of [9]. Nevertheless, the behavior of the Staffware model shown in Figure 10 matches the behavior of the P/T net shown in Figure 1.

Figures 9 and 10 illustrate the differences between workflow modeling languages used by today’s workflow manage-

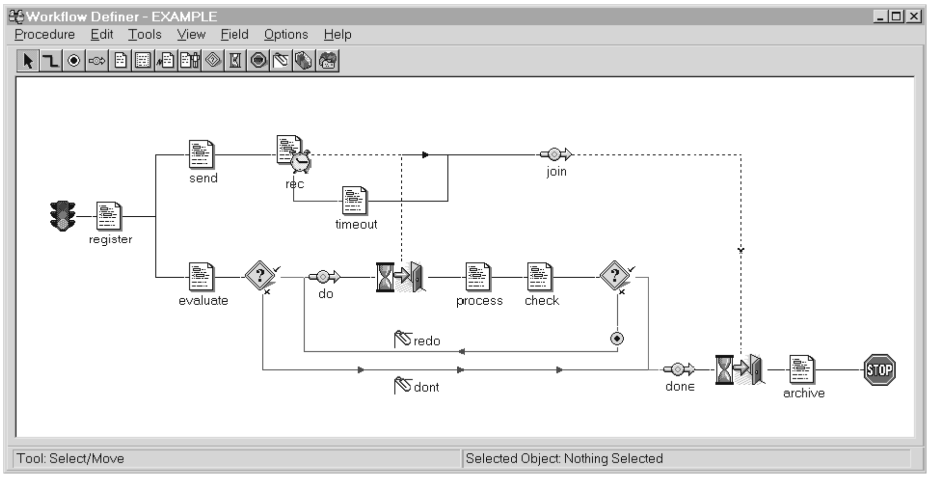

FIGURE 10. The Staffware specification of the process of Figure 1

ment systems. Both designs model the process corresponding to the Petri net shown in Figure 1. In the remainder, it is shown that this workflow process is incorrect, e.g., the workflow will deadlock if a redo is needed. As a result, both COSA and Staffware may deadlock if the workflow is executed. This example is no exception: In the current generation of workflow management systems, there are hardly any verification capabilities. Therefore, it is relevant to develop tools which can detect anomalies in workflow designs. Instead of building a specific workflow verification tool for every workflow management system, we propose the approach illustrated by Figure 11.

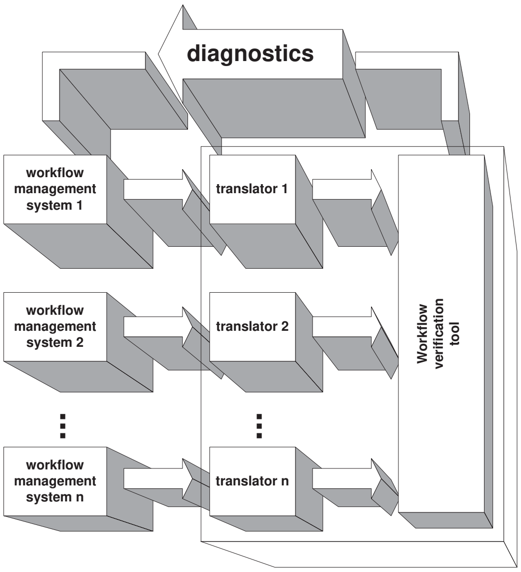

FIGURE 11. The approach supported by Woflan

As Figure 11 shows, there is a specific *translator* for each workflow management system. Such a translator translates a workflow process definition into a P/T net. During the translation, the abstraction discussed in the previous subsection is used to extract the information required for qualitative verification. It is important to note that the workflow verification tool is *not* used to edit the workflow process definition. If the verification tool detects errors, then the diagnostics provided by the verification tool are used to correct the errors using the design tools of the workflow management system itself. As Figure 11 shows, the process of correcting the errors is iterative: The workflow process definition constructed using the workflow management system is translated and analyzed using the verification tool. Then, the diagnostics are used to correct (if necessary) the process definition using the workflow management system. This procedure is repeated until all errors have been repaired. Note that the approach illustrated in Figure 11 stands or falls with the assumption that the diagnostics are of high-quality and workflow-system independent. Since most workflow management systems model workflows in terms of a graph structure connecting tasks, it is possible to make the diagnostics relatively system independent. For example, the verification tool can present a list of tasks which cannot be executed or show execution sequences in terms of tasks which lead to a deadlock. These diagnostics can be interpreted in the context of any workflow management system. To improve the feedback to the workflow modeler, it is possible to use the diagnostics to highlight the errors directly in the design tools of workflow management systems. Note that the latter requires extensions of the workflow management system itself.

4. WORKFLOW NETS

In this section, we introduce the class of *workflow nets* (WF nets) being the subclass of P/T nets used for modeling workflow process definitions as originally introduced in [1]. In addition, we formalize the soundness property introduced in Section 1 in terms of WF nets. We also briefly consider the subclass of free-choice WF nets. Finally, we present techniques for analyzing whether or not a given WF net is sound.

The soundness property is the least requirement that a WF net must satisfy in order to model a correct workflow process definition. As explained, a WF net is an abstraction of the actual workflow process, i.e., only the control-flow perspective is considered. We do not propose WF nets as a complete modeling language. They are merely introduced for the purpose of (qualitative) verification. When importing a workflow process definition from some workflow tool, our verification tool Woflan distills the aspects it needs from the workflow process definition and translates this information to a WF net.

4.1. Structural restrictions

Not every P/T net corresponds to a proper workflow process definition. A P/T net modeling a workflow must satisfy several structural properties.

First, we want a P/T-net model of a workflow process to have a well-defined beginning and end. Therefore, we require that such a P/T-net model has one place indicating the condition that a case has been created and one place indicating that a case has been completed. In the example of Figure 1, these places are called `i` and `o`, but they also could have been called `start` and `finish`. From now on, we assume that i (in) and o (out) identify these places. There can be no tasks that fulfill the condition corresponding to i: The workflow cannot generate its own cases. Also, there can be no tasks for which the condition corresponding to o has to be fulfilled: Once a P/T-net model of a workflow signals that a case has been completed, no more tasks should be executed for this case.

Second, observe that there is not much use in having a task that can never be executed or in having a task from which a case cannot be completed. Thus, we want to exclude such tasks. In terms of the structure of a workflow net, this means that it must satisfy at least the following requirement: For every transition t in a workflow net, there must be a directed path from i to t and a directed path from t to o. In P/T-net terms, this conforms to strongly connectedness (see Definition 2.5) under the assumption that there is a directed path from o to i. This assumption can be fulfilled if we short-circuit the net as illustrated in Figure 3.

DEFINITION 4.1. (*Workflow net*) A P/T net $N = (P, T, F)$ is a workflow net (WF net) iff

(i). $i \in P \wedge \bullet i = \emptyset$,
(ii). $o \in P \wedge o\bullet = \emptyset$, and
(iii). the short-circuited P/T net $(P, T \cup \{\underline{t}\}, F \cup \{(o, \underline{t}), (\underline{t}, i)\})$, denoted $\underline{N}$, is strongly connected, where $\underline{t} \notin T$.

The example P/T net `N` of Figure 1 satisfies all three conditions, using place `i` as input place i and `o` as output place o. Thus, it is a WF net.

4.2. Behavioral restrictions

Considering the behavioral correctness of a workflow, we are, as explained Section 3.3, interested in the behavior of a single case. Assuming a WF net $N = (P, T, F)$, it is an obvious choice to have $[i]$ as the initial marking, because it corresponds to the creation of a new case. So, $S = (N, [i])$ is the WF system corresponding to N that we are interested in.

The behavioral restrictions we impose on a WF system in its initial state can be derived from the soundness requirement introduced in Section 1. Recapitulating, a workflow process must always have the option to complete, completion must always be proper, and every task should contribute to at least one possible execution of the workflow. In a WF net, completion of a case is signaled by a token in the special place o. Thus, the completion option means that it must always be possible to put a token in o. Proper completion means that, as soon as a token is put in o, all other places must be empty. The last requirement strengthens the third structural requirement of Definition 4.1. It simply means that a WF system may not have any dead transitions (see Definition 2.17).

DEFINITION 4.2. (*Soundness*) A WF net $N = (P, T, F)$ is sound iff

(i). $\forall M \in B(P), [i] \longrightarrow M : \exists M_1 \in B(P), M \longrightarrow M_1 : M_1 \geq [o]$ (option to complete),
(ii). $\forall M \in B(P), [i] \longrightarrow M : M \geq [o] \Rightarrow M = [o]$ (proper completion), and
(iii). no transition $t \in T$ is dead in $(N, [i])$ (no dead tasks).

Soundness is originally defined in [1], where it says that it should always be possible to complete the case properly (option to complete properly). Our definition is slightly different, but it is not difficult to prove that they are equivalent.

Soundness of a WF net N can, for example, be determined from a CG of the WF system $(N, [i])$. If we take a look at our WF system `S` in Figure 2 and its OG in Figure 7 (which is also the unique CG of `S`), we see that `N` is not sound because the first two restrictions are not satisfied:

(i). In [`c4`, `c5`], there is no option to complete;
(ii). in [`c8`, `o`], we have improper completion.

The third restriction is satisfied, because for every transition we have at least one arc labeled with it in the CG.

In [1], it has been shown that soundness of a WF net corresponds to liveness (see Definition 2.18) and boundedness (see Definition 2.19) of the short-circuited WF system. Recall that, for a WF net N, the short-circuited net $(P, \underline{T} = T \cup \{\underline{t}\}, \underline{F} = F \cup \{(o, \underline{t}), (\underline{t}, i)\})$ with $\underline{t} \notin T$ is denoted $\underline{N}$.

THEOREM 4.1. (*Soundness vs. liveness and boundedness*) A WF net $N = (P, T, F)$ is sound iff the short-circuited WF system $(\underline{N}, [i])$ is live and bounded.

Proof. See [1]. □

From the CG in Figure 8, we conclude that the short-circuited WF system $\underline{S}$ of Figure 3 is not bounded and not live. It is not bounded, because we have infinite markings in the CG; it is not live, because, for instance, marking [`c4`, `c5`] has no outgoing arcs. Hence, the WF net `N` of Figure 1 is not sound, which conforms to our earlier conclusion.

4.3. Free-choice WF nets

The class of free-choice WF nets (see Definition 2.8) is an interesting one for two reasons. First, it appears that many workflow management systems allow only workflow process definitions that result in free-choice WF nets. Most of the workflow management systems available at the moment abstract from states between tasks, i.e., states are not represented explicitly. Such workflow management systems use the AND-split, AND-join, OR-split, and OR-join as standard building blocks to specify workflow procedures. Because these systems abstract from states, every choice is made *inside* an OR-split building block. If we model such an OR-split in terms of a WF net, the OR-split corresponds to a number of transitions sharing the same set of input places. Thus, it appears that for these workflow management systems a workflow procedure always corresponds to a free-choice WF net. Only a few workflow management systems (e.g., COSA, INCOME, LEU, and MOBILE) allow arbitrary non-free-choice constructs. Second, for a free-choice WF net, it can be decided in polynomial time whether or not the net is sound, because it is possible to verify in polynomial time whether the corresponding short-circuited WF system is live and bounded [16].

Given these two facts, one could envision a verification tool that focuses on free-choice WF nets. However, for Woflan, we decided differently. One of the main requirements for Woflan mentioned in the introduction is that it is workflow-product independent. Allowing non-free-choice WF nets means that Woflan can support a wider range of (future) workflow management systems. Furthermore, standard routing constructs, such as parallelism, sequential routing, conditional routing, and iteration, can be modeled without violating the free-choice property. However, sometimes, complex routing constructs cannot be modeled with free-choice WF nets. For example, Staffware is a workflow management system that abstracts from states (see also Section 3.3) but it supports one (rarely used) construct that can only be translated to a non-free-choice construct in the corresponding WF net (see [9], for more details). In other occasions, non-free-choice constructs yield more concise models than the corresponding free-choice ones. A second requirement for Woflan mentioned in the introduction is that it must provide to-the-point diagnostic information in case of design errors. Unfortunately, efficient algorithms for verifying soundness are not necessarily a good basis for meaningful diagnostic information in case a WF net is not sound.

4.4. Analyzing WF nets

Theorem 4.1 is an interesting result, because it shows that for the analysis of WF nets we can focus on boundedness and liveness of short-circuited WF systems. Boundedness and liveness have been studied extensively in the Petri-net literature. Existing results can be tuned to the analysis of WF nets. In the remainder of this section, we present results that form the foundation of Woflan, emphasizing results that are useful for providing meaningful diagnostic information in case of errors in a WF net.

4.4.1. Structural techniques

In Section 2.2.1, a number of structural techniques for analyzing P/T nets have been introduced. Despite the fact that Woflan is not restricted to free-choice WF nets, the free-choice property does play a role in diagnosing WF nets. Also, PT- and TP-handles, S-components and S-coverability, and (place-)invariants all play an important role in Woflan. The interpretation of non-free-choice constructs, PT/TP-handles, and S-components in the workflow domain is explained in more detail in the next section. In this subsection, we present results relating structural techniques to soundness of WF nets.

THEOREM 4.2. (*Sound and free-choice vs. S-coverable*) Let N be a sound, free-choice WF net. The short-circuited WF net $\underline{N}$ is S-coverable.

Proof. This follows directly from Theorem 4.1 and the fact that a net which is free-choice, live, and bounded must be S-coverable ([16]). □

In the analysis of WF nets, this theorem can be used as follows. If N is a free-choice WF net such that $\underline{N}$ is not S-coverable, then N cannot be sound. Places that are not part of any S-component are a potential source of the error. For example, the WF net `N` of Figure 1 is free-choice, but $\underline{\texttt{N}}$ of Figure 3 is not S-coverable, as explained in Section 2.2.1. Place `c8` is not part of an S-component. Thus, net `N` is not sound, as we have concluded earlier.

DEFINITION 4.3. (*Well-structuredness*) A WF net N is well-structured iff $\underline{N}$ is well-handled, i.e., the short-circuited net has no PT-handles and TP-handles (see Definition 2.7).

THEOREM 4.3. (*Sound and well-structured vs. S-coverable*) Let N be a sound, well-structured WF net. The short-circuited WF net $\underline{N}$ is S-coverable.

Proof. See [4]. □

Theorem 4.3 can be used in the analysis of WF nets in a similar way as Theorem 4.2 can be used. Theorem 4.3 does not provide useful information for our running example, because short-circuited WF net $\underline{\texttt{N}}$ of Figure 3 is not well-structured.

As a side remark, note that for a given well-structured WF net, it can be decided in polynomial time whether or not it is sound. (See [4]; the proof uses Theorem 4.1 and the fact that short-circuited WF nets without PT-handles and TP-handles are elementary extended non-self controlling [11].) Also note that the classes of free-choice WF nets and well-structured WF nets are incomparable. That is, there are free-choice nets that are not well-structured and vice versa.

S-coverability of a short-circuited WF net is a sufficient (but not necessary) condition for safeness and, hence, boundedness of the corresponding system.

THEOREM 4.4. (*S-coverability vs. boundedness*) Let N be a WF net and let the short-circuited WF net $\underline{N}$ be S-coverable. The short-circuited WF system $(\underline{N}, [i])$ is safe and bounded.

Proof. It follows from Definition 2.11 that the number of tokens in any reachable marking of $(\underline{N}, [i])$ in an S-component of $\underline{N}$ is constant. Because we initially have one token (in i), the number of tokens in any S-component is either zero or one. Therefore, the number of tokens in any place in any S-component is always either zero or one. Because all places in $\underline{N}$ are contained in some S-component, $(\underline{N}, [i])$ is safe and thus bounded. □

Note that a consequence of Theorem 4.4 is that both sound free-choice WF nets and sound well-structured WF nets correspond to safe WF systems.

Considering again our running example, we have already seen that $\underline{\texttt{N}}$ of Figure 3 is not S-coverable and that system $(\underline{\texttt{N}}, \texttt{[i]})$ is not bounded. Since place `c8` is not part of an S-component, again the diagnostic information points to place `c8` as a possible error: It *might* be unsafe or unbounded. (We, of course, already know that `c8` is unbounded.)

It is also well-known that place-invariants with only non-negative weights, the so-called *semi-positive* place-invariants, can be used to formulate a sufficient condition for boundedness. A place occurring with a positive weight in a semi-positive place-invariant is said to be *covered* by that invariant.

THEOREM 4.5. (*semi-positive place-invariants vs. boundedness*) Let N be a WF net. If every place of N is covered by a semi-positive place-invariant of the short-circuited net $\underline{N}$, then the short-circuited WF system $(\underline{N}, [i])$ is bounded.

Proof. It follows immediately from Theorem 2.31 of [16]. □

Places not covered by a semi-positive place-invariant of a short-circuited WF net might be indications of an error. In the running example, place `c8` is the only place not covered by a semi-positive place-invariant of $\underline{\texttt{N}}$.

4.4.2. *Liveness and boundedness vs. soundness*

In this paragraph, we investigate the relation between the soundness of a WF net and the liveness and boundedness of the corresponding short-circuited WF system in some more detail.

As the following result shows, an unbounded place in a short-circuited WF system may be a sign of improper completion.

THEOREM 4.6. (*Improper completion vs. unboundedness*) Let N be a WF net that can complete improperly. Then, the short-circuited WF system $(\underline{N}, [i])$ has unbounded places.

Proof. It follows from the assumption and the definition of proper completion (Definition 4.2) that there exists a non-empty marking $M \in B(P)$ such that $[i] \longrightarrow M + [o]$ in N. Then, $[i] \longrightarrow M + [o]$ in $\underline{N}$ and, because of the short-circuiting transition $\underline{t}$, $[i] \longrightarrow M + [i]$ in $\underline{N}$. We conclude that all places in M are unbounded in $(\underline{N}, [i])$. □

In the OG of Figure 7, we see that WF net `N` of Figure 1 may complete improperly, because marking `[c8,o]` is reachable. The CG of Figure 8 shows that system $(\underline{\texttt{N}}, \texttt{[i]})$ has unbounded place `c8`.

Non-live transitions in a short-circuited WF system are a potential sign that a WF net does not satisfy the completion option.

THEOREM 4.7. (*Option to complete vs. liveness*) Let $N = (P, T, F)$ be a WF net that does not satisfy the completion option. Then, the short-circuited WF system $(\underline{N}, [i])$ has non-live transitions.

Proof. Suppose $(\underline{N}, [i])$ has only live transitions. Then, the short-circuiting transition $\underline{t}$ is live, i.e., for all $M \in B(P)$ with $[i] \longrightarrow M$, there exists an $M_1 \in B(P)$ with $M \longrightarrow M_1$ such that $\bullet\underline{t} \leq M_1$. Since $\bullet\underline{t} = \{o\}$, we immediately conclude that N has the option to complete. □

Let us return to WF net `N` of Figure 1 once more. The CG of `S=(N, [i])` in Figure 7 has a deadlock marking, namely `[c4,c5]`; thus, `N` does not satisfy the completion option. Since the CG of $\underline{\texttt{S}}$=$(\underline{\texttt{N}}, \texttt{[i]})$ in Figure 8 has the same deadlock marking, all transitions of $\underline{\texttt{S}}$ are non-live. Although this observation is consistent with Theorem 4.7, it does unfortunately not provide any useful diagnostic information on WF net `N`.

Part of the soundness requirement on a WF net is the absence of dead transitions in the corresponding WF system. A dead transition in a WF system corresponds to a task in the workflow that can never be executed. Non-live transitions in the short-circuited WF system, in particular dead transitions, might be a sign of dead transitions in the non-short-circuited WF system. The question is how dead transitions in a WF system $S = (N, [i])$ and the short-circuited WF system $\underline{S}$ relate to each other. Observe that any occurrence sequence of S is also an occurrence sequence of $\underline{S}$, but that the converse is not necessarily true. Thus, a transition that is dead in $\underline{S}$ is also dead in S, but a transition that is dead in S might not be dead in the short-circuited system $\underline{S}$. However, under the assumption of boundedness of $\underline{S}$, a transition that is dead in S is also dead in $\underline{S}$.

THEOREM 4.8. (*Dead transitions in bounded short-circuited WF systems*) Let $S = (N, [i])$ with $N = (P, T, F)$ be a WF system such that the short-circuited system $\underline{S} = (\underline{N}, [i])$ is bounded. Transition $t \in T$ is dead in S iff it is dead in $\underline{S}$.

Proof. The result follows immediately from the observation that, under the boundedness assumption, either the OGs of $\underline{S}$ and S are identical (in case marking $[o]$ is not reachable in S) or the OG of $\underline{S}$ extends the OG of S with the arc $([o], \underline{t}, [i])$ (in case $[o]$is reachable). □

4.4.3. *Behavioral error sequences*

Structural errors in a P/T net modeling a workflow, i.e., violations of the requirements of Definition 4.1, are generally easy to find and to correct. Behavioral errors, i.e., violations of Definition 4.2, are more difficult to locate and to correct.

The results in Section 4.4.2 show that the sets of unbounded places in a short-circuited WF net, as well as the lists of non-live and dead transitions may provide useful information for diagnosing behavioral errors. Unbounded places, non-live transitions, and dead transitions all point to different types of behavioral errors in a WF net. However, experience with verification of workflow processes has shown that this information is not always sufficient for finding the exact cause of an error. In particular, it might be difficult to diagnose violations of requirements (i) (option to complete) and (ii) (proper completion) of Definition 4.2. To overcome this problem, we introduce so-called behavioral error sequences. The idea for these sequences is relatively simple: We want to find firing sequences of minimal length such that *every* continuation of that sequence leads to an error. Such a firing sequence is required to be minimal in the sense that no prefix has the property that every continuation leads to an error. Thus, one can think of behavioral error sequences as *scenarios* that capture the essence of errors made in the workflow design. Depending on the kind of error one is interested in, different types of behavioral error sequences can be helpful for diagnosing the design. In the next two paragraphs, we introduce two types of behavioral error sequences called *non-live sequences* and *unbounded sequences* that are particularly useful for diagnosing liveness-related (requirement (i) of Definition 4.2, option to complete) and boundedness-related (requirement (ii) of Definition 4.2, proper completion) behavioral errors, respectively.

4.4.4. *Non-live sequences*

Intuitively, a non-live sequence is a firing sequence of a workflow system of minimal length such that completion is no longer possible (i.e., it is no longer possible to reach a marking with a token in the special place o). By now, it is clear that the completion-option requirement of a WF net is strongly related to the liveness of the corresponding short-circuited system. Liveness analysis is only feasible for bounded systems. Thus, we assume a WF system $S = (N, [i])$ such that the short-circuited system $\underline{S} = (\underline{N}, [i])$ is bounded. We also assume the absence of dead transitions in S (or equivalently in $\underline{S}$; see Theorem 4.8). In the next section, it is explained in more detail how these assumptions are enforced in the diagnosis process of Woflan. The precise definition of non-live sequences is based on the following theorem.

THEOREM 4.9. (*Liveness of bounded short-circuited WF systems*) Let $S = ((P, T, F), [i])$ be a WF system without dead transitions such that the short-circuited system $\underline{S}$ is bounded. Then, $\underline{S}$ is live iff $\forall M \in B(P), [i] \longrightarrow M : M \longrightarrow [o]$.

Proof. The implication from left to right follows in a straightforward way from Definition 4.2 (Soundness) and Theorem 4.1 (Soundness vs. liveness and boundedness). The other implication follows directly from Definitions 2.17 (Dead transitions) and 2.18 (Liveness). □

Based on this theorem, we define a non-live sequence as a firing sequence of WF system S of minimal length that ends in a marking from which it is no longer possible to reach $[o]$. Non-live sequences can be computed from the OG of S. Note that the OG of S is finite, because $\underline{S}$ and hence also S is bounded. In terms of the OG of S, a (non-empty) non-live sequence is a firing sequence corresponding to a path in the OG that starts in marking $[i]$ and ends in a marking M

(i). from which there is no path to $[o]$ and
(ii). whose immediate predecessor M_1 on the path has a path to $[o]$.

Apparently, the transition leading from marking M_1 to marking M removes the option to complete. To determine which markings in the OG can act as M and M_1, we partition the markings into three parts:

(i). red markings, from which there is no path to $[o]$,
(ii). green markings, from which all paths lead to $[o]$, and
(iii). yellow marking, from which some but not all paths lead to $[o]$.

Only a red marking can possibly act as M, whereas only a yellow marking can possibly act as M_1. All we need to do now is to find arcs in the OG which connect a yellow marking to a red marking. The label of such an arc gives us the name of the transition whose firing removes the option to complete. Any path from the initial marking $[i]$ to M in the OG corresponds to a non-live sequence.

The definition of non-live sequences can be formalized as follows. Note that the definition does not require the absence of dead transitions in the WF system under consideration. Let $M_1 \Longrightarrow M$ denote that there exists a path in the OG from node M_1 to node M.

DEFINITION 4.4. (*OG partitions for non-liveness*) Let $N = (P, T, F)$ be a WF net such that its WF system $(N, [i])$ is bounded. Let $G = (H, A)$ be the OG of $(N, [i])$. We partition H into three parts:

(i). $H_R = \{M \in H \mid \neg(M \Longrightarrow [o])\}$,
(ii). $H_G = \{M \in H \mid \neg\exists M_R \in H_R : M \Longrightarrow M_R\}$ and
(iii). $H_Y = H \setminus (H_G \cup H_R)$.

Remarks:

- If there are no red markings, there can be no yellow markings: $H_R = \emptyset$ implies $H_Y = \emptyset$.
- If there are no green markings, there can be no yellow markings: $H_G = \emptyset$ implies $H_Y = \emptyset$.
- If there is no way to complete properly, then all markings are red: $[o] \notin H$ implies $H = H_R$.
- If there is a way to complete properly, then the target marking is green (because $o\bullet = \emptyset$): $[o] \in H$ implies $[o] \in H_G$.

DEFINITION 4.5. (*Non-live sequences*) Let $(N, [i])$ be a bounded WF system with OG $G = (H, A)$. Let H_R and H_Y be defined as in Definition 4.4. If $[i] \in H_R$, then the occurrence sequence $[i]$ is called non-live. An occurrence sequence $[i]t_0M_1 \ldots t_{n-2}M_{n-1}t_{n-1}M_n$, for some positive natural number n, with all markings distinct is called non-live iff $M_n \in H_R$ and $M_{n-1} \in H_Y$. A *firing* sequence of a WF system is called non-live iff it is derived from a non-live occurrence sequence.

The most valuable information in a non-live sequence is the combination of its last two markings ($M_{n-1} \in H_Y$ and $M_n \in H_R$) and its last transition (t_{n-1}). The only interest we have in the sequence's prefix ($[i]t_0M_1 \ldots t_{n-2}$) is that it gives us a path which leads to the last-but-one marking. Note that we have excluded firing sequences containing cycles (by requiring that all markings in a non-live sequence must be distinct); cycles do not provide any additional useful information. Also note that it is possible that several non-live sequences have the same suffix $M_{n-1}t_{n-1}M_n$.

THEOREM 4.10. (*Non-live sequences vs. liveness*) Let S be a WF system without dead transitions such that the short-circuited system $\underline{S}$ is bounded. Then, $\underline{S}$ is live iff S has no non-live sequences.

Proof. The theorem follows immediately from Theorem 4.9 (Liveness of bounded short-circuited WF systems) and Definition 4.5 (Non-live sequences). □

Note that, based on Theorem 4.1, Theorem 4.10 can alternatively be formulated as follows. If $S = (N, [i])$ is a WF system without dead transitions such that the short-circuited system $\underline{S}$ is bounded, then N is sound iff S has no non-live sequences.

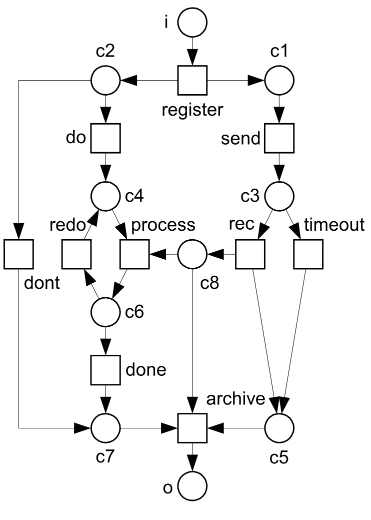

FIGURE 12. WF net $\mathtt{N}_1$

As an example, consider the WF net $\mathtt{N}_1$ of Figure 12. It is a variant of WF net `N` of Figure 1 with an extra arc from place `c8` to transition `archive`. The OG of $\mathtt{S}_1$=($\mathtt{N}_1$, `[i]`) is shown in Figure 13. The meaning of the thick arcs is explained in the next section. Clearly, $\mathtt{S}_1$ has no dead transitions. Since the OG of $\underline{\mathtt{S}}_1$=($\underline{\mathtt{N}}_1$, `[i]`) is simply the graph in Figure 13 extended with the arc (`[o]`, `shortcircuit`, `[i]`), where `shortcircuit` is the short-circuiting transition, we see that $\underline{\mathtt{S}}_1$ is bounded. Figure 13 also shows the partitioning of the OG of $\mathtt{S}_1$ according to Definition 4.4. We

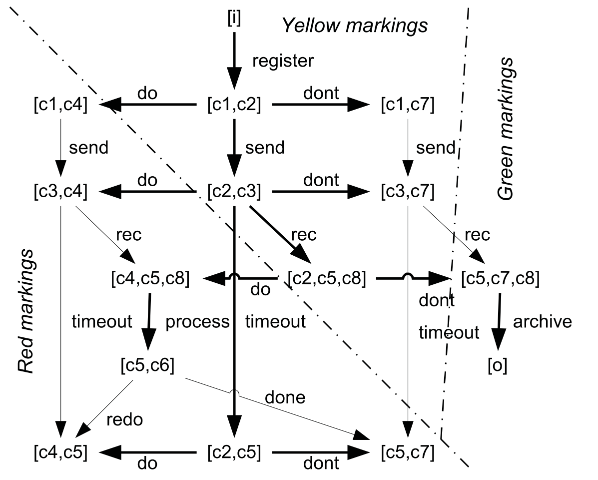

FIGURE 13. The OG of $\mathtt{S}_1$ partitioned for non-live sequences

can deduce, among others, the following five non-live sequences:

(i). `register send timeout`,
(ii). `register send dont timeout`,
(iii). `register send rec do`,
(iv). `register send do`, and
(v). `register do`.

Since $\mathtt{S}_1$ has non-live sequences, we can deduce from Theorem 4.10 that $\underline{\mathtt{S}}_1$ is not live, which means that $\mathtt{N}_1$ is not sound. It is also possible to arrive at this conclusion by investigating the OG of $\underline{\mathtt{S}}_1$. Since it contains deadlock marking `[c4,c5]`, it follows that all transitions of $\underline{\mathtt{S}}_1$ are non-live. Unfortunately, the information that all transitions are non-live is not sufficiently specific to be useful. By examining the above five non-live sequences, we can obtain more detailed information. Note that non-live sequence (ii) provides almost the same information as sequence (i). Together, they show that the combination `send` and `timeout` is the possible cause of an error and that `dont` is not important. From sequence (i), we conclude that, whatever happens, place `c8` does not get a token. As a result, transitions `process` and `archive` cannot fire. The sequences (iii), (iv), and (v) provide the information that firing transition `do` always results in an error. We may conclude that the cycle to which `do` leads might cause a problem. For now, we do not go into details about possible solutions to correct the errors.

4.4.5. Unbounded sequences

Intuitively, an unbounded sequence is a firing sequence of a WF system of minimal length such that every continuation implies a violation of the proper-completion requirement of Definition 4.2. Such a violation can have two causes. The first one is the most straightforward one. Clearly, proper completion is violated if a reachable marking is strictly greater than the marking $[o]$ that signals proper completion. The second cause is more implicit. If a WF system is unbounded, then the proper-completion requirement is also violated. To see this, consider a WF system $S = (N, [i])$ with two reachable markings M and M_1 such that $M < M_1$ (which by Definition 2.19 means that S is unbounded). Assuming that proper completion is possible from M, i.e., $M \longrightarrow [o]$, we may deduce that $M_1 \longrightarrow [o] + M_1 - M$ which is strictly greater than $[o]$. Thus, assuming that completion is possible at all, unboundedness of a WF system implies a violation of the proper-completion requirement.

As we have seen, the proper-completion requirement of a WF net is strongly related to the boundedness of the corresponding short-circuited system. The following theorem confirms this observation. It forms the basis for formalizing unbounded sequences.

THEOREM 4.11. (*Boundedness of short-circuited WF systems*) Let $S = ((P, T, F), [i])$ be a WF system. System $\underline{S} = ((P, \underline{T}, \underline{F}), [i])$ is bounded iff system S is bounded and, for all markings $M \in B(P)$ reachable from $[i]$ in S, $\neg(M > [o])$.

Proof. To prove the theorem, we show that $\underline{S}$ is unbounded iff S is unbounded or there is a marking $M \in B(P)$ reachable from $[i]$ in S such that $M > [o]$. Recall Definition 2.19 (Boundedness). The implication from right to left is straightforward (see also the proof of Theorem 4.6). The other implication is more involved. Assume that $s = M_0 t_1 M_1 \ldots t_n M_n$, for some natural number n, is an occurrence sequence of $\underline{S}$ such that $M_0 = [i]$ and such that there exists a $k < n$ with $M_k < M_n$. Distinguish two cases. First, assume that the short-circuiting transition $\underline{t}$ is not an element of $\{t_1, \ldots, t_n\}$. In this case, s is also an occurrence sequence of S, which means that S is unbounded. Second, assume that $\underline{t}$ is an element of $\{t_1, \ldots, t_n\}$. Without loss of generality, we may assume that s is minimal in the following sense: First, all markings $M_0, \ldots, M_n$ are different; second, there are no natural numbers k and l with $k < l < n$ such that $M_k < M_l$. The first assumption means that s contains no cycles; the second assumption means that s contains no strict prefix from which unboundedness can be derived. The crux of the proof is that $\underline{t}$ must be t_n. Suppose that $\underline{t}$ equals t_k, with $k < n$. Since $\bullet\underline{t} = \{o\}$ and $\underline{t}\bullet = \{i\}$, $M_{k-1} \geq [o]$ and either $M_k = [i] = M_0$ or $M_k > [i] = M_0$. In both cases, the minimality of s is violated. Thus, $\underline{t}$ equals t_n. It follows from the definition of $\underline{t}$ and s that $M_n > [i]$ and that the occurrence sequence $M_0 t_1 M_1 \ldots t_{n-1} M_{n-1}$ is an occurrence sequence of S such that $M_{n-1} > [o]$. □

Unbounded sequences can be computed from a coverability graph of a WF system S (see Section 2.2.4). Assuming we have a CG of S, an unbounded sequence is a firing sequence of S of minimal length which inevitably leads either to an infinite marking in the CG or to a marking greater than $[o]$ in that CG. The above theorem means that such a sequence corresponds to a sequence of $\underline{S}$ that inevitably leads to an infinite marking when the CG of S is extended to a CG of $\underline{S}$.

To compute unbounded sequences, we partition a given CG of S in a way similar to the partitioning of the OG for computing non-live sequences given in Definition 4.4:

(i). The green markings are those markings from which infinite markings or markings greater than $[o]$ are not reachable;

(ii). the red markings are those markings from which infinite markings or markings greater than $[o]$ are unavoidable, i.e., those markings from which no green marking is reachable;

(iii). the yellow markings are those markings from which infinite markings or markings greater than $[o]$ are reachable but avoidable.

DEFINITION 4.6. (*CG partitions for unboundedness*) Let $N = (P, T, F)$ be a WF net, let $G = (H, A)$ be a CG of WF system $(N, [i])$, and let $H^\omega = H \setminus B(P) \cup \{M \in B(P) \mid M > [o]\}$ be the set of markings in H that are infinite or greater than $[o]$. We partition H into three parts:

(i). $H^\omega_G = \{M \in H \mid \neg\exists M_1 \in H^\omega : M \Longrightarrow M_1\}$,
(ii). $H^\omega_R = \{M \in H \mid \neg\exists M_1 \in H^\omega_G : M \Longrightarrow M_1\}$ and
(iii). $H^\omega_Y = H \setminus (H^\omega_G \cup H^\omega_R)$.

Remarks:

- If there are no red markings, there can be no yellow markings: $H^\omega_R = \emptyset$ implies $H^\omega_Y = \emptyset$.
- If there are no green markings, there can be no yellow markings: $H^\omega_G = \emptyset$ implies $H^\omega_Y = \emptyset$.

Given the above partitioning of a CG of a WF system, we can define its unbounded sequences.

DEFINITION 4.7. (*Unbounded sequences*) Let $(N, [i])$ be a WF system with CG (H, A). Let H^ω_R and H^ω_Y be defined as in Definition 4.6. If $[i] \in H^\omega_R$, then the occurrence sequence $[i]$ is called unbounded. An occurrence sequence $[i] t_0 M_1 \ldots t_{n-2} M_{n-1} t_{n-1} M_n$, for some positive natural number n, with all markings distinct is called unbounded iff $M_n \in H^\omega_R$ and $M_{n-1} \in H^\omega_Y$. A *firing* sequence of a WF system is called unbounded iff it is derived from an unbounded occurrence sequence.

THEOREM 4.12. (*Unbounded sequences vs. boundedness*) A short-circuited WF system $\underline{S}$ is bounded iff S has no unbounded sequences.

Proof. The theorem follows immediately from Theorem 4.11 (Boundedness of short-circuited WF systems) and Definition 4.7 (Unbounded sequences). □

Unbounded sequences have been defined on the basis of a CG of a WF system. However, CGs of WF systems can become very large, even to the extent that the computation of unbounded sequences may become intractable. A simple observation alleviates the problem of large CGs: Infinite markings in a CG have only infinite successors. For determining unbounded sequences, it is not necessary to consider successors of infinite markings, because they are guaranteed to be red. This observation leads to the notion of a *restricted CG* (RCG) of a system. Let $S = ((P, T, F), M_0)$ be some P/T system. An RCG of S is constructed via the algorithm of Definition 2.16 with one important difference, namely that we restrict the marking M in step (ii) to be finite. As an example, compare the CG of the short-circuited system of

Figure 3 depicted in Figure 8 with the RCG of Figure 14. For this simple example, the RCG is approximately half the size of the CG. Note that if a system is bounded the RCG-generation algorithm and the CG-generation algorithm both yield the OG of the system.

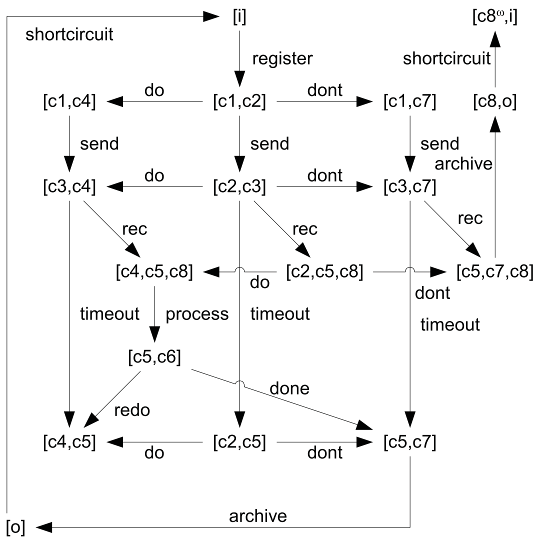

FIGURE 14. The RCG of the short-circuited example net

It is straightforward to see that an RCG can be used to compute the unbounded sequences of a WF system. Consider the partitioning of a CG given in Definition 4.6. Since infinite markings are always red, it is clear that successors of infinite markings are also red. Therefore, the part of a CG that is omitted in an RCG is not used when constructing unbounded sequences. This means that unbounded sequences can be computed by applying the partitioning of Definition 4.6 to an RCG.

The idea to restrict a CG of a system to an RCG is similar to one of the ideas behind the notion of an MCG (minimal CG) of [21]. In general, an RCG of a system is still larger than its MCG. Unfortunately, the MCG of a WF system is not suitable for computing unbounded sequences. For more details, the interested reader is referred to [21].

Figure 15 shows the partitioned RCG of the example system `S` of Figure 2. Note that this RCG is the OG of `S`, because `S` is bounded. `S` has among others the following unbounded sequences:

(i). `register send rec dont` and
(ii). `register send dont rec`.

These two sequences show that firing the combination of `rec` and `dont` inevitably leads to unboundedness of the short-circuited system. The reason is that `rec` puts a token in place `c8`, whereas firing `dont` removes the option to remove this token via transition `process`.

5. WOFLAN

This section describes *Woflan* (WOrkFLow ANalyzer, see http://www.tm.tue.nl/it/woflan) version 2.1. Woflan is a tool

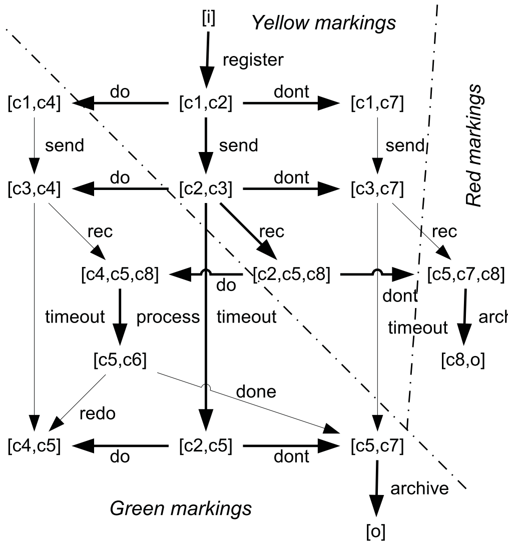

FIGURE 15. The RCG partitioned for unboundedness

that analyzes workflow process definitions specified in terms of Petri nets. It has been designed to verify process definitions that are downloaded from a workflow management system, as explained in Section 3.3. As indicated in the introduction, there is a clear need for such a verification tool.

Based on some of the results presented in the previous section, the development of the tool Woflan started at the end of 1996 and the first version was released in 1997 [8]. Basically, Woflan takes a workflow process definition imported from some workflow product, translates it into a P/T net, and tells whether or not the net is a sound WF net. Furthermore, using some standard P/T net-analysis techniques as well as those tailored to WF nets presented in the previous section, the tool provides diagnostic information about the net in case it is not a sound WF net. Woflan implements a predefined diagnosis process illustrated in Figure 16. The diagnosis process is in fact a workflow process modeled in Protos [31]. In the next subsection, the diagnosis process of Figure 16 is explained in detail. In Section 5.2, the P/T net of Figure 1 is analyzed by means of Woflan. Version 2.1 of Woflan extends version 1.0 as described in [8] with some new analysis techniques of which the technique of behavioral error sequences is the most important one, with a predefined, detailed diagnosis process that uses a new, workflow-oriented nomenclature, and with an import facility for COSA, Staffware, METEOR, and Protos. A brief overview of the material of this section was presented at the 2000 International Conference on Application and Theory of Petri nets [45].

5.1. Diagnosis process

In Sections 2 and 4, we have seen a wide range of analysis techniques for P/T nets in general and WF nets in particular. The goal is to apply these techniques in the analysis of workflow processes in a logical and meaningful order, and to distill useful diagnostic information from the analysis results in case of errors in the workflow. The diagnosis process

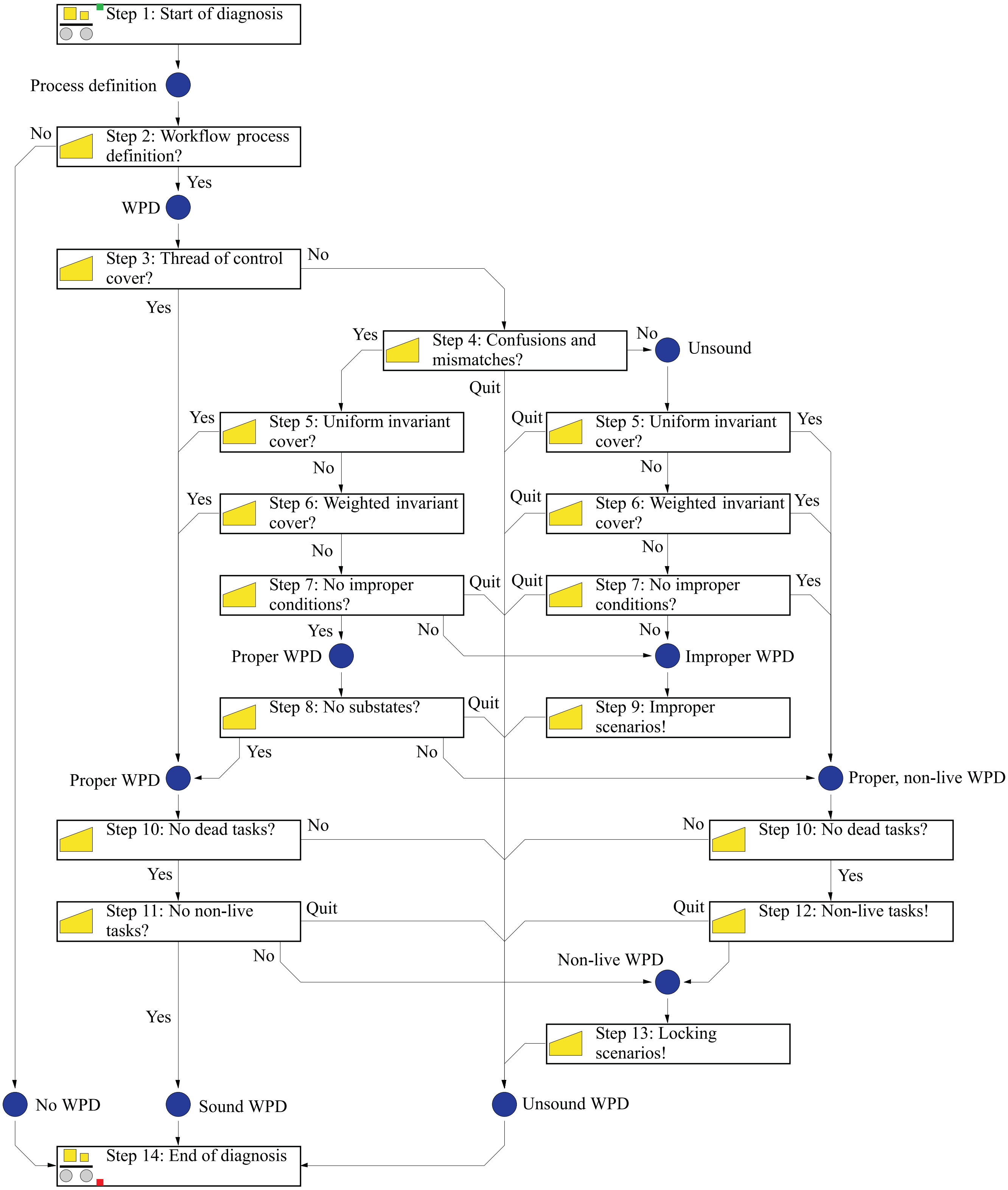

FIGURE 16. Diagnosis process, modeled using Protos

implemented in Woflan, version 2.1, achieves this goal. Figure 16 illustrates the process. As mentioned, the process is in fact a workflow itself modeled in Protos. The rectangles are the basic tasks in the process, where special symbols are used for the initial and the final task (Steps 1 and 14). The circles are similar to places in P/T nets. They are only included at some relevant points in the workflow (as explained below). Steps 2 through 8 and 10 through 12 are OR-splits and Step 14 is an OR-join. Analyzing the Protos model of Figure 16 in Woflan yields that it corresponds to a sound WF net.

The basis for the diagnosis process in Figure 16 is Theorem 4.1 (Soundness vs. liveness and boundedness). That is, the diagnosis process aims at establishing the soundness of a WF net by showing that the corresponding short-circuited system is live and bounded. As mentioned earlier, liveness analysis is only feasible for bounded systems. Thus, we have decided to center the diagnosis process around the following three milestones. The naming of the milestones is chosen in such a way that it fits with standard workflow terminology.

Workflow Process Definition (WPD) Does the imported process definition correspond to a WF net?

Proper WPD Is the short-circuited system corresponding to the WF net bounded?

Sound WPD Is the (bounded) short-circuited system corresponding to the WF net live (and thus the WF net sound)?

The order in which analysis techniques are applied in the diagnosis process is based on two criteria, namely efficiency of the technique and usefulness of the diagnostic information. Since structural analysis techniques are (usually) computationally much more efficient than behavioral ones, we see that structural analysis techniques are used as much as possible in the diagnosis process before switching to behavioral techniques.

5.1.1. Step 1: Start of diagnosis

The diagnosis process is started by importing a process definition from some workflow tool. In this step, the process definition is translated to a P/T-net representation, applying the abstractions discussed in Section 3.2.

5.1.2. Step 2: Workflow process definition?

In this step, it is verified whether the first milestone is satisfied. The first milestone is included to guarantee that the process definition that is being imported from some workflow tool corresponds to a WF net. Woflan simply checks whether all the requirements of Definition 4.1 are satisfied (one place must correspond to a point of creation, one place must correspond to a point of completion, and all nodes must be related to both places). If the milestone is not satisfied, the diagnosis process ends and the workflow designer must make a correction to the process definition. In this case, Woflan provides diagnostic information such as, for example, the list of tasks that are not connected to the point of creation and/or the point of completion.

5.1.3. Step 3: Thread of control cover?

From a workflow point of view, we would like to see a case as a set of parallel *threads of control*: Each such a thread specifies that certain tasks have to be executed in a certain (sequential) order to get a certain piece of work completed. In the running example of Figure 1, we have two such threads:

(i). The first thread handles the piece of work associated with the complaint form: After registration, first, the form has to be sent to the complainant. Second, it is either received back or a timeout occurs. Finally, the returned form or the fact that it was not returned in time is archived.

(ii). The second thread handles the piece of work associated with the complaint itself: After registration, first, the complaint has to be evaluated. Second, depending on the evaluation (`do` or `dont`), it may be processed followed by a check. Third, depending on the result of the check (`done` or `redo`), it may be processed again. Finally, it is archived.

The idea of threads is reflected by the S-components in the short-circuited WF net: Every S-component in that short-circuited net corresponds to a logical piece of work in the workflow. (See, for example, Figure 6 that shows the two S-components for the running example.) Recall that an S-component is a strongly connected state machine which is embedded in a P/T net (see Definition 2.11). For each S-component in a P/T system, the total number of tokens in its places is always constant. From the strongly connectedness of S-components and the structure of WF nets, it follows that an S-component in a short-circuited sound WF net always contains the short-circuiting transition $\underline{t}$ and the two special places i and o. Assuming the initial marking $[i]$, every place in an S-component is safe and bounded, and the system corresponding to a short-circuited WF net that is S-coverable is safe and thus bounded (see also Theorem 4.4). In addition, since i is an element of all S-components in an S-coverable net, every S-component contains exactly one token in every marking reachable from $[i]$. This observation conforms to the intuitive notion of threads of control.

It appears that any WF net should satisfy the requirement that its short-circuited net is S-coverable. A place that does not belong to a thread of control is a suspicious place, because it cannot be related to a logical piece of work. Although it is possible to construct a sound WF net with a short-circuited net that is not S-coverable, the places that are not S-coverable in sound WF nets typically do not restrict transitions from being enabled and are thus superfluous. Note that S-coverability is not a sufficient requirement: It is possible to construct an unsound WF net with an S-coverable short-circuited net.

The diagnostic information that Woflan provides is the list of S-components of the short-circuited WF net, as well as a list of places not contained in any of these S-components. This information can generally be computed efficiently. If there are no uncovered places, the second milestone of the diagnosis process (Proper WPD) has been achieved (see

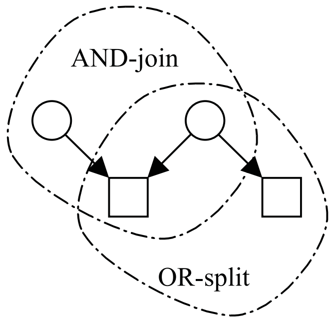

FIGURE 17. A non-free-choice cluster (confusion)

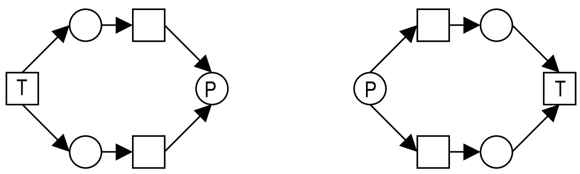

FIGURE 18. AND/OR mismatches

Theorem 4.4), which means that we can continue with liveness analysis (see Figure 16).

5.1.4. Step 4: Confusions and mismatches?
At this point, we know that our workflow process definition is not covered by threads of control; in Petri-net terminology, the short-circuited WF system is not S-coverable. Based on Theorems 4.2 and 4.3, we may conclude that the WF net under consideration should *not* be free-choice or well-structured. If it is free-choice or well-structured, we know that it cannot be sound. It is indeed possible to have a sound WF-net that is neither free-choice nor well-structured. For some more advanced routing constructs, non-free-choice nets and/or non-well-structured nets are inevitable. Notwithstanding these observations, in many practical workflows, non-free-choiceness or non-well-structuredness are signs of design errors, as explained in some more detail below.

5.1.4.1. Confusions The diagnostic information that Woflan provides on the free-choice property is the set of so-called *confusions*. A confusion is a non-free-choice cluster, where a cluster is a connected component of a net that remains after all arcs from transitions to places are removed from the net. A cluster is non-free-choice iff it does not satisfy the free-choice property of Definition 2.8. An example of a non-free-choice cluster is shown in Figure 17.

Two transitions that do not satisfy the free-choice property have different presets that are not disjoint. In a workflow context, this means that two tasks share some but not all preconditions. Usually, tasks that share a precondition start alternative branches: They form an OR-split. Also, a task that has multiple preconditions (note that at least one of the transitions has multiple preconditions) usually ends a set of parallel branches: It is an AND-join. A non-free-choice cluster is therefore often a mixture of an OR-split with an AND-join (see Figure 17). The OR-split is troubled by such an AND-join, because one alternative may be enabled while the other is not. The AND-join is troubled by the OR-split, because a fulfilled parallel branch may get unfulfilled before the AND-join is enabled. If possible, the OR-split and AND-join must be separated. The routing of a case should be independent of the order in which tasks are executed.

As explained in Section 4.3, most of the workflow management systems available at the moment abstract from states between tasks which means that process definitions imported from these workflow systems yield, in principle, free-choice WF nets. Clearly, the search for confusions is only meaningful for workflow management systems that allow non-free-choice constructs.

5.1.4.2. Mismatches A good workflow design is characterized by a balance between 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. From a workflow point of view, the situations as depicted in Figure 18 are suspicious.

In the leftmost situation, an AND-split is terminated by an OR-join. Tasks of a case are executed in parallel, but fulfilling one branch implies that both branches are fulfilled. The condition corresponding to place `P` can even be fulfilled twice. In a workflow, such a condition is often an error. In P/T-net terminology, this means that usually all places of a WF net should be safe. Note that this kind of error may lead to unboundedness of the short-circuited system and hence to unsoundness.

In the rightmost situation, an OR-split is terminated by an AND-join. One of the alternative tasks will be executed for the case. However, the task corresponding to transition `T` synchronizes both branches and needs both its preconditions to be fulfilled; it will never be executed. Note that this kind of error may lead to a non-live short-circuited system and hence to unsoundness.

Both situations depicted in Figure 18 describe a so-called non-well-handled pair: A transition-place or place-transition pair with two disjoint paths leading from one to the other. The leftmost situation describes a TP-handle, the rightmost a PT-handle (see Definition 2.6). Recall from Definition 4.3 that a WF net is well-structured iff the short-circuited net is well-handled (see Definition 2.7). Although a non-well-handled pair in the short-circuited net is often a sign of potential errors, a WF net that is not well-structured can still be sound.

The diagnostic information that Woflan provides is a list of all non-well-handled pairs in the short-circuited net; usually, the subset of non-well-handled pairs fully embedded in the non-short-circuited net (i.e., both paths between the two nodes of the pair do not contain the short-circuiting transition) provides the most useful information, because they often correspond to the undesirable AND-OR and OR-AND mismatches discussed above.

At this point in the diagnosis method, there are several possibilities. Quite often, the combination of a number of places not covered by a thread of control (Step 3) and information on confusions plus AND-OR / OR-AND mismatches reveals one or more errors in the process definition. (Note that, theoretically, the workflow process definition may still be sound.) Thus, the workflow designer might decide to end the diagnosis process, to correct the process definition in the workflow tool being used to design the workflow, and to restart the diagnosis process on the new process definition (the Quit-option of Step 4 in Figure 16). In other occasions, the designer may decide to continue the diagnosis process, even if it is already known that the workflow process definition cannot be sound (based on Theorems 4.2 and 4.3, as explained above).

5.1.5. *Step 5: Uniform invariant cover?*

A uniform invariant is a (semi-positive) place-invariant with only weights zero and one. Uniform invariants of a WF net can in general be computed efficiently, although it requires theoretically in the worst-case exponential space. Such place-invariants can provide useful information concerning the proper-completion property of a WF net. As mentioned before, the net of Figure 1 has a place-invariant $\mathtt{i}+\mathtt{c1}+\mathtt{c3}+\mathtt{c5}+\mathtt{o}$. Because we know that initially there is one token in place `i` and upon completion there is one token in `o`, we conclude from this invariant that `c1`, `c3`, and `c5` are empty upon completion. Furthermore, we can deduce from Theorem 4.5 (semi-positive place-invariants vs. boundedness) that a short-circuited WF system is bounded if all places are covered by uniform invariants. A place that is not covered by a uniform invariant *might* be unsafe or even unbounded. From a workflow point of view, this means that a condition might be fulfilled more than once at a single point in time, which is often undesirable. Note that this check is less discriminating than the check for S-coverability (Step 3): Every S-component corresponds to a uniform invariant. Thus, every place belonging to an S-component is covered by a uniform invariant. However, a place that does not belong to any S-component might still be covered by a uniform invariant.

The diagnostic information that Woflan provides in this step is the set of uniform invariants of the short-circuited WF net as well as the places that are not covered by these invariants. If all places are covered, the Proper-WPD milestone has been achieved.

5.1.6. *Step 6: Weighted invariant cover?*

Another structural technique for deciding boundedness of the short-circuited WF net is simply the check whether all places in the net are covered by some semi-positive place-invariant (thus allowing weights greater than one when compared to the previous step). Semi-positive place-invariants are simply called *weighted* invariants in Woflan. Clearly, this check is less discriminating than the check performed in the previous step. Places that are not covered by a weighted invariant might be unbounded. From a workflow point of view, this means that a condition *might* be fulfilled an arbitrary number of times.

The diagnostic information that Woflan provides in this step is (a representation of) the set of weighted invariants of the short-circuited WF net as well as the places that are not covered by these invariants. If all places are covered, the Proper-WPD milestone has been achieved.

5.1.7. *Step 7: No improper conditions?*

At this point in the diagnosis process, we have indications that some places of the short-circuited system *might* be unbounded. In Woflan, unbounded places are referred to as improper conditions. An improper condition in the short-circuited system always indicates a soundness error (related to improper completion; see also Sections 4.4.2 and 4.4.5). To determine improper conditions, Woflan computes the MCG (Minimal Coverability Graph [21]) of the short-circuited system. This computation can be time and space consuming, but it turns out that computing the MCG is feasible for most practical workflows (up to several hundreds of tasks). (Particularly for workflows corresponding to *bounded* short-circuited WF systems the computation does not take very long.)

The diagnostic information provided by Woflan consists of the set of improper conditions. If this set is empty, the Proper-WPD milestone has been achieved.

5.1.8. *Step 8: No substates?*

A substate of a system is a reachable marking M such that there is another reachable marking M_1 with $M < M_1$. It is not difficult to see that a *bounded* short-circuited WF system with substates cannot be live. Assume M is a substate of such a system with M_1 a marking reachable from the initial marking such that $M < M_1$. (Note that M_1 cannot be reachable from M, because this would contradict the boundedness of the system (see Definition 2.19).) It is impossible to reach marking $[o]$ from substate M, because otherwise we could reach $[o] + M_1 - M$ from M_1 which by Theorem 4.11 (Boundedness of short-circuited WF systems) contradicts the boundedness assumption. Since the short-circuiting transition has o as its only precondition, this transition cannot be live, which implies that also the short-circuited system cannot be live. The MCG algorithm that is used for computing improper conditions in the previous step allows the easy detection of substates (see [21]). The current version of Woflan provides a warning if a bounded short-circuited system has substates; it does not provide any detailed information about substates because this information is rather technical.

5.1.9. *Step 9: Improper scenarios!*

If the set of improper conditions in Step 7 of the diagnosis process is not empty, we know that the short-circuited WF system is unbounded. In case the set of improper conditions provides insufficient information for diagnosing the error(s), Woflan offers the workflow designer the possibility to compute the unbounded sequences of the WF system, called improper scenarios in Woflan.

As explained in Section 4.4.5, unbounded sequences are computed by constructing and partitioning an RCG of the WF system. Recall that it is not possible to use the MCG for this purpose (see [21]). It is not difficult to see that sequences that are

- permutations of the same set of transitions and
- end with the same transition

all provide the same diagnostic information. Thus, it suffices to consider only a single sequence of such a set. In order to minimize the set of improper scenarios presented to the workflow designer, Woflan computes a spanning tree of the RCG. A spanning tree of a graph is a connected subgraph in the form of a tree that contains all the nodes. The tree-constraint means that between every two nodes there is exactly one undirected path. A spanning tree of an RCG can be constructed in a straightforward way during the construction of the RCG. In the RCG of Figure 15, for example, the thick arcs denote a spanning tree. If Woflan is applied to our running example, it computes precisely the partitioned RCG of Figure 15 with the visualized spanning tree. Using this tree, it presents the two unbounded sequences given in Section 4.4.5 for this example.

Since at this point in the diagnosis process we know that the short-circuited system is unbounded and, hence, that the Proper-WPD milestone cannot be achieved, the workflow designer must make a correction to the workflow process definition and restart the diagnosis process with this corrected process definition.

5.1.10. Step 10: No dead tasks?

At some point during the diagnosis, the Proper-WPD milestone has been achieved, possibly after one or more corrections to the original process definition have been made. It remains to establish the third milestone of the diagnosis process. Recall that this part of the process is aimed at analyzing the liveness of the short-circuited WF system.

Using the MCG of the short-circuited WF system, Woflan provides the set of dead transitions of this system. Recall that Theorem 4.8 (Dead transitions in bounded short-circuited WF systems) implies that this set is precisely the set of dead transitions of the non-short-circuited system. These transitions correspond to dead tasks in the workflow process. Note that the MCG might already be available from Step 7 (No improper conditions?) of the diagnosis process; if this is not the case the MCG is computed at this point. If the WF system has dead tasks, the workflow designer must correct the error(s) and restart the diagnosis process with the new process definition.

5.1.11. Step 11: No non-live tasks?

At this point in the diagnosis process, we know that the short-circuited WF system is bounded and that it does not have any dead transitions. Woflan computes the OG of the short-circuited system to determine the set of non-live tasks, which it presents to the workflow designer. If all tasks are live, the diagnosis process is complete and successful: It has been shown that the short-circuited WF system is bounded and live which by Theorem 4.1 implies that the underlying WF net is sound.

5.1.12. Step 12: Non-live tasks!

At this point in the diagnosis process, we know that the short-circuited WF system is bounded, that it does not have any dead transitions, but that it is not live. Also in this case Woflan computes the set of non-live tasks via the OG of the short-circuited system.

5.1.13. Step 13: Locking scenarios!

If the result of Step 11 or Step 12 indicates that there are non-live transitions, but if this information is not sufficient for diagnosing the error(s), Woflan provides the option to compute the non-live sequences of the WF system. In Woflan, non-live sequences are referred to as locking scenarios (because they generally lead to livelocks and/or deadlocks in the workflow process). The set of locking scenarios is computed from the OG of the WF system (see Section 4.4.4) and minimized via a spanning tree of the OG. As in Step 9 (Improper scenarios!) of the process, the reason for minimizing the set of scenarios presented to the workflow designer is that non-live sequences being permutations of the same set of transitions and ending with the same transition provide the same diagnostic information.

5.1.14. Step 14: End of diagnosis

The diagnosis process ends with one of three possible conclusions, namely that the imported process definition does not correspond to a WF net, that it does correspond to a WF net but is not sound, or that it corresponds to a sound WF net. In case of errors, the process definition must be corrected in the workflow tool being used (see Section 3.3, Figure 11), after which the diagnosis process has to be restarted.

5.2. Diagnosing the example net

In this subsection, we diagnose the example workflow process illustrated in Figure 1 in Woflan. Following the approach explained in Section 3.3, Figure 11, we used Protos as the front-end workflow tool for designing and correcting the process definition (Protos can be used as the design tool for, e.g., COSA, Flower, and ECHO). As an alternative, we could also have chosen CONE, the design tool of COSA. Both tools support a modeling language that is sufficiently expressive for modeling arbitrary P/T nets.

Figure 19 shows a Protos model of the example workflow process. Note that we have modeled the two choices in the process via tasks `evaluate` and `check`, as explained in Section 3.3. Figure 20 shows a number of Woflan dialogs for the various steps of the diagnosis process of Figure 16.

The upper dialog in Figure 20 shows the information provided by Woflan when importing our Protos process definition. (Protos definitions are imported via the COSA import facility, which clarifies the title of the dialog window.) Using this dialog the workflow designer can preview

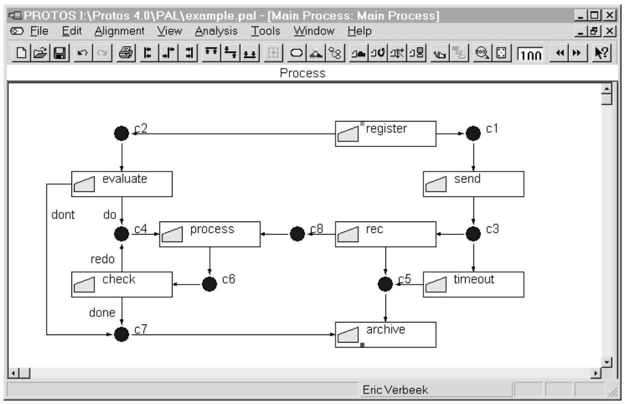

FIGURE 19. The Protos specification of the process of Figure 1

the P/T net resulting from the conversion before it is analyzed. In this case, the information is a straightforward list of conditions and tasks. Note that Woflan reports a task named `check_0`. As explained, Woflan splits choices into a number of tasks corresponding to the possible outcomes of a choice. In this case, Woflan splits task `check` into `check_0` and `check_1`, and task `evaluate` into `evaluate_0` and `evaluate_1`. These four new tasks correspond to tasks `done`, `redo`, `dont`, and `do` of Figure 1, respectively. The information provided by Woflan during the conversion may vary depending on the workflow tool being used. If Staffware is used, for example, some errors in the process definition may already be detected during the conversion. The reason is that Staffware uses a proprietary modeling language of which the mapping to WF nets is nontrivial (see [9]). In the next section, we briefly return to this point when discussing the Staffware case study.

The second dialog in Figure 20 is the Workflow-Process-Definition dialog that corresponds to Step 2 of the diagnosis process of Figure 16. It clearly shows that the net is a workflow process definition (i.e., a WF net).

The third dialog corresponds to Step 3 (Thread of control cover?) of the diagnosis process. It lists two threads of control, corresponding to the two S-components shown in Figure 6, and one condition that is not covered, namely condition `c8`. This information indicates that there might be a problem with `c8`; it may be improper (unbounded).

Because not all conditions are covered by threads of control, the diagnosis process continues with Step 4 (Confusions and mismatches?). The corresponding dialog is also shown in Figure 20. This dialog shows that our example net is unsound: Either condition `c8` needs to be covered by a thread of control or a confusion needs to be introduced somewhere.

It may be worthwhile to consider the mismatches at this point. Woflan indicates that the short-circuited net has four OR-AND mismatches and five AND-OR mismatches. One of the OR-AND mismatches is fully embedded in the non-short-circuited net and corresponds to the PT-handle shown in Figure 4; Woflan marks this OR-AND mismatch with the label ‘local’. Two of the AND-OR mismatches are local to the non-short-circuited net and correspond to the TP-handles of Figure 5. Unfortunately, in this example, it is not straightforward to derive any useful information from these mismatches other than the already known fact that condition `c8` is probably the cause of the unsoundness.

Steps 5 and 6 of the diagnosis process that compute uniform and weighted invariants, respectively, do not provide any additional information. In both cases, it turns out that condition `c8` is uncovered.

Step 7 (No improper conditions?) provides us with the definite information that condition `c8` is improper. Step 9 (Improper scenarios!) yields two improper scenarios, as shown in the dialog in Figure 20. Both scenarios result in the marking [`c5`, `c7`, `c8`]. (Recall that `evaluate_0` corresponds to transition `dont` of Figure 1. Executing task `archive` at that point results in marking [`c8`, `o`], which corresponds to improper completion.

At this point in the diagnosis, we have to make a correction. Clearly, the diagnostic information obtained so far suggests that transition `archive` must remove a token from `c8`. We correct the process definition in our modeling tool Protos (see Figure 11) by adding an arc between condition `c8` and task `archive`. The resulting process definition is not shown, but it corresponds to the WF net of Figure 12 (assuming the appropriate renamings as explained above).

We restart the diagnosis process on the new process definition. In Steps 1 through 6, Woflan provides the following diagnostic information. The process definition is still not covered by threads of control or invariants; in all cases, condition `c8` is still uncovered. However, the process definition is also not free-choice and not well-structured. Thus, it might still be sound. Step 7 (No improper conditions?) shows that the process definition is proper. Thus, our correction in the first iteration of the diagnosis process has been an improvement.

It turns out that the process definition has no substates and no dead tasks (Steps 8 and 10 of the diagnosis process; Step 9 is skipped in this iteration). However, Woflan reports in Step 11 that all tasks are non-live (in the short-circuited system). At this point, we know that the process definition is not sound. Unfortunately, the information is not sufficiently specific for diagnosing the error(s). Thus, we let Woflan compute the locking scenarios of the process definition (Step 13). Woflan reports the five scenarios already presented (as non-live sequences) in Section 4.4.4. From the discussion in that section, we may conclude that the execution of task `timeout` is the probable cause of an error and that also the cycle consisting of tasks `process` and `check` (option `redo`) is very likely the cause of a problem. A closer look at the workflow process definition reveals that there are indeed two problems. First, the execution of task `timeout` does not mark condition `c8`, which means that tasks `process` and `archive` cannot be executed after `timeout` is executed. To correct this error, we add an arc from `timeout` to `c8`. Second, the cycle consisting of tasks `process` and `check` can only be executed once, because `c8` is only an input condition (and not an output condition) of the cycle. We correct this error by adding an arc from `process` to `c8`.

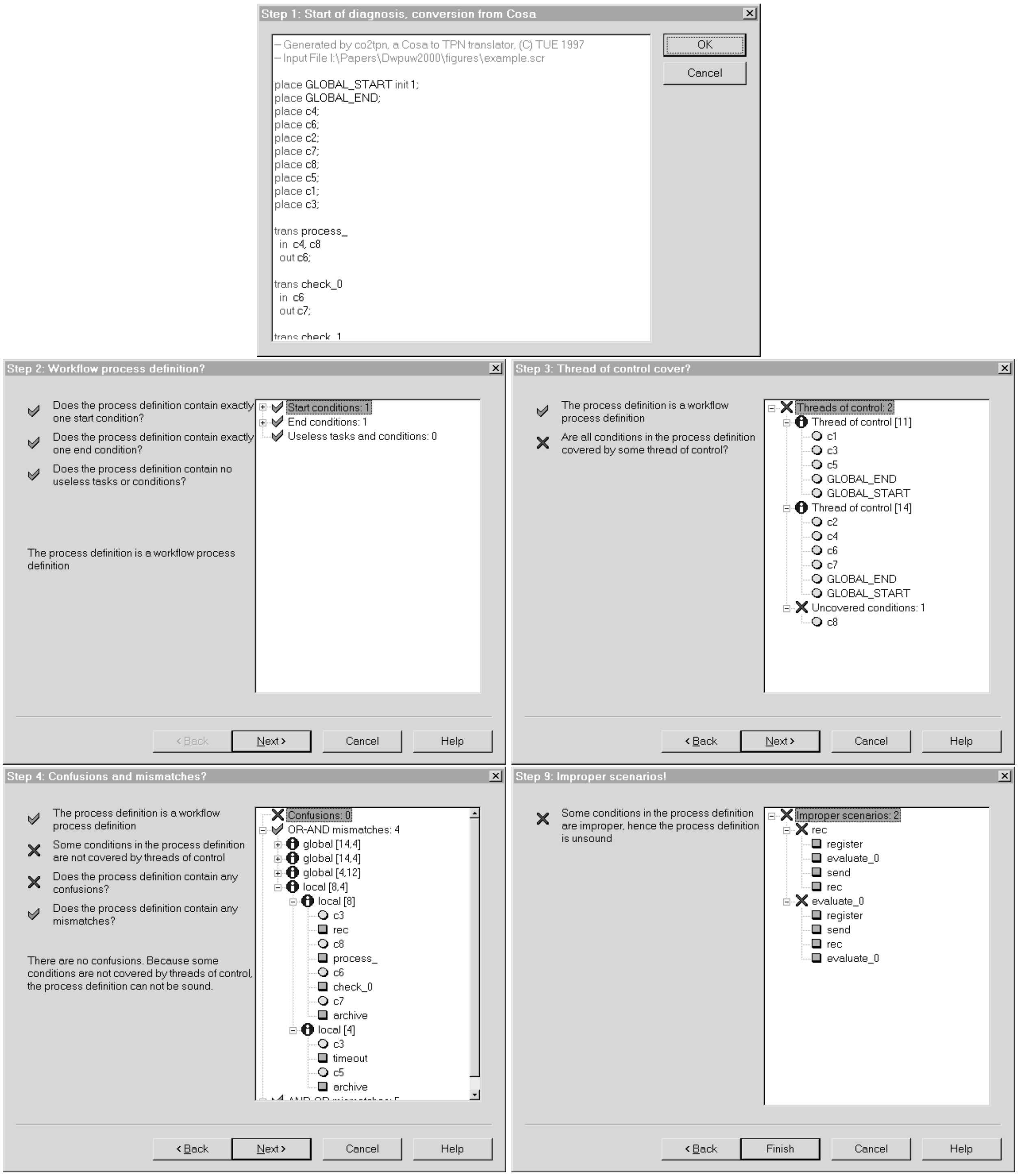

FIGURE 20. Example diagnosis, dialogs

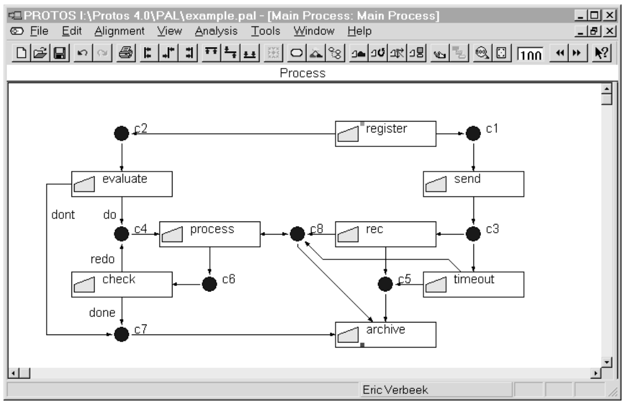

FIGURE 21. A sound version of the example process definition

After adding the two arcs mentioned above in our Protos model, the process definition looks as in Figure 21. A third iteration of the diagnosis process shows that the process definition is sound. (The iteration goes via Steps 1, 2, 3, 10, 11, and 14, which is the shortest path through the diagnosis process.) Note that the process definition of Figure 21 is not free-choice (cf. tasks `process` and `archive`). Consequently, this process definition is only feasible when using workflows tools as Protos or COSA. Staffware, for example, does not allow the non-free-choice construct in the process definition. It is important to note that corrections to process definitions may depend on the workflow system at hand. When we would have used Staffware for designing our workflow process, we would have had to think of another way to correct the errors. (It is an interesting exercise to come up with a free-choice variant of the process definition of Figure 21.)

6. CASE STUDIES

To evaluate the applicability of Woflan, we performed two case studies, one focusing on the usefulness of all the steps in the diagnosis process of Figure 16 supported by Woflan and another one testing the applicability of Woflan and the approach of Figure 11 on a workflow process developed in a real-world context.

For testing the usefulness of the steps of the diagnosis process of Figure 16, we used seventeen Protos models of the workflow process of a travel agency at a university. These seventeen models were chosen from the work of twenty groups of students that designed Protos models from an informal description of the workflow process, as part of an assignment for a course on workflow management. There are two reasons why this case study is particularly useful for evaluating the diagnosis process of Woflan. First, Protos supports P/T nets as a modeling language. Consequently, all steps in the diagnosis process of Woflan may in principle provide useful information concerning possible errors in workflow process definitions designed in Protos. Second, the assignment was set up in such a way that the students had to use a wide variety of routing constructs in their models. By evaluating seventeen models of this workflow process, it is almost guaranteed that these models also contain a wide variety of errors.

For testing our approach to workflow verification on a real-world example, we cooperated with Staffware Benelux. We set up an experiment where a workflow designer of Staffware Benelux introduced a number of non-trivial errors in a large workflow that was known to be correct. We were not familiar with the workflow process. Also, the type of errors was not known to us and neither did we know the total number of errors. The reason for choosing Staffware, instead of for example COSA, is that Staffware supports a proprietary modeling language of which the mapping onto P/T nets is non-trivial. Thus, this case study is a real test of the approach illustrated in Figure 11, in particular of the interpretation of the diagnostic information provided by Woflan in the Staffware model.

In the remainder of this section, we discuss the results of both case studies in some detail.

6.1. Protos case

The input for this case study consisted of workflow process definitions developed by 42 industrial-engineering students of the course *Workflow Management & Groupware* (1R420; Eindhoven University of Technology) and 15 computing-science students of the course *Workflow Management: Models, Methods, and Tools* (25756; University of Karlsruhe). These students formed 20 groups which independently designed Protos [31] models of the workflow in a travel agency. Fourteen of these groups consisted of students from the Eindhoven University of Technology; the other six consisted of students of the University of Karlsruhe.

From the Eindhoven collection of models, we selected eleven reasonably looking solutions; three models were so poor that analyzing them by means of Woflan was not very meaningful. From the Karlsruhe collection, all models were selected. The number of tasks and other building blocks of the models ranged from 54 to 89. These numbers show that the case study was performed on workflow models of more than reasonable size. A snapshot of a(n unsound) Protos model of the travel-agency workflow is shown in Figure 22.

The groups of Eindhoven consisted of industrial engineers, which had only a little prior experience in modeling and no background in formal verification. Verification of workflows was only a minor topic of the course *Workflow Management & Groupware* (1R420) and the students did not practice with Woflan. Although the groups were told to simulate the workflow process by hand (play the *token game*) to test their model, not one of them was able to produce a sound model.

In contrast to the groups of Eindhoven, the groups taking the course *Workflow Management: Models, Methods, and Tools* (25756) in Karlsruhe consisted of computing-science engineers, which did have a background in modeling and verification. Furthermore, the importance of making a correct workflow was emphasized and analysis techniques for P/T nets and WF nets were treated in the course. In addition, they practiced with a prior version of Woflan on small ex-

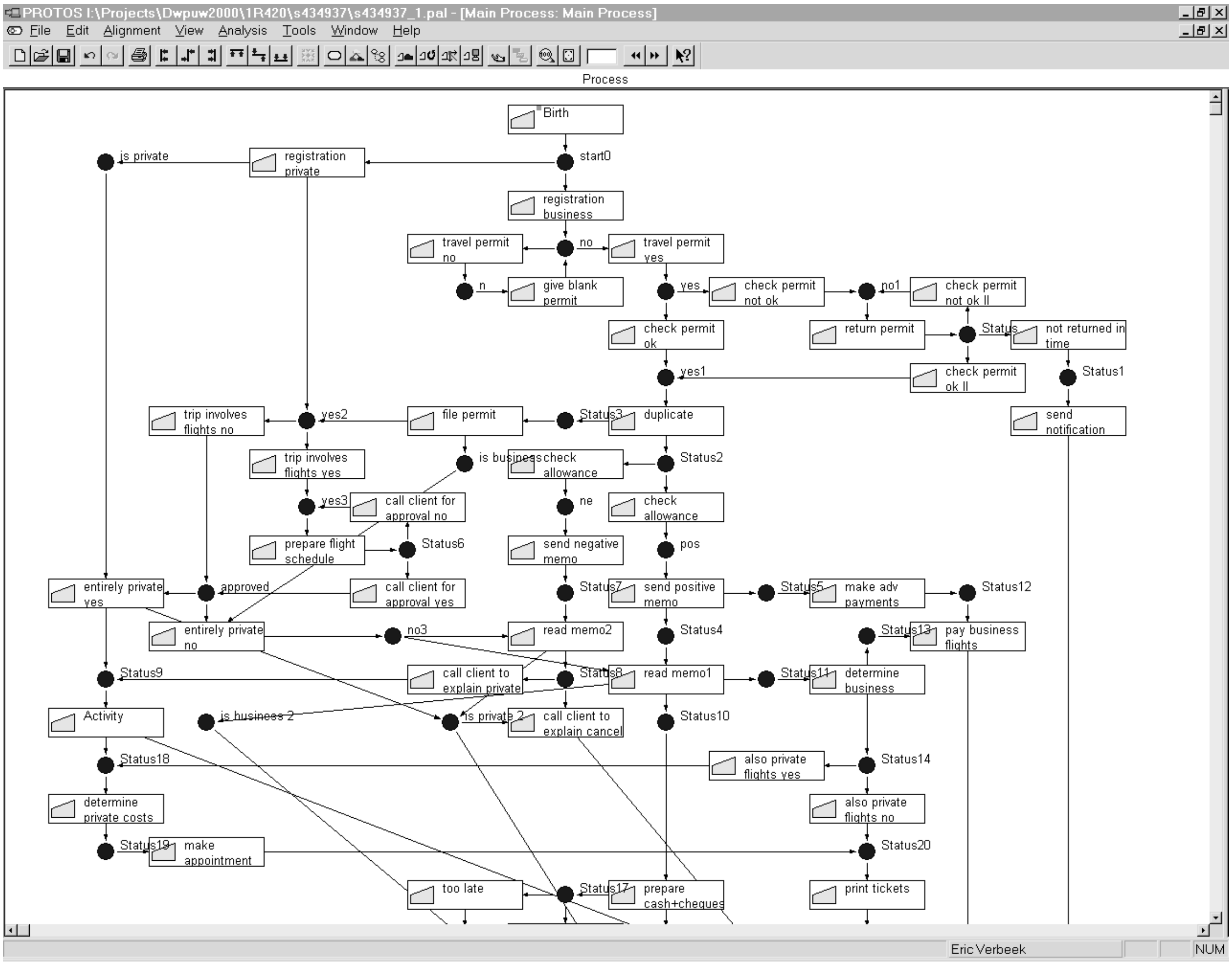

FIGURE 22. A snapshot of a Protos model of the travel-agency workflow

amples. However, none of the groups used Woflan to check their solution to the assignment. In the end, the Karlsruhe groups delivered better models than the Eindhoven groups. Of the seventeen models we analyzed with Woflan, five appeared to be sound, all from Karlsruhe groups.

Table 1 shows an overview of our efforts to diagnose the seventeen workflow models. It contains the following information:

- The number of iterations with Woflan needed to produce a sound workflow process definition.
- Diagnostic information (see below for more details).
- The estimated time it took us to produce a sound workflow process definition.

The case study was performed on a Pentium 200 PC with 128 Mb of RAM running Windows NT 4.0.

The numbers in the column of Table 1 containing diagnostic information refer to the corresponding steps of the diagnosis process of Figure 16. An entry implies that, based on the information provided in that step, a correction was made in the model being diagnosed. In case a correction was made in Step 4, it is specified whether this correction was based on a confusion or on a mismatch. The entries are simply given in increasing order; the corrections are not necessarily made in that order. For example, when diagnosing the model of group 3, four corrections based on Step 4 were made in the initial model, one correction based on Step 13 was made in the second model, and one correction based on Step 3 was made in the third model.

The information in Table 1 shows that Steps 3 (Threads of control cover?), 4 (Confusions and mismatches?), 7 (No improper conditions?), 9 (Improper scenarios!), 10 (No dead tasks?), and 13 (Locking scenarios!) of the diagnosis process of Figure 16 are all used to make one or more corrections. In particular Steps 3, 4, 9, and 13 are used quite often. To us, this does not come as a surprise because the diagnostic information provided in these steps has a clear interpretation in the workflow domain.

Of course, it is also interesting to see which steps are not used. All Protos models considered in the case study corresponded to workflow process definitions. Consequently, no corrections were made in Step 2 of the diagnosis pro-

Group	**Iterations**	**Diagnosis**	**Time** (min)	**University**
1	2	4 (mism)	5	Eindhoven
2	9	4 (mism: 4×), 7 (3×), 9, 13 (2×)	90	Eindhoven
3	4	3, 4 (mism: 4×), 13	30	Eindhoven
4	8	3, 4 (mism: 12×), 13	75	Eindhoven
5	3	3, 4 (conf: 1×; mism: 6×)	30	Eindhoven
6	3	3 (3×)	30	Eindhoven
7	7	3 (2×), 4 (conf: 1×; mism: 8×), 9	60	Eindhoven
8	3	3 (2×)	20	Eindhoven
9	2	4 (mism: 4×)	20	Eindhoven
10	2	3	5	Eindhoven
11	7	3 (2×), 4 (conf: 2×), 9 (3×), 10	50	Eindhoven
12	1	sound	< 5	Karlsruhe
13	1	sound	< 5	Karlsruhe
14	2	13	5	Karlsruhe
15	1	sound	< 5	Karlsruhe
16	1	sound	< 5	Karlsruhe
17	1	sound	< 5	Karlsruhe

TABLE 1. Overview of the results of the travel-agency case study

cess. However, this step is essential in the process because the WPD milestone guarantees that the remainder of the diagnosis process is meaningful. The information in Table 1 furthermore shows that Steps 5, 6, 8, 11, and 12 were not used to make corrections. However, in one occasion (Group 11; final model), Step 5 (Uniform invariant cover?) showed that the process definition was proper; interestingly, that process definition was not covered by threads of controls, which is usually the case. Step 11 (No non-live tasks?) is simply required in the diagnosis process for showing soundness of a workflow process definition. Nevertheless, the results of the case study show that a list of non-live tasks is generally not sufficient for diagnosing an error; in all relevant cases, locking scenarios (Step 13) were computed to obtain more detailed information. Further experience with Woflan might point out that Steps 11 and 13 can be integrated. For similar reasons, also Step 12, which is simply a variant of Step 11, might be integrated with Step 13. This leaves Steps 6 (Weighted invariant cover?) and 8 (No substates?). These steps are usually only relevant if the process definition is non-safe (see Definition 2.19). In practice, this is rarely true. However, both steps might turn out to be useful in these rare occasions and, furthermore, come almost for free after Steps 5 and 7, respectively.

Besides the above observations about the usefulness of the steps in the diagnosis process of Woflan, two other interesting observation can be made. In the informal description of the travel-agency workflow process, a distinction was made between private trips and business trips. At several points in the process, the execution of certain tasks or the order of execution depended on this distinction. Consequently, a workflow process definition of the travel-agency process almost always contains a number of choices (OR-splits) that must be kept consistent. In several models used for the case study, this consistency was not enforced by the workflow process definition. The type of a trip is a typical example of a piece of control data (see Section 3.2.1). As mentioned in Section 3.2.3, in our opinion, one should avoid situations where the logical correctness of a process definition depends on the invariance of a piece of control data. Fortunately, the diagnostic information provided by Woflan made it straightforward to correct these models enforcing the consistency via the process definition.

Another interesting observation is that the industrial-engineering students of Eindhoven did not produce a single correct workflow, whereas the computing-science students of Karlsruhe handed in only one flawed model, which was straightforward to correct. In our opinion, the different background of the students causes this discrepancy. Industrial-engineering students have little background in modeling and verification; computing-science students are trained in both skills. Many designers of workflow processes in practice have also little experience in formal verification. Woflan can be a useful aid in designing correct workflow processes that helps to prevent a lot of problems caused by the implementation of erroneous workflow processes.

Summarizing the results of the travel-agency case study, Woflan proved to be useful for diagnosing and correcting all the seventeen models in reasonable time and with reasonable effort. The results indicate that the diagnosis process of Figure 16 is appropriate for verifying complex workflow processes.

6.2. Staffware case

As explained in the introduction to this section, we set up an experiment in cooperation with Staffware Benelux to test our approach on a real-world workflow process. The starting point of the case study was a complex process of 114 tasks and other building blocks (wait steps, complex routers, etc.), developed by Staffware Benelux using Staffware 2000 [43]. The model contained a number of errors that were not known to us in advance, but that were known to Staffware Benelux. We diagnosed the Staffware model with Woflan 2.1, corrected the Staffware model, and discussed our diagnosis results with Staffware Benelux. It turned out that we

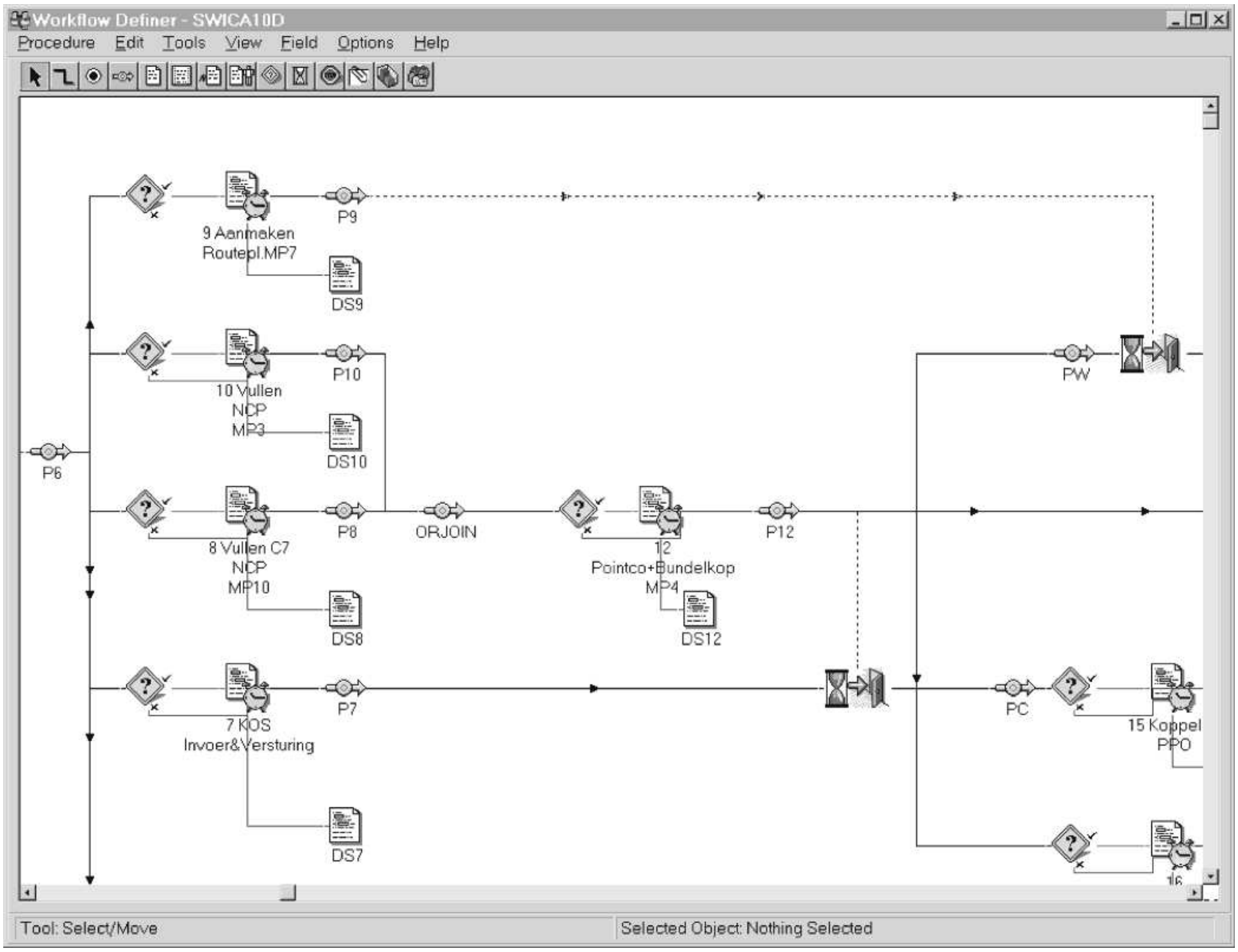

FIGURE 23. A snapshot of the Staffware process definition

found six out of seven errors in the process definition. Another positive result is that the corrections we made proved to be the appropriate ones. The error that we did not find was lost in the conversion from Staffware to Woflan. As already mentioned, the mapping from Staffware models to P/T nets is non-trivial. Apparently, the error was lost in the abstraction (see Section 3.2) applied during the conversion. However, in our discussion with Staffware Benelux after the completion of the case study, it turned out that there is a straightforward check that can be incorporated in the conversion process to detect the type of error that we missed. In the remainder of this subsection, we discuss the conversion of Staffware models to P/T nets and the results of the case study in some more detail. Figure 23 shows a snapshot of the (unsound) Staffware model.

6.2.1. Conversion

Two important aspects of the Staffware modeling language play a role when converting Staffware models to P/T nets and, in particular, to WF nets. The first one has already been mentioned before. The Staffware modeling language abstracts from states in a workflow process. The second one is that Staffware models do not necessarily have a single point of exit. Staffware models may diverge in several independent branches. A Staffware case is completed if all branches are completed. These two aspects have consequences for the application of Woflan to Staffware models.

To start with the second aspect, the problem is to map the notion of completion used in Staffware onto our notion of completion. In [9], a solution to this problem is given. Essentially, the approach of [9] means that a standard P/T-net construction is used to detect the completion of all the branches in the Staffware model. The most important consequence of this construction is that the resulting P/T net is always bounded and almost always a WF net.[2] Consequently, the first milestone of the diagnosis process discussed in Section 5.1 is almost always satisfied by a Staffware model and the second one is always satisfied, possibly hiding some

[2]The translation proposed in [9] results in a P/T net which may have multiple arcs between pairs of nodes. However, multiple arcs can only occur in the special completion-detection construct. Furthermore, the results presented in this paper extend in a straightforward way to P/T nets allowing multiple arcs between pairs of nodes and also Woflan can cope with such nets.

errors related to the structure of the process (WPD milestone) or to improper completion (Proper WPD milestone). As already mentioned before, a consequence of the first aspect mentioned above is that a WF net corresponding to a Staffware model is, in principle, free-choice. However, as already mentioned, Staffware allows one particular construct that cannot be mapped onto a corresponding free-choice P/T-net construct. Furthermore, the construct for detecting successful completion is generally not free-choice.

It may be clear that the above observations have implications for the diagnosis process supported by Woflan. In particular, we have to be careful with the interpretation of the diagnostic information provided by Woflan.

Step 1 (Start of diagnosis) During the conversion from Staffware to Woflan, diagnostic information on the structure of the process is generated. In the current version of the conversion, this information focuses on the connectedness of the model.

Step 2 (Workflow process definition?) As already mentioned, the P/T net resulting from the conversion is almost always a WF net. In some rare occasions, this may not be true; in such a case, the information provided by Woflan can be used to correct the error.

Step 3 (Threads of control cover?) The abovementioned construction for detecting completion introduced during the conversion implies that the WF net is generally not covered by threads of control. However, the diagnostic information provided by Woflan in this step can still be useful.

Step 4 (Confusions and mismatches?) Most likely, the WF net resulting from a Staffware model has only one confusion, which is the result of the construction for detecting completion. Also many of the mismatches in the net are often caused by this special construction. Mismatches that are inherent in the original Staffware model are identified by Woflan and may, of course, still provide useful information.

Step 5 (Uniform invariant cover?) The conversion is such that the WF net is generally not covered by uniform invariants.

Step 6 (Weighted invariant cover?) The completion-detection construction guarantees that the WF net is covered by weighted invariants. (For this reason, the Proper-WPD milestone is always satisfied.)

Steps 7, 8, 9, and 12 These steps are always skipped (because of the outcomes in the earlier steps).

Steps 10, 11, 13, and 14 These steps are unaffected.

At a first glance, the above list might seem to contradict our claim that Woflan is workflow-tool independent. However, note that the tool itself has not been changed in order to make it useful for analyzing Staffware models. The only programming effort was put into the conversion program.

Iterations	**Diagnosis**	**Time** (min)
3	1, 4 (mism: 3×), 13 (2×)	90

TABLE 2. The results of the Staffware case study

Furthermore, some items in the above list are just simplifications of the diagnosis process of Figure 16 that are not visible to users of Woflan; some other items explain how certain diagnostic information should be interpreted in terms of Staffware models. One could even argue that, despite the large differences between Staffware models and WF nets, a surprisingly large part of the diagnosis process and the provided diagnostic information is still relevant.

6.2.2. Diagnosis

In this paragraph, we discuss the actual diagnosis of the Staffware model used for this case study. Table 2 summarizes the results. The case study was performed on a Pentium III 500 PC with 256 Mb of RAM running Windows NT 4.0.

Three iterations were needed for the diagnosis, taking in total about one and a half hour. Given the size of the workflow process and the fact that we were not familiar with the process, in our opinion, this effort is reasonable. In the first two iterations, we found and corrected six (out of seven) errors; the third iteration showed that the model resulting from the first two iterations was sound.

During the first iteration, one error was detected during the conversion (Step 1: Start of diagnosis). A small part of the process definition was not connected to the main part. Furthermore, three structural errors were found and corrected via mismatches reported in Step 4 (Confusions and mismatches). An OR-join had to be replaced by an AND-join and two arcs had to be removed. The first error is visible in Figure 23: The complex router labeled `P6`, which acts as an AND-split, is (partly) complemented by the router labeled `ORJOIN`, acting as an OR-join. The latter should be replaced by a wait step, which acts as an AND-join.

In the second iteration, we did not find any more structural errors, but we did find two behavioral ones. The locking scenarios of Step 13 of the diagnosis process clearly indicated that the process contained two erroneous OR-splits. Both are visible in Figure 23. The first one is the choice (the diamond) just before the task labeled `9 Aanmaken Routepl.MP7`. If the choice has a negative result, the branch terminates. In this particular case, this implies an error because furtheron the synchronization via the wait step following complex router `PW` will fail. (This mistake might seem obvious given the three visually similar constructs also shown in the snapshot; however, recall that the total workflow consists of over 100 building blocks which makes it much harder to find the mistake simply via visual inspection.) The second erroneous OR-split is the step labeled `10 Vullen NCP MP3`. Note that this step is visually identical (!) to the step labeled `8 Vullen C7 NCP MP10` and two of the other steps shown in the snapshot. However, the scenarios reported by Woflan indicate that it is not. The erroneous step is disabled (withdrawn in Staffware terminology) in case of a timeout, thus causing a synchronization error furtheron. The timeouts associated with step `8 Vullen C7 NCP MP10` and the other similar steps do not disable the corresponding steps, but simply generate some kind of warning message.

The two errors found in the second iteration were straightforward to correct yielding a workflow process definition that was proved sound in a third iteration.

As already mentioned, we only found six out of seven errors in the original Staffware model, despite the fact that Woflan reports that the model resulting after the corrections described above is sound. The one error Woflan fails to diagnose is lost in the conversion. It concerns a type of error that may occur in the timeout construct of Staffware. As explained in Section 3, it is inherent to our approach that some errors are lost in the abstractions we apply, particularly if these errors are not or not closely related to the routing of cases. However, in this particular case, it is possible to incorporate a simple check in the conversion process to filter out this specific type of error. In fact, further experience might show that also other types of errors can be filtered out during the conversion of process definitions for use with Woflan. It is even possible that (some of) the conversion programs coupling Woflan with the various workflow products evolve into workflow-tool-specific extensions of Woflan for diagnosing errors that are specific for that particular workflow tool.

Summarizing, the main conclusion of this case study is that Woflan can be a useful aid for detecting and correcting errors in Staffware process definitions. The results support our belief that workflow-tool-independent verification as visualized in Figure 11 is feasible. Further experience is needed to optimize the interface between Staffware and Woflan.

7. RELATED WORK

Petri nets have been proposed for modeling workflow process definitions long before the term "workflow management" was coined and workflow management systems became readily available. An example is the work on Information Control Nets [17, 18], a variant of classical Petri nets, originally developed in the late seventies. For the reader interested in the application of Petri nets to workflow management, we refer to the two most recent workshops on workflow management held in conjunction with the annual International Conference on Application and Theory of Petri Nets [14, 6] and an elaborate paper on workflow modeling using Petri nets [2]. Only a few papers in the literature focus on the verification of workflow process definitions. In [24], some verification issues have been examined and the complexity of selected correctness issues has been identified, but no concrete verification procedures have been suggested. In [1], [4], and [10], concrete verification procedures based on Petri nets have been proposed. Woflan builds upon the techniques presented in [1, 4]. The technique presented in [10] has been developed for checking the consistency of transactional workflows including temporal constraints. However, the technique is restricted to acyclic workflows and only gives necessary conditions (i.e., not sufficient conditions) for consistency. In [37], a reduction technique has been proposed. This reduction technique uses a correctness criterion which corresponds to soundness and the class of workflow processes considered are in essence acyclic free-choice P/T nets. Some work on the compositional verification of workflows, using well-known Petri-net results such as the refinement rules in [44], can be found in [4, 5, 46].

As far as we know, only one other tool has been developed for verifying workflows: *FlowMake* [36]. FlowMake is a tool based on the reduction technique described in [37] and can interface with the IBM MQSeries Workflow product. FlowMake can only handle acyclic workflows and provides fewer diagnostics than Woflan: Only the reduced workflow graph is shown.

The work presented in this paper builds on previous research reported by the authors [1, 4, 8, 45]. The main contribution of this paper is a complete description of the latest version of Woflan, the diagnosis process it supports, and the techniques it is based on. The concept, computation, and application of behavioral error sequences have not been addressed in previous publications. Moreover, the experimental results have not been presented before.

8. CONCLUSIONS AND FUTURE WORK

Workflow-management technology is rapidly gaining popularity in the support of business processes. A thorough analysis of workflow processes before their actual implementation is necessary to guarantee effectiveness and efficiency. To guide a workflow designer in finding and correcting errors in a workflow process, we developed a diagnosis process and the accompanying tool Woflan, both based on Petri-net techniques. We have evaluated Woflan, version 2.1, in two case studies: one involving seventeen models of a fairly complex workflow designed by students in Protos [31] and one involving a large real-world workflow process designed in Staffware [43]. A novel analysis technique of behavioral error sequences proved to be a useful aid in diagnosing the workflows. The results are encouraging. They show that the diagnosis process supported by Woflan is useful and that our approach to workflow-product-independent verification of workflow processes is feasible. Nevertheless, we would like to evaluate Woflan and its analysis techniques in other experiments, in order to further optimize the diagnosis process.

We are also working on extending the set of workflow tools Woflan can interface with. The current version of Woflan (version 2.1) can import workflow process definitions from COSA, Staffware, METEOR, and Protos. On paper, we have also designed translations from BaanERP/DEM (BaaN), SAP/Workflow (SAP AG), and ARIS (IDS Prof. Scheer) to Woflan. The Dynamic Enterprise Modeler (DEM) of BaanERP is based on a subclass of Petri nets, which means that the translation is straightforward. SAP/Workflow and ARIS are both based on event-driven process chains. A translation of event-driven process chains to WF nets is described in [3]. In the future, we plan to build the corresponding interfaces.

Furthermore, we are looking into visualizing Woflan's output in a graphical way. The current interface is entirely textual. There are several ways for displaying the diagnostics in a graphical manner: either via diagrams shown directly by Woflan, via dedicated tools such as VIPtool [15], or via an interface in the workflow tool used to design the workflow process. The last option is clearly preferable from the viewpoint of interpreting the diagnostic information provided by Woflan in terms of the original workflow process definition. However, it also means that the workflow tool itself has to be extended. Any of the first two options might be a reasonable compromise between the amount of effort needed for realizing visual diagnostic information and ease of interpretation by workflow designers.

A direction for future research is the use of the inheritance-preserving transformation rules presented in [5] for incremental design and verification of workflows. Starting from a correct workflow template [30] or an already verified existing workflow process definition, these rules allow for safe extensions which preserve the soundness property. Correctness by design is obviously preferable over the approach where correctness is verified only after the design of the complete workflow has been completed.

As a final remark, note that Woflan can be helpful in the design and verification of correct workflow process definitions. However, this does not mean that the entire workflow is correct. It is still possible that errors are made in the implementation of the workflow process or that the process suffers bottlenecks in the performance due to a poor allocation of resources. To prevent such kinds of errors, other techniques are needed to complement Woflan.

ACKNOWLEDGEMENTS

The authors whish to thank Geert-Jan Houben, Marc Voorhoeve, Jaap van der Woude, and the anonymous referees for their useful comments. Furthermore, we are obliged to Edmar Kok of Staffware Benelux for providing us with the Staffware case and helping us out with it.

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, Proceedings*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Toulouse, France, June 1997. Springer, Berlin, Germany, 1997.

[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.

[3] W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[4] W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In Van der Aalst et al. [7], pages 161–183.

[5] W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. To appear in Theoretical Computer Science.

[6] W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors. *Workflow Management: Net-based Concepts, Models, Techniques, and Tools (WFM'98), Proceedings*, Lisbon, Portugal, June 1998. Eindhoven University of Technology, Eindhoven, The Netherlands, Computing Science Report 98/7, 1998.

[7] W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 2000.

[8] 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, *Petri Nets in System Engineering (PNSE'97), Proceedings*, pages 78–90, Hamburg, Germany, September 1997. University of Hamburg, FBI-HH-B-205/97, 1997.

[9] W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of Workflow Task Structures: A Petri-net-based Approach. *Information Systems*, 25(1):43–69, 2000.

[10] N.R. Adam, V. Atluri, and W.K. Huang. Modeling and Analysis of Workflows using Petri Nets. *Journal of Intelligent Information Systems*, 10(2):131–158, March 1998.

[11] 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, Proceedings*, volume 935 of *Lecture Notes in Computer Science*, pages 25–44, Torino, Italy, June 1995. Springer, Berlin, Germany, 1995.

[12] E. Best. Fairness and Conspiracies. *Information Processing Letters*, 18:215–220, 1984.

[13] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.

[14] 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.

[15] J. Desel. Validation of Information Systems by Analyzing Partially Ordered Petri Net Processes. Technical report 375, AIFB, University of Karlsruhe, Karlsruhe, Germany, 1998.

[16] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

[17] C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, USA, 1979. ACM Press.

[18] 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, Proceedings*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16, Chicago, Illinois, June 1993. Springer, Berlin, Germany, 1993.

[19] J. Esparza and M. Nielsen. Decidability Issues for Petri Nets - A Survey. *Journal of Information Processing and Cybernetics*, 30(3):143–160, 1994.

[20] 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, Berlin, Germany, 1990.

[21] A. Finkel. The Minimal Coverability Graph for Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1993*, volume

674 of *Lecture Notes in Computer Science*, pages 210–243. Springer, Berlin, Germany, 1993.

[22] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[23] K. Hayes and K. Lavery. Workflow Management Software: The Business Opportunity. Technical report, Ovum Ltd, London, UK, 1991.

[24] 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.

[25] R.R.A. Issa and R.F. Cox. Using Process Modeling and Workflow Integration to gain (ISO 9000) Certification in Construction. In *CIB W89 Beijing International Conference on Construction, Modernization, and Education*, Beijing, China, 1996.

[26] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.

[27] M. Klein, C. Dellarocas, and A. Bernstein, editors. *Towards Adaptive Workflow Systems, CSCW-98 Workshop, Proceedings*, Seattle, Washington, November 1998. http://ccs.mit.edu/klein/cscw98/.

[28] T.M. Koulopoulos. *The Workflow Imperative*. Van Nostrand Reinhold, New York, USA, 1995.

[29] P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, USA, 1997.

[30] T.W. Malone, K. Crowston, J. Lee, B. Pentland et al. Tools for Inventing Organizations: Toward a Handbook for Organizational Processes. *Management Science*, 45(3):425–443, 1999.

[31] Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, The Netherlands, 1997.

[32] M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflows without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[33] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1985.

[34] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.

[35] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.

[36] W. Sadiq and M.E. Orlowska. FlowMake Product Information, Distributed Systems Technology Centre, Queensland, Australia. http://www.dstc.edu.au/Research/Projects/FlowMake/productinfo/index.html.

[37] W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In M. Jarke and A. Oberweis, editors, *Advanced Information Systems Engineering, 11th. International Conference, CAiSE'99, Proceedings*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209, Heidelberg, Germany, June 1999. Springer, Berlin, Germany, 1999.

[38] T. Schäl. *Workflow Management for Process Organisations*, volume 1096 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1996.

[39] A. Sheth. From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration. In R. Wagner, editor, *Database and Expert Systems Applications, 8th. International Workshop, DEXA'97, Proceedings*, pages 24–27, Toulouse, France, September 1997. IEEE Computer Society Press, Los Alamitos, California, USA, 1997.

[40] A. Sheth, K. Kochut, and J. Miller. Large Scale Distributed Information Systems (LSDIS) laboratory, METEOR project page. http://lsdis.cs.uga.edu/proj/meteor/meteor.html.

[41] M. Silva and R. Valette. Petri Nets and Flexible Manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424 of *Lecture Notes in Computer Science*, pages 274–417. Springer, Berlin, Germany, 1990.

[42] Software-Ley. *COSA 2.0 User Manual*. Software-Ley GmbH, Pullheim, Germany, 1998.

[43] Staffware. *Staffware GWD Procedure Definer's Guide, Version 8, Issue 2*. Staffware Plc, Berkshire, UK, 1999.

[44] R. Valette. Analysis of Petri Nets by Stepwise Refinements. *Journal of Computer and System Sciences*, 18(1):35–46, 1979.

[45] H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-Net-Based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000, Proceedings*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484, Aarhus, Denmark, June 2000. Springer, Berlin, Germany, 2000.

[46] M. Voorhoeve. Compositional Modeling and Verification of Workflow Processes. In Van der Aalst et al. [7], pages 184–200.

[47] WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, Belgium, 1996.

[48] M. Wolf and U. Reimer, editors. *Practical Aspects of Knowledge Management (PAKM'96), 1st. International Conference, Workshop on Adaptive Workflow, Proceedings*, Basel, Switzerland, October 1996.