Diagnosing workflow processes using Woflan

H.M.W. Verbeek[1], T. Basten[2]*, and W.M.P. van der Aalst[3]†

[1] *Dept. of Computing Science, Eindhoven University of Technology, The Netherlands*
wsineric@win.tue.nl

[2] *Dept. of Electrical Engineering, Eindhoven University of Technology, The Netherlands*
tbasten@ics.ele.tue.nl

[3] *Dept. of Computing Science, Eindhoven University of Technology, The Netherlands*
wsinwa@win.tue.nl

Abstract

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 new diagnostics to locate the source of a design error. Based on a case study (involving twenty groups of students designing a complex workflow process), these new diagnostics have been evaluated and the results have been used to develop a method to guide the user of Woflan in finding and correcting errors in the design of workflows.

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

Contents

*Part of this work was done while the author was employed at the Department of Computing Science of the Eindhoven University of Technology.

†Part of this work was done while the author was at sabbatical leave at the Large Scale Distributed Information Systems (LSDIS) laboratory of the University of Georgia.

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 [20, 22, 25, 26, 31]. A critical challenge for workflow management systems is their ability to respond effectively to changes [9, 13, 24, 29, 36]. 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. Even a simple change as adding a task can cause a deadlock or livelock. Creating or modifying a complex process that combines parallel and conditional routing is an activity subject to errors. Contemporary workflow management systems do not support advanced techniques to verify the correctness of workflow process definitions [5, 6, 23]. 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 [21]. Clearly, rigorous verification of workflow processes can be used as a tool to ensure certain levels of quality.

The development of *Woflan* 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 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, 4, 5]. This notion expresses the minimal requirements any workflow should satisfy and includes properties such as the absence of deadlocks and livelocks, and proper termination. The current version 1.3 of Woflan can analyze workflows designed with the workflow products *COSA* and *Protos*. COSA (COSA Solutions/Software Ley, [33]) 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. Fortunately, 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 process definitions made with Protos. Protos (Pallas Athena, [28]) supports Business Process Reengineering (BPR) and can be used to model and analyze business processes. The tool is very easy to use and supports Petri nets. To facilitate the modeling of simple workflows by users not familiar with Petri nets, it is possible to abstract from states. However, Protos cannot detect subtle design flaws that may result in deadlocks or livelocks. Therefore, it is useful to download workflows specified with Protos and analyze them with Woflan.

This paper focuses on the new features recently added to Woflan 1.3. These features allow for the generation of so-called *behavioral error sequences*. One can think of such a sequence as a doomsday scenario that clearly shows the roots of the error. These sequences are used for diagnosing errors that are not easy to detect with analysis techniques available in earlier versions of Woflan. The functionality of Woflan 1.3 has been evaluated using a case study. This case study was part of the final assignment 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 (see Appendix A for the complete assignment text). These workflows were de-

signed with Protos. Afterwards, we collected the workflows and analyzed them with Woflan 1.3. Most of the designed workflows contained several errors that were repaired using the diagnostics provided by Woflan. Based on this case study and earlier experience, we have developed a method to guide users of Woflan in detecting and repairing design errors in workflows.

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. Section 3 introduces workflow management, a subclass of P/T nets called WF nets for modeling workflows, and a soundness property on these WF nets. Section 4 introduces version 1.2 of the tool Woflan that uses standard P/T-net techniques to analyze WF nets. Using these techniques, Woflan can decide whether or not a WF net satisfies the soundness property. However, sometimes Woflan 1.2 does not guide a designer towards locating errors; it will only tell that they exist. In Section 5, we propose a behavioral technique that guides a user towards correcting these errors. This technique has been implemented yielding the current version 1.3 of Woflan. Woflan 1.3 has been tested using a case study, which is presented in Section 6. The results from this case study are used in Section 7, where we introduce a method for diagnosing and correcting workflow nets. Section 8 presents conclusions and topics for future work, among which implementing the method in Woflan is an important one. The informal description of the workflow used in the case study and a formal model in Protos can be found in Appendix A.

2 P/T nets

2.1 Introduction

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 [3, 5] 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 [15, 30].

2.2 Basic definitions

2.2.1 Net structure

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. To identify the elements in a P/T net, we introduce the set of identifiers U.

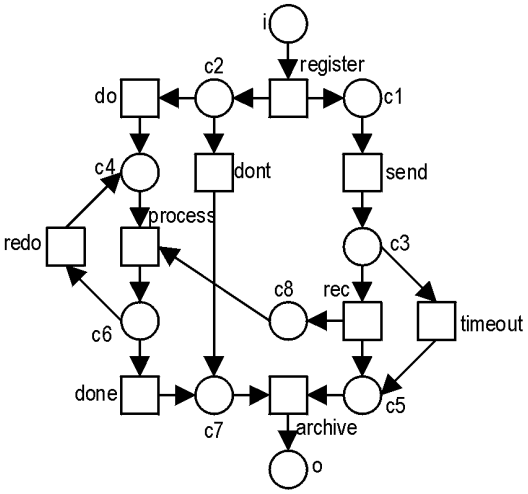


Figure 1: The example P/T net N

Definition I (P/T net)
The triple $N \in (P, T, F)$ is a P/T net iff:

i. $P \subseteq U$ is a finite, non-empty, set of places.

ii. $T \subseteq U$ 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 I 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. In this paper, we allow at most one arc between any two nodes of a P/T net.

2.2.2 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 small 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.

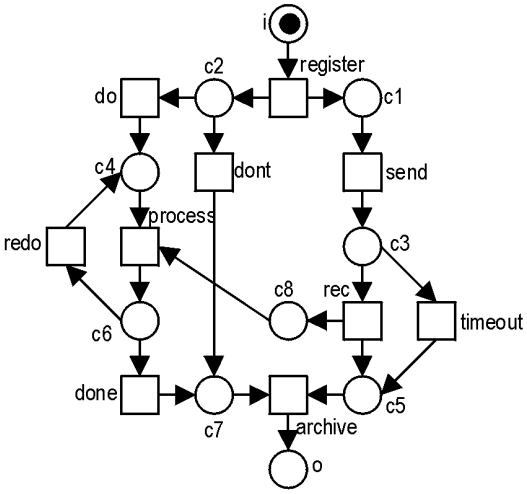


Figure 2: An example system S for net N

Notation (Bags)
A bag over some alphabet A is defined as a function from A to the set of natural numbers. 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 *set* of elements from A is also a bag over alphabet A, namely the function yielding 1 for every element in the set and 0 otherwise. The set of all bags over an alphabet A is denoted $B(A)$. We use brackets to explicitly enumerate a bag and superscripts to denote the cardinalities of the elements. For example, $[a^2, b^3, c]$ denotes the bag with two elements a, three elements b, and one element 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. To denote individual elements of a bag, the same symbol "$\in$" is used as for sets. The sum (addition) 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 (subtraction) of X and Y, denoted $X - Y$, is defined as $[a^n|a \in A \wedge n = (X(a) - Y(a))\underline{max}0]$. Bag X is a subbag of bag Y, denoted $X \leq Y$, iff, for all $a \in A$, $X(a) \leq Y(a)$.

Definition II (Marking)
A bag $M \in B(P)$ is called a marking of a P/T net (P, T, F).

Definition III (System)
The pair $S = (N, M)$ is a system iff N is a P/T net and M is a marking of N (called the initial marking of S).

2.2.3 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 behaves dynamically. We need a way to get from one state to another. For this reason, we define the so-called firing rule.

Definition IV (Preset, postset)
Let $N = (P, T, F)$ be a P/T net. For $n \in P \cup T$:

- $\bullet n = \{n_0 \in P \cup T | (n_0, n) \in F\}$ (the preset of n) and
- $n\bullet = \{n_0 \in P \cup T | (n, n_0) \in F\}$ (the postset of n)

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

Definition V (Firing rule)
Let $N = (P, T, F)$ be a P/T net, M a marking of N, and $t \in T$.

i. M enables t iff $\bullet t \leq M$.

ii. M_1 is reached from M by firing t (denoted $M \xrightarrow{t} M_1$) iff M enables t and $M_1 = M - \bullet t + t\bullet$.

So, a transition is *enabled* iff its preset is a subbag of the actual marking. 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 actual state 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`.

2.3 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.

2.3.1 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. All the structural properties we consider in this paper can be computed in polynomial time (with respect to the number of nodes of the net).

Nodes in a P/T net are connected by paths, which are sequences of arcs.

Definition VI (Directed path)
Let $N = (P, T, F)$ be a P/T net. The sequence $s = (n_0, n_1, \ldots, n_k)$, for some natural number k, is called a (directed) path from n_0 to n_k iff $\forall i, 0 \leq i < k : (n_i, n_{i+1}) \in F$.

Definition VII (Undirected path)
Let $N = (P, T, F)$ be a P/T net. The sequence $s = (n_0, n_1, \ldots, n_k)$, for some natural number k, is called an *undirected* path from n_0 to n_k iff $\forall i, 0 \leq i < k : (n_i, n_{i+1}) \in F \cup F^{-1}$.

The set of nodes $n_0, n_1, \ldots, n_k$ in a (directed or undirected) path $s = (n_0, n_1, \ldots, n_k)$ is called the alphabet of s, denoted $a(s)$.

A path is called elementary iff all nodes in the path are different.

Definition VIII (Elementary path)
Let $N = (P, T, F)$ be a P/T net, and let $s = (n_0, n_1, \ldots, n_k)$, for some natural number k, be a (directed or undirected) path from n_0 to n_k. Path s is called *elementary* iff $\forall i, 0 \leq i \leq k : \forall j, 0 \leq j \leq k : i \neq j \Rightarrow n_i \neq n_j$. The set of all elementary directed (undirected) paths from n_0 to n_k is denoted $E_d(n_0, n_k)$ ($E_u(n_0, n_k)$).

A P/T net is called (strongly) connected iff there exists a (directed) path between every two nodes.

Definition IX ((Strongly) connected)
Let $N = (P, T, F)$ be a P/T net. Net N is connected iff $\forall n_0, n_1 \in P \cup T : E_u(n_0, n_1) \neq \emptyset$. It is strongly connected iff $\forall n_0, n_1 \in P \cup T : E_d(n_0, n_1) \neq \emptyset$.

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`. Figure 4 shows a net that is strongly connected.

A place-transition pair is called a PT-handle iff there exist different elementary paths from the place to the transition; a transition-place pair is called a TP-handle iff there exist different elementary paths from the transition to the place [17].

Definition X (PT-handle, TP-handle)
Let $N = (P, T, F)$ be a P/T net and let $p \in P, t \in T$ be nodes of N. The pair (p, t) is called a PT-handle iff $\exists s_0, s_1 \in E_d(p, t) : s_0 \neq s_1 \wedge a(s_0) \cap a(s_1) = \{p, t\}$. The pair (t, p) is called a TP-handle iff $\exists s_0, s_1 \in E_d(t, p) : s_0 \neq s_1 \wedge a(s_0) \cap a(s_1) = \{p, t\}$.

Since PT-handles and TP-handles can easily introduce design flaws [4], we name nets without these potentially correctness-threatening constructs well-handled. The P/T net `N` of Figure 1 is not well-handled, because it contains one PT-handle (see Figure 12) and two TP-handles (see Figure 9 and Figure 24).

Definition XI (Well-handled)
A P/T net (P, T, F) is called well-handled iff it has no PT-handles and no TP-handles.

A P/T net is called *free-choice* iff for every two transitions t_0 and t_1 their presets are either disjoint or identical. Net `N` of Figure 1 is free-choice.

Definition XII (Free-choice)
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 one input and one output place.

Definition XIII (State machine)
A P/T net (P, T, F) is a state machine iff $\forall t \in T : |{\bullet t}| = |t {\bullet}| = 1$.

Definition XIV (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 XV (S-component)
Let $N = (P, T, F)$ be a P/T net; let $N_0 = (P_0, T_0, F_0)$ be a subnet of N such that $P_0 \neq \emptyset$ and 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 strongly connected state machines, it is S-coverable. The P/T net `N` of Figure 1 has no S-components. The P/T net `N` of Figure 4 has two S-components (see Figure 10) but is not S-coverable: place `c8` is not covered by these S-components.

Definition XVI (S-coverable)
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) such that $p \in P_0$.

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

Definition XVII (Place-invariant)
Let $N = (P, T, F)$ be a P/T net and let w be a weight function from P to the set of 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 $\mathtt{i} + \mathtt{c1} + \mathtt{c3} + \mathtt{c5} + \mathtt{o}$.

It is not difficult to see that if w_0 and w_1 are place-invariants, $w_0 + w_1$ and $w_0 - w_1$ 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.

Definition XVIII (Transition-invariant)
Let $N = (P, T, F)$ be a P/T net and let w be a weight function from T to the set of integer numbers. Function w is a transition-invariant of N iff $\forall p \in P : \sum t \in \bullet p : w(t) = \sum t \in p\bullet : w(t)$.

For example, a transition-invariant of net N of Figure 1 is $\mathtt{rec} + \mathtt{process} + \mathtt{redo} - \mathtt{timeout}$.

As with place-invariants, summation and subtraction of transition-invariants yields new transition-invariants.

2.3.2 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 XIX (Occurrence sequences)
Let $S = (N, M_0)$ be a system, let $M_0, M_1, \ldots, M_n$, 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.

i. $s = M_0 t_0 M_1 \ldots t_{n-1} M_n$ is an occurrence sequence of S iff $\forall i, 0 \leq i < n : M_i \xrightarrow{t_i} M_{i+1}$.

ii. By $first(s)$, we denote the first marking of $s = M_0 t_0 M_1 \ldots t_{n-1} M_n$, i.e., M_0. By $last(s)$, we denote the last marking of s, i.e., M_n.

The set of all occurrence sequences of a system S is denoted S_s.

An occurrence sequence of a system can be projected onto the set of transitions, yielding a so-called *firing sequence*.

Notation (Firing sequences)
The restriction of occurrence sequences to sequences of transitions: $s = M_0 t_0 M_1 t_1 M_2 t_2 \ldots t_{n-1} M_n \rightarrow t = t_0 t_1 t_2 \ldots t_{n-1}$ is called the firing sequence associated to s. The set of all firing sequences of a system S is denoted S_t.

Consider again P/T net N of Figure 1. For the system $(\mathtt{N}, [\mathtt{c5}, \mathtt{c7}, \mathtt{c8}])$, $\mathtt{S}_t$ equals {archive}. If our initial marking is $[\mathtt{c4}, \mathtt{c5}, \mathtt{c8}]$, then $\mathtt{S}_t$ equals {process, process redo, process done, process done archive}. Note that S_t (and S_s) are prefix-closed, i.e., every prefix of a firing (occurrence) sequence is also a firing (occurrence) sequence.

2.3.3 Occurrence graph

Given a system $S = (N, M_0)$, its set of occurrence sequences S_s can be embedded into a graph. Every occurrence sequence corresponds to some path in that graph and vice versa.

Notation (Reachability)
Let $N = (P, T, F)$ be a P/T net; let M_0 and M_1 be markings of N and let $S = (N, M_0)$. The marking M_1 is reachable from marking M_0, denoted $M_0 \longrightarrow M_1$, iff there exists an occurrence sequence $s \in S_s$ such that $first(s) = M_0$ and $last(s) = M_1$.

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

Definition XX (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:

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

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

The construction of this graph is straightforward, although termination is not guaranteed.

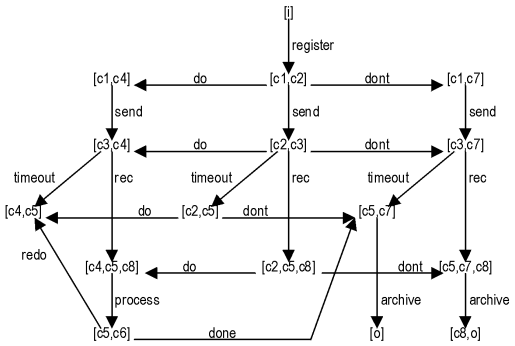


Figure 3: The OG of system S

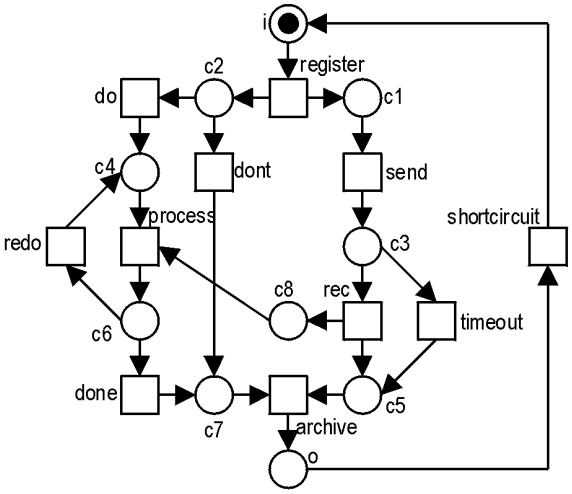


Figure 4: The short-circuited system $\underline{S} = (\underline{N}, [i])$

Construction (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. The OG $G = (H, A)$ of S 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_2 to H and (M, t, M_2) to A. Repeat this step until no new arcs can be added.

For our example system S of Figure 2, the OG looks as in Figure 3.

The OG embeds exactly all possible behaviors of the system. However, for some systems, the OG is infinite- and can thus not be finitely constructed. For example, if we short-circuit net N in our system S of Figure 2 with the `shortcircuit` transition from o to i, as is shown in Figure 4, we get an OG which has infinitely many nodes. In this extended system, firing the transitions `register send rec dont archive shortcircuit` over and over again, leads to infinitely many markings $[o, c8^n]$, 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; therefore, this place is called *unbounded*. As a result, the number of markings in the OG is infinite.

2.3.4 Coverability graph

A solution to cope with unbounded places is the notion of a so-called coverability graph. A coverability graph is a variant of the occurrence graph that is finite in size. 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.

Notation (Infinity)
Infinity is denoted by ω. For every natural number n, it holds that $n \leq \omega$, $\omega + n = \omega$, and $\omega - n = \omega$. For every natural number $n > 0$, it holds that $n\omega = \omega n = \omega$.

Notation (Extended bags)
An extended bag over some alphabet A is defined as a function from A to the set of natural numbers extended with ω. The set of all extended bags over an alphabet A is denoted $B^\omega(A)$.

All operations on bags can be defined for extended bags in a straightforward way.

Definition XXI (Finite, extended, infinite markings)
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 in a set of *finite* markings ($B(P)$) and a set of *infinite* markings ($B^\omega(P) \setminus B(P)$).

The markings as we have used them before are now called finite markings.

A coverability graph of a system is a variant of the OG, where paths containing infinitely many different markings in the OG are represented by a finite number of infinite markings. An infinite marking is introduced in a coverability graph if we encounter a marking that has a smaller marking as one of its predecessors. Suppose we have the markings M_1 and M_2 such that in the occurrence graph there is a path from M_1 to M_2 and $M_1 < M_2$. Then, there has to be a path from M_2 to the marking $M_2 + (M_2 - M_1)$ that corresponds to firing all the transitions on the path from M_1

to M_2. Consequently, there also is a path from marking $M_2 + (M_2 - M_1)$ to $M_2 + 2(M_2 - M_1)$, from marking $M_2 + 2(M_2 - M_1)$ to $M_2 + 3(M_2 - M_1)$, and so on. We conclude that places which occur in the marking $M_2 - M_1$ are unbounded. Whereas the OG of a P/T net contains infinitely many finite markings of the form $M_2 + n(M_2 - M_1)$, for $n > 0$, a coverability graph contains one infinite marking with ω-values for each place in $M_2 - M_1$. It is a known fact ([30], p. 70) that a coverability graph of a system is always finite and can thus be constructed.

Definition XXII (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 exists a node M_1 in H such that $M_1 \leq M_2$, $M_1(p) < M_2(p)$, and there exists a 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 [30] for more details). Nevertheless, a CG of a system can be used to analyze the behavior of the system. The short-circuited net $\underline{\mathsf{S}}$ of Figure 4 has a unique CG which is shown in Figure 5.

Given a system and a CG of this system, every occurrence sequence of the system corresponds to some path in the CG and is thus embedded in it. The converse is not necessarily true: There may be paths in the CG that do not correspond to any occurrence sequence. However, a path that contains only finite markings does correspond to some occurrence sequence. This is in accordance with the fact that the CG is identical to the OG if there are no infinite markings in the CG. The theoretical worst-case complexity of generating a CG is non-primitive recursive space. Nevertheless, for small to medium size systems, which includes many examples in workflow practice, the performance of the CG-generation algorithm is acceptable. In Sections 5 and 6, we return to this point.

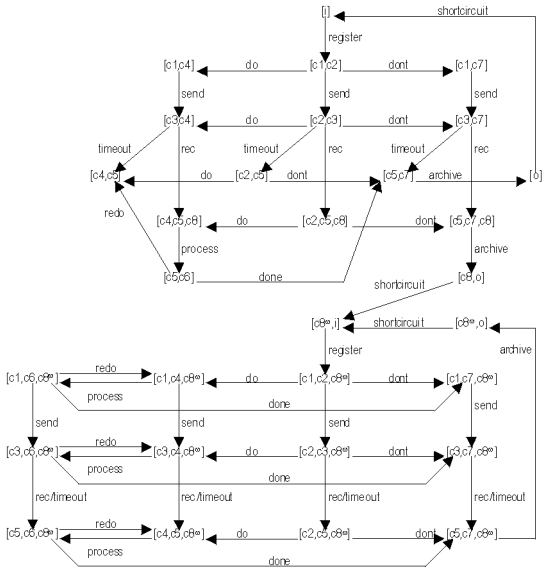


Figure 5: The CG for the short-circuited $\underline{\mathsf{S}}$

2.3.5 Behavioral properties

Behavioral properties of a P/T net are those properties that depend on the marking of the P/T net. Therefore, these properties are defined on systems, not on P/T nets. In the remainder, we do not go into detail about the precise complexities of the algorithms to determine behavioral properties (see [17] for more information on this topic). For the purpose of this paper, it suffices to know that the theoretical complexity of computing behavioral properties is often much worse than the complexity of computing structural properties.

A transition is *dead* iff it is never enabled.

Definition XXIII (Dead transition)
A transition $t \in T$ of a system $((P, T, F), M_0)$ is dead iff $\forall M \in B(P), M_0 \longrightarrow M : t$ is not enabled in M.

A transition is live iff it can always fire again. A system is called live iff every transition is live.

Definition XXIV (Live)
A transition $t \in T$ of a system $((P, T, F), M_0)$ 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. A system $((P, T, F), M_0)$ is live iff $\forall t \in T : t$ is live.

The system S of Figure 2 is not live: For instance, no more transition firings are possible from reachable marking [o] (see Figure 3). The short-circuited system $\underline{\mathsf{S}}$ of Figure 4 is also not live: No more transition firings are possible from reachable marking [c4, c5] (see Figure 5).

A system is called bounded iff all places are bounded. A system is called safe iff all places in any reachable marking contain at most one token.

Definition XXV (Bounded, safe)
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 $\forall M \in B(P), M_0 \longrightarrow M : \forall p \in P : M(p) \leq 1$.

An alternative definition for boundedness is to require that the number of reachable markings is finite. Note that, for a bounded P/T system, the CG-generation algorithm of Definition XXII yields the OG of the system.

The 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 4 is unbounded, which follows directly from the fact that there are infinite markings in the CG of Figure 5.

3 Workflow management

3.1 Introduction

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 [5, 20, 22, 25, 26, 31, 32]. 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 [3, 5, 16]), P/T nets constitute a good starting point for a solid theoretical 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.

3.2 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*. 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 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 independent of the specific bicycle for which the wheel will be used.

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 most important aspect of a workflow process is the *routing definition*. The routing definition specifies which *tasks* need to be executed in what *order*. Alternative terms for routing definition are: 'procedure', 'flow diagram' and '*workflow process 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 [26]. 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 so-called *workflow attributes*. A workflow attribute is a specific piece of information used for the routing of a case. One can think of a workflow attribute as a control variable or a logistic parameter. A workflow attribute may be the age of a customer, the department responsible, or the registration date. Routing decisions may also be based on other data than workflow attributes, such as for example 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). 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 is another important part of a workflow process. 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 definitions of efficient and effective workflow processes is the basis of the successful introduction of a workflow system. Formal verification can be a useful aid in obtaining the desired effectiveness and efficiency.

3.3 Verification of workflows

Our work is aimed at the verification of *workflow process definitions*. Two types of verification are possible, namely qualitative verification and quantitative verification. Qualitative verification focuses on functional properties of a workflow process definition, whereas quantitative verification focuses on the performance of the workflow process. Examples of functional properties are the existence or absence of deadlocks or livelocks. Examples of performance properties are average throughput times and service levels. The current version of the tool Woflan only implements techniques for qualitative verification, although we are considering extending Woflan with techniques for quantitative verification as well.

The focus on qualitative verification allows us to abstract from resources and triggers. The allocation of resources to work items and the occurrence of triggers is a crucial factor in the *performance* of a workflow process. However, for qualitative verification, it suffices to assume that sufficient resources are available to execute all the required tasks and that any trigger will occur eventually. Another abstraction is that we consider only one case in isolation. The only way cases interact with each other is via the competition for resources. Therefore, if we abstract from resources, it also suffices to consider only one case in isolation. Finally, we abstract from workflow attributes and other data. This means that we consider each choice (OR-split) to be a non-deterministic one. The reason is the following. If we are able to prove certain desirable properties in the situation where all choices are taken non-deterministically, they will also be satisfied in the situation where choices are based on workflow attributes or other data. An important consequence of these abstractions is that they allow us to use P/T nets rather than Petri nets with data and time. From an analysis point of view, the class of P/T nets is preferable because of the availability of efficient algorithms and powerful analysis tools.

3.4 Workflow nets

In this subsection, we introduce a subclass of P/T nets, namely the class of *workflow nets* (WF nets). In addition, we formalize the so-called *soundness* property for WF nets. 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 a workflow process. A workflow process contains for example information about applications, workflow attributes, triggers, case data, and resource constraints. 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.

3.4.1 P/T-net representation

The P/T net `N` 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. This initial marking `[i]` corresponds to the fact that a new complaint has been received. First, the complaint is registered (`register`). The 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 complaint'. 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*. If a task fulfils a condition when it is completed successfully, an *arc* is drawn from the corresponding transition to the corresponding place. If a task needs a condition to be fulfilled before it can start, an *arc* is drawn from the corresponding place to the corresponding transition. 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 [26, 35]) can be modeled by P/T nets.

By verifying the behavior of a P/T net modeling a workflow, we can verify some properties of the workflow in reality. Not every P/T net corresponds to a workflow. A P/T net modeling a workflow must satisfy several structural properties. It must, for example, have a well-defined beginning and end, a s in the example described above. Therefore, in the next paragraph, we impose some structural restrictions on P/T nets yielding so-called *workflow nets* (WF) nets. In Section 3.4.3, we impose restrictions on the *behavior* of WF nets, defining so-called *sound* workflow nets. As mentioned, soundness is the least requirement that a WF net must satisfy in order to be a proper abstraction of a workflow process definition.

3.4.2 Structural restrictions

We want a P/T-net model of a workflow process to have 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 in the universe U.

There can be no tasks that fulfill the condition corresponding to i: The workflow cannot generate cases. Also, there can be no tasks for which the condition corresponding to o has to be fulfilled: Once a case has been completed, no more tasks should be executed for this case.

Furthermore, there is not much use in having a task that can never be executed or in having a task from which the case cannot be completed. This means that the structure of a workflow net must satisfy at least the following requirement: For every transition t in a workflow net, there must be a path (see Definition VI) from i to t and a path from t to o. In P/T-net terms, this conforms to strongly connectedness (see Definition IX) under the assumption that there is a path from o to i. This assumption can be fulfilled if we short-circuit the net as in Figure 4. Nets that satisfy the above restrictions are called workflow nets or WF nets.

Definition XXVI (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 $\underline{t} \in U \setminus T$.

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. Therefore, it is a WF net.

3.4.3 Behavioral restrictions

To analyze the behavioral correctness of a workflow, we are interested in its behavior for one case. Under the assumption that the workflow system can distinguish cases, it is an obvious choice to have $[i]$ as the initial marking, because it corresponds to the creation of a new case. So, if $N = (P, T, F)$ is a WF net, then $S = (N, [i])$ is the corresponding WF system that we are interested in.

The behavioral restrictions we impose on a WF system in its initial state are straightforward:

i. It should always be possible to complete a case (option to complete).

ii. It should not be possible to complete a case improperly (proper completion), where improper completion means that there is still work in progress after completion of the case.

iii. For every task, there should be an execution of the workflow that executes it (no dead tasks).

Completion of a case is signaled by a token in the special place o. Thus, rule i means that it must always be possible to put a token in o. The second rule means

that, as soon as a token is put in o, all other places must be empty. The third rule strengthens the third structural requirement of Definition XXVI. It simply means that a WF system may not have any dead transitions (see Definition XXIII).

If a WF system in its initial state behaves according to the above rules, then the corresponding WF net is called *sound*.

Definition XXVII (Soundness)
A workflow 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 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 3 (which is also a CG of S), we see that N is not sound because the first two restrictions are not satisfied:

- In [c4, c5], there is no option to complete;
- 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 and boundedness of the short-circuited WF system. Recall that, for a WF net N, the short-circuited net $(P, T \cup \{\underline{t}\}, F \cup \{(o, \underline{t}), (\underline{t}, i)\})$ with $\underline{t} \in U \setminus T$ is denoted $\underline{N}$.

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

Proof
See [1].

From the CG in Figure 5, we conclude that the short-circuited WF system $\underline{\text{S}}$ of Figure 4 is not bounded and not live. It is not bounded, because we have infinite markings in the CG; it is not live, because we for instance cannot escape from the marking [c4, c5]. Hence, the WF net N of Figure 1 is not sound, which conforms to our earlier conclusion.

Theorem I is an interesting result, because the Petri-net literature contains many results on liveness and boundedness of systems. These results form the basis for the efficient verification of WF nets. In the remainder of this section, we present the most important results that are used in Woflan.

3.4.4 Properties

For free-choice WF nets (see Definition XII), there is an efficient algorithm to decide soundness.

Theorem II (Free-choice vs. soundness)
Given a free-choice WF net, it can be decided in polynomial time whether or not the net is sound.

Proof
Let N be a WF net. It is possible to verify in polynomial time whether the short-circuited WF system $(\underline{N}, [i])$ is live and bounded [15].

Theorem III (Soundness and free-choice vs. S-coverability)
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 I and the fact that a net which is free-choice, live, and bounded must be S-coverable ([15]).

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. Theorem II and Theorem III show that the class of free-choice WF nets is an interesting class from a viewpoint of analysis.

Another interesting class is the class of so-called well-structured WF nets. A WF net is well-structured iff the short-circuited net is well-handled, i.e., the short-circuited net has no PT-handles and TP-handles (see Definition XI).

Definition XXVIII (Well-structured)
A WF net N is well-structured iff $\underline{N}$ is well-handled.

Theorem IV (Well-structuredness vs. soundness)
Given a well-structured WF net, it can be decided in polynomial time whether or not the WF net is sound.

Proof
See [4]. The proof uses the fact that short-circuited WF nets without PT-handles and TP-handles are elementary extended non-self controlling [10].

A sound well-structured WF net is S-coverable.

Theorem V (Soundness and well-structuredness vs. S-coverability)
Let N be a sound, well-structured WF net. The short-circuited WF net $\underline{N}$ is S-coverable.

Proof
See [4].

Theorem V can be used in the analysis of WF nets in a similar way as Theorem III can be used.

If a short-circuited WF net $\underline{N}$ is S-coverable, then the short-circuited WF system $(\underline{N}, [i])$ is safe (and bounded).

Theorem VI (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 XV that the number of tokens in any reachable marking of $(\underline{N}, [i])$ in an S-component of 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 S-coverable place is either zero or one. Because all places in N are S-coverable, $(\underline{N}, [i])$ is safe and thus bounded.

A consequence of Theorem VI is that both sound free-choice WF nets and sound well-structured WF nets correspond to safe WF systems. So, a free-choice or well-structured WF net of which the corresponding system is not safe cannot be sound.

Improper termination of a WF net N always leads to an unbounded short-circuited WF system $(\underline{N}, [i])$. Thus, an unbounded place in a short-circuited WF system may be a sign of improper completion.

Theorem VII (Improper completion vs. unboundedness)
Let Nbe 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 XXVII) 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 $(\underline{N}, [i])$.

A WF net N that has no option to complete always has a short-circuited WF system $(\underline{N}, [i])$ containing non-live transitions. Thus, in case of non-liveness of the short-circuited system, the WF net may not have the option to complete. Thus, non-live transitions are a potential sign of errors.

Theorem VIII (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 $(\underline{N}, [i])$, with $\underline{N} = (\underline{P}, \underline{T}, \underline{F})$, has non-live transitions.

Proof
Suppose $(\underline{N}, [i])$ has only live transitions. Then, the short-circuiting transition t is live, i.e., $\forall M \in B(P), [i] \longrightarrow M : \exists M_1 \in B(P), M \longrightarrow M_1 : M_1$ enables $\underline{t}$. Since $\bullet\underline{t} = \{o\}$ and $P = \underline{P}$, we conclude that $\forall M \in B(P), [i] \longrightarrow M : \exists M_1 \in B(P), M \longrightarrow M_1 : M_1 \geq [o]$, i.e., N has the option to complete.

4 Woflan

4.1 Introduction

This section describes *Woflan* (WOrkFLow ANalyzer), a tool that analyzes workflow process definitions specified in terms of Petri nets. Woflan has been designed to verify process definitions that are downloaded from a workflow management system (cf. [5, 7]). As indicated in the introduction, there is a clear need for such a verification tool. Today's workflow management systems do not verify the correctness of workflow process definitions. Therefore, errors made at design time 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. To avoid these costly problems, it is important to verify the correctness of a workflow process definition before it becomes operational.

Based on 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 [2, 19]. Basically, Woflan takes a workflow processes definition imported from some workflow product, translates it into a P/T net, and tells whether or not: (1) the net is a WF net and (2) the net is sound. Furthermore, using existing P/T net analysis routines, the tool provides information about some properties in case it is not a sound WF net. These properties can be either

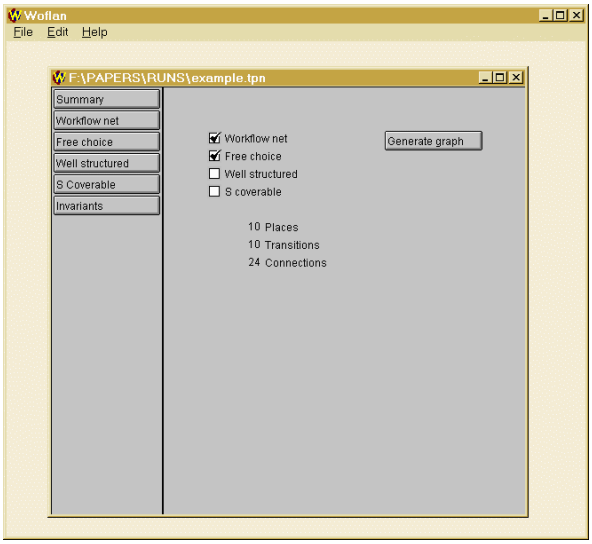


Figure 6: Woflan's summary on N before it has generated a CG

structural or behavioral. The information on structural properties is straightforward to compute; for the information on behavioral properties Woflan first needs to compute the behavior of the system. Woflan generates a CG for the short-circuited system and checks whether the short-circuited system is bounded and live. If so, the WF net is sound.

This section reviews the basic functionality of Woflan present in version 1.2. This version extends version 1.0 as described in [2, 19] with an import facilitiy for Cosa and Protos. In the remaining sections of the paper, we focus on the new features, the case study, and a method to support the use of Woflan.

4.2 Structural properties

Figure 6 shows us the summary information on the structural properties of WF net N of Figure 1. The following information is available in this summary:

- Whether N is a WF net;
- whether N is free-choice;
- whether N is well-structured;
- whether the short-circuited net $\underline{N}$ (see Figure 4) is S-coverable and
- the numbers of places, transitions, and arcs (connections).

Net N is a WF net that is free-choice but not well-structured; the short-circuited net $\underline{N}$ is not S-coverable.

Using the buttons on the left-hand side, detailed information on the properties becomes available.

Please note that we use the standard Petri-net terminology to avoid confusion, i.e., we use the terms transition and place rather than task and condition. As indicated in Section 3.4, the WF net is an abstraction of the workflow process definition downloaded from the workflow management system. One task in the workflow process definition may correspond to several transitions and the WF net abstracts from information about applications, workflow attributes, triggers, case data, and resources.

4.2.1 WF net

The detailed diagnostic information calculated by Woflan on the WF-net property of some WF net N (see Definition XXVI) consists of six items:

1. The set of places with an empty preset, i.e., so-called source places. Every case has to enter the workflow at the same point, called the point of creation. Therefore, to satisfy the WF-net property, there should be exactly one source place (denoted i), which corresponds to this point of creation.

2. The set of places with an empty postset, i.e., so-called sink places. Every case has to leave the workflow at the same point, called the point of completion. Therefore, there should be exactly one sink place (denoted o), which corresponds to this point of completion.

3. The set of transitions with an empty preset, i.e., so-called source transitions. A task that corresponds to a source transition, does not have to wait for a case to be created: there is no path from the point of creation i to the task. Therefore, the task cannot be related to a case. Note that, if a source transition exists, the short-circuited net is not strongly connected.

4. The set of transitions with an empty postset, i.e., so-called sink transitions. A task that corresponds to a sink transition does not help completing a case: There is no path from the task to the point of completion o. Therefore, the task cannot be related to the completion of the case. Note that, if a sink transition exists, the short-circuited net is not strongly connected.

5. The set of unconnected nodes. A node is unconnected iff there is no undirected path in the short-circuited net $\underline{N}$ to the short-circuiting transition $\underline{t}$. The conditions and/or tasks corresponding to these unconnected nodes cannot be related to the points of creation i and completion o (because both are by definition connected to $\underline{t}$).

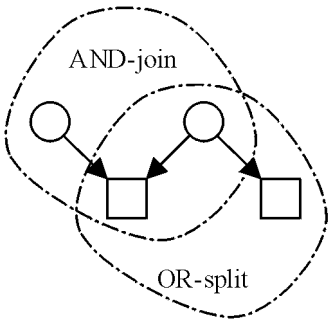


Figure 7: A non-free-choice cluster

6. The set of strongly unconnected nodes. A node is strongly unconnected iff there is no path in $\underline{N}$ from that node to $\underline{t}$ or no path from $\underline{t}$ to that node. There should be no strongly unconnected nodes. The conditions and/or tasks corresponding to strongly unconnected nodes cannot be related to either the point of creation i or the point of completion o (because both are by definition strongly connected to $\underline{t}$). Note that, if a node is unconnected, it is also strongly unconnected.

If all the requirements are satisfied (one place corresponding to a point of creation, one place corresponding to a point of completion, and all nodes can be related to both places), the net is a WF net. Net N is obviously a WF net.

4.2.2 Free-choice property

The detailed information on the free-choice property gives us the set of so-called non-free-choice clusters, where a cluster is one of the connected components that remain 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 XII. An example of a non-free-choice cluster is shown in Figure 7.

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 (see Figure 7) is therefore often a mixture of an OR-split with an AND-join. 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.

There may be several reasons for warning for non-free-choice constructs. Most of the workflow management systems available at the moment abstract from states between tasks, i.e., states are not represented explicitly. These 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 P/T net, the OR-split corresponds to a number of transitions sharing the same set of input places. This means that, for these workflow management systems, a workflow procedure corresponds to a free-choice P/T net. Only a few workflow management systems (e.g. COSA, INCOME, LEU, and MOBILE) allow for constructs that yield a non-free-choice WF net. Therefore, it makes sense to consider free-choice P/T nets. Clearly, parallelism, sequential routing, conditional routing, and iteration can be modeled without violating the free-choice property. Another reason for restricting WF nets to free-choice P/T nets is the following. If we allow non-free-choice P/T nets, then the choice between conflicting tasks may be influenced by the order in which the preceding tasks are executed. (In P/T-net terminology, this is called *confusion*.) The routing of a case should be independent of the order in which tasks are executed. As indicated before, a situation where the free-choice property is violated is often a mixture of parallelism and choice. Therefore, Woflan supplies diagnostics for tracing non-free-choice constructs. However, note that non-free-choice clusters are not always incorrect. Sometimes, complex routing constructs cannot be modeled with free-choice WF nets; in other occasions, non-free-choice constructs might yield more concise models.

Net N of Figure 1 has no non-free-choice clusters and is hence free-choice.

4.2.3 Well-structuredness

A balance between AND/OR-splits and AND/OR-joins characterizes a good workflow design. 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 8 are suspicious.

In the leftmost situation, an AND-split is terminated by an OR-join. Tasks of a case are executed in parallel, but, after one branch gets fulfilled, 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 net and hence to unsoundness.

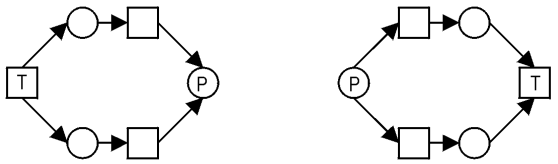


Figure 8: AND/OR mismatches

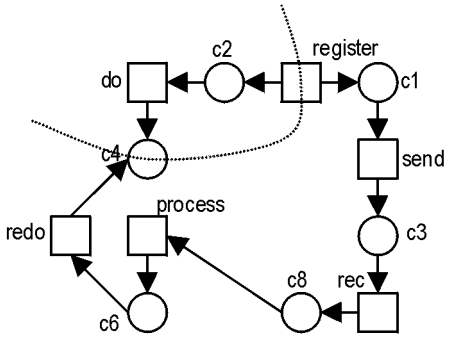


Figure 9: A non-well-handled pair in N

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 net and hence to unsoundness.

Both situations describe a so-called non-well-handled pair (see Definition XI): A transition-place 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 X).

Recall from Definition XXVIII that a WF net N is well-structured iff the short-circuited net $\underline{N}$ is well-handled. Although a non-well-handled pair in $\underline{N}$ is often a sign of potential errors, a WF net that is not well-structured can still be sound.

According to the analysis of Woflan depicted in Figure 6, our example net N is not well-structured. The detailed diagnostic information provided by Woflan shows that there are three non-well-handled pairs. One of them is displayed graphically by the P/T net in Figure 9, where the non-well-handled pair is (register, c4). Note that, because of the non-well-handled pair, the condition c4 might get fulfilled twice, which endangers the boundedness of system ($\underline{N}$, [i]) and thus the soundness of net N.

Well-structured WF nets and free-choice WF nets have similar properties. In both cases, soundness can be verified in polynomial time and soundness implies safeness (see Section 3.4.4). In spite of these similarities, there are sound well-structured WF-nets which are not free-choice and there are sound free-choice WF-nets which are not well-structured. In fact, it is possible to have a sound WF-net that is neither free-choice nor well-structured. Notwithstanding these observations, the two structural characterizations (free-choice and well-structuredness) turn out to be very useful for the analysis of workflow process definitions. Both well-structuredness and the free-choice property correspond to desirable properties of a workflow. A WF net satisfying one of these properties can be analyzed very efficiently and is easy to interpret. However, for more advanced routing constructs non-free-choice nets and/or non-well-structured nets are inevitable. In such nets, soundness-endangering constructs are compensated by other constructs.

4.2.4 S-coverability

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

1. The first thread handles the piece of work associated with the complaint form: After registration, we first have to send the form to the complainant. Second, we can either receive it back or a timeout occurs. Finally, the returned form or the fact that it was not returned in time is archived.

2. The second thread handles the piece of work associated with the complaint itself: After registration, we first have to evaluate the complaint. Second, depending on the evaluation (do or dont), we process it followed by a check. Third, depending on the result of the check (done or redo), we process it again. Finally, we archive it.

A place that does not belong to one of these threads is a suspicious place, because it cannot be related to a logical piece of work.

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. Recall that an S-component is a (strongly connected) state machine which is embedded in a P/T net (see Definition XV) and that for each S-component in a P/T net the 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

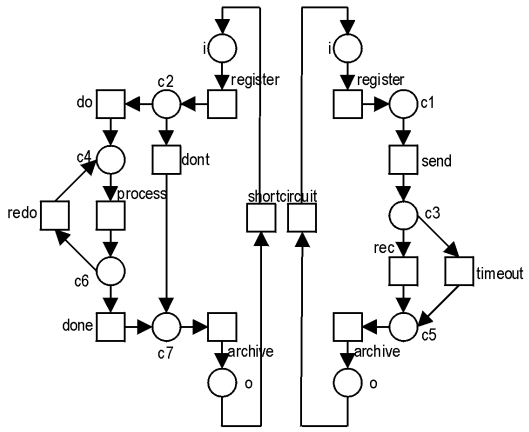


Figure 10: S-components in the short-circuited net N

and bounded, and the system corresponding to a short-circuited WF net that is S-coverable is safe and thus bounded too (see Theorem VI). 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.

It appears that any WF net should satisfy the requirement that its short-circuited net is S-coverable ([4]). 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 detailed information that Woflan provides on S-coverability consists of the set of all S-components (in textual format) and the set of all nodes which are not covered by these S-components. (For readability, the short-circuiting transition $\underline{t}$, which is part of all S-components, is not listed.) Figure 10 shows us both S-components for the short-circuited net N. Note that place c8 is not covered by any of them and is thus not S-coverable.

4.2.5 Invariants

Woflan provides detailed information on place- and transition-invariants of a WF net.

Place-invariants can provide useful information in case a WF net does not satisfy the proper-completion property. As mentioned before, the net of Figure 1 has a place-invariant i + c1 + c3 + c5 + o. Because we know that initially there is one token in place i and upon completion there is one token in o, we conclude that c1, c3, and c5 are empty upon completion. The only place which does not occur in any place-invariant is place c8. Therefore, this place is suspicious: It may be unbounded. In general, place-invariants provide additional information when compared to S-components. In our example, the information that c8 may be unbounded was already deduced in the previous paragraph from the fact that it is not S-coverable.

Transition-invariants provide useful information about cycles and alternative routes. Consider for instance the invariant rec + process + redo − timeout. From this invariant, we deduce that performing the task timeout (all negative weighted transitions) results in the same state of the workflow as performing the tasks rec, process, and redo (all positive weighted transitions). Note that an invariant containing only nonnegative weights corresponds to a cycle in the workflow: The state of the workflow does not change when all the tasks with a positive weight in the invariant are performed (as many times as indicated by their weight). Therefore, in our example of Figure 1, we would expect the invariant process + redo. However, this is not an invariant of the net, which suggests that process and redo do not form a cycle. Clearly, this is a sign of a potential error.

As mentioned in Section 2, a P/T net has either no place-(transition-)invariants or it has infinitely many place-(transition-)invariants. It is impossible to display infinitely many invariants. Therefore, we have to make a representative selection out of these invariants. Woflan offers three of such selections: base, extended, and semi-positive invariants.

- The base invariants as computed by Woflan form a minimal set of invariants needed to construct all invariants using addition and subtraction as operators.
- For the extended invariants as computed by Woflan, every possible set of places and transitions is checked whether there is an invariant for them.
- An invariant is called semi-positive iff all its weights are nonnegative. The semi-positive invariants as computed by Woflan form a minimal set of invariants, needed to construct all semi-positive invariants.

4.3 Behavioral properties

To determine the behavioral properties of a WF net N, Woflan computes a CG of the short-circuited WF system $(\underline{N}, [i])$. The construction of a CG requires, in the worst case, non-primitive recursive space. It is unknown what the exact complexity of deciding soundness for an arbitrary WF net is. Since soundness corresponds to liveness and boundedness, it is likely that the complexity of deciding soundness is at least

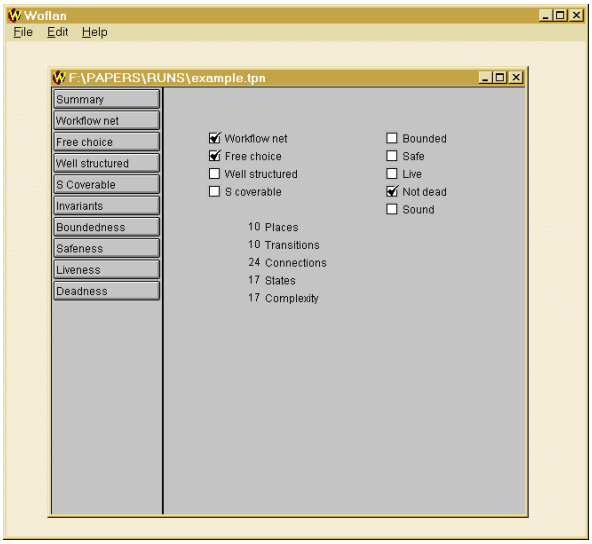


Figure 11: Woflan's summary after the CG has been generated

EXPSPACE-hard. See [11(Section 4.3), 17] for pointers to literature on this topic.

Woflan generates a CG for the short-circuited WF system when the button `Generate graph` in Figure 6 is pressed. After generating the CG, the summary display looks as in Figure 11. The number of markings in the graph (`States`) is reported and so is an indication of the graph's complexity. This complexity is given by the following formula:

$$\text{Complexity} = \frac{\text{Places}}{5} + \frac{\text{Transitions}}{10} + \frac{\text{Connections}}{10} + 10(^{10}log(\text{States}))$$

This formula is based on experience with practical workflow processes. It is an indicator for the complexity of the routing. The more tasks and connections there are and the more reachable states there are, the higher the complexity. Workflow processes with a complexity of less than 25 are easy to understand. Processes with a complexity of more than 100 are really difficult to grasp. Furthermore, Woflan reports information on four behavioral properties of the *short-circuited* WF system $\underline{S} = (\underline{N}, [i])$ and on the soundness of N itself:

i. Whether $\underline{S}$ is bounded;

ii. whether $\underline{S}$ is safe;

iii. whether $\underline{S}$ is live;

iv. whether $\underline{S}$ contains dead transitions and

v. whether N is sound.

Recall that the WF net N is sound iff $\underline{S}$ is bounded and live (Theorem I).

4.3.1 Boundedness

The detailed information on the boundedness property provided by Woflan consists of the set of unbounded places. An unbounded place in $\underline{S}$ always indicates a soundness error in N. Most likely, this error is due to improper completion: Improper completion of N always results in unbounded places in $\underline{S}$ (Theorem VII).

In our system $\underline{\texttt{S}}$ of Figure 4, the place `c8` turns out to be unbounded. As a result, the WF net `N` of Figure 1 cannot be sound. Note that this place is the only place that is not S-coverable (see Section 4.2.4).

4.3.2 Safeness

The detailed information on the safeness property consists of the set of unsafe places. Recall that an unbounded place is also unsafe. We already know that unbounded places always indicate errors. However, also a bounded place which is unsafe is suspicious, although the WF net may still be sound. Note that such sound WF nets cannot be free-choice (Theorem III and Theorem VI) or well-structured (Theorem V and Theorem VI).

From a workflow point of view, an unsafe place corresponds to a condition that can be fulfilled more than once at a single point in time, which is often an abnormal situation.

In system $\underline{\texttt{S}}$ of Figure 4, only the (unbounded) place `c8` is unsafe. Because net `N` is free-choice, we know that `N` cannot be sound.

4.3.3 Deadness

The detailed information on the deadness property consists of the set of dead transitions in $\underline{S}$. As a workflow designer, we are interested in the transitions that are dead in $S = (N, [i])$. A dead transition in S corresponds to a task in the workflow that can never be executed. A transition that is dead in $\underline{S}$ is also dead in S. However, a transition that is dead in S might not be dead in the short-circuited system $\underline{S}$. This means that the information provided by Woflan is useful but not necessarily complete. The reason that Woflan provides the dead transitions in $\underline{S}$ is that it uses the CG it has computed for $\underline{S}$ to calculate these transitions. However, under the assumption of boundedness, a transition that is dead in S is also dead in $\underline{S}$ (see Theorem XII in Section 7.2.3). In this case, the information provided by Woflan is complete and accurate.

Our example system $\underline{\texttt{S}}$ of Figure 4 contains no dead transitions. However, because $\underline{\texttt{S}}$ is not bounded, we

may not conclude that net N contains no tasks that cannot be executed.

4.3.4 Liveness

The detailed information of Woflan on the liveness property consists of a (sub)set of the transitions that are not live in the short-circuited system $\underline{S}$. Woflan determines these transitions from the computed CG of $\underline{S}$. As a result, Woflan may not find the complete set of non-live transitions. However, under the assumption of boundedness, which means that the CG of $\underline{S}$ equals its OG, the set is complete.

Recall that soundness of a net N is equivalent to the boundedness and liveness of the short-circuited system $\underline{S} = (\underline{N}, [i])$. Thus, a non-live transition in $\underline{S}$ always indicates a soundness error in N. Most likely, this error is caused because N lacks the option to complete (Theorem VIII).

In system $\underline{S}$ of Figure 4, all transitions are non-live, while none of them is reported dead. Although we now know that there are liveness errors, the set of non-live transitions does not help us towards locating the error: The information that all transitions in $\underline{S}$ are non-live is not sufficiently specific.

4.4 Diagnosing the example net

As we have seen, the net N of Figure 1 is a WF net that is not sound, because the short-circuited system $\underline{S}$ is unbounded and non-live. Woflan's diagnosis clearly indicates that there is something wrong with place c8: It is not S-coverable in the short-circuited net $\underline{N}$ and it is unbounded in the system $\underline{S}$. This unbounded behavior can be caused only when transition process is not executed as many times as transition rec. This is clearly the case, because we do not have to process a complaint even if the form has been returned in time. Adding an arc from c8 to archive can solve this problem. In case a complaint is not processed, archive removes the token in c8, thus guaranteeing proper termination.

Recall that PT-handles endanger liveness, whereas TP-handles endanger boundedness. If we take a look at the TP-handle of Figure 9, we must ask ourselves the question how c8 can be unbounded, while c4 and c6 are safe. There appears to be something wrong with transition process. The arc connecting c4 to process prevents c4 and c6 from being unsafe, but introduces a cyclic path process c6 redo c4. This path needs c8 as input, but c8 does not get any output from the cycle. Such a cycle cannot function correctly: It can only be executed once. Adding an arc from process to c8 can solve this error. Informally, condition c8 means that the complainant has returned the form. In the original process, task process falsifies this condition by removing the token in c8: This is clearly incorrect.

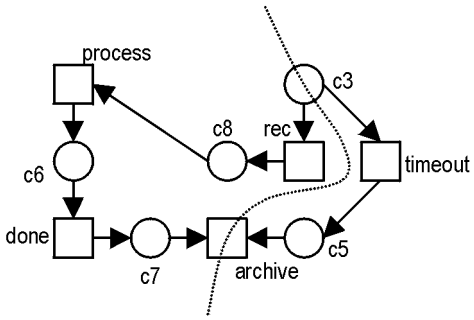


Figure 12: The only PT-handle in net N

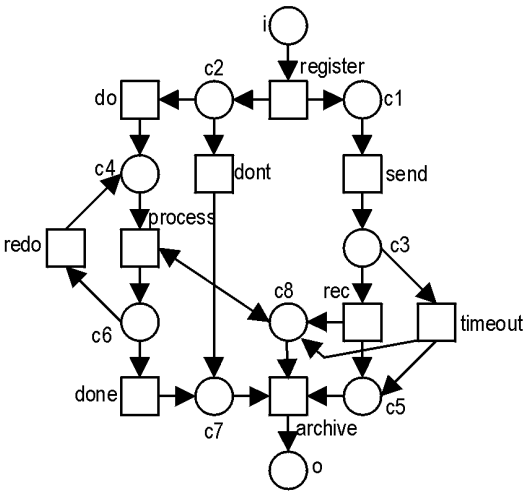


Figure 13: The corrected example net

The correction suggested above means that process only checks the condition modeled by c8.

Woflan also diagnoses that all transitions in the short-circuited system $\underline{S}$ are non-live. To solve this error, we take a look at the only PT-handle (c3, archive) that Woflan reports, which is displayed in Figure 12. From this PT-handle, we conclude that a problem occurs when a form is not received in time (timeout) while it needs to be processed (process). According to the original workflow, a complaint can only be processed if we receive the form of the complainant in time! It is obvious that we must be able to process even in case of a time-out. Adding an arc from timeout to c8 can solve this problem.

After adding the arcs mentioned above, the net looks as in Figure 13 and is sound. In general, a workflow net need not be sound after one correction iteration, but in this case it is. Note that the non-well-handled pair (register, c4) is still present in Figure 13. Soundness does not imply well-structuredness.

Although the net in Figure 13 is sound, it may still be worthwhile to have a closer look at Woflan's diagnosis. The net in Figure 13 has a place-invariant c5 – c8. Be-

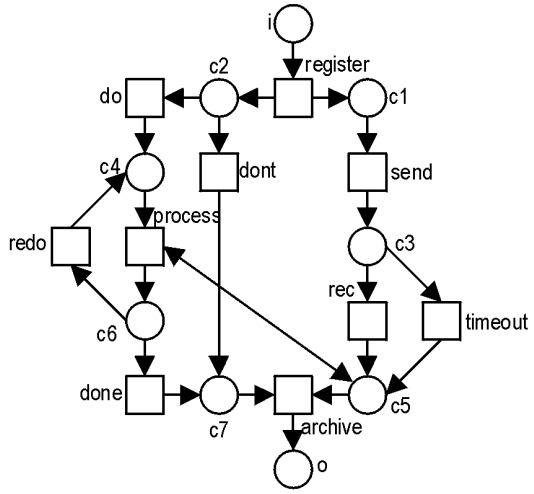


Figure 14: The corrected net with c8 removed

cause c5 and c8 initially have the same number of tokens (initially, they are both empty), this invariant tells us that c5 and c8 will always have the same number of tokens. So, if there is a token in c8, there also must be a token in c5. As a result, we can add arcs from process to c5 and vice versa without changing the behavior of the net. After these arcs have been added, the places c5 and c8 share their preset and their postset: they have become equivalent places. Therefore, one of them can be removed without changing the behavior. If we remove c8, the net becomes as is shown in Figure 14. The net of Figure 14 has the same behavior as the net of Figure 13, assuming [i] as the initial marking. This example shows that Woflan can also be used to simplify WF nets that are already sound.

4.5 Concluding remarks

Woflan can tell whether a given P/T net is a WF net and whether a WF net is sound. However, there are two points that can be improved.

First, in case of a behavioral error, i.e., a violation of the requirements of Definition XXVII, Woflan's diagnosis might not always be helpful in *locating* the error. In the example net N, it was detected that the short-circuited system $\underline{S} = (\underline{N}, [i])$ was not live, but all transitions were reported to be non-live. We may have a similar situation with unboundedness: if a net is unbounded, all places may be unbounded. The diagnosis of the running example also shows that the interpretation of structural properties to solve behavioral errors, like unboundedness and non-liveness, is not straightforward. This problem becomes worse for workflow processes with a larger complexity than the example. Based on these observations, we would like to have additional properties that guide us towards locating behavioral errors.

Second, some errors are easier to detect and to correct than others. It might be the case that correcting one simple error solves several, possibly hard to detect, other errors. For instance, structural errors are often a source for behavioral errors. Thus, if Woflan detects a structural error, there is no need to try to solve behavioral errors: Solving the structural error affects the behavior of the system and might solve the behavioral errors. In case of several related errors in a WF net, we need Woflan to guide us to finding and correcting that error that is most likely the source of the other errors.

To obtain the desired improvements, in the remainder of this paper, we

- introduce behavioral error sequences to help us locate behavioral errors,
- report on a case study which was used to investigate whether these sequences are useful in obtaining the above mentioned goals, and
- introduce a method based on the results of the case study and incorporating behavioral error sequences.

For more information on (older versions of) Woflan, we refer to [2, 5, 7, 19]. The architecture and a brief overview of the functionality of Woflan 1.0 is given in [2, 19]. More information on the interfaces with workflow products such as COSA and Protos can be found in [5, 7].

5 Behavioral error sequences

5.1 Introduction

Structural errors in a P/T net modeling a workflow, i.e., violations of the requirements of Definition XXVI, are generally easy to find and to correct. Behavioral errors, i.e., violations of Definition XXVII, are more difficult to locate and to correct. To overcome this problem, we introduce so-called behavioral error sequences. The idea for these sequences is relatively simple: Determine firing sequences of minimal length, such that *every* continuation of that sequence leads to an error. 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 a workflow designer is interested in, different types of behavioral error sequences can be helpful to diagnose the design. In this section, we introduce a type of behavioral error sequences called *unsound sequences* that are particular useful for diagnosing soundness-related behavioral errors.

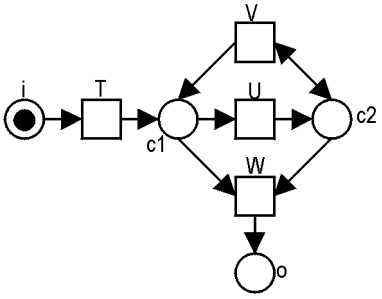


Figure 15: $[o]$ reachable in OG, but not present in CG

5.2 Unsound sequences

In a WF system, a soundness-related behavioral error means that either the case cannot be completed or it can be completed but not properly. Recall that the marking $[o]$ corresponds to proper completion. For defining unsound sequences, we would like to use the OG of the WF system. Given the OG of a WF system $S = (N, [i])$, an unsound sequence is a firing sequence of minimal length from which it is impossible to complete properly, i.e., from which it is impossible to reach the state $[o]$.

However, as mentioned earlier, the OG of a system may be infinite and thus cannot always be constructed. As a result, unsound sequences as defined above cannot always be computed from the OG. In Section 2.3.4, we have introduced the notion of a CG of a system as an alternative for its OG that is always finite. Unfortunately, a CG cannot be used to compute the unsound sequences accurately, as the example system of Figure 15 shows. It is straightforward to see that, in this example system, the firing sequence T U V W leads to state [o] and that the firing sequence T U V V is unsound. However, the state [o] does not even occur in its (only) CG, as can be seen in Figure 16. In this CG, firing sequence T U V W leads to state $[c1^{\omega}, c2^{\omega}, o^{\omega}]$ and not to [o]! According to the CG, proper completion is not possible.

Despite the above observation, we have decided to compute unsound sequences of a WF system from a CG of this system. The CG-generation algorithm is already available in Woflan and there is no straightforward alternative to calculate unsound sequences as defined above accurately. Furthermore, in many cases, a CG is a sufficiently accurate approximation of the OG of a WF system to provide the proper set of unsound sequences.

Let S be a WF system and G a coverability graph of this system. In the remainder we define an (non-empty) unsound sequence as a firing sequence that ends in a marking M

- from which there is no path in G to $[o]$ and
- that has an immediate predecessor M_1 from which there is a path in G to $[o]$.

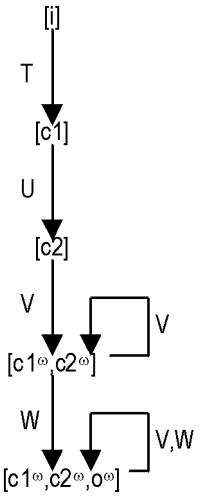


Figure 16: CG of the example from Figure 15

Apparently, the transition which led from marking M_1 to marking M removes the option to complete properly (according to the CG G). To determine which markings in G 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 G 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 properly. Any minimal path from the initial marking $[i]$ to M in G corresponds to an unsound sequence.

Notation (CG reachability)
Let $S = (N, [i])$ be a WF system; let $G = (H, A)$ be a CG of S and let $M, M_1 \in H$ be extended markings of N. The marking M is CG reachable from marking M_1, denoted $M_1 \Longrightarrow M$, iff there exists a path in G from M_1 to M.

Definition XXIX (CG partitions for soundness)
Let $S = (N, [i])$ be a WF system; let $G = (H, A)$ be a CG of S. We partition H into three parts:

i. $H_R = \{M \in H | \neg(M \Longrightarrow [o])\}$,

ii. $H_G = \{M \in H | M \Longrightarrow [o] \wedge \neg \exists M_R \in H_R : M \Longrightarrow M_R\}$, 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 $H_Y = \emptyset$.
- Successors of an infinite marking are always infinite; so, from an infinite marking, $[o]$ is never reachable. All infinite markings are red, which means that all green and yellow markings are finite: $H_G \cup H_Y \subseteq B(P)$.
- Soundness of a WF net implies that all markings are green: $H = H_G$, but not vice versa (there still may be dead transitions!).
- 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 $[o] \in H_G$.

Given a CG of a WF system and the above partitioning of this CG, we can define the unsound sequences formally.

Definition XXX (Unsound sequences)
Let $(N, [i])$ be a WF system with CG $G = (H, A)$. Let H_R and H_Y be defined as in Definition XXIX. If $[i] \in H_R$, then the occurrence sequence $[i]$ is called unsound. An occurrence sequence $s = [i]t_0M_1 \ldots t_{n-2}M_{n-1}t_{n-1}M_n$, for some positive natural number n, is called unsound iff $M_n \in H_R$ and $M_{n-1} \in H_Y$. A *firing* sequence of a WF system is called unsound iff its associated occurrence sequence is unsound.

By examining unsound sequences of a WF system, we can correct the WF net.

Theorem IX (Unsound sequences vs. soundness)
Let $(N, [i])$ be a WF system without dead transitions. Then, N is sound iff $(N, [i])$ has no unsound sequences.

Proof This follows immediately from Definition XXVII (soundness) and Definition XXX (unsound sequences).

The most valuable information in an unsound 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 it is possible that several unsound sequences have the same suffix $M_{n-1}t_{n-1}M_n$.

5.3 Diagnosing the example net

Using the technique of unsound sequences, we diagnose the WF system S of Figure 2. It is interesting to compare this diagnosis with the diagnosis made in Section 4.4.

First, we compute and partition a CG, as is shown in Figure 17. Note that it is not necessary to short-circuit system S for determining unsound sequences. Also note that this CG equals the OG of S, because it contains no infinite markings. The meaning of the thick arrows in Figure 17 is explained in the next subsection. Second, we compute from the partitioned CG the unsound sequences. In this case, there are nine unsound sequences.

i. `register send rec dont`

ii. `register send dont rec`

iii. `register dont send rec`

iv. `register send rec do process redo`

v. `register send do rec process redo`

vi. `register do send rec process redo`

vii. `register send timeout do`

viii. `register send do timeout`

ix. `register do send timeout`

Observe that sequences that are

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

all provide the same diagnostic information. In our example, the sequences ii and iii provide identical information, as well as sequences iv, v, and vi and sequences viii and ix. Thus, it suffices to consider only a single sequence of such a set.

Sequences i and ii tell us that the combination `rec` and `dont` is fatal, because the token in `c8` is not removed, which results in the marking [`c5`, `c7`, `c8`]. Firing transition `archive` in that marking results in the marking [`c8`, `o`]. Note that, due to this error, the place

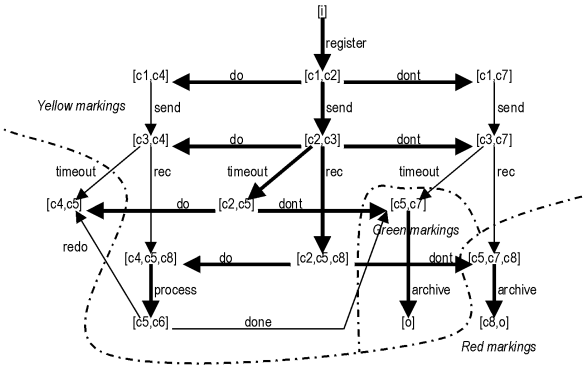


Figure 17: The partitioned CG for system S

c8 is unbounded in the short-circuited system. Requiring that archive removes the token in c8 can repair this error.

Sequences vii and viii tell us that timeout has to put a token in c8; otherwise, the complaint cannot be processed in case the form is not received in time.

Sequence iv tells us that a complaint can only be processed once because no transition is putting a token back in c8 once it is removed. Transition process could take care of this. Note that the pair redo/dont could also take care of this.

The net that results of the suggested corrections is the sound WF net of Figure 13. The above examples clearly indicate that the unsound sequences can be seen as doomsday scenarios. As such, they are particularly useful for tracing the source of an error. They are a useful technique that is complementary to the analysis techniques already implemented in Woflan 1.2 (see Section 4).

5.4 Implementation

The technique of unsound sequences is implemented in a new version of Woflan, namely version 1.3. This version incorporates all the functionality of version 1.2, extending it with information on unsound sequences of a WF system. Based on the observation made in the previous subsection that there may be unsound sequences that provide identical diagnostic information, Woflan 1.3 provides only one of those unsound sequences. In this subsection, we discuss implementation issues concerning Woflan 1.3. When performing the case study as presented in Section 6 with a prototype of Woflan 1.3, we encountered problems with the size of the CGs of workflow systems. Therefore, we implemented in the final release of Woflan 1.3 a restricted CG that still contains sufficient information to compute behavioral properties.

5.4.1 Restricted CG

During the case study, problems arose when trying to construct CGs for the short-circuited WF systems. In particular for *unbounded* short-circuited WF systems, the CGs could be too large to handle by the prototype of Woflan 1.3. As a result, behavioral properties could not be determined. In Woflan 1.3, three kinds of behavioral properties are provided: boundedness-related properties (boundedness and safeness), liveness-related properties (liveness and dead transitions), and behavioral error sequences (unsound sequences).

A simple observation partly alleviates the problem of large CGs: Infinite markings have only infinite successors. For determining boundedness-related properties and behavioral error sequences, it is not necessary to consider successors of infinite markings. This observation leads to the following notion of a restricted CG.

Definition XXXI (Restricted CG) Let $S = ((P, T, F), M_0)$ be a system, let $H \in B^{\omega}(P)$ 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 a *finite* 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 exists a node M_1 in H such that $M_1 \leq M_2$, $M_1(p) < M_2(p)$, and there exists a 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 restricted CG (RCG) of S.

Note that the only difference with the construction of a CG of Section 2.3.4 is 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 4 depicted in Figure 5 with the RCG of Figure 18. 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.

Woflan 1.3 uses an RCG of a short-circuited system $\underline{S}$ to compute boundedness-related and liveness-related properties of $\underline{S}$, whereas Woflan 1.2 uses a CG of $\underline{S}$. This has several consequences.

First, it is clear that $\underline{S}$ is bounded iff there are no infinite markings in the computed RCG. This means that an RCG is sufficient to accurately compute the boundedness property. The same is true for safeness. Recall that

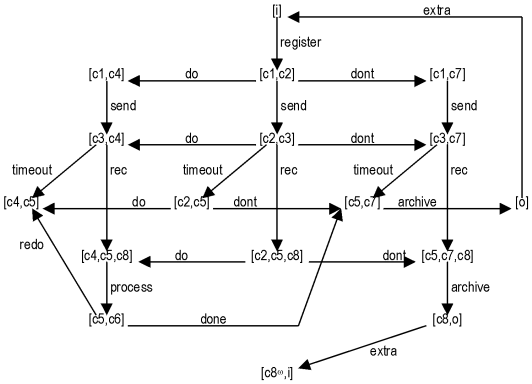


Figure 18: The RCG of the short-circuited example net

Woflan 1.2 also provides detailed information about unbounded places and unsafe places. When the same system is analyzed by Woflan 1.2 and Woflan 1.3, the set of unbounded places found by Woflan 1.3 is a subset of the set found by Woflan 1.2. The reason is that Woflan 1.3 does not find places that are only unbounded because other places are unbounded. This is a positive effect when considering the goals set out in Section 4.4. It is also possible that Woflan 1.3 does not find some places that are unsafe. However, in such cases the unsafeness is always caused by another error.

Second, in case a system is unbounded, Woflan 1.3 omits liveness-related information, because the RCG is not usable to determine non-live transitions or dead transitions. Recall that in case of unbounded systems the liveness property, as presented by Woflan 1.2, is not always accurate in the sense that the set of non-live transitions found by Woflan 1.2 might not be complete. Since for bounded systems the RCG equals the CG and the OG, Woflan 1.3 presents liveness-related information in this case (which is identical to the information as presented by Woflan 1.2).

Finally, it is straightforward to see that an RCG can be used to compute the unsound sequences of a WF system. Consider the partitioning of a CG given in Definition XXIX. 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 unsound sequences. This means that unsound sequences can be computed by applying the partitioning of Definition XXIX to an RCG.

Summarizing, by using an RCG instead of a CG, boundedness-related information becomes more accurate, liveness-related information is unusable in case a system is unbounded, and behavioral error sequences do not change. A clear advantage of using an RCG instead of a CG is that for unbounded systems an RCG is often much smaller than a CG.

5.4.2 Generating an RCG

The routines needed for generating CGs are already available in Woflan 1.2. For navigating a CG, Woflan builds a *spanning tree* for every CG. In general, 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 a CG can be constructed in a straightforward way during the construction of the CG. An example of a spanning tree can be found in Figure 17: The thick arcs constitute a spanning tree.

It is straightforward to adapt the existing routines in Woflan 1.2 such that an RCG (with a spanning tree) is constructed instead of a CG. To compute possible unsound sequences of a system S, Woflan 1.3 first computes an RCG of S. Second, it partitions the computed RCG. Third, it computes the unsound sequences, using the spanning tree to determine a minimal set of sequences as explained in Section 5.3. Finally, to compute boundedness-related and liveness-related properties, Woflan 1.3 extends the RCG of S to an RCG of the short-circuited system $\underline{S}$.

5.4.3 Computing unsound sequences

The algorithm for partitioning an RCG as implemented in Woflan 1.3 is linear in the size (nodes+arcs) of the RCG. An RCG is partitioned by the following steps.

1. Color all markings red.
2. Search for marking $[o]$; if it is found, color it green and, repeatedly, color all red predecessors of 'fresh' green markings green. After this step, which consist of at most two complete traversals of the RCG, the red markings are exactly those markings from which $[o]$ is not reachable (H_R of Definition XXIX); the green markings correspond to the union of the sets H_G and H_Y of Definition XXIX.
3. Search for red markings with an immediate green predecessor; for each such a marking found, color its green predecessors yellow and, repeatedly, color all green predecessors of 'fresh' yellow markings yellow as well. After this step, which also consists of at most two complete traversals of the RCG, the yellow markings are exactly those markings, from which $[o]$ is reachable and a red marking is reachable (H_Y of Definition XXIX). As a result, the green markings are exactly the markings from which it is always possible to reach $[o]$ (H_G of Definition XXIX).

As mentioned before, Woflan uses the spanning tree of an RCG to compute a minimal set of unsound sequences. In general, a CG has many possible spanning

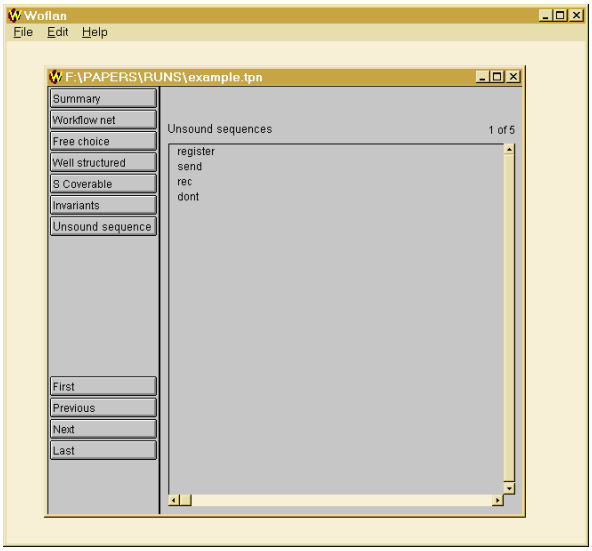


Figure 19: Unsound sequences for system S

trees. The *root* of a spanning tree is the only node that has no incoming arcs. For our application, it is unimportant which spanning tree is used, as long as node $[i]$ is the root. The example spanning tree as depicted in Figure 17 satisfies this requirement.

Given a partitioning of an RCG and a spanning tree with root $[i]$, unsound sequences can be calculated by at most two complete traversals of the RCG. First, Woflan searches for red markings with an immediate yellow predecessor. Second, if it finds such a pair, it traces the path from the yellow marking back to $[i]$ using the spanning tree. This path together with the transition that connects the yellow to the red node forms an unsound firing sequence.

Consider the CG of Figure 17, which is also an RCG, because it contains no infinite markings. Given the depicted spanning tree, we can minimize the set of nine unsound sequences given in Section 5.3 to a set of five unsound sequences, namely the set containing i, ii, iv, vii, and viii. Figure 19 shows sequence i as presented by Woflan 1.3.

5.5 Concluding remarks

The technique for behavioral error sequences appears to be useful in the analysis of WF nets. The technique of unsound sequences is implemented in Woflan 1.3. Woflan 1.3 uses a variant of a CG called a restricted CG (RCG) to compute behavioral properties of a WF system. The main reason for using an RCG is that it improves performance when compared to using a CG. The complexity of the algorithm to compute unsound sequences for a system is linear in the size of the computed RCG of the system.

An interesting future extension of Woflan is the visualization of behavioral error sequences. A good way to visualize sequences is by using so-called *runs* [12]. The set of runs of a P/T net is a compact partial-order-based representation of the semantics of the P/T net. An interesting aspect of a run is that it can be visualized in an intuitive way as a P/T net itself.

In the running example, the five sequences calculated by Woflan 1.3 (sequences i, ii, iv, vii, and viii of Section 5.3) can be displayed graphically by only three runs, which are shown in Figure 20. The first run embeds the sequences i and ii; the second one corresponds to sequence iv and the third run embeds the sequences vii and viii. At the moment, work is being done on an interface between Woflan and VIPtool. VIPtool [14] is a software package developed by members of AIFB of the University of Karlsruhe that is capable of generating and visualizing runs for both P/T systems and colored-Petri-net systems.

6 Case study: travel agency

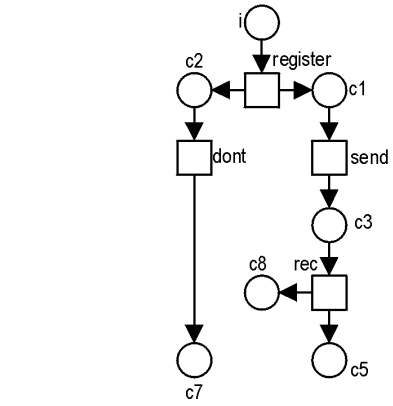
6.1 Introduction

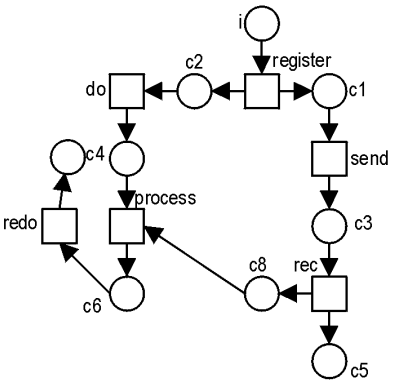
To test the applicability of Woflan 1.3 in general and its extension with unsound sequences in particular, we performed a case study. Given a description of a travel agency at a university (see Appendix A.1), twenty groups of students had to model the workflow as a final assignment for a course on workflow management. Fourteen of these groups consisted of industrial engineers from the Eindhoven University of Technology; the other six consisted of computing-science engineers of the University of Karlsruhe. For this assignment, the students involved used Protos (Pallas Athena) as a modeling tool (see Figure 21).

Using Protos' export facility to Woflan it is easy to analyze a Protos model with Woflan (see Figure 22).

From the Eindhoven collection of reports, we selected eleven reasonably looking solutions; three reports were so poor that analyzing the Protos model by means of Woflan was not very meaningful. From the Karlsruhe collection, all reports were selected. We analyzed the selected nets using Woflan 1.3 and tried to correct them if necessary, i.e., we tried to get them sound if they were not. The number of transitions of the models analyzed ranges from 54 to 89 and the complexity calculated by Woflan ranges from 43 to 70. These numbers show that the case study was performed on workflow models of more than reasonable size. An example of a sound Protos model can be found in Appendix A.2.

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





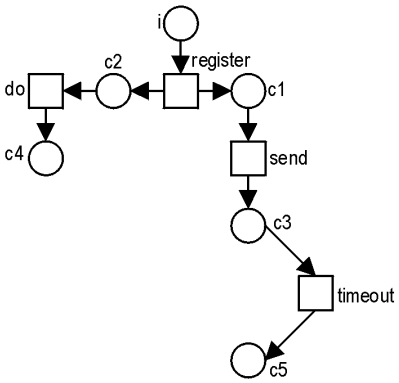


Figure 20: Three runs embedding all five unsound sequences

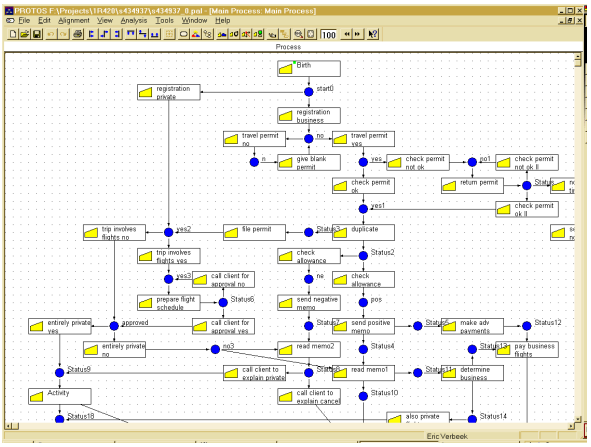


Figure 21: Example Protos model

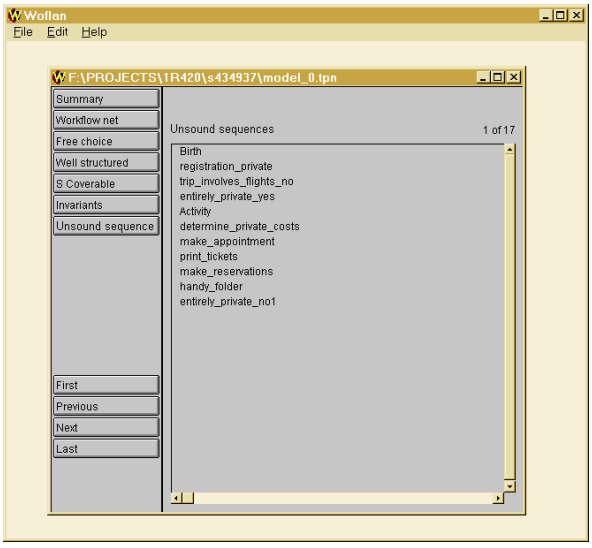


Figure 22: Example Woflan diagnosis

6.2 Diagnosis

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 system by hand (play the *token game*) to test their model, not one of them was able to produce a model which was sound.

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 Woflan 1.2 on small examples. However, none of the groups used Woflan to check their solution to the assignment. In the end, the Karlsruhe groups delivered better nets than the Eindhoven groups. Of the seventeen nets, five appeared to be sound, which were all from Karlsruhe groups. Table 1 shows an overview of our efforts to diagnose the workflow processes in the seventeen reports. It contains the following information:

- The elapsed time, i.e., the estimated time it took to produce a sound WF net.
- The number of iterations needed to produce a sound WF net.
- Diagnostic information about the initial net.
- Diagnosis of the initial net.

For the diagnosis, the following keywords are used (see also Section 3.2):

- OR-split: an AND-split (start of parallel branches) should have been an OR-split (alternative branches).
- OR-join: an AND-join (end of parallel branches) should have been an OR-join (end of alternative branches).
- AND-join: an OR-join should have been an AND-join.
- Attributes: attributes were used to keep different but dependent choices consistent. Woflan cannot handle attributes; they have to be modeled explicitly in the workflow.
- Flow error: Arcs had to be added or deleted between existing nodes.

Group	Models			Diagnosis	
	Elapsed time (h:m)		Symptoms		University
1	0:11	3	Unbounded, not live	Attributes, flow error	Eindhoven
2	0:02	1	Unbounded	OR-split	Eindhoven
3	2:39	10	Unbounded, not live	Attributes, OR-split, flow errors	Eindhoven
4	1:08	1	Unbounded, not live	Attributes, OR-joins	Eindhoven
5	1:04	7	Unbounded, not live	Attributes, flow errors	Eindhoven
6	0:20	3	Unbounded, not live	Attributes, OR-split, flow error	Eindhoven
7	0:17	3	Not live	Attributes, OR-join	Eindhoven
8	0:52	6	Unbounded, not live	Attributes, OR-splits, flow errors	Eindhoven
9	0:20	3	Unbounded, not live	Attributes, OR-join, AND-join	Eindhoven
10	1:28	12	Not live	Attributes, OR-joins, flow errors	Eindhoven
11	0:01	1	Unbounded	OR-split	Eindhoven
12	0:14	1	Not live	Attributes	Karlsruhe
13	0:00	0	None	Sound	Karlsruhe
14	0:00	0	None	Sound	Karlsruhe
15	0:00	0	None	Sound	Karlsruhe
16	0:00	0	None	Sound	Karlsruhe
17	0:00	0	None	Sound	Karlsruhe

Table 1: Overview of the results of the case study

In ten out of seventeen models, attributes were used to keep dependent choices consistent: nine models of Eindhoven and one of Karlsruhe. We had to incorporate these attributes explicitly in the models. In the Eindhoven course, the use of attributes to keep dependent choices consistent was explicitly allowed, which means that it cannot be counted as a real mistake. However, all nine Eindhoven models that used attributes in this way also contained other errors. In the Karlsruhe course, the use of attributes to keep dependent choices consistent was explicitly disallowed.

The short-circuited systems corresponding to nine of the models that students handed in appeared to be unbounded. As mentioned earlier, the first prototype of Woflan 1.3 had problems constructing CGs of these systems. For this reason, it is interesting to have a look at the number of extended markings in these CGs. The prototype of Woflan 1.3, running on the computer used for the case study, was able to handle at least 100000 markings in a CG. For the nine models with unbounded short-circuited systems, Table 2 shows the exact num-

Group	First
1	100000+
2	100000+
3	100000+
4	89138
5	8817
6	100000+
8	100000+
9	100000+
11	297

Table 2: Initial CG sizes for the initially unbounded systems

Group	First	RCG	Factor
1	100000+	3371	29+
2	100000+	488	204+
3	100000+	7388	13+
4	89138	257	346.84
5	8817	4568	1.93
6	100000+	2585	38+
8	100000+	452	221+
9	100000+	619	161+
11	297	126	2.36

Table 3: CG size vs. RCG sizes for the initially unbounded systems

ber of markings in the CG as computed by the prototype or the entry `100000+` if the number of markings in the computed CG exceeded 100000.

In Section 5.4, we claimed that for unbounded systems the number of markings in an RCG is often significantly smaller than the number of markings in a CG. Table 3 confirms this claim.

Using Woflan 1.3, we were able to diagnose and correct all the seventeen models in reasonable time and with reasonable effort. An interesting observation is that we mainly used the technique of unsound sequences, the main reason being that the complexity of the models complicated the interpretation of the structural properties of the models.

6.3 Concluding remarks

The main conclusion is that Woflan 1.3 and in particular the technique of behavioral error sequences is very useful in the analysis of complex workflow processes. An interesting discovery during the case study was that the size of a CG of the (short-circuited) systems corresponding to the workflow processes, although finite, might still be a problem. Using an RCG instead of a CG alleviated this problem. The workflow process that is the basis of this case study is a fairly complex one. In our practical experience with workflow modeling, most workflows we encountered had simpler process definitions. This means that the current techniques implemented in Woflan 1.3 appear to be sufficiently powerful to handle practical workflow processes, despite the theoretical complexity of the (R)CG algorithm.

As mentioned before, for bounded systems, all behavioral properties can be checked on an RCG. However, when a system is unbounded, some behavioral properties cannot be checked using an RCG. Combined with our experience with the case study, it is a good idea to correct errors causing unboundedness before trying to correct other types of behavioral errors. This observation is incorporated in our method, which is introduced in the next section.

Another interesting observation is that many groups used attributes to keep dependent choices consistent. Recall that a workflow attribute is a specific piece of information used for the routing of a case. As explained in Section 3.3, a WF net abstracts from workflow attributes. One of the reasons for abstracting from workflow attributes is that properties of the WF net that are valid under the assumption of non-deterministic choices are also valid when choices are based on workflow attributes. The case study shows that it is not always possible to prove soundness of a WF net assuming non-deterministic choices. Our solution was to manually encode the value of workflow attributes explicitly by means of places. In real-world applications, information about workflow attributes should be available in the workflow definition made in for example Protos or COSA. It is interesting to investigate whether it is possible to automate the encoding of workflow attributes in WF nets to some extent when importing process definitions in Woflan.

A final conclusion is that the industrial-engineering students of Eindhoven did not produce a single correct workflow, whereas the computing-science-engineering 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-engineering students are trained in both skills. Many designers of workflow processes in practice have also little experience in formal verification. Thus, 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.

7 Method

7.1 Introduction

As explained, we want to have a method that supports Woflan in guiding the user towards the most basic er-

rors in a net. Based on the results of our experiment, we want to construct and use an RCG instead of a CG to check behavioral properties. Therefore, it seems reasonable to check whether a system is bounded as soon as possible: If a system is bounded, the RCG-generation algorithm yields the OG, which means that other behavioral properties can be verified accurately. Boundedness can easily be checked by means of an RCG. Although an RCG is usually small enough to compute, we still want to minimize its use. If possible, we want to use structural properties instead, because they can be computed much more efficiently.

The current version of Woflan provides an overview of diagnostic information about a P/T net. The goal of the method is to provide the workflow designer with appropriate diagnostic information at the right time and in the right order. Most of the diagnostic techniques used in the method are already implemented in Woflan 1.3. However, Woflan does not yet support the method itself. The order in which techniques are applied is based on two criteria, namely efficiency of the technique and usefulness of the information.

7.2 Milestones

Based on the above observations, our method consists of the following three milestones. Let N be a P/T net.

1. The net N is a WF net.
2. The short-circuited system $(\underline{N}, [i])$ is bounded, which implies proper completion of N.
3. The short-circuited net $(\underline{N}, [i])$ is live, which implies the option to complete for N and the absence of dead tasks.

If we cannot achieve a milestone, we do not attempt to achieve the next milestones. So, we will not attempt to prove liveness, if we do not already have boundedness.

7.2.1 WF net

The first milestone is very straightforward. It covers anything which is not in accordance with the given requirements for a WF net (see Definition XXVI and Section 4.2). Woflan guides the user towards correcting these errors by providing the information explained in Section 4.2.1.

A special error is when the net is empty. In this case, the conversion from the native file format to the TPN file format (the format used by Woflan [34]) possibly failed.

7.2.2 Boundedness

At this point, we know that the net N under consideration is a WF net, because the WF-net milestone is achieved. To achieve the boundedness milestone, the short-circuited WF system $\underline{S} = (\underline{N}, [i])$ needs to be bounded. As explained, structural analysis techniques are generally more efficient than behavioral ones. Therefore, we try to prove boundedness of $\underline{S}$ by means of the following structural property. A net is *structurally* bounded iff for every possible initial marking the corresponding system is bounded.

Definition XXXII (Structurally bounded)
A P/T net $N = (P, T, F)$ is structurally bounded iff $\forall M \in B(P) : (N, M)$ is bounded.

To help us decide whether or not $\underline{N}$ is structurally bounded, we have three structural properties at our disposal:

i. Whether $\underline{N}$ is S-coverable;

ii. whether $\underline{N}$ is covered by safe place-invariants (place-invariants containing only weights 0 and 1); and

iii. whether $\underline{N}$ is covered by semi-positive place-invariants (see Section 4.2.5).

S-coverability

It is possible to decide in polynomial time whether or not net $\underline{N}$ is S-coverable. If $\underline{N}$ is S-coverable, it is structurally bounded, which means that $\underline{S}$ is safe and bounded (Theorem VI). Thus, we have proven the milestone and we can continue with the third milestone.

A place that is not S-coverable is a suspicious place. Such a place is called *uncovered*. If a net is not S-coverable, Woflan provides the following

Diagnostic information:

- Uncovered places (see Section 4.2.4).

Recall Theorem III which states that $\underline{N}$ is S-coverable if N is a sound, free-choice WF net. In other words, if $\underline{N}$ is not S-coverable, then either N is not sound or it is not free-choice. So, if $\underline{N}$ is not S-coverable, it is a good moment to test the free-choice property: If N is free-choice, it cannot be sound. In this case, Woflan provides the following

Diagnostic information:

- Non-free-choice clusters (see Section 4.2.2).

In a similar way, using Theorem V, we test the well-structuredness property: If $\underline{N}$ is not S-coverable and N is well-structured, N cannot be sound.

Diagnostic information:

- TP-handles (see Section 4.2.3).

If a correction is made based on the diagnostic information of Woflan, the diagnosis of the corrected net starts again with the first step of the method. However, if we deduce from one of the above steps that N is not sound, there is a good chance that we still do not know *why* N is not sound. If this is the case, the diagnosis can be continued with the next step.

Safe place-invariants

At this point, we know that $\underline{N}$ is not S-coverable. The next step is to decide whether or not all places in net $\underline{N}$ occur in some *safe* place-invariant, which can be done in polynomial time. A place is *covered* by a safe place invariant iff it occurs in such an invariant with weight one. If all places in $\underline{N}$ are covered by a safe place-invariant, the net $\underline{N}$ is structurally bounded.

A place that is not covered by a safe place-invariant *might* be unsafe. Such a place is called a *structurally unsafe* place. From a workflow point of view, this means that a condition might be fulfilled more than once at a single point in time. A structurally unsafe place cannot be S-coverable: Every S-component corresponds to some safe place-invariant. However, a place that is not S-coverable, might be structurally safe. Therefore, this check is less selective than the check on S-coverability.

Diagnostic information:

- Structurally unsafe places.

Semi-positive place-invariants

At this point, we know that there are structurally unsafe places. The next step is to decide whether or not all places in net $\underline{N}$ occur in some *semi-positive* place-invariant. If all places in $\underline{N}$ are covered by a semi-positive place-invariant, the net $\underline{N}$ is structurally bounded, which means that $\underline{S}$ is bounded although it need not be safe. Because $\underline{S}$ is not necessarily safe if it satisfies this check, this check is less discriminating than the previous two checks.

Places that are not covered by a semi-positive place-invariant might be unbounded. From a workflow point of view, this means that a condition *might* be fulfilled an arbitrary number of times. Such a place is called *structurally unbounded*.

Diagnostic information:

- Structurally unbounded places.

Unbounded sequences

If we cannot decide boundedness of our short-circuited workflow system $\underline{S}$ by proving structurally boundedness of workflow net $\underline{N}$, we have to decide whether or not $\underline{S}$ is bounded by means of behavioral properties. For this purpose, we introduce a new type of behavioral error sequences called *unbounded* sequences. The basis of unbounded sequences is the following theorem.

Theorem X (Boundedness of short-circuited WF systems)
Let $S = ((P, T, F), [i])$ be a WF system. System $\underline{S} = ((\underline{P}, \underline{T}, \underline{F}), [i])$ is bounded iff system S is bounded $\wedge \forall M \in B(P), [i] \longrightarrow M : \neg(M > [o])$.

Proof
To prove the theorem, we prove that $\underline{S}$ is unbounded iff S is unbounded $\vee \exists M \in B(P), [i] \longrightarrow M : M > [o]$. According to the definition of boundedness (Definition XXV), we have to prove that $\exists \underline{M}, \underline{M_1} \in B(P) : [i] \longrightarrow \underline{M} \longrightarrow \underline{M_1} \wedge \underline{M_1} > \underline{M}$ iff $\exists M, M_1 \in B(P) : [i] \longrightarrow M \longrightarrow M_1 \wedge M_1 > M \vee \exists M \in B(P), [i] \longrightarrow M : M > [o]$. The implication from right to left is straightforward (see also Theorem VII). 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 $k < n$. Then, $M_{k-1} \geq [o]$, i.e., either $M_0 = [i] = M_k$ or $M_0 < M_k$. In both cases, the minimality of s is violated. Thus, $\underline{t}$ equals t_n. It follows from the definition of $\underline{t}$ and the minimality of 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 $M_{n-1} > [o]$.

Intuitively, an unbounded sequence is a firing sequence of S of minimal length which inevitably leads either to an infinite marking in a given RCG of S or to a marking greater than $[o]$ in that RCG. The above theorem means that such a sequence corresponds to a sequence of $\underline{S}$ that inevitably leads to an infinite marking when the RCG of S is extended to an RCG of $\underline{S}$.

At this point in our method, unbounded sequences are more appropriate than unsound sequences (see Section 5.2). Unbounded sequences provide only diagnostic information on unboundedness, whereas unsound sequences provide mixed information on unboundedness and non-liveness.

To calculate unbounded sequences, we have to partition a given RCG of S in a way that is slightly different from the partitioning given in Definition XXIX:

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 form which infinite markings or markings greater than $[o]$ are reachable but avoidable.

Definition XXXIII (RCG partitions for unboundedness)
Let $N = (P, T, F)$ be a WF net, let $G = (H, A)$ be an RCG of WF system $(N, [i])$, and let $H^\omega = H \setminus B(P)$ be the set of markings in H that are infinite or greater than $[o]$. We partition H into three parts:

i. $H^\omega_G = \{M \in H | \neg\exists M_1 \in H^\omega : M \Longrightarrow M_1\}$,

ii. $H^\omega_R = \{M \in H | \neg\exists M_1 \in H^\omega_G : M \Longrightarrow M_1\}$ 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 $H^\omega_Y = \emptyset$.
- Boundedness of $\underline{S}$ implies that all markings are green ($H = H^\omega_G$) and vice versa.

Given this partitioning of an RCG of a WF system S, we can compute the unbounded sequences.

Definition XXXIV (Unbounded sequences)
Let $(N, [i])$ be a WF system with RCG (H, A). Let H^ω_R and H^ω_Y be defined as in Definition XXXIII. If $[i] \in H^\omega_R$, then the occurrence sequence $[i]$ is called unbounded. An occurrence sequence $s = [i]t_0M_1 \ldots t_{n-2}M_{n-1}t_{n-1}M_n$, for some positive natural number n, 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 its associated occurrence sequence is unbounded.

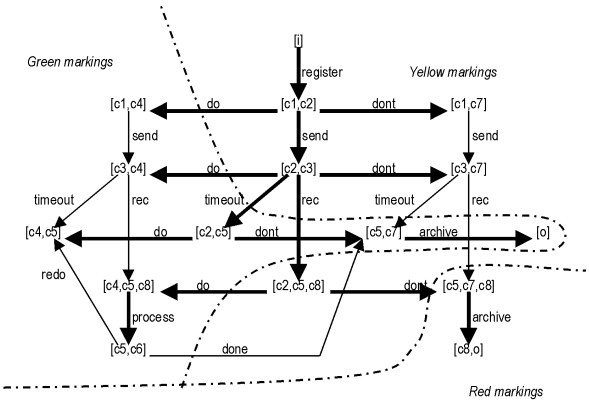


Figure 23: The RCG partitioned for unboundedness

Theorem XI (Unbounded sequences vs. boundedness)
A WF system $\underline{S}$ is bounded iff S has no unbounded sequences.

Proof
This follows immediately from Theorem X (Boundedness of short-circuited WF systems) and Definition XXXIV (Unbounded sequences).

Figure 23 shows the partitioned RCG of the example system S of Figure 2. Using the spanning tree of this partitioned RCG, we compute the following minimal set of (see Section 5.3) unbounded sequences:

- `register send rec dont` and
- `register send dont rec`.

If a WF system S has no unbounded sequences, then $\underline{S}$ is bounded (Theorem X), which means that we have achieved the second milestone. In this case, we can continue with the next milestone. If S has unbounded sequences, Woflan provides the following

Diagnostic information:

- Unbounded sequences.

At this point, we have proven $\underline{S}$ to be unbounded. Thus, we have to make a correction and start again with the first step of the method. The technique of unbounded sequences is not yet implemented in Woflan 1.3. However, it is straightforward to adapt the algorithm to compute unsound sequences as presented in Section 5.

7.2.3 Liveness

At this point, we know that the system $\underline{S}$ is a bounded, short-circuited WF system. We have to decide whether

or not $\underline{S}$ is live. The information that is already available is in some cases sufficient to deduce that $\underline{S}$ is non-live, namely in the case that the underlying net $\underline{N}$ is not S-coverable (see **S-coverability**). In this case, Woflan continues with the step **Free-choice property**; otherwise Woflan skips the steps **Free-choice property** and **Well-structuredness** and continues with the step **Occurrence graph**. Currently, we do not know of any structural technique to prove liveness.

Free-choice property

It follows from Theorem III and the fact that $\underline{S}$ is bounded that $\underline{S}$ cannot be live if N is free-choice and $\underline{N}$ is not S-coverable.

Diagnostic information:

- Uncovered places.

Well-structuredness

It follows from Theorem V and the fact that $\underline{S}$ is bounded that $\underline{S}$ cannot be live if N is well-structured and $\underline{N}$ is not S-coverable.

Diagnostic information:

- Uncovered places.
- PT-handles.

In both cases, the diagnostic information might be sufficient to correct the net, in which case we start the method again with the corrected net. In case non-liveness has been proved and the information so far is not sufficient, Woflan continues with the step **Dead transitions**.

Occurrence graph

At this point, we know that both systems S and $\underline{S}$ are bounded. Recall that the RCG-generation algorithm yields the OG of a system if it is bounded. Furthermore, note that the OG of S might already be available from the boundedness milestone. Also note that, because $\underline{S}$ is bounded, either the OGs of $\underline{S}$ and S are identical or the OG of $\underline{S}$ extends the OG of S with the arc $([o], \underline{t}, [i])$. Thus, at this point, Woflan computes the OG of $\underline{S}$ and determines whether the liveness-property is satisfied.

In case liveness is proven, we have shown that N is sound, which means that our diagnosis is complete. In case $\underline{S}$ is not live, we are interested in detailed diagnostic information on non-liveness.

Dead transitions

At this point, we know that $\underline{S}$ is not live. In case non-liveness is caused by dead transitions in S, we want to remove these as soon as possible.

Theorem XII (Dead transitions in bounded WF systems)
Let $S = ((P, T, F), [i])$ be a bounded WF system; let $t \in T$. Transition t is dead in S iff it is dead in $\underline{S}$.

Proof
The result follows immediately from the earlier observation that either the OGs of $\underline{S}$ and S are identical or the OG of $\underline{S}$ extends the OG of S with the arc $([o], \underline{t}, [i])$.

To decide whether there are dead transitions, the OG of the system S is necessary. If the OG is not already available, it is calculated at this point.

Diagnostic information:

- Dead transitions.

Non-live sequences

At this point, we know that $\underline{S}$ is bounded and contains no dead transitions, but that it is not live. To provide useful diagnostic information, we introduce another type of behavioral error sequences, called non-live sequences. These sequences are based on the following theorem.

Theorem XIII (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 theorem follows in a straightforward way from Definition XXVII (Soundness), Theorem I (Soundness vs. boundedness and liveness), and Theorem X (Boundedness of short-circuited WF systems).

Intuitively, a non-live sequence is a firing sequence of S of minimal length that ends in a marking from which it is no longer possible to reach $[o]$. To compute non-live sequences, we partition the OG of S as follows.

Definition XXXV (OG partitions for non-liveness)
Let $N = (P, T, F)$ be a marked WF net, let $G = (H, A)$ be the OG of its WF system $(N, [i])$. We partition H into three parts:

i. $H^o_R = \{M \in H | \neg(M \Longrightarrow [o])\}$,

ii. $H^o_G = \{M \in H | M \Longrightarrow [o] \wedge \neg\exists M_R \in H^o_R : M \Longrightarrow M_R\}$ and

iii. $H^o_Y = H \setminus (H^o_G \cup H^o_R)$.

Definition XXXVI (Non-live sequences)
Let $(N, [i])$ be a WF system with OG $G = (H, A)$. Let H^o_R and H^o_Y be defined as in Definition XXXV. If $[i] \in H^o_R$, then the occurrence sequence $[i]$ is called non-live. An occurrence sequence $s = [i]t_0M_1 \ldots t_{n-2}M_{n-1}t_{n-1}M_n$, for some positive natural number n, is called non-live iff $M_n \in H^o_R$ and $M_{n-1} \in H^o_Y$. A *firing* sequence of a WF system is called non-live iff its associated occurrence sequence is non-live.

Theorem XIV (Non-live sequences vs. liveness)
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 S has no non-live sequences.

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

Diagnostic information:

- Non-live sequences

Using non-live sequences, we have to correct the WF net under consideration and start again with the first step of our method. In the next subsection, we apply the method to our running example. This subsection also contains examples of non-live sequences.

7.3 Example

Although the method and the techniques of unbounded sequences and non-live sequences have not been implemented yet in Woflan 1.3, our running example of Figure 1 is small enough to apply the method by hand. Note that this net is diagnosed in Section 4.4 using Woflan 1.2.

7.3.1 WF net

Net N is a WF net, which is straightforward to check.

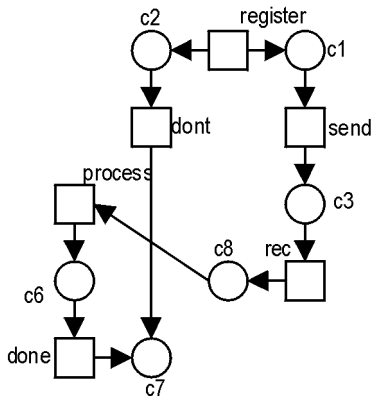


Figure 24: A TP-handle in N

7.3.2 Boundedness

S-coverability

Net $\underline{N}$ is not S-coverable (see Figure 10 for the S-components).

Diagnostic information:

- Uncovered places: c8.

Net N is free-choice, so it cannot be sound.

Diagnostic information:

- Non-free-choice clusters: None.

Net N is not well-structured.

Diagnostic information:

- TP-handles: See Figure 9 and Figure 24. Note that both handles contain the uncovered place c8.

There is obviously something wrong with place c8. At this point, we can either correct the net or continue with our method. Suppose, we still do not know which correction to make. We do observe that either the net has to become S-coverable, or we have to introduce non-free-choice clusters. We continue with the method.

Safe place-invariants

Net $\underline{N}$ is not covered by safe place-invariants.

Diagnostic information:

- Structurally unsafe places: c8.

We already suspect place c8 because it is uncovered.

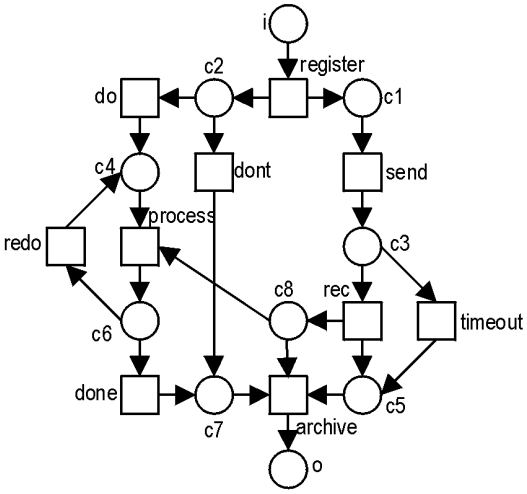


Figure 25: Net N after one correction iteration: N_1

Semi-positive place-invariants Net N is not covered by semi-positive place-invariants.

Diagnostic information:

- Structurally unbounded places: c8.

Unbounded sequences System S has unbounded sequences, which means S is unbounded.

Diagnostic information:

- Unbounded sequences: register send rec dont and register send dont rec.

Both sequences result in the marking [c5, c7, c8]. Firing transition archive from that marking results in marking [c8, o]. A this point of our method, we have to make a correction. Transition archive must remove the token in c8. After correcting the net N by adding this arc, the resulting net N_1 is shown in Figure 25. We start our method again on this net.

7.3.3 WF net

Net N_1 is a WF net.

7.3.4 Boundedness

S-coverability

Net N_1 is not S-coverable.

Diagnostic information:

- Uncovered places: c8.

Net N_1 is not free-choice.

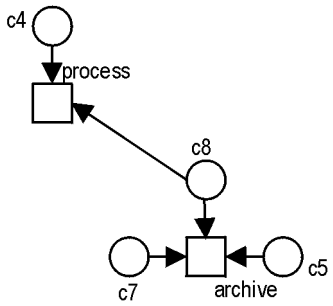


Figure 26: A non-free-choice cluster in N_1

Diagnostic information:

- Non-free-choice clusters: See Figure 26.

Net N_1 is not well-structured.

Diagnostic information:

- TP-handles: See Figure 9 and Figure 24, both TP-handles are still present in N_1.

Safe place-invariants

Net N_1 is not covered by safe place-invariants.

Diagnostic information:

- Structurally unsafe places: c8.

Semi-positive place-invariants

Net N_1 is not covered by semi-positive place-invariants.

Diagnostic information:

- Structurally unbounded places: c8.

Unbounded sequences

System S_1 has no unbounded sequences. Therefore, S_1 is bounded.

7.3.5 Liveness

Free-choice property

As mentioned before, net N_1 does not satisfy the free-choice property.

Diagnostic information:

- Uncovered places: c8.

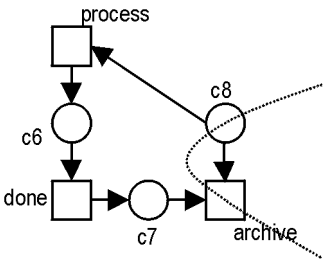


Figure 27: Second PT-handle in N_1

Well-structuredness

Net N_1 is also not well-structured.

Diagnostic information:

- Uncovered places: c8.
- PT-handles: See Figure 12 and Figure 27. Note that the PT-handle displayed in Figure 27 is not present in the original net N.

Occurrence graph

The system S_1 is not live.

Dead transitions

System S_1 has no dead transitions.

Diagnostic information:

- Dead transitions: None.

Non-live sequences

After partitioning the OG of S_1 according to Definition XXXV, it looks as in Figure 28. Using the spanning tree, we deduce a minimal set of five non-live sequences:

i. register send rec do,

ii. register send timeout,

iii. register send dont timeout,

iv. register send do, and

v. register do.

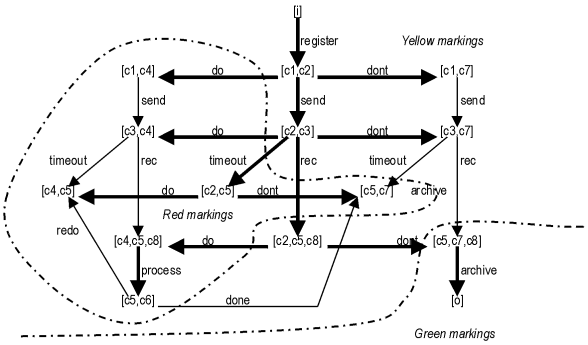


Figure 28: The OG of S_1 partitioned for non-live sequences

By examining these five firing sequences, we note that iii provides almost the same information as ii, where the combination send and timeout is crucial and dont is not important. From sequence ii, we conclude that, whatever happens, place c8 will not get a token. As a result, transitions process and archive cannot fire. To correct this error, we add an arc from timeout to c8.

The sequences i, iv, and v provide the same information, namely that firing transition do always results in an error. From sequence v, we conclude that the cycle to which do leads might be the problem. Recall that place c8 is uncovered. Considering the cycle and place c8 leads to the observation that the cycle can only be executed once, because c8 is only an input place (and not an output place) of the cycle. Also the PT-handle of Figure 27 suggests that there is a problem with c8. We correct the net by adding an arc from process to c8.

Applying the two corrections mentioned above results in the sound net we found earlier in Figure 13. Obviously, applying the method to this net shows that it is sound.

8 Concluding remarks 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 the tool Woflan and a diagnosis method that are both based on Petri-net techniques. Although the method has not yet been implemented in Woflan, it looks promising and most of the necessary diagnostics are already available in Woflan 1.3. We have tested

Woflan in a case study involving seventeen models of a fairly complex workflow designed by students. A novel analysis technique of behavioral error sequences proved to be a useful aid in diagnosing the workflows. Twelve unsound workflows could be corrected in reasonable time with reasonable effort. The experience with the case study was a source of inspiration for the method.

An interesting conclusion of the case study is that a coverability graph of a system representing an erroneous workflow can be too large to handle by Woflan. By introducing and using a *restricted* coverability graph, we have alleviated this problem.

In version 2 of Woflan, we want to implement the method presented in Section 7. Furthermore, we would like to test the method on more, practical, examples.

We are also working on extending the set of workflow tools Woflan can interface with. The current version of Woflan (Woflan 1.3) can import workflow process definitions of COSA and Protos. On paper, we have also designed translations from BaanERP/DEM (BaaN), Staffware (Staffware), 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. The translation of Staffware to WF nets is described in [6]. 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 [8]. In the future, we plan to build the corresponding interfaces.

Furthermore, we want to visualize Woflan's output in a graphical way. There are several ways to display the diagnostics in a graphical manner: either via diagrams shown directly by Woflan, via dedicated tools such as VIPtool [14], or via an interface in the workflow tool used to design the workflow process.

A direction for future research is the use of the inheritance-preserving transformation rules presented in [11] for incremental design and verification of workflows. Starting from a correct workflow template [27] 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, and Jaap van der Woude for their fruitful comments.

A Travel agency

Section A.1 contains the description of a travel agency at a university. The models mentioned in the case study (Section 6) are all based on this description. Section A.2 gives one possible formalization of the workflow process used in the travel agency. The model is made in Protos.

A.1 Informal description

Some time ago the board of Somewhere University (SU) decided to open a travel agency at the campus. The new agency is supposed to organize both business and private trips for employees of SU. However, the service is not as the board expected. The most important complaint is that both the organization of a trip and the financial settlement take too long. Therefore, the board has started an investigation. Interviews with several people involved have provided the following process description. (To avoid confusion between employees of SU that want to book a trip and employees that are involved in the organization of the trip, in the remainder, the former are called clients.)

The whole process starts when someone drops in at the travel agency to book a trip. An employee of the agency registers all the relevant information of the client. The agency maintains a separate file for each trip. An important issue is whether the client wants to book a private trip, a business trip, or a combination of both. Approximately 20 percent of all the trips organized by the agency is private.

Private trips are easy. The agency has one employee dedicated to the organization of private trips. As soon as the wishes of a client are registered, she can start with the organization of the trip.

Business trips are more complicated. The agency has two employees for the organization of business trips (although one of them works only three days a week). For each trip, there is always a single employee responsible, who also carries out as many tasks as possible for this trip. In this way, the service to clients should be guaranteed. For business trips, a client needs a travel permit. Usually, clients that are familiar with the process have already filled out a permit. Clients that arrive without a permit are given a blank permit that they can fill out later, after which they must return the permit to the agency. Travel permits are always checked before any other action is taken. If a permit is not filled

out properly, it is returned to the client with the request to provide the missing information and send the permit back as soon as possible. In case a permit is not returned in time, the travel agency can no longer guarantee a timely organization of the trip. In the rare occasion that this happens, a notification is sent to the client and the file is closed. If a travel permit is okay, it is filed and the actual organization of the trip can start. First, however, a copy of the file is sent to the finance department of SU, because this department is responsible for the financial aspects of the trip.

An employee of the finance department of SU checks whether the client is allowed to make business trips paid by SU. The results of this check are sent to the travel agency in an internal memo. If the result is negative for the client, which is hardly ever the case because clients usually know when they are permitted to make business trips, the finance department does not make any payments. If the result is positive, the finance department makes an advance payment on the bank account of the client. It also pays any registration fees that might need to be paid in case of conference visits. Finally, it pays those flights of the trip that are made for business purposes. However, these payments can only be made after the finance department has received detailed pricing information from the travel agency. After all the necessary payments have been made, the finance department is no longer involved in the preparations of the trip. However, after the client returns, the finance department handles the client's declaration (see below).

To prepare a trip (private or business), the travel agency always starts with flight arrangements. If a trip involves one or more flights, the responsible employee of the travel agency starts by preparing a flight schedule that includes departure and arrival times of all flights as well as pricing information. Then, the client is called to approve the schedule. If the client does not approve the schedule, a new proposal is prepared and the client is contacted again. When a client approves the schedule, arrangements must be made to pay the flight(s). In case the trip is private, an appointment is made with the client to pay cash or by credit card. In case the trip is (partly) business, the travel agency has to wait for the memo of the finance department which states whether or not the client is allowed to make business trips for SU. If the memo is negative, the employee of the travel agency responsible for the trip calls the client to explain the problem. If the client still wants to make the trip, he or she has to pay all the costs and an appointment is made to pay for the flights. However, often the client decides to cancel the trip, in which case the file is closed. If the memo is positive, the travel agency determines the costs of business flights and, if applicable, the costs of private flights. Relevant information on business flights is sent to the finance department that handles the actual payment. In case of private flights, the client is contacted to make an appointment to arrange the payment.

The internal memo that the travel agency receives from the finance department, is also used to determine whether a request must be sent to the in-house bank office (which is situated at the campus close to the travel agency) to prepare cash and travel cheques for the client. Such a request is always made when a business trip is allowed. (In case of private trips, the client has to take care of acquiring cash and cheques him- or herself.)

The task of the bank in the process is very straightforward. Upon receipt of a request, a bank employee prepares cash and travel cheques and sends them to the travel agency. If a client returns cash and/or cheques after the trip, information about the exact amount that is used by the client is sent to the finance department. The finance department needs this information to process the client's declaration. In case a client does not return cash or cheques in time, the amount supposedly spent by the client is fixed to the value of the cash and cheques handed out to the client before the trip.

After flight arrangements have been made and any private flights have been paid, the responsible employee of the travel agency books hotels and makes reservations for local transportation (train, car, etc.). She also prints vouchers for any hotels that are booked. When cash and cheques have been received from the bank and all flight tickets have been received from the central office of the travel agency in SomewhereElse where they are printed, the employee puts all the documents together in a handy folder for the client. The agency has to make sure that everything is ready at least three working days before the trip starts, because, then, the client picks up the documents. At that point, the involvement of the agency with the trip is finished. In case of a private trip, this also means that the process is complete. In case of a business trip, however, the declaration of the client still needs to be processed.

As mentioned, the finance department takes care of processing declarations. When it has received a client's declaration and the necessary information of the bank, an employee of the finance department processes the declaration and calculates the balance. The result must be approved by the director of the finance department. In case of mistakes, the employee must make the necessary corrections. After the declaration has been approved by the director, the balance is settled with the next salary payment of the client. In addition, the total cost of the trip is deducted from the travel budget of the faculty or other unit where the client is employed. If a client does not hand in his or her declaration in time (within a month after completion of the trip), the finance department assumes that the total cost of the trip equals the sum of the advance payment and the value of the cash and cheques given to the client.

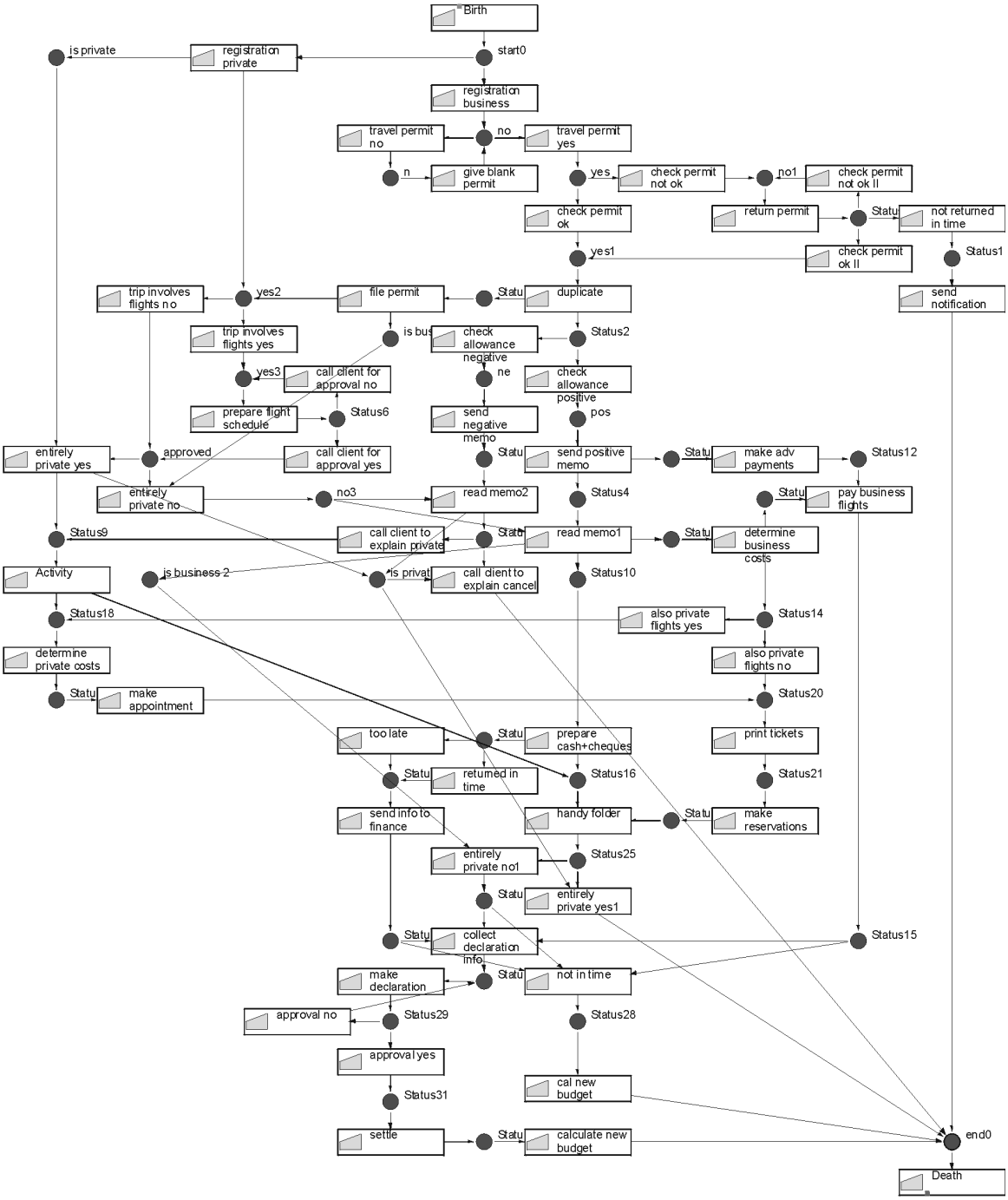


Figure 29: An example Protos model

The board of SU thinks that the main reason why the above process takes so long is that the co-ordination between the three departments involved is poor. It believes that a workflow system might provide a solution. As a starting point, it would like to receive a report covering the following subjects.

1. A resource classification of all the resources involved in the current process, distinguishing roles and groups.
2. A process model of the current situation developed in Protos, including information about roles and triggers.
3. An analysis of the resource classification and the process model, using the guidelines for process (re-)design discussed in the book and the slides.
4. An improved resource classification/process model developed in Protos, based on the results of the analysis. (Include only the graphical representation of the resource classification/process model and any information which is new compared to the original resource classification/process model.)

A.2 Protos model

See Figure 29.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407-426. Springer, Berlin, Germany, 1997.
2. W.M.P. van der Aalst, H.M.W. Verbeek, and D. Hauschildt. A Petri-net-based Tool to Analyze Workflows. In B. Farwer, D. Moldt, and M.O. Stehr, *Petri Nets in System Engineering (PNSE'97)*, pages 78-89. Technical report FBI-HH-B-205/97, University of Hamburg, Hamburg, Germany, 1997.
3. W.M.P. van der Aalst. Chapter 10: Three Good reasons for Using a Petri-net-based Workflow Management System. In T. Wakayama et al., editor, *Information and Process Integration in Enterprises: Rethinking Documents*. The Kluwer International Series in Engineering and Computer Science, pages 161-182. Kluwer Academic Publishers, Norwell, MA, 1998.
4. W.M.P. van der Aalst. Finding Errors in the Design of a Workflow Process: A Petri-net-based Approach. In [9], pages 60-81.
5. 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.
6. W.M.P. van der Aalst and A.H.M. ter Hofstede. *Verification of Workflow Task Structures: A Petri-net-based Approach*. Technical report 380, AIFB, University of Karlsruhe, Karlsruhe, Germany, 1998.
7. W.M.P. van der Aalst. Chapter 1: Putting Petri Nets to Work in the Workflow Arena. In *Petri Net Approaches for Modelling and Validation*, volume 1 of *LINCOM Studies in Computer Science*, pages 1-19, Lincom, Mnchen, Germany, 1999.
8. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. To appear in *Information and Software Technology*, 1999.
9. 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)*. UNINOVA, Lisbon, Portugal, 1998.
10. K. Barkaoui, J.M. Couvreur, and C. Dutheillet. On liveness in Extended Non Self-Controlling Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 25-44. Springer, Berlin, Germany, 1995.
11. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD Thesis. Eindhoven University of Technology, Department of Computing Science, Eindhoven, the Netherlands, 1998.
12. E. Best and C. Fernández C. *Non-sequential Processes: A Petri Net View*, volume 13 of *EACTS Monographs on Theoretical Computer Science*, Springer, Berlin, Germany, 1988.
13. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211-238, 1998.
14. J. Desel. *Validation of Information Systems by Analyzing Partially Ordered Petri Net Processes*. Technical report 375, AIFB, University of Karlsruhe, Karlsruhe, Germany, 1998.
15. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, UK, 1995.

16. C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1-16. Springer, Berlin, Germany, 1993.

17. J. Esparza and M. Nielsen. Decidability Issues for Petri Nets - a Survey. *Journal of Information Processing and Cybernetics*, 30(3): 210-242, 1994.

18. 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.

19. D. Hauschildt, H.M.W. Verbeek, and W.M.P. van der Aalst. *WOFLAN: a Petri-net-based Workflow Analyzer*. Computing Science Report 97/12, Eindhoven University of Technology, Eindhoven, the Netherlands, 1997.

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

21. R.R.A. Issa and R.F. Cox. Using Process Modeling and Workflow Integration to Gain ISO 9000 Certification in Construction. In proceedings of *CIB W89 Beijing International Conference on Construction, Modernization, and Education*. Beijing, China, 1996.

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

23. 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.

24. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems*, Seattle, USA, 1998.

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

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

27. T. Malone, W. Crowston, J. Lee, B. Pentland. Tools for Inventing Organizations: Toward a Handbook for Organizational Processes. To appear in *Management Science*, 1999.

28. Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, the Netherlands, 1997.

29. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93-129, 1998.

30. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EACTS Monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1985.

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

32. A. Sheth. From Contemporary Workflow Process Automation to Dynamic Work Activity Coordination and Collaboration. *Siggroup Bulletin*, 18(3):17-20, 1997.

33. Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1996.

34. SMIS group, Eindhoven University of Technology. *Woflan*. http://www.win.tue.nl/∼woflan/.

35. WFMC. *Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011)*. Technical report, Workflow Management Coalition, Brussels, Belgium, 1996.

36. M. Wolf and U. Reimer, editors. *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow*, Basel, Switzerland, 1996.