Challenges in Business Process Management: Verification of business processes using Petri nets

W.M.P. van der Aalst

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

Abstract. Most scientists working on formal methods are mainly focusing on technical systems such as circuit design, embedded systems, traffic control, etc. Few are working on the application of formal methods to business processes. As a result, interesting problems in the domain of Business Process Management (BPM) are not addressed. To stimulate the application of formal methods to BPM, the following two conferences, taking place in June 2003, are co-located: (1) the International Conference on Applications and Theory of Petri nets (Petri nets 2003) [8] and (2) the International Conference on Business Process Management: On the Application of Formal Methods to Process-Aware Information Systems (BPM 2003) [13]. Both conferences precede the International Colloquium on Automata, Languages and Programming (ICALP 2003) also taking place in Eindhoven (The Netherlands). By co-locating these events we hope to trigger cooperation between people working on BPM and formal methods.
This survey/tutorial discusses the need for formal methods in BPM. Although different formal methods could be applied in this domain, we focus on the application of Petri nets in this domain. In particular, we focus on the verification of workflow processes using Petri-net-based results. By this we hope to stimulate scientists working on Petri nets to address some of the challenges posed by BPM.

1 Introduction

The goal of this paper is to discuss the relation between Petri nets and BPM. This way we hope to interest researchers working on formal methods in some of the scientific challenges in this domain. The definition of a BPM system used throughout this paper is: *a generic software system that is driven by explicit process designs to enact and manage operational business processes*. The system should be process-aware and generic in the sense that it is possible to modify the processes it supports. The process designs are often graphical and the focus is on structured processes that need to handle many cases.

In the remainder of this paper, we will first put BPM and related technology in its historical context. Then, we will discuss models for process design. Since BPM systems are driven by explicit models, it is important to use the right techniques. Next, we will discuss techniques for the analysis of process models. We will argue that it is vital to have techniques to assert the correctness of workflow designs. Based on this we introduce the class of workflow nets: A subclass of Petri nets.

2 Business process management from a historical perspective

Only the wisest and stupidest of men never change.
Confucius

To show the relevance of BPM systems, it is interesting to put them in a historical perspective. Consider Figure 1, which shows some of the ongoing trends in information systems. This figure shows that today's information systems consist of a number of layers. The center is formed by the operating system, i.e., the software that makes the hardware work. The second layer consists of generic applications that can be used in a wide range of enterprises. Moreover, these applications are typically used within multiple departments within the same enterprise. Examples of such generic applications are a database management system, a text editor, and a spreadsheet program. The third layer consists of domain specific applications. These applications are only used within specific types of enterprises and departments. Examples are decision support systems for vehicle routing, call center software, and human resource management software. The fourth layer consists of tailor-made applications. These applications are developed for specific organizations.

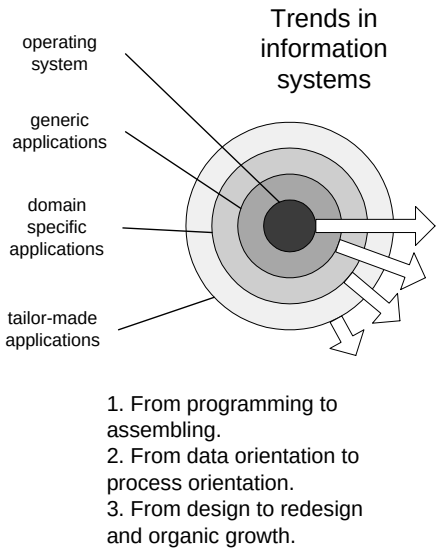

Fig. 1. Trends relevant for BPM.

In the sixties the second and third layer were missing. Information systems were built on top of a small operating system with limited functionality. Since no generic nor domain specific software was available, these systems mainly consisted of tailor-made applications. Since then, the second and third layer have developed and the ongoing trend is that the four circles are increasing in size, i.e., they are moving to the outside while absorbing new functionality. Today's operating systems offer much more

functionality. Database management systems that reside in the second layer offer functionality which used to be in tailor-made applications. As a result of this trend, the emphasis shifted from programming to assembling of complex software systems. The challenge no longer is the coding of individual modules but orchestrating and gluing together pieces of software from each of the four layers.

Another trend is the shift from data to processes. The seventies and eighties were dominated by data-driven approaches. The focus of information technology was on storing and retrieving information and as a result data modeling was the starting point for building an information system. The modeling of business processes was often neglected and processes had to adapt to information technology. Management trends such as business process reengineering illustrate the increased emphasis on processes. As a result, system engineers are resorting to a more process driven approach.

The last trend we would like to mention is the shift from carefully planned designs to redesign and organic growth. Due to the omnipresence of the Internet and its standards, information systems change on-the-fly. As a result, fewer systems are built from scratch. In many cases existing applications are partly used in the new system. Although component-based software development still has it problems, the goal is clear and it is easy to see that software development has become more dynamic.

The trends shown in Figure 1 provide a historical context for BPM systems. BPM systems are either separate applications residing in the second layer or are integrated components in the domain specific applications, i.e., the third layer. Notable examples of BPM systems residing in the second layer are workflow management systems [32, 36] such as Staffware, MQSeries, and COSA, and case handling systems such as FLOWer. Note that leading enterprise resource planning systems populating the third layer also offer a workflow management module. The workflow engines of SAP, Baan, PeopleSoft, Oracle, and JD Edwards can be considered as integrated BPM systems. The idea to isolate the management of business processes in a separate component is consistent with the three trends identified. BPM systems can be used to avoid hard-coding the work processes into tailor-made applications and thus support the shift from programming to assembling. Moreover, process orientation, redesign, and organic growth are supported. For example, today's workflow management systems can be used to integrate existing applications and support process change by merely changing the workflow diagram. Give these observations, we hope to have demonstrated the practical relevance of BPM systems. In the remainder of this paper we will focus more on the scientific importance of these systems. Moreover, for clarity we will often restrict the discussion to clear cut BPM systems such as workflow management systems.

An interesting starting point from a scientific perspective is the early work on office information systems. In the seventies, people like Skip Ellis [22], Anatol Holt [31], and Michael Zisman [44] already worked on so-called office information systems, which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. During the seventies and eighties there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and research almost stopped for a decade. Consequently, hardly any advances were made in the eighties. In the nineties,

there again was a huge interest in these systems. The number of workflow management systems developed in the past decade and the many papers on workflow technology illustrate the revival of office information systems. Today workflow management systems are readily available [36]. However, their application is still limited to specific industries such as banking and insurance. As was indicated by Skip Ellis it is important to learn from these ups and downs [23]. The failures in the eighties can be explained by both technical and conceptual problems. In the eighties, networks were slow or not present at all, there were no suitable graphical interfaces, and proper development software was missing. However, there were also more fundamental problems: a unified way of modeling processes was missing and the systems were too rigid to be used by people in the workplace. Most of the technical problems have been resolved by now. However, the more conceptual problems remain. Good standards for business process modeling are still missing and even today's workflow management systems enforce unnecessary constrains on the process logic (e.g., processes are made more sequential).

To summarize we state that, although the relevance of BPM systems is undisputed, many fundamental problems remain to be solved. In the remainder of this paper we will try to shed light on some of these problems.

3 Models for process design

A camel is a horse designed by committee.
Sir Alec Issigonis

BPM systems are driven by models of processes and organizations. By changing these models, the behavior of the system adapts to its environment and changing requirements. These models cover different perspectives. Figure 2 shows some of the perspectives relevant for BPM systems [32]. The process perspective describes the control-flow, i.e., the ordering of tasks. The information perspective describes the data that are used. The resource perspective describes the structure of the organization and identifies resources, roles, and groups. The task perspective describes the content of individual steps in the processes. Each perspective is relevant. However, in this paper we restrict ourselves to the process perspective.

Many techniques have been proposed to model the process perspective. Some of these techniques are informal in the sense that the diagrams used have no formally defined semantics. These models are typically very intuitive and the interpretation shifts depending on the modeler, application domain, and characteristics of the business processes at hand. Examples of informal techniques are ISAC, DFD, SADT, and IDEF. These techniques may serve well for discussing work processes. However, they are inadequate for directly driving information systems since they are incomplete and subject to multiple interpretations. Therefore, more precise ways of modeling are required.

Figure 3 shows an example of an order handling process modeled in terms of a so-called workflow net [2]. Workflow nets are based on the classical Petri-net model invented by Carl Adam Petri in the early sixties [37]. The squares are the active parts of the model and correspond to tasks. The circles are the passive parts of the model and are used to represent states. In the classical Petri net, the squares are named transitions and

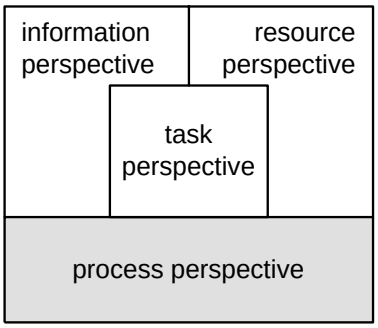

Fig. 2. Perspectives of models driving BPM systems.

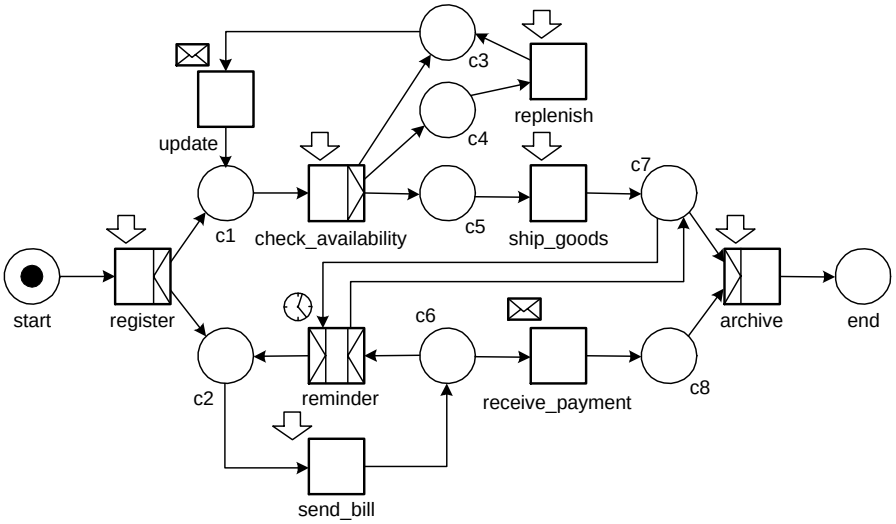

Fig. 3. WF-net.

the circles places. A workflow net models the life-cycle of one case. Examples of cases are insurance claims, tax declarations, and traffic violations. Cases are represented by tokens and in this case the token in *start* corresponds to an order. Task *register* is a so-called AND-split and is enabled in the state shown. The arrow indicates that this task requires human intervention. If a person executes this task, the token is removed from place *start* and two tokens are produced: one for *c1* and one for *c2*. Then, in parallel, two tasks are enabled: *check_availability* and *send_bill*. Depending on the eagerness of the workers executing these two tasks either *check_available* or *send_bill* is executed first. Suppose *check_availability* is executed first. If the ordered goods are available, they can be shipped by executing task *ship_goods*. If they are not available, either a replenishment order is issued or not. Note that *check_availability* is an OR-split and produces one token for *c3*, *c4*, or *c5*. Suppose that not all ordered goods are available, but the appropriate replenishment orders were already issued. A token is produced for

c3 and task *update* becomes enabled. Suppose that at this point in time task *send_bill* is executed, resulting in the state with a token in *c3* and *c6*. The token in *c6* is input for two tasks. However, only one of these tasks can be executed and in this state only *receive_payment* is enabled. Task *receive_payment* can be executed the moment the payment is received. Task *reminder* is an AND-join/AND-split and is blocked until the bill is sent and the goods have been shipped. Note that the reminder is sent after a specified period as indicated by the clock symbol. However, it is only possible to send a reminder if the goods have been actually shipped. Assume that in the state with a token in *c3* and *c6* task *update* is executed. This task does not require human involvement and is triggered by a message of the warehouse indicating that relevant goods have arrived. Again *check_availability* is enabled. Suppose that this task is executed and the result is positive. In the resulting state *ship_goods* can be executed. Now there is a token in *c6* and *c7* thus enabling task *reminder*. Executing task *reminder* again enables the task *send_bill*. A new copy of the bill is sent with the appropriate text. It is possible to send several reminders by alternating *reminder* and *send_bill*. However, let us assume that after the first loop the customer pays resulting in a state with a token in *c7* and *c8*. In this state, the AND-join *archive* is enabled and executing this task results in the final state with a token in *end*.

This very simple workflow net shows some of the routing constructs relevant for business process modeling. Sequential, parallel, conditional, and iterative routing are present in this model. There also are more advanced constructs such as the choice between *receive_payment* and *reminder*. This is a so-called *implicit choice* since it is not resolved by the system but by the environment of the system. The moment the bill is sent, it is undetermined whether *receive_payment* or *reminder* will be the next step in the process. Another advanced construct is the fact that task *reminder* is blocked until the goods have been shipped. The latter construct is a so-called *milestone*. The reason that we point out both constructs is that many systems have problems supporting these rather fundamental process patterns [11, 12].

Workflow nets have clear semantics. The fact that we are able to play the so-called token game using a minimal set of rules shows the fact that these models are executable. None of the informal informal techniques mentioned before (i.e., ISAC, DFD, SADT, and IDEF) have formal semantics. Besides workflow nets there are many other formal techniques. Examples are the many variants of process algebra [14] and statecharts [29]. The reason we prefer to use a variant of Petri nets is threefold [2]:

- Petri nets are graphical and yet precise.
- Petri nets offer an abundance of analysis techniques.
- Petri nets treat states as first-class citizens.

The latter point deserves some more explanation. Many techniques for business process modeling focus exclusively on the active parts of the process, i.e., the tasks. This is rather surprising since in many administrative processes the actual processing time is measured in minutes and the flow time is measured in days. This means that most of the time cases are in-between two subsequent tasks. Therefore, it is vital to model these states explicitly.

4 Techniques for process analysis

From the errors of others, a wise man corrects his own.
Syrus

BPM systems allow organizations to change their processes by merely changing the models. The models are typically graphical and can be changed quite easily. This provides more flexibility than conventional information systems. However, by reducing the threshold for change, errors are introduced more easily. Therefore, it is important to develop suitable analysis techniques. However, it is not sufficient to just develop these techniques. It is as least as important to look at methods and tools to make them applicable in a practical context.

Traditionally, most techniques used for the analysis of business processes, originate from operations research. All students taking courses in operations management will learn to apply techniques such as simulation, queueing theory, and Markovian analysis. The focus mainly is on *performance analysis* and less attention is paid to the correctness of models. *Verification* and *validation* are often neglected. As a result, systems fail by not providing the right support or even break down [3, 40]. Verification is needed to check whether the resulting system is free of logical errors. Many process designs suffer from deadlocks and livelocks that could have been detected using verification techniques. Validation is needed to check whether the system actually behaves as expected. Note that validation is context dependent while verification is not. A system that deadlocks is not correct in any situation. Therefore, verifying whether a system exhibits deadlocks is context independent. Validation is context dependent and can only be done with knowledge of the intended business process.

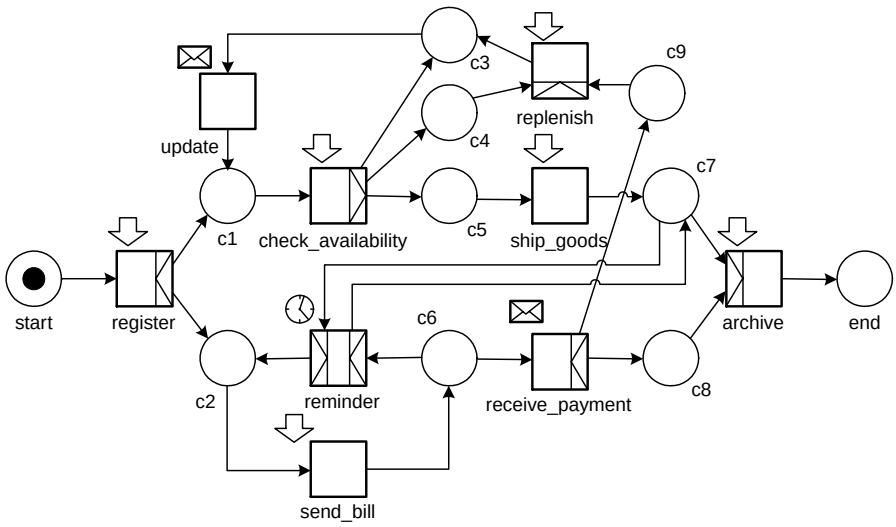

Fig. 4. An incorrect WF-net.

To illustrate the relevance of validation and verification and to demonstrate some of the techniques available, we return to the workflow net shown in Figure 3. This workflow process allows for the situation where a replenishment is issued before any payment is received. Suppose that we want to change the design such that replenishments are delayed until receiving payment. An obvious way to model this is to connect task *receive_payment* with *replenish* using an additional place *c9* as shown in Figure 4. Although this extension seems to be correct at first glance, the resulting workflow net has several errors. The workflow will deadlock if a second replenishment is needed and something is left behind in the process if no replenishments are needed. These are logical errors that can be detected without any knowledge of the order handling process. For verification, application independent notions of correctness are needed. One of these notions is the so-called *soundness property* [2]. A workflow net is sound if and only if the workflow contains no dead parts (i.e., tasks that can never be executed), from any reachable state it is always possible to terminate, and the moment the workflow terminates all places except the sink place (i.e., place *end*) are empty. Note that soundness rules out logical errors such as deadlocks and livelocks. The notion of soundness is applicable to any workflow language. An interesting observation is that soundness corresponds to liveness and boundedness of the short-circuited net [2]. The latter properties have been studied extensively [39, 21]. As a result, powerful analysis techniques and tools can be applied to verify the correctness of a workflow design. Practical experience shows that many errors can be detected by verifying the soundness property. Moreover, Petri-net theory can also be applied to guide the designer towards the error.

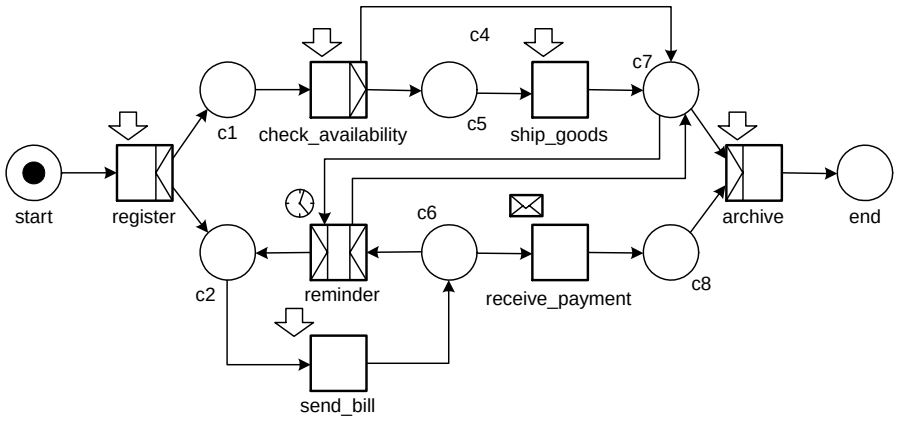

Fig. 5. A sound but incorrect WF-net.

Soundness does not guarantee that the workflow net behaves as intended. Consider for example, the workflow net shown in Figure 5. Compared to the original model, the shipment of goods is skipped if some of the goods are not available. Again this may seem to be a good idea at first glance. However, customers are expected to pay even if the goods are never delivered. In other words, task *receive_payment* needs to

be executed although task *ship_goods* may never be executed. The latter error can only be detected using knowledge about the context. Based on this context one may decide whether this is acceptable or not. Few analysis techniques exist to automatically support this kind of validation. The only means of validation offered by today's workflow management systems is gaming and simulation.

An interesting technique to support validation is inheritance of dynamic behavior. Inheritance can be used as a technique to compare processes. Inheritance relates subclasses with superclasses [16]. A workflow net is a subclass of a superclass workflow net if certain dynamic properties are preserved. A subclass typically contains more tasks. If by hiding and/or blocking tasks in the subclass one obtains the superclass, the subclass inherits the dynamics of the superclass.[1] The superclass can be used to specify the minimal properties the workflow design should satisfy. By merely checking whether the actual design is a subclass of the superclass, one can validate the essential properties. Consider for example Figure 6. This workflow net describes the minimal requirements the order handling process should satisfy. The tasks *register*, *ship_goods*, *receive_payment*, and *archive* are mandatory. Tasks *ship_goods* and *receive_payment* may be executed in parallel but should be preceded by *register* and followed by *archive*. The original order handling process shown in Figure 3 is a subclass of this superclass. Therefore, the minimal requirements are satisfied. However, the order handling process shown in Figure 5 is not a subclass. The fact that task *ship_goods* can be skipped demonstrates that not all properties are preserved.

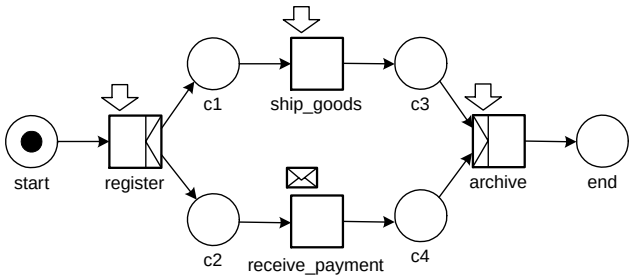

Fig. 6. A superclass WF-net.

Inheritance of dynamic behavior is a very powerful concept that has many applications. Inheritance-preserving transformation rules and transfer rules offer support at design-time and at run-time [7]. Subclass-superclass relationships also can be used to enforce correct processes in an E-commerce setting. If business partners only execute subclass processes of some common contract process, then the overall workflow will be executed as agreed. It should be noted that workflows crossing the borders of organizations are particularly challenging from a verification and validation point of view [4]. Errors resulting from miscommunication between business partners are highly dis-

[1] We have identified four notions of inheritance [7, 16]. In this paper, we only refer to life-cycle inheritance.

ruptive and costly. Therefore, it is important to develop techniques and tools for the verification and validation of these processes.

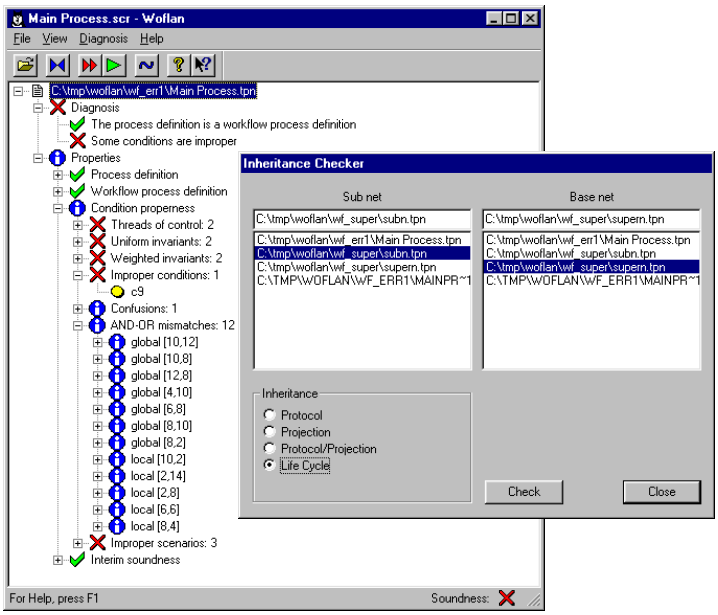

Fig. 7. A screenshot showing the verification and validation capabilities of Woflan.

Few tools aiming at the verification of workflow processes exist. Woflan [42] and Flowmake [40] are two notable exceptions. We have been working on Woflan since 1997. Figure 7 shows a screenshot of Woflan. Woflan combines state-of-the-art scientific results with practical applications [10, 42, 43]. Woflan can interface with leading workflow management systems such as Staffware and COSA. It can also interface with BPR-tools such as Protos. Workflow processes designed using any of these tools can be verified for correctness. It turns out that the challenge is not to decide whether the design is sound or not. The real challenge is to provide diagnostic information that guides the designer to the error. Woflan also supports the inheritance notions mentioned before. Given two workflow designs, Woflan is able to decide whether one is a subclass of the other. Tools such as Woflan illustrate the benefits of a more fundamental approach. Large scale experiments with experienced students show that workflow designers frequently make errors and that these design errors can be detected using Woflan [42].

5 Formalization of sound workflow nets

The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals. We cause accidents.
Nathaniel Borenstein

In the first part of this paper, an informal introduction was given into the BPM domain. In this introduction, we focused on workflow processes. As demonstrated, the process perspective can be modeled in terms of a WF-net. In this section, we formalize the notions mentioned in previous sections. First, we introduce some Petri net notation. Then we define WF-nets and the soundness property.

5.1 Petri Nets

This section introduces the basic Petri net terminology and notations. Readers familiar with Petri nets can skip this section.[2]

The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 1 (Petri net). *A Petri net is a triple* (P, T, F)*:*

- P *is a finite set of places,*
- T *is a finite set of transitions* ($P \cap T = \emptyset$)*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation)*

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g., $p\bullet$ is the set of transitions sharing p as an input place. Note that we do not consider multiple arcs from one node to another. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \rightarrow \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$. To compare states we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

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

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.
(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

Given a Petri net (P, T, F) and a state M_1, we have the following notations:

- $M_1 \stackrel{t}{\rightarrow} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \stackrel{t}{\rightarrow} M_2$
- $M_1 \stackrel{\sigma}{\rightarrow} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n via a (possibly empty) set of intermediate states $M_2, \ldots M_{n-1}$, i.e., $M_1 \stackrel{t_1}{\rightarrow} M_2 \stackrel{t_2}{\rightarrow} \ldots \stackrel{t_{n-1}}{\rightarrow} M_n$

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

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

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

Definition 2 (Live). *A Petri net (PN, M) is live iff, for every reachable state M' and every transition t there is a state M'' reachable from M' which enables t.*

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

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

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

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

Paths connect nodes by a sequence of arcs.

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

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

Definition 6 (Strongly connected). *A Petri net is strongly connected iff, for every pair of nodes (i.e., places and transitions) x and y, there is a path leading from x to y.*

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

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

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

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

See [21, 39] for a more elaborate introduction to these standard notions.

5.2 WF-Nets

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

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

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

A WF-net has one input place (i) and one output place (o) because any case handled by the procedure represented by the WF-net is created when it enters the WFMS and is deleted once it is completely handled by the WFMS, i.e., the WF-net specifies the life-cycle of a case. The third requirement in Definition 11 has been added to avoid 'dangling tasks and/or conditions', i.e., tasks and conditions which do not contribute to the processing of cases.

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

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

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

Figures 3, 4 and 5 show examples of WF-nets. In each net the source place i is named *start* and the sink place o is named *end*. Note that some syntactic sugaring is used. There is a one-to-one correspondence between AND-splits/AND-joins and transitions. However, OR-splits and OR-joins correspond to clusters of transitions: one for each choice. For example, task *check_availability* corresponds to three transitions. Each of these three transitions has *c1* as input place and one output place. The first one produces a token for *c5*, the second for *c4*, and the third for *c3*. Using this translation it is easy to see that each of the nets shown in figures 3, 4 and 5 is indeed a WF-net.

5.3 Soundness

In this section we summarize some of the basic results for WF-nets presented in [1]. The remainder of this paper will build on these results.

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

> *For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there should be no dead tasks, i.e., it should be possible to execute an arbitrary task by following the appropriate route though the WF-net. These two additional requirements correspond to the so-called *soundness property*.

Definition 12 (Sound). *A procedure modeled by a WF-net $PN = (P, T, F)$ is sound if and only if:*

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state o. Formally:[3]

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

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

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

(iii) There are no dead transitions in (PN, i). Formally:

$$\forall_{t \in T} \exists_{M, M'} \ i \xrightarrow{*} M \xrightarrow{t} M'$$

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 12 states that starting from the initial state (state *i*), it is always possible to reach the state with one token in place *o* (state *o*). If we assume a strong notion of fairness, then the first requirement implies that eventually state o is reached. Strong fairness means in every infinite firing sequence, each transition fires infinitely often. The fairness assumption is reasonable in the context of workflow management:

[3] Note that there is an overloading of notation: the symbol *i* is used to denote both the *place i* and the *state* with only one token in place *i* (see Section 5.1).

All choices are made (implicitly or explicitly) by applications, humans or external actors. Clearly, they should not introduce an infinite loop. Note that the traditional notions of fairness (i.e., weaker forms of fairness with just local conditions, e.g., if a transition is enabled infinitely often, it will fire eventually) are not sufficient. See [2, 34] for more details. The second requirement states that the moment a token is put in place *o*, all the other places should be empty. Sometimes the term *proper termination* is used to describe the first two requirements [27]. The last requirement states that there are no dead transitions (tasks) in the initial state *i*.

Note that the second requirement is implied by the first one. Suppose the second requirement does not hold. This implies that there is a state M reachable from i such that $M \geq o$ and $M \neq o$, i.e., a state with at least two tokens. There is at least one token in o which cannot be removed. The other token cannot be consumed without producing a new one. Therefore, it is not possible to reach state o from M. This contradicts with the first requirement. This shows that the second requirement can be removed. Nevertheless, Definition 12 lists this requirement since it corresponds to an intuitive notion of correctness.

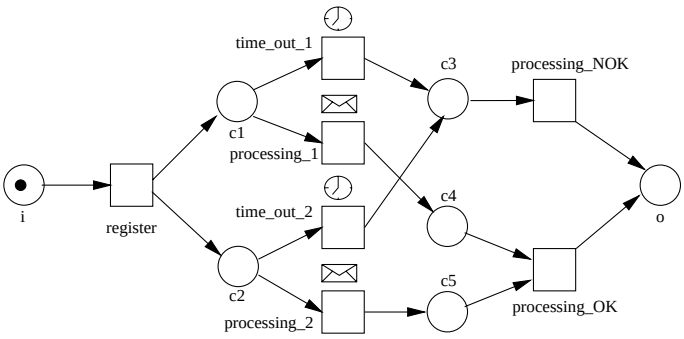

Fig. 8. Another WF-net for the processing of complaints.

Figure 8 shows a WF-net which is not sound. There are several deficiencies. If *time_out_1* and *processing_2* fire or *time_out_2* and *processing_1* fire, the WF-net will not terminate properly because a token gets stuck in *c4* or *c5*. If *time_out_1* and *time_out_2* fire, then the task *processing_NOK* will be executed twice and because of the presence of two tokens in *o* the moment of termination is not clear.

Figures 3, 4 and 5 also show examples of WF-nets. In each net the source place i is named *start* and the sink place o is named *end* and task *check_availability* corresponds to multiple transitions as described before. The WF-net shown in Figure 3 is sound. The WF-net shown in Figure 4 is not sound: The process may deadlock before reaching the sink place or the process may leave a superfluous token in place *c9*. The WF-net shown in Figure 5 is sound (although it is not a subclass of Figure 3).

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. In [1] we have shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$.

$\overline{PN}$ is the Petri net obtained by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\}$. In the remainder we will call such an extended net the *short-circuited* net of PN. The short-circuited net allows for the formulation of the following theorem.

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

Proof. See [1]. □

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

Several authors have proposed other (typically weaker) notions of soundness. In [19, 24] the notion of relaxed soundness is used. This notion is weaker since it focuses on the possibility to have sound executions. In [30] the notion of k-soundness is introduced. This notion takes k tokens in place i as a starting point. In [35] soundness is also defined for multiple connected WF-nets.

6 Structural Characterization of Soundness

A classic is classic not because it conforms to certain structural rules, or fits certain definitions (of which its author had quite probably never heard). It is classic because of a certain eternal and irrepressible freshness.
Edith Wharton

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

- For a complex WF-net it may be intractable to decide soundness. (For arbitrary WF-nets liveness and boundedness are decidable but also EXPSPACE-hard, cf. Cheng, Esparza and Palsberg [18].)
- Soundness is a minimal requirement. Readability and maintainability issues are not addressed by Theorem 1.
- Theorem 1 does not show how a non-sound WF-net should be modified, i.e., it does not identify constructs which invalidate the soundness property.

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

6.1 Free-Choice WF-Nets

Most of the WFMS's available at the moment, abstract from states between tasks, i.e., states are not represented explicitly. These WFMS's use building blocks such as the AND-split, AND-join, OR-split and OR-join to specify workflow procedures. The

AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. Because these systems abstract from states, every choice is made *inside* an OR-split building block. If we model an OR-split in terms of a Petri net, the OR-split corresponds to a number of transitions sharing the same set of input places. This means that for these WFMS's, a workflow procedure corresponds to a free-choice Petri net (cf. Definition 7).

It is easy to see that a process definition composed of AND-splits, AND-joins, OR-splits and OR-joins is free-choice. If two transitions t_1 and t_2 share an input place ($\bullet t_1 \cap \bullet t_2 \neq \emptyset$), then they are part of an OR-split, i.e., a 'free choice' between a number of alternatives. Therefore, the sets of input places of t_1 and t_2 should match ($\bullet t_1 = \bullet t_2$). Figure 8 shows a free-choice WF-net. The WF-net shown in Figure 3 is not free-choice; *archive* and *reminder* share an input place but the two corresponding input sets differ.

We have evaluated many WFMS's and just one of these systems (COSA [41]) allows for a construct which is comparable to a non-free choice WF-net [12, 33]. Therefore, it makes sense to consider free-choice Petri nets in more detail. Clearly, parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property. Another reason for restricting WF-nets to free-choice Petri nets is the following. If we allow non-free-choice Petri nets, then the choice between conflicting tasks *may* be influenced by the order in which the preceding tasks are executed. The routing of a case should be independent of the order in which tasks are executed. A situation where the free-choice property is violated is often a mixture of parallelism and choice. Figure 9 shows such a situation. Firing transition *t1* introduces parallelism. Although there is no real choice between *t2* and *t5* (*t5* is not enabled), the parallel execution of *t2* and *t3* results in a situation where *t5* is not allowed to occur. However, if the execution of *t2* is delayed until *t3* has been executed, then there is a real choice between *t2* and *t5*. In our opinion parallelism itself should be separated from the choice between two or more alternatives. Therefore, we consider the non-free-choice construct shown in Figure 9 to be improper. In literature, the term *confusion* is often used to refer to the situation shown in Figure 9.

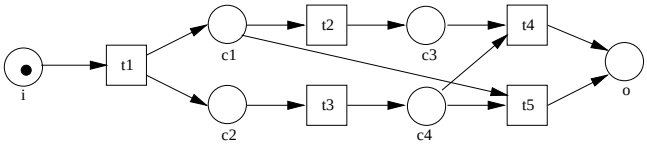

Fig. 9. A non-free-choice WF-net containing a mixture of parallelism and choice.

Free-choice Petri nets have been studied extensively (cf. Best [17], Desel and Esparza [21, 20, 25], Hack [28]) because they seem to be a good compromise between expressive power and analyzability. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist. For example, the well-known Rank Theorem (Desel and Esparza [21]) enables us to formulate the following corollary.

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

Proof. Let PN be a free-choice WF-net. The short-circuited net $\overline{PN}$ is also free-choice. Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time (Rank Theorem [21]). By Theorem 1, this corresponds to soundness. □

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

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

Proof. Let PN be a sound free-choice WF-net. $\overline{PN}$ is the Petri net PN extended with a transition connecting o and i. $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable [21], i.e., each place is part of an embedded strongly connected state-machine component. Since initially there is just one token $(\overline{PN}, i)$ is safe and so is (PN, i). □

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

Although most WFMS's only allow for free-choice workflows, free-choice WF-nets are not a completely satisfactory structural characterization of 'good' workflows. On the one hand, there are non-free-choice WF-nets which correspond to sensible workflows (cf. Figure 3). On the other hand there are sound free-choice WF-nets which make no sense. Nevertheless, the free-choice property is a desirable property. If a workflow can be modeled as a free-choice WF-net, one should do so. A workflow specification based on a free-choice WF-net can be enacted by most workflow systems. Moreover, a free-choice WF-net allows for efficient analysis techniques and is easier to understand. Non-free-choice constructs such as the construct shown in Figure 9 are a potential source of anomalous behavior (e.g., deadlock) which is difficult to trace.

6.2 Well-Structured WF-Nets

Another approach to obtain a structural characterization of 'good' workflows, is to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. As shown in Figure 10, an AND-split should be complemented by an AND-join and an OR-split should be complemented by an OR-join.

One of the deficiencies of the WF-net shown in Figure 8 is the fact that the AND-split *register* is complemented by the OR-join *c3* or the OR-join *o*. To formalize the concept illustrated in Figure 10 we give the following definition.

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

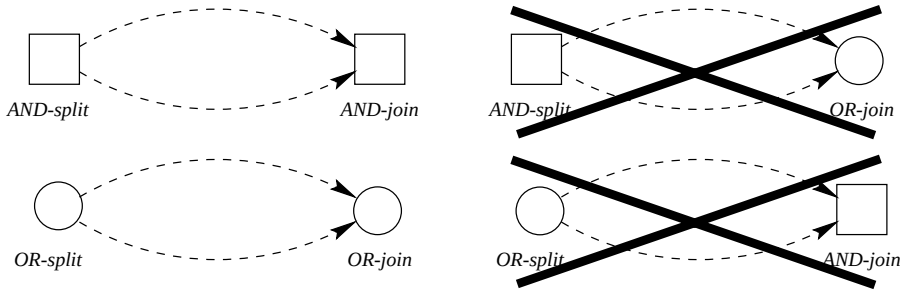

Fig. 10. Good and bad constructions.

Note that the WF-net shown in Figure 8 is not well-handled. Well-handledness can be decided in polynomial time by applying a modified version of the max-flow min-cut technique described in [9]. A Petri net which is well-handled has a number of nice properties, e.g., strong connectedness and well-formedness coincide.

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

Proof. Let PN be a strongly connected well-handled Petri net. Clearly, there are no circuits that have PT-handles nor TP-handles [26]. Therefore, the net is structurally bounded (See Theorem 3.1 in [26]) and structurally live (See Theorem 3.2 in [26]). Hence, PN is well-formed. □

Clearly, well-handledness is a desirable property for any WF-net PN. Moreover, we also require the short-circuited $\overline{PN}$ to be well-handled. We impose this additional requirement for the following reason. Suppose we want to use PN as a part of a larger WF-net PN'. PN' is the original WF-net extended with an 'undo-task'. See Figure 11. Transition $undo$ corresponds to the undo-task, transitions $t1$ and $t2$ have been added to make PN' a WF-net. It is undesirable that transition $undo$ violates the well-handledness property of the original net. However, PN' is well-handled iff $\overline{PN}$ is well-handled. Therefore, we require $\overline{PN}$ to be well-handled. We use the term *well-structured* to refer to WF-nets whose extension is well-handled.

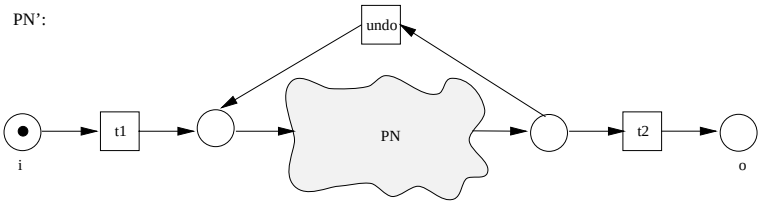

Fig. 11. The WF-net PN' is well-handled iff $\overline{PN}$ is well-handled.

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

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

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

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

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

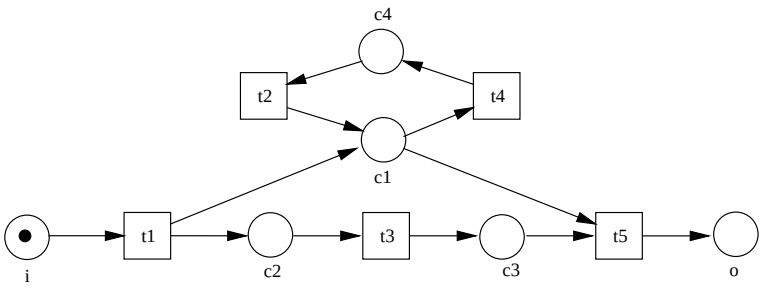

Fig. 12. A well-structured WF-net.

Consider for example the WF-net shown in Figure 12. The WF-net is well-structured and, therefore, also elementary extended non-self controlling. However, the net is not free-choice. Nevertheless, it is possible to verify soundness for such a WF-net very efficiently.

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

Proof. Let PN be a well-structured WF-net. The short-circuited net $\overline{PN}$ is elementary extended non-self controlling (Theorem 2) and structurally bounded (see proof of Lemma 2). For bounded elementary extended non-self controlling nets the problem of deciding whether a given marking is live, can be solved in polynomial time (See [15]). Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time. By Theorem 1, this corresponds to soundness. □

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

Proof. Let $\overline{PN}$ be the net PN extended with a transition connecting o and i. $\overline{PN}$ is extended non-self controlling. $\overline{PN}$ is covered by state-machines (S-components), see Corollary 5.3 in [15]. Hence, $\overline{PN}$ is safe and so is PN (see proof of Lemma 1). □

Well-structured WF-nets and free-choice WF-nets have similar properties. In both cases soundness can be verified very efficiently and soundness implies safeness. In spite of these similarities, there are sound well-structured WF-nets which are not free-choice (Figure 12) and there are sound free-choice WF-nets which are not well-structured. In fact, it is possible to have a sound WF-net which is neither free-choice nor well-structured (Figures 3 and 9).

6.3 S-Coverable WF-Nets

What about the sound WF-nets shown in Figure 3 and Figure 9? The WF-net shown in Figure 9 can be transformed into a free-choice well-structured WF-net by separating choice and parallelism. The WF-net shown in Figure 3 cannot be transformed into a free-choice or well-structured WF-net without yielding a much more complex WF-net. Place *c7* acts as some kind of milestone which is tested by the task *reminder*. Traditional workflow management systems which do not make the state of the case explicit, are not able to handle the workflow specified by Figure 3. Only workflow management systems such as COSA [41] have the capability to enact such a state-based workflow. Nevertheless, it is interesting to consider generalizations of free-choice and well-structured WF-nets: *S-coverable WF-nets* can be seen as such a generalization.

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

The WF-nets shown in Figure 3 and Figure 9 are S-coverable. The WF-net shown in Figure 8 is not S-coverable. The following two corollaries show that S-coverability is a generalization of the free-choice property and well-structuredness.

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

Proof. The short-circuited net $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable (cf. [21]). □

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

Proof. $\overline{PN}$ is extended non-self controlling (Theorem 2). Hence, $\overline{PN}$ is S-coverable (cf. Corollary 5.3 in [15]). □

All the sound WF-nets presented in this paper are S-coverable. Every S-coverable WF-net is safe. The only WF-net which is not sound, i.e., the WF-net shown in Figure 8, is not S-coverable. These and other examples indicate that there is a high correlation between S-coverability and soundness. It seems that S-coverability is one of the basic requirements any workflow process definition should satisfy. From a formal point of view, it is possible to construct WF-nets which are sound but not S-coverable. Typically, these nets contain places which do not restrict the firing of a transition, but which are not in any S-component. (See for example Figure 65 in [38].) From a practical point of view, these WF-nets are to be avoided. WF-nets which are not S-coverable are difficult to interpret because the structural and dynamical properties do not match. For example, these nets can be live and bounded but not structurally bounded. There seems to be no practical need for using constructs which violate the S-coverability property. Therefore, we consider S-coverability to be a basic requirement any WF-net should satisfy.

Another way of looking at S-coverability is the following interpretation: S-components corresponds to *document flows*. To handle a workflow several pieces of information are created, used, and updated. One can think of these pieces of information as physical documents, i.e., at any point in time the document is in one place in the WF-net. Naturally, the information in one document can be copied to another document while executing a task (i.e., transition) processing both documents. Initially, all documents are present but a document can be empty (i.e., corresponds to a blank piece paper). It is easy to see that the flow of one such document corresponds a state machine (assuming the existence of a transition t^*). These document flows synchronize via joint tasks. Therefore, the composition of these flows yields an S-coverable WF-net. One can think of the document flows as threads. Consider for example the short-circuited net of the WF-net shown in Figure 3. This net can be composed out of the following two threads: (1) a thread corresponding to the physical process (places *start*, $c1$, $c3$, $c4$, $c5$, $c7$ and *end*) and (2) a thread corresponding to the financial process (places *start*, $c2$, $c6$, $c8$, and *end*). Note that the tasks *register*, *reminder* and *archive* are used in both threads.

Although a WF-net can, in principle, have exponentially many S-components, they are quite easy to compute for workflows encountered in practice (see also the above interpretation of S-component as document flows or threads). Note that S-coverability only depends on the structure and the degree of connectedness is generally low (i.e., the incidence matrix of a WF-net typically has few non-zero entries [9]). Unfortunately, in general, it is not possible to verify soundness of an S-coverable WF-net in polynomial time. The problem of deciding soundness for an S-coverable WF-net is PSPACE-complete. For most applications this is not a real problem. In most cases the number of tasks in one workflow process definition is less than 100 and the number of states is less than 200,000. Tools using standard techniques such as the construction of the coverability graph have no problems in coping with these workflow process definitions.

6.4 Summary

The three structural characterizations (free-choice, well-structured and S-coverable) turn out to be very useful for the analysis of workflow process definitions. Based on

our experience, we have good reasons to believe that S-coverability is a desirable property any workflow definition should satisfy. Constructs violating S-coverability can be detected easily and tools can be build to help the designer to construct an S-coverable WF-net. S-coverability is a generalization of well-structuredness and the free-choice property (Corollary 3 and 4). Both well-structuredness and the free-choice property also correspond to desirable properties of a workflow. A WF-net satisfying at least one one of these two properties can be analyzed very efficiently. However, we have shown that there are workflows that are not free-choice and not well-structured. Consider for example Figure 3. The fact that task *reminder* tests whether there is a token in *c7*, prevents the WF-net from being free-choice or well-structured. Although this is a very sensible workflow, most workflow management systems do not support such an advanced routing construct. Even if one is able to use state-based workflows (e.g., COSA) allowing for constructs which violate well-structuredness and the free-choice property, then the structural characterizations are still useful. If a WF-net is not free-choice or not well-structured, one should locate the source which violates one of these properties and check whether it is really necessary to use a non-free-choice or a non-well-structured construct. If the non-free-choice or non-well-structured construct is really necessary, then the correctness of the construct should be double-checked, because it is a potential source of errors. This way the readability and maintainability of a workflow process definition can be improved.

7 Conclusion

In this paper, the application of Petri nets to BPM was discussed.[4] First, BPM was put in its historical perspective. Then, the topics of process design and process analysis were discussed. These topics have been illustrated using Petri nets. In the second part of the paper, we investigated a basic property that any workflow process definition should satisfy: the soundness property. For WF-nets, this property coincides with liveness and boundedness. In our quest for a structural characterization of WF-nets satisfying the soundness property, we have identified three important subclasses: free-choice, well-structured, and S-coverable WF-nets. The identification of these subclasses is useful for the detection of design errors. We hope that these results demonstrate the relevance of formal methods for BPM in general and workflow management in particular.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[4] Note that parts of the paper are taken from [5, 6].

4. W.M.P. van der Aalst. Loosely Coupled Interorganizational Workflows: Modeling and Analyzing Workflows Crossing Organizational Boundaries. *Information and Management*, 37(2):67–75, March 2000.
5. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
6. W.M.P. van der Aalst. Making Work Flow: On the Application of Petri nets to Business Process Management. In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, Berlin, 2002.
7. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
8. W.M.P. van der Aalst and E. Best, editors. *Application and Theory of Petri nets*, volume 2679 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2003.
9. W.M.P. van der Aalst, D. Hauschildt, and H.M.W. Verbeek. A Petri-net-based Tool to Analyze Workflows. In B. Farwer, D. Moldt, and M.O. Stehr, editors, *Proceedings of Petri Nets in System Engineering (PNSE'97)*, pages 78–90, Hamburg, Germany, September 1997. University of Hamburg (FBI-HH-B-205/97).
10. W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of Workflow Task Structures: A Petri-net-based Approach. *Information Systems*, 25(1):43–69, 2000.
11. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, Berlin, 2000.
12. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
13. W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors. *Busines Process Management*, volume 2678 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2003.
14. J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1990.
15. K. Barkaoui, J.M. Couvreur, and C. Dutheillet. On liveness in Extended Non Self-Controlling Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag, Berlin, 1995.
16. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
17. E. Best. Structure Theory of Petri Nets: the Free Choice Hiatus. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 168–206. Springer-Verlag, Berlin, 1987.
18. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. In R.K. Shyamasundar, editor, *Foundations of software technology and theoretical computer science*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, Berlin, 1993.
19. J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer-Verlag, Berlin, 2001.
20. J. Desel. A proof of the Rank theorem for extended free-choice nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 134–153. Springer-Verlag, Berlin, 1992.

21. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
22. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
23. C.A. Ellis and G. Nutt. Workflow: The Process Spectrum. In A. Sheth, editor, *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 140–145, Athens, Georgia, May 1996.
24. R. Eshuis and J. Dehnert. Reactive Petri nets for Workflow Modeling. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 295–314. Springer-Verlag, Berlin, 2003.
25. J. Esparza. Synthesis rules for Petri nets, and how they can lead to new results. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 182–198. Springer-Verlag, Berlin, 1990.
26. J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer-Verlag, Berlin, 1990.
27. K. Gostellow, V. Cerf, G. Estrin, and S. Volansky. Proper Termination of Flow-of-control in Programs Involving Concurrent Processes. *ACM Sigplan*, 7(11):15–27, 1972.
28. M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.
29. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
30. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.
31. A. W. Holt. Coordination Technology and Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 278–296. Springer-Verlag, Berlin, 1985.
32. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
33. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, 2003.
34. E. Kindler and W.M.P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 70(6):269–274, June 1999.
35. E. Kindler, A. Martens, and W. Reisig. Inter-Operability of Workflow Applications: Local Criteria for Global Soundness. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 235–253. Springer-Verlag, Berlin, 2000.
36. P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.
37. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.
38. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer-Verlag, Berlin, 1985.
39. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
40. W. Sadiq and M.E. Orlowska. Analyzing Process Models using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.

41. Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1998.
42. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
43. Woflan Home Page. http://www.tm.tue.nl/it/woflan.
44. M.D. Zisman. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.