Verification of Workflow Task Structures: A Petri-net-based approach

W.M.P. van der Aalst[1,*] and A.H.M. ter Hofstede[2]

[1] *Department of Mathematics and Computing Science, Eindhoven University of Technology GPO Box 513, NL-5600 MB Eindhoven, The Netherlands, e-mail: wsinwa@win.tue.nl*
[2] *Cooperative Information Systems Research Centre, Queensland University of Technology GPO Box 2434, Brisbane, Qld 4001, Australia, e-mail: arthur@icis.qut.edu.au*

Abstract

While many workflow management systems have emerged in recent years, few of them provide any form of support for verification. Consequently, most workflows become operational before they have been thoroughly checked. This frequently results in runtime errors which need to be corrected on-the-fly at, typically, prohibitive costs. This paper shows how verification of a typical process control specification, which is at the heart of most workflow specifications, can benefit from state-of-the-art Petri-net based analysis techniques. To illustrate the applicability of the approach, a verification tool has been developed. This tool can download and verify the correctness of process definitions designed with Staffware, one of the leading workflow management systems.

1 Introduction

Recent years have seen the proliferation of workflow management systems developed for different types of workflows and based on different paradigms (see e.g. [Aal98a, EN93, EKR95, GHS95, JB96, Kou95, Law97, Sch96, WFM96]). Despite the abundance of such tools, the critical issue of workflow verification is virtually neglected. Few tools provide any form of verification support.

This lack of support can be explained from the fact that the verification of workflows is hard from a computational as well as an algorithmic point of view (see e.g. [Aal97, AAH98, HOR98]). The consequences, however, are that few workflows are thoroughly checked before they are

*Part of this work was done at AIFB (University of Karlsruhe, Germany) and CTRG (University of Colorado, USA) during a sabbatical leave.

deployed in practice, which often results in errors having to be corrected in an ad hoc fashion often at prohibitive costs.

Workflow specifications address many issues including data flow, exception handling, recovery etc. Hence, verification of a full workflow specification is typically not feasible. However, typically the specification of *process control* is at the heart of most workflow specifications. Workflow specification languages need to support the specification of moments of choice, sequential execution, parallelism, synchronization, and iteration.

In this paper we focus on Task Structures (see e.g. [HN93]) which is a powerful language for the specification of process control. Task structures can be seen as a good general representative of process control specification languages used in workflow management. The specification language as used in [CCPP98] is essentially the same as Task Structures. In [BHP97, BH97] Task Structures were extended with advanced workflow concepts and used for a real-life workflow application involving road closures in Queensland. There are also workflow management systems that use a language close to Task Structures. In fact, we show that there is a one-to-one correspondence between Task Structures and the diagramming technique used in Staffware [Sta97]. Staffware is one of the leading workflow management systems with more than 550,000 users worldwide. In fact, according to the Gartner Group, Staffware is the market leader with 25 percent of the global market [Cas98].

Petri nets have been around since Carl Adam Petri's PhD thesis in the early sixties [Pet62] and have found many applications in computer science. Petri nets have a rigorous mathematical foundation and a substantial body of theory for their formal analysis has been developed. In this paper this theory is exploited and state-of-the-art Petri-net based techniques are used for the verification of Task Structures. The results provide an important impetus for the further automation of workflow verification, in particular as many sophisticated automated tools for the analysis of Petri nets exist. One such tool, Woflan [AHV97], and its application, will be briefly discussed in this paper. In particular, it will be demonstrated how Woflan can be used for the verification of workflow specifications in Staffware.

The organization of this paper is as follows. In section 2 the various perspectives of workflow modeling are discussed. In Section 3, Task Structures are introduced. In Section 4 Task Structures are first mapped to Petri nets and then an extension of this mapping is described to a particular form of Petri nets, WF-nets, particularly suitable for workflow modeling. Section 5 then applies formal Petri net analysis techniques to the results of such mappings. In Section 6 we describe the functionality of Woflan and in particular the implementation of the link between Staffware and Woflan. Section 7 provides a concrete case study highlighting the main aspects of the approach presented. Section 8 gives pointers to related work and Section 9 provides conclusions and some topics for future research.

2 Workflow perspectives

The primary task of a workflow management system is to enact case-driven business processes by joining several perspectives. The following perspectives are relevant for workflow modeling and workflow execution: (1) *control flow* (or process) perspective, (2) *resource* (or organization) perspective, (3) *data* (or information) perspective, (4) *task* (or function) perspective, (5) *operation* (or application) perspective. (These perspectives are similar to the perspectives given in [JB96].) In the control-flow perspective, *workflow process definitions* (workflow schemas) are defined to specify which *tasks* need to be executed and in what order (i.e., the routing or control flow). A task is an atomic piece of work. Workflow process definitions are instantiated for specific *cases* (i.e., workflow instances). Examples of cases are: a request for a mortgage loan, an insurance claim, a tax declaration, an order, or a request for information. Since a case is an instantiation of a process definition, it corresponds to the execution of concrete work according to the specified routing. In the *resource* perspective, the organizational structure and the population are specified. The organizational structure describes relations between roles (resource classes based on functional aspects) and groups (resource classes based on organizational aspects). Thus clarifying organizational issues such as responsibility, availability, and authorization. Resources, ranging from humans to devices, form the organizational population and are allocated to roles and groups. The data perspective deals with *control* and *production data*. Control data are data introduced solely for workflow management purposes, e.g., variables introduced for routing purposes. Production data are information objects (e.g., documents, forms, and tables) whose existence does not depend on workflow management. The task perspective describes the elementary operations performed by resources while executing a task for a specific case. In the operational perspective the elementary actions are described. These actions are often executed using applications ranging from a text editor to custom build applications to perform complex calculations. Typically, these applications create, read, or modify control and production data in the information perspective.

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

The resource perspective can only restrict the routing of cases, i.e., it does not enable execution paths excluded in the control flow perspective. Therefore, it suffices to focus on deadlocks as a result of restrictions imposed by the resource perspective. A potential deadlock could arise (1) when multiple tasks try to allocate multiple resources at the same time, or (2) when there are tasks imposing such demanding constraints that no resource qualifies. The first type of deadlock often occurs in flexible manufacturing where both space and tools are needed to complete operations thus potentially resulting in locking problems [SV90]. However, given today's workflow technology, such deadlocks cannot occur in a workflow management system: At any time there is only one resource working on a task which is being executed for a specific

case. In today's workflow management systems it is not possible to specify that several resources are collaborating in executing a task. Note that even if multiple persons are executing one task, e.g., writing a report, only one person is allocated to that task from the perspective of the workflow management system: This is the person that selected the work item from the in-basket (i.e., the electronic worktray). Since a person is working on one task at a time and each task is eventually executed by one person (although it may be allocated to a group a people), it suffices to check for the presence of sufficient resources. Therefore, from the viewpoint of verification, i.e., analyzing the logical correctness of a workflow process, it is reasonable to abstract from these locking problems. (Nevertheless, if in the future collaborative features are explicitly supported by the workflow management system, then these problems should be taken into account.) The second type of deadlock occurs when there is no suitable resource to execute a task for a given case, e.g., there is not a single resource with the specified role. Generally, such problems can be avoided quite easily by checking whether all role/group expressions yield a non-empty set of resources. However, there may be more subtle errors resulting from case management (a subset of tasks for a given case is required to be executed by the same resource) and function separation (two tasks are not to be executed by the same resource to avoid security violations). For example: Task 1 should be executed by the same person as task 2 and task 2 should be executed by the same person as task 3. However, task 3 should not be executed by the person who executed task 1. Clearly, there is no person qualified to execute task 3. Such problems highly depend on the workflow management system being used and are fairly independent of the routing structure. Therefore, we think it is reasonable to abstract from these resource-driven deadlocks.

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

hold for the situation with workflow attributes. Last but not least, we abstract from triggers and workflow attributes because it allows us to use ordinary Petri nets (i.e., P/T nets) rather than high-level Petri nets. From an analysis point of view, this is preferable because of the availability of efficient algorithms and powerful analysis tools.

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

Note that we do not explicitly consider transactional workflows [GHS95]. There are several reasons for this. First of all, most workflow management systems (in particular the commercial ones) do not support transactional features at the workflow modeling language (e.g., Staffware, the system considered in this paper, has no such facilities). Second, as is shown in [AAH98] the various transactional dependencies can easily be modeled in terms of Petri nets. Therefore, we can straightforwardly extend the approach in this paper to transactional workflows.

3 Workflow modeling using Task Structures

The specification of workflows in general is known to be quite complex and many issues are involved. Workflow specifications should incorporate execution dependencies between tasks, information flow between tasks, access to distributed databases, temporal constraints, exception handling etc. In this paper though, focus is solely on *control flow* aspects in workflow specifications. (See previous section.) Any conceptual workflow specification language should at least be capable of capturing moments of choice, sequential composition, parallel execution, and synchronization. Task Structures are capable of modeling these task dependencies. Moreover, Task Structures are very close to the diagramming languages used by today's workflow management systems. As mentioned in the introduction, Staffware uses a diagramming language corresponding to Task Structures. Compared to diagramming languages based on Petri nets, e.g., the modeling technique used by COSA (Software Ley/COSA Solutions) or Income (Promatis), the expressive power is limited. However, compared to Petri-net-based languages, Task Structures result in a more compact representation and termination is implicit, i.e., there is no need to identify a final task or final state. Section 3.1 introduces Task Structures. A formal definition of Task Structures is given in Section 3.2.

3.1 Informal explanation of Task Structures

Task Structures were introduced in [Bot89] to describe and analyze problem solving processes. In [WH90, WHO92] they were extended and used as a meta-process modeling technique for describing the strategies used by experienced information analysts. In [HN93] they were extended again and a formal semantics in terms of Process Algebra was given [BW90].

In Figure 1, the main concepts of Task Structures are graphically represented. They are discussed subsequently.

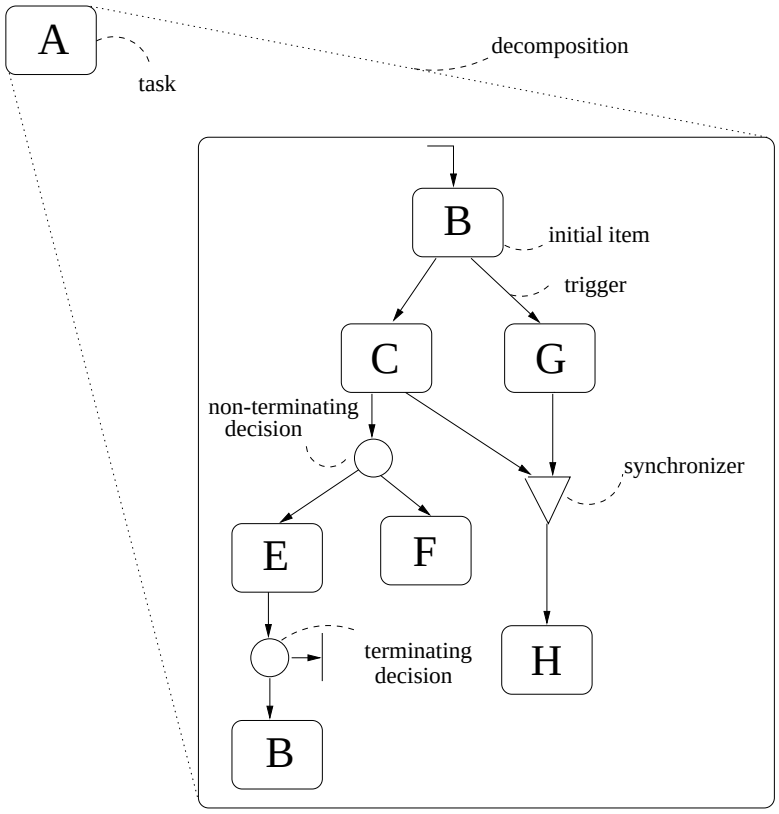

Figure 1: Graphical representation of Task Structure concepts

The central notion in Task Structures is the notion of a *task*. In a workflow context, tasks are basic work units that collectively achieve a certain goal. A task can be defined in terms of other tasks, referred to as its *subtasks*. This *decomposition* may be performed repeatedly until a desired level of detail has been reached. Tasks with the same name have the same decomposition, e.g. the tasks named B in Figure 1. Performing a task may involve choices between subtasks, *decisions* represent these moments of choice. Decisions coordinate the execution of tasks. Two kinds of decisions are distinguished: *terminating* and *non-terminating* decisions. A

decision that is terminating, may lead to termination of the execution path of that decision. If this execution path is the only active execution path of the supertask, the supertask terminates as well.

Triggers, graphically represented as arrows, model sequential order. In Figure 1 the task with name G can start after termination of the top task named B. *Initial items* are those tasks or decisions, that have to be performed first as part of the execution of a task that has a decomposition. Due to iterative structures, it may not always be clear which task objects are initial. Therefore, this has to be indicated explicitly. Finally, *synchronizers* deal with explicit synchronization. In Figure 1 the task named H can only start when the tasks with names C and G have terminated. It is important to note that tasks have XOR-join/AND-split semantics [Law97], i.e., a task can be triggered via *any* of the ingoing arcs (XOR-join) and triggers subsequent tasks via *all* of the outgoing arcs (AND-join). Decisions have an XOR-join/XOR-split semantics and synchronizers have an AND-join/AND-split semantics. There is no need for an explicit XOR-join building block or an explicit AND-split building block, because these routing constructs are already provided by normal tasks.

As a simple example of a Task Structure, consider Figure 2, which models an example taken from [CCPP98]. The example concerns a simple assembly line for desktop computers. The construction begins by preparing a cabinet, which may be either a *tower* or a *minitower*. At the same time, the motherboard is prepared, its CPU is inserted followed by the disk controller. When both cabinet and motherboard are ready, the motherboard is inserted in the cabinet and then step by step all other components are added. After the FDD is inserted, a CD-ROM is added if the cabinet is a tower. The assembly ends with the insertion of a hard drive and video ram.

3.2 Formal definition of Task Structures

In this paper we translate Task Structures into Petri nets. To allow for an unambiguous translation and to prove the correctness of the verification technique, we provide a formal definition of Task Structures. In this paper, we will only give the formal semantics implicitly (via the mapping onto Petri nets).

Definition 3.1

Formally, a Task Structure $\mathcal{W} = (\mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{S}, \mathcal{D}, \mathcal{D}_t, \mathsf{Trig}, \mathsf{Name}, \mathcal{I})$ *without decomposition consists of the following components:*

1. *A set* $\mathcal{X}$ *of task objects.* $\mathcal{X}$ *is the union of a set of synchronizers* $\mathcal{S}$*, a set of tasks* $\mathcal{T}$ *and a set of decisions* $\mathcal{D}$*. In* $\mathcal{D}$ *we distinguish a subset* $\mathcal{D}_t$ *consisting of the terminating decisions. For convenience, we define the set* $\mathcal{U}$*, the set of non-synchronizers, as* $\mathcal{T} \cup \mathcal{D}$*.*
2. *A relation* $\mathsf{Trig} \subseteq \mathcal{X} \times \mathcal{X}$ *of triggers, capturing which task object can start which other task object(s) (if any).*

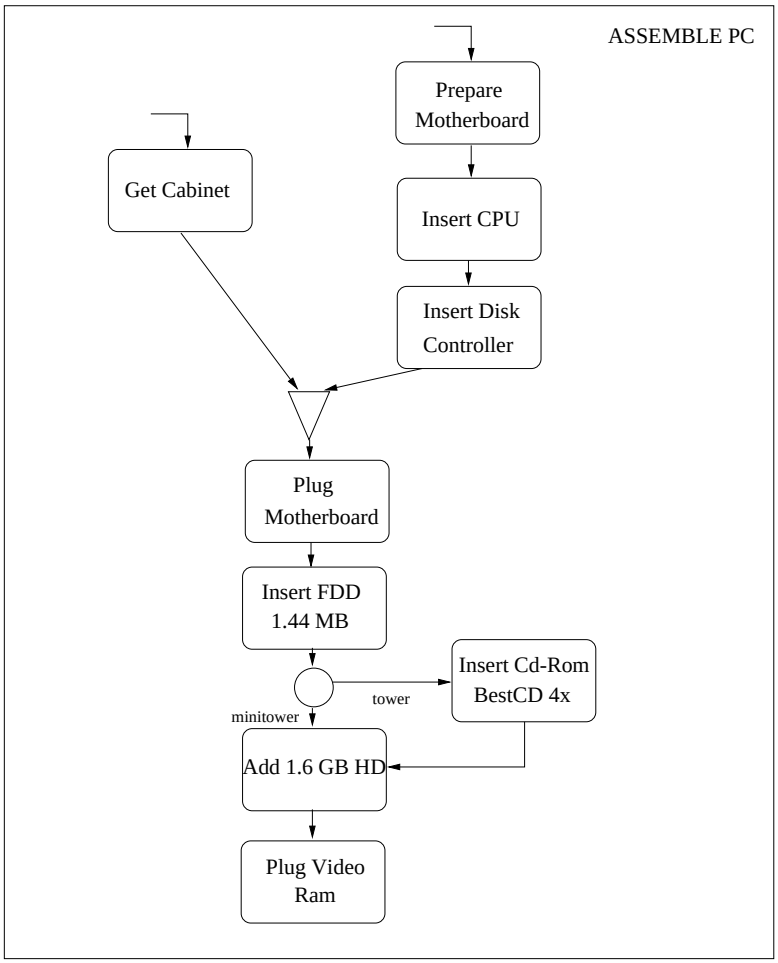

Figure 2: Main task for desktop assembly line

3. *A function* $\mathsf{Name}: \mathcal{T} \to \mathcal{N}$ *yielding the name of a task, where* $\mathcal{N}$ *is a set of names.*
4. *A subset* $\mathcal{I}$ *of the set of non-synchronizers* $\mathcal{U}$*, consisting of the initial items.*

□

In this paper decomposition of Task Structures in not considered. Hierarchical decomposition can be incorporated in a trivial way, but recursive decomposition increases the expressive power of Task Structures to such an extent that verification becomes computationally intractable (see [HO99]).

Example 3.1 The Task Structure of Figure 3 is formally captured by

$$\mathcal{X} \quad = \quad \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, s_1, d_1, d_2\},$$

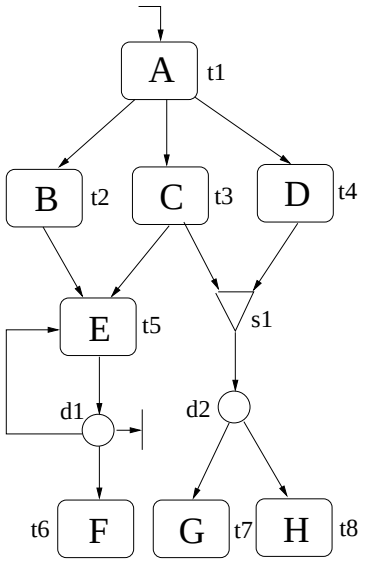

Figure 3: Example Task Structure

$$\begin{aligned} \mathcal{T} &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}, \\ \mathcal{S} &= \{s_1\}, \\ \mathcal{D} &= \{d_1, d_2\}, \\ \mathcal{D}_t &= \{d_1\}. \end{aligned}$$

Further, $t_1 \mathsf{Trig} t_2$, $t_1 \mathsf{Trig} t_3$, $t_1 \mathsf{Trig} t_4$, $t_2 \mathsf{Trig} t_5$, $t_3 \mathsf{Trig} t_5$ etc, and $\mathsf{Name}(t_1) = A$, $\mathsf{Name}(t_2) = B$, etc. Finally, $\mathcal{I} = \{t_1\}$. □

At this point it is important to emphasize an important difference between Task Structures and a special class of Petri nets, called workflow nets, which will be used as the basis for their formal verification. Task Structures do not have a unique final task as opposed to workflow nets. Therefore, it is difficult to identify the point where a Task Structure terminates. Hence, a major challenge in the mapping from Task Structures to workflow nets is to ensure that the result of the mapping indeed has a unique final place. The notions of *outdegree* and *indegree* play an important role in this mapping as they facilitate keeping track of the number of parallel streams at any point in time.

Definition 3.2

The outdegree of a task or synchronizer u is the number of task objects u triggers upon termination.

$$\mathsf{out}(u) = \#\{x \in \mathcal{X} \mid u \mathsf{Trig} x\}$$

The indegree of a synchronizer s is the number of its input task objects:

$$\mathsf{in}(s) = \#\{x \in \mathcal{X} \mid x \mathsf{Trig} s\}$$

□

Example 3.2 In Figure 3, the task named A has outdegree 3 indicating that three parallel streams are started after its termination. □

4 Mapping Task Structures onto WF-nets

In this section we consider the formal mapping of Task Structures to workflow nets, which are a special class of Petri nets. First, we introduce some standard concepts and notations for Petri nets. Then, we provide the mapping which is used to verify Task Structures using state-of-the-art Petri net technology.

4.1 Introduction to Petri nets

This section introduces the basic Petri net terminology and notations used in the remainder. Readers familiar with Petri nets can skip this section. Readers interested in more background material are referred to [DE95, Jen96, Mur89, Rei85].

The classical Petri net is a directed bipartite graph with two node types called *places* (graphically represented by circles) and *transitions* (graphically represented by thick lines). The nodes are connected via directed *arcs*. In this paper we consider Petri nets with *arc weights*. Arc weights represent the number of connections between a certain place and a certain transition.

Definition 4.1 (*Petri net with arc weights*)
A Petri net with arc weights is a quadruple (P, T, F, W)*:*

- P *is a finite set of places,*
- T *is a finite set of transitions* ($P \cap T = \varnothing$)*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation),*
- $W: F \to \mathbb{N}^+$ *is a function assigning weights to arcs.*

□

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.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state* M, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \to \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two

tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$. To compare states, we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

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

(1) A transition t is said to be *enabled* iff each input place p of t contains at least $W(p,t)$ tokens.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* $W(p,t)$ tokens from each input place p of t and *produces* $W(t,p)$ tokens for each output place p of t.

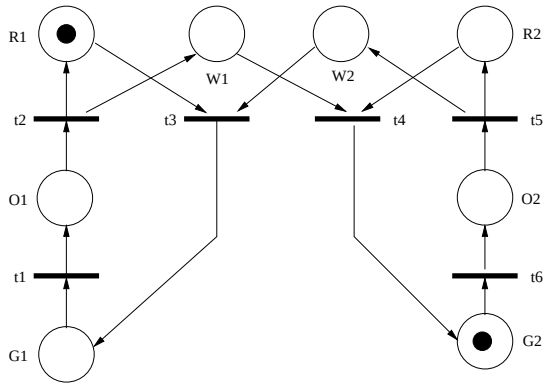

Figure 4: A Petri-net modeling two traffic lights.

Example 4.1 The Petri-net in Figure 4 models two traffic lights for the same intersection. The initial state is such that the light of the first traffic light is red (token in place $R1$) and the light of the second traffic light is green (token in place $G2$). Note that the moment a traffic light turns red, control is transferred (via places $W1$ and $W2$) to the other traffic light. In the initial state the only transition that is enabled is transition $t6$. Firing transition $t6$ would lead to the consumption of the token in $G2$ and the production of a token for $O2$. In this example all arc weights are equal to 1. Throughout this paper, we will only depict arc weights not equal to 1. Therefore, no arc weights are portrayed in Figure 4. □

Given a Petri net with arc weights $PN = (P, T, F, W)$ and a state M_1, we have the following notations:

- $M_1[t\rangle_{PN} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2

- $M_1[\,\rangle_{PN} M_2$: there is a transition t such that $M_1[t\rangle_{PN} M_2$
- $M_1[\sigma\rangle_{PN} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1} \in T^*$ leads from state M_1 to state M_n, i.e., $M_1[t_1\rangle_{PN} M_2[t_2\rangle_{PN} ...[t_{n-1}\rangle_{PN} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1[*\rangle M_n$) iff there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $M_1[\sigma\rangle M_n$. The subscript $_{PN}$ is omitted if it is clear which Petri net is considered. Note that the empty firing sequence is also allowed, i.e., $M_1[*\rangle M_1$.

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 $M[*\rangle M'$. Let us define some standard properties for Petri nets (cf. [DE95, Jen96, Mur89, Rei85]). These definitions have been added to make the paper self-contained.

Liveness and boundedness correspond to the dynamic behavior of a Petri net in a given state.

Definition 4.2 (*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. □

Definition 4.3 (*Bounded, safe*)
A Petri net (PN, M) is bounded iff, exists a natural number n such that for every reachable state and every place p 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. □

Connectedness, the free-choice property, and place invariants are static properties which do not depend on some initial marking.

Definition 4.4 (*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 4.5 (*Free-choice*)
A Petri net is a free-choice Petri net iff, for every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \varnothing$ implies $\bullet t_1 = \bullet t_2$. □

Definition 4.6 (*Semi-positive place invariant*)
A semi-positive place invariant Z is a function mapping places onto natural numbers, i.e., $Z: P \rightarrow I\!N$, such that for any transition $t \in T$, $\sum_{p \in \bullet t} W((p,t))Z(p) = \sum_{p \in t\bullet} W((t,p))Z(p)$. □

If for every place there is a semi-positive place invariant which assigns a positive weight to that place, then the Petri net is bounded for any initial marking [DE95, Mur89, Rei85]. This well-known property will be used to prove one of the main results presented in this paper.

Example 4.2 The Petri net shown in Figure 4 is live, bounded, safe, strongly connected, and free-choice. The net is live because from any of the 6 reachable states it is possible to enable any transition. The net is safe because the number of tokens in any of the 8 places in each of the 6 reachable states is either 0 or 1. Since the net is safe, it is also bounded. The net is strongly connected, because there is a directed path between any pair of nodes. The Petri net shown in Figure 4 is conflict free, i.e., no two transitions are sharing an input place. Therefore, it is also free-choice. Examples of semi-positive place invariants are: $R1 + O1 + G1$, $R2 + O2 + G2$, and $W1 + O1 + G1 + W2 + O2 + G2$. (Note that we use a notation similar to states.) Since every place is covered by some semi-positive place invariant, the net is bounded for any initial state. □

4.2 The mapping

Mapping Task Structures to classical Petri nets is relatively straightforward. Each task $t \in \mathcal{T}$ is mapped onto a place E_t, and a transition C_t is created which has as input place E_t and as output places all places corresponding to the task objects triggered by that task (if such places exist!). An exception is the treatment of synchronizers. For each synchronizer $s \in \mathcal{S}$ and each task object $x \in \mathcal{X}$ such that $x\mathsf{Trig}s$, a place with the name $\sigma_{x,s}$ is created. Synchronization is now achieved by creating a transition H_s which has all these places as input places and has as output places the places corresponding to the task objects triggered by that synchronizer.

Each decision $d \in \mathcal{D}$ is mapped to a place E_d and has for each of its choices $e \in \mathcal{X}$ an arc to a unique transition $G_{d,e}$ which has an outgoing arc to place E_e. If d is terminating as well, there is an arc from the place corresponding to that decision to a transition F_d without output places (if that transition fires it will simply consume a token from that place).

Finally, the initial marking of the net is a marking with exactly one token in each of the places E_i with i an initial item. The following definition captures this mapping formally.

Definition 4.7
Given a Task Structure $\mathcal{W} = (\mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{S}, \mathcal{D}, \mathcal{D}_t, \mathsf{Trig}, \mathsf{Name}, \mathcal{I})$ *the corresponding Petri net* $\mathcal{P}_\mathcal{W} = (P_\mathcal{W}, T_\mathcal{W}, F_\mathcal{W}, W_\mathcal{W})$ *and its initial marking* M_0 *are defined by:*

$$
\begin{aligned}
P_\mathcal{W} &= \{E_x \mid x \in \mathcal{T} \cup \mathcal{D}\} \cup \{\sigma_{x,s} \mid x\mathsf{Trig}s \wedge s \in \mathcal{S}\} \\
T_\mathcal{W} &= \{C_t \mid t \in \mathcal{T}\} \cup \{F_d \mid d \in \mathcal{D}_t\} \cup \{G_{d,e} \mid d\mathsf{Trig}e \wedge d \in \mathcal{D}\} \cup \{H_s \mid s \in \mathcal{S}\} \\
F_\mathcal{W} &= \{(E_t, C_t) \mid t \in \mathcal{T}\} \cup \\
&\quad \{(E_d, F_d) \mid d \in \mathcal{D}_t\} \cup \\
&\quad \{(E_d, G_{d,e}) \mid d \in \mathcal{D} \wedge d\mathsf{Trig}e\} \cup \\
&\quad \{(\sigma_{x,s}, H_s) \mid s \in \mathcal{S} \wedge x\mathsf{Trig}s\} \\
&\quad \{(C_t, E_x) \mid t \in \mathcal{T} \wedge t\mathsf{Trig}x \wedge x \notin \mathcal{S}\} \cup \\
&\quad \{(C_t, \sigma_{t,s}) \mid t \in \mathcal{T} \wedge t\mathsf{Trig}s \wedge s \in \mathcal{S}\} \cup
\end{aligned}
$$

$$
\begin{array}{rcl}
 & & \{(G_{d,x}, E_x) \mid d \in \mathcal{D} \wedge d\mathsf{Trig}x \wedge x \notin \mathcal{S}\} \cup \\
 & & \{(G_{d,x}, \sigma_{d,x}) \mid d \in \mathcal{D} \wedge d\mathsf{Trig}x \wedge x \in \mathcal{S}\} \cup \\
 & & \{(H_s, E_x) \mid s \in \mathcal{S} \wedge s\mathsf{Trig}x \wedge x \notin \mathcal{S}\} \cup \\
 & & \{(H_s, \sigma_{s,s'}) \mid s \in \mathcal{S} \wedge s\mathsf{Trig}s' \wedge s' \in \mathcal{S}\} \\
W_{\mathcal{W}} & = & \lambda t \in F_{\mathcal{W}}.1 \\
M_0 & = & \lambda p \in P_{\mathcal{W}}.\mathit{if}\ p \in \{E_i \mid i \in \mathcal{I}\}\ \mathit{then}\ 1\ \mathit{else}\ 0\ \mathit{fi}
\end{array}
$$

□

The above definition uses the λ-notation which is the standard notation for unnamed functions, i.e., $W_{\mathcal{W}}$ is a function which maps every arc onto weight 1 and M_0 is a function which maps places onto either 1 or 0 tokens (depending on whether they correspond to an initial item or not).

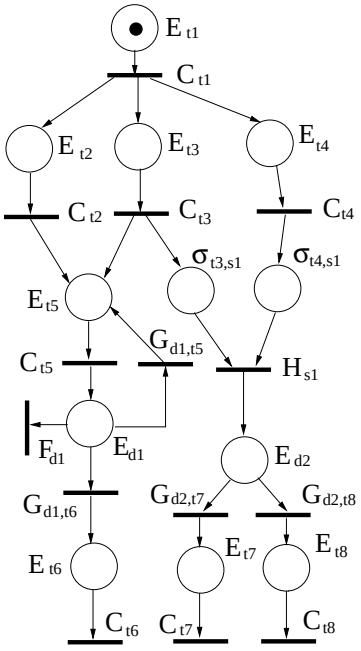

Figure 5: Mapping of Task Structure of Figure 3 to Petri net

Example 4.3 Figure 5 contains the result of applying the previously described translation to the Task Structure of Figure 3. Note that all arc weights are 1. Therefore, they are not depicted. □

A Petri net which models the process aspect of a workflow, is called a *WorkFlow net* (WF-net, [Aal98c, Aal97]). It should be noted that a WF-net specifies the dynamic behavior of a

single case in isolation. For the verification of Task Structures we consider the mapping of Task Structures to WF-nets.

Definition 4.8 (*WF-net*)
A Petri net $PN = (P, T, F, W)$ is a WF-net (Workflow net) if and only if:

(i) PN has two special places: i and o. Place i is a source place: $\bullet i = \varnothing$. Place o is a sink place: $o\bullet = \varnothing$.

(ii) If we add a transition t^* to PN which connects place o with i (i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.

□

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 if it enters the workflow management system and is deleted once it is completely handled by the workflow management system, i.e., the WF-net specifies the life-cycle of a case. The second requirement in Definition 4.8 (the Petri net extended with t^* should be strongly connected) states that for each transition t (place p) there should be a path from place i to o via t (p). This requirement has been added to avoid 'dangling tasks and/or conditions', i.e., tasks and conditions which do not contribute to the processing of cases.

The readers familiar with [Aal98c, Aal97] will note that, in contrast to earlier definitions, Definition 4.8 allows for arbitrary arc weights. This extension is needed to allow for the translation of Task Structures to WF-nets.

Mapping a Task Structure to a WF-net is slightly more complex as the result of the fact that a WF-net should have a unique output place. This is achieved through the introduction of a "shadow place" S which keeps track of the number of parallel streams at any point in time (this will be inversely proportional to the number of tokens in S). Every transition C_t with n output arcs ($n > 1$) has an input arc from S with weight $n - 1$. This is a situation where the original task t starts n task objects in parallel upon termination, hence we have $n - 1$ extra parallel streams at that point in time.

Every transition H_s corresponding to a synchronizer with m input arcs and p output arcs has an output arc to S with weight $p - m$ if $m > p$, as this reflects m parallel streams coming together and p new streams being generated; the nett result of this being $m - p$ less parallel streams. Similarly, if $p > m$ there will be an arc from S to H_s with weight $p - m$ as in that case the nett result of the synchronizer is that $p - m$ new parallel streams are generated. In addition, every transition C_t with no output arcs and every transition F_d (which never has an output arc) have an arc to S. Such transitions reflect termination of a stream.

There is a transition t_{init} with an input arc from the input place i and an output arc to S with weight $c - |\mathcal{I}|$. The number c is the maximal number of parallel streams that could be active

at any point in time. $|\mathcal{I}|$ represents the number of initial items, each of which will initially generate a parallel stream. From t_{init} there are also output arcs to all places corresponding to initial items. There is also a transition t_{end} with an output arc to output place o and an input arc from S with weight c.

The constant c corresponds to the maximal number of parallel streams in the Task Structure. As it turns out, it is very hard to statically determine this constant for a particular given Task Structure. However, it is not necessary to have a precise estimation of c. The number c has to be chosen in such a way that no transition (except t_{end}) will ever get blocked because place S does not contain sufficient tokens. Therefore, any upperbound will do. For the moment, it suffices to know that for correct Task Structures such an upperbound exists (see Sections 4.3 and 5). It is important to note that during the execution of the Task Structure (i.e., after firing t_{init} and before firing t_{end}), the number of active parallel streams is equal to c minus the number of tokens in S.

Definition 4.9

Given a Task Structure $\mathcal{W} = (\mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{S}, \mathcal{D}, \mathcal{D}_t, \mathsf{Trig}, \mathsf{Name}, \mathcal{I})$*, the corresponding WF-net* $\mathcal{N}_{\mathcal{W}} = (P_{\mathcal{W}_\mathcal{N}}, T_{\mathcal{W}_\mathcal{N}}, F_{\mathcal{W}_\mathcal{N}}, W_{\mathcal{W}_\mathcal{N}})$ *and its initial marking* M_0 *are defined as follows.*

$$
\begin{aligned}
P_{\mathcal{W}_\mathcal{N}} &= P_{\mathcal{W}} \cup \{i, o, S\} \\
T_{\mathcal{W}_\mathcal{N}} &= T_{\mathcal{W}} \cup \{t_{init}, t_{end}\} \\
F_{\mathcal{W}_\mathcal{N}} &= F_{\mathcal{W}} \cup \{(i, t_{init}), (t_{init}, S), (S, t_{end}), (t_{end}, o)\} \cup \\
&\quad \{(S, C_t) \mid t \in \mathcal{T} \wedge \mathsf{out}(t) > 1\} \cup \{(H_s, S) \mid s \in \mathcal{S} \wedge \mathsf{out}(s) < \mathsf{in}(s)\} \cup \\
&\quad \{(S, H_s) \mid s \in \mathcal{S} \wedge \mathsf{in}(s) < \mathsf{out}(s)\} \cup \{i \in \mathcal{I} \mid (t_{init}, E_i\} \cup \\
&\quad \{(C_t, S) \mid t \in \mathcal{T} \wedge \mathsf{out}(t) = 0\} \cup \{(F_d, S) \mid d \in \mathcal{D}_t\}
\end{aligned}
$$

The arc weight function $W_{\mathcal{W}_\mathcal{N}}$ *assigns* 1 *to every arc with the following exceptions:*

$$
\begin{aligned}
W_{\mathcal{W}_\mathcal{N}}((S, C_t)) &= \mathsf{out}(t) - 1 \textit{ if } \mathsf{out}(t) > 1 \\
W_{\mathcal{W}_\mathcal{N}}((S, H_s)) &= \mathsf{out}(t) - \mathsf{in}(s) \textit{ if } \mathsf{out}(s) > \mathsf{in}(s) \\
W_{\mathcal{W}_\mathcal{N}}((H_s, S)) &= \mathsf{in}(s) - \mathsf{out}(s) \textit{ if } \mathsf{in}(s) > \mathsf{out}(s) \\
W_{\mathcal{W}_\mathcal{N}}((t_{init}, S)) &= c - |\mathcal{I}| \\
W_{\mathcal{W}_\mathcal{N}}((S, t_{end})) &= c
\end{aligned}
$$

where c *is an upperbound for the maximal number of parallel streams. In the initial marking place* i *has a token and no other place has a token.*

$$M_0 = \lambda p \in P_{\mathcal{W}_\mathcal{N}}.\textit{if } p = i \textit{ then } 1 \textit{ else } 0 \textit{ fi}$$

□

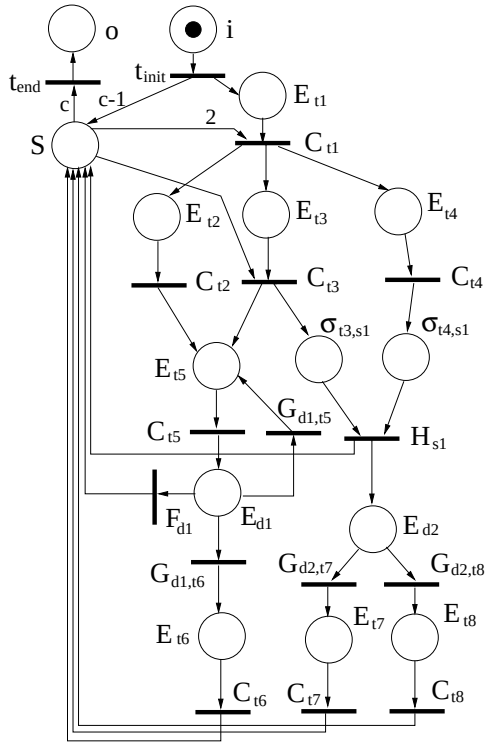

Figure 6: Mapping of Task Structure of Figure 3 to WF-net

Example 4.4 Figure 6 shows the result of applying the previously defined mapping to the Task Structure of Figure 3. Since the maximal number of parallel streams in the Task Structure is 4, the constant c has to be at least 4. Note that place S contains exactly four tokens the moment the Task Structure terminates, i.e., t_{end} fires the moment the Task Structure of Figure 3 terminates. □

4.3 An upperbound for the number of parallel streams

Place S, the shadow place, has been added to keep track of the number of parallel streams. This place is crucial for deciding whether a Task Structure has terminated but is not part of the semantics. Therefore, it is of the utmost importance that S does not change the behavior. Consider for example the WF-net shown in Figure 6. If $c = 2$, then C_{t1} is blocked right from the start. If $c = 3$, then C_{t3} is blocked until F_{d1} or C_{t6} fires. These examples show that, if we choose a value for c which is too small, the behavior changes. Adding a new place will never extend the behavior of a Petri net. (An additional place can only restrict the behavior because the place can block transitions but it cannot enable transitions which are not enabled in the net without the place, see [DE95].) Therefore, it suffices to choose c in such a way that place

S can never block a transition (other than t_{end}). If the number of maximal parallel streams is assumed to be finite, which seems to be a reasonable assumption, then it is always possible to find such a c. However, the fact that such a c exists is not very helpful, because it does not give us a concrete value.

Fortunately, there is a very pragmatic solution. Set c to *maxint*, the largest integer value that can be handled by the application or programming language that is used for verification. It is reasonable to assume that the maximum number of parallel streams does not exceed this value. Moreover, during the verification phase (see next section) it is possible to check whether c was not too small. Since the complexity of the verification algorithm does not depend upon c, it is possible to choose such a large value without any harm. From a practical point of view, it is no problem to choose maxint as an upperbound for the number of parallel streams. Nevertheless, more elegant solutions are possible using the rich theory of Petri nets. In fact, a place which does not restrict the firing of transitions is called an *implicit place* and this notion has been studied quite well in Petri-net literature, cf., [CS90, Ber86, Ber87].

An implicit place, also called a redundant place, is a place which always contains sufficient tokens to allow for the firing of the transitions connected to it. The constant c in the WF-net constructed using Definition 4.9, should be chosen in such a way that place S is implicit in the net without t_{end}. Several authors have investigated techniques to find so-called structural implicit places ([CS90, Ber86, Ber87]). A structural implicit place is a place which is guaranteed to be implicit by the structure of the Petri net. Every structural implicit place is an implicit place, but there may be implicit places which are not structurally implicit. Since structural implicit places can be found without constructing the reachability graph (polynomial time), it is possible to use these techniques for efficiently establishing a suitable value for c.

5 Verification of soundness

The correctness, effectiveness, and efficiency of the business processes supported by the workflow management system are vital to the organization (cf. [Aal98c, GHS95, JB96]). A workflow process definition which contains errors may lead to angry customers, back-log, damage claims, and loss of goodwill. Flaws in the design of a workflow definition may also lead to high throughput times, low service levels, and a need for excess capacity. This is why it is important to *analyze* a workflow process definition before it is put into production. Basically, there are three types of analysis:

- *validation*, i.e., testing whether the workflow behaves as expected,
- *verification*, i.e., establishing the correctness of a workflow, and
- *performance analysis*, i.e., evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization.

Validation can be done by interactive simulation: a number of fictitious cases are fed to the system to see whether they are handled well. For verification and performance analysis more advanced analysis techniques are needed.

Today's workflow management systems give limited support to performance analysis. Most workflow management systems provide a rudimentary simulator or provide a gateway to a simulation tool. Simulation can be used to estimate key performance indicators by experimenting with the specified workflow under the assumption of a specific behavior of the environment. Examples of key performance indicators are: average throughput time of cases, average waiting time, occupation rates of resources, service levels, and the average number of pending cases.

Most workflow management systems do not give any support for the verification of workflows. As a result, workflow process definitions become operational before they are thoroughly checked for correctness. This often results in runtime errors, which need to be repaired on-the-fly at high costs. Examples of such errors are:

- Deadlock: A case gets stuck in some state where it is not possible to execute any tasks.
- Livelock: A case is trapped in an infinite loop where it is possible to execute tasks but no real progress is possible.
- Dead task: A task can never be executed for any case.

The above errors can be detected without knowing anything about the particular application, i.e., the errors correspond to domain independent anomalous behavior. There are also errors which can only be detected with knowledge about the application. An example of such an error could be the scenario where a customer receives goods but not the bill (or receives the bill twice). Another example is the situation where tasks are executed in wrong order (e.g., the bill is sent before the goods). In this paper, we focus on *domain or application independent errors* because we are interested in a general-purpose verification tool which can be applied without adding additional information for the purpose of verification.

Both manufacturers and users of workflow management systems see the need for analysis tools which take care of the verification of workflows. Unfortunately, most manufacturers do not have the technology to build such tools. In this section, we will show that workflows specified in terms of Task Structures can be verified using state-of-the-art Petri-net-based analysis techniques.

5.1 Sound Task Structures

Before we focus on verification techniques, we need establish the correctness criteria we want to use. In our opinion any Task Structure should satisfy the following four properties:

1. *The Task Structure should be connected, i.e., there should be a path between any two task objects ignoring the direction of triggers.*

2. *Every task object should be on a path from some initial item to a terminating task object (i.e., a task object with no outgoing triggers or a terminating decision).*

3. *There are no dead tasks, i.e., any task can be executed by choosing the appropriate route through the Task Structure.*

4. *From any reachable state, it is possible to reach a terminal state.*

These requirements are quite reasonable. Task Structures composed out of parts which are not connected or task objects which are not on a path from an initial item to a final task object do not make any sense. Moreover, dead tasks or Task Structures which cannot terminate clearly correspond to design errors.

Since we use Petri nets (in particular WF-nets) to verify the correctness of Task Structures, we have to map the four requirements onto WF-nets. The first two requirements correspond to the properties that were already stated in the definition of WF-nets (Def. 4.8). These two properties can be verified statically, i.e., they only relate to the structure of the corresponding WF-net. In the remainder of this paper, we will assume that every Task Structure satisfies these two properties. The third requirement corresponds to the property that no transition is dead in the corresponding WF-net, i.e., starting in state i it is possible to fire any transition at least once. The last requirement corresponds to the property that from any state reachable in the WF-net (starting from state i), it is possible to reach a state with a token in o and the moment a transition puts a token in o all other places should be empty. These last two requirements correspond to the so-called *soundness property.*

Definition 5.1 (*Sound*)
A Task Structure $\mathcal{W}$ mapped onto a WF-net $\mathcal{N}_{\mathcal{W}} = (P, T, F, W)$ 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:

$$\forall_M (i[*\rangle M) \Rightarrow (M[*\rangle o)$$

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

$$\forall_M (i[*\rangle M \ \wedge \ M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in $(\mathcal{N}_{\mathcal{W}}, i)$. Formally:

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

□

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 4). The soundness property relates to the dynamics of the corresponding WF-net. The first requirement in Definition 5.1 states that starting from the initial state (state i), it is always possible to reach the state with one token in place o (state o). If we assume a strong notion of fairness, then the first requirement implies that eventually state o is reached. Strong fairness means in every infinite firing sequence, each transition fires infinitely often. The fairness assumption is reasonable in the context of workflow management: All choices are made (implicitly or explicitly) by applications, humans or external actors. Clearly, they should not introduce an infinite loop. Note that the traditional notions of fairness (i.e., weaker forms of fairness with just local conditions, e.g., if a transition is enabled infinitely often, it will fire eventually) are not sufficient. See [KA99] 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 in Definition 5.1 [GCEV72]. The last requirement states that there are no dead transitions (tasks) in the initial state i.

For the WF-net shown in Figure 6, which corresponds to the Task Structure shown in Figure 3, it is quite easy to see that it is sound. However, for complex Task Structures it is far from trivial to check the soundness property.

5.2 A necessary and sufficient condition for soundness

Given a Task Structure which corresponds to the WF-net $\mathcal{N}_{\mathcal{W}} = (P, T, F, W)$, we want to decide whether it is sound. In the remainder of this section, we will talk about the soundness of the WF-net, rather than the Task Structure, because all proofs will be done in a Petri net setting. Since a Task Structure can be mapped onto a WF-net using Definition 4.9, there is a clear correspondence between the Task Structure $\mathcal{W}$ and the WF-net $\mathcal{N}_{\mathcal{W}} = (P, T, F, W)$. Note that a Task Structure can be mapped onto a WF-net in polynomial time.

Since most of the results presented in this section hold for arbitrary WF-nets, we use $\mathcal{N}$ rather than $\mathcal{N}_{\mathcal{W}}$. Only for the results specific for WF-nets originating from a Task Structure $\mathcal{W}$ (Definition 4.9), we will use the notation $\mathcal{N}_{\mathcal{W}}$.

For verification purposes, we define an extended net $\overline{\mathcal{N}} = (\overline{P}, \overline{T}, \overline{F}, \overline{W})$. $\overline{\mathcal{N}}$ is the Petri net that we obtain by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{\mathcal{N}} = (\overline{P}, \overline{T}, \overline{F}, \overline{W})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, $\overline{F} = F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\}$, and for $\langle x, y \rangle \in F$, $\overline{W}(\langle x, y \rangle) = W(\langle x, y \rangle)$, $\overline{W}(\langle o, t^* \rangle) = 1$, and $\overline{W}(\langle t^*, i \rangle) = 1$.

For an arbitrary WF-net $\mathcal{N}$ and the corresponding extended Petri net $\overline{\mathcal{N}}$ we will prove the following result:

> *$\mathcal{N}$ is sound if and only if $(\overline{\mathcal{N}}, i)$ is live and bounded.*

First, we prove the 'if' direction.

Lemma 5.1 If $(\overline{\mathcal{N}}, i)$ is live and bounded, then $\mathcal{N}$ is a sound WF-net.

Proof:

$(\overline{\mathcal{N}}, i)$ is live, i.e., for every reachable state M there is a firing sequence which leads to a state in which t^* is enabled. Since o is the input place of t^*, we find that for any state M reachable from state i it is possible to reach a state with at least one token in place o. Consider an arbitrary reachable state $M' + o$, i.e., a state with at least one token in place o. In this state t^* is enabled. If t^* fires, then the state $M' + i$ is reached. Since $(\overline{\mathcal{N}}, i)$ is also bounded, M' should be equal to the empty state. Hence requirements (i) and (ii) hold and proper termination is guaranteed. Requirement (iii) follows directly from the fact that $(\overline{\mathcal{N}}, i)$ is live. Hence, $\mathcal{N}$ is a sound WF-net. □

To prove the 'only if' direction, we first show that the extended net is bounded.

Lemma 5.2 If $\mathcal{N}$ is sound, then $(\overline{\mathcal{N}}, i)$ is bounded.

Proof:

Assume that $\mathcal{N}$ is sound. The set of reachable markings of $(\mathcal{N}, i)$ is equal to the set of reachable markings of the extended net $(\overline{\mathcal{N}}, i)$, because if transition t^* in $\overline{\mathcal{N}}$ fires, the net returns to the initial state i which was already reachable. Therefore, $(\overline{\mathcal{N}}, i)$ is bounded if and only if $(\mathcal{N}, i)$ is bounded. Now assume that $(\mathcal{N}, i)$ is not bounded. Since $\mathcal{N}$ is not bounded there are two states M_i and M_j such that $i[*\rangle M_i$, $M_i[*\rangle M_j$ and $M_j > M_i$. (See for example the proof that the coverability tree is finite in Peterson [Pet81] (Theorem 4.1)). However, since $\mathcal{N}$ is sound we know that there is a firing sequence σ such that $M_i[\sigma\rangle o$. Therefore, there is a state M such that $M_j[\sigma\rangle M$ and $M > o$. Hence, it is not possible that $\mathcal{N}$ is both sound and not bounded and the lemma holds. □

Now we can prove that $(\overline{\mathcal{N}}, i)$ is live.

Lemma 5.3 If $\mathcal{N}$ is sound, then $(\overline{\mathcal{N}}, i)$ is live.

Proof:

Assume $\mathcal{N}$ is sound. By Lemma 5.2 we know that $(\overline{\mathcal{N}}, i)$ is bounded. Because $\mathcal{N}$ is sound we know that state i is a so-called home-marking of $\overline{\mathcal{N}}$, i.e., for every state M' reachable from $(\overline{\mathcal{N}}, i)$ it is possible to return to state i. In the original net $(\mathcal{N}, i)$, it is possible to fire an arbitrary transition t (requirement (iii)). This is also the case in the extended net. Therefore, $(\overline{\mathcal{N}}, i)$ is live because for every state M' reachable from $(\overline{\mathcal{N}}, i)$ it is possible to reach a state which enables an arbitrary transition t. □

Theorem 5.1 A WF-net $\mathcal{N}$ is sound if and only if $(\overline{\mathcal{N}}, i)$ is live and bounded.

Proof:

Follows directly from Lemmas 5.1, 5.2 and 5.3. □

Theorem 5.1 is an extension of the results presented in [Aal97, Aal98c]. In [Aal97, Aal98c] we restrict ourselves to WF-nets with arc weights 1. The extension to WF-nets with arbitrary arc weights is straightforward. Theorem 5.1 holds for any WF-net. However, if the WF-net $\mathcal{N}$ originates from a Task Structure, then $(\overline{\mathcal{N}}, i)$ is bounded by definition.

Lemma 5.4 Let $\mathcal{W}$ be a Task Structure and let $\mathcal{N}_{\mathcal{W}}$ be the WF-net constructed using Definition 4.9. $(\overline{\mathcal{N}_{\mathcal{W}}}, i)$ is bounded.

Proof:

To prove boundedness, we construct a semi-positive place invariant (see Def. 4.6) by assigning weight c (see Def. 4.9) to the places i and o and assigning weight 1 to all other places. To prove that this is a place invariant, we consider all types of transitions that can be constructed using Definition 4.9. Let t be a task, d be a (terminating) decision, s a synchronizer, and e be a task object. Transitions of type C_t consume one token from E_t and $\mathsf{out}(t) - 1$ tokens from S and produce one token for each of the $\mathsf{out}(t)$ output places of C_t. Transitions of type F_d consume one token from E_d and produce one token for S. Transitions of type $G_{d,e}$ consume one token from E_d and produce one token for E_e or $\sigma_{d,e}$. A transition of type H_s consumes one token from each of the $\mathsf{in}(s)$ input places of predecessor task objects and produces one token for each of the $\mathsf{out}(s)$ successor task objects. If $\mathsf{in}(s) > \mathsf{out}(s)$, then the transition of type H_s will also produce $\mathsf{in}(s) - \mathsf{out}(s)$ tokens for S. If $\mathsf{in}(s) < \mathsf{out}(s)$, then the transition will consume $\mathsf{out}(s) - \mathsf{in}(s)$ tokens from S. Hence, for transitions of type C_t, F_d, $G_{d,e}$, and H_s, the number of tokens consumed is equal to the number of tokens produced, i.e., the input/output behavior of these transitions is consistent with the place invariant. Transition t_{init} consumes one token with weight c from i and produces one token for each of the $|\mathcal{I}|$ initial items and $c - |\mathcal{I}|$ tokens for place S. Transition t_{out} consumes c tokens of weight one from place S and produces one token with weight c for place o. Transition t^* consumes 1 token of weight c from place o and produces one token with weight c for place i. Hence, the remaining transitions t_{init}, t_{out}, and t^* also do not invalidate the place invariant. Since the place invariant assigns a positive weight to all places in the extended WF-net $\overline{\mathcal{N}_{\mathcal{W}}}$, the net is structurally bounded. □

Since the $(\overline{\mathcal{N}_{\mathcal{W}}}, i)$ is bounded by definition, it suffices to check liveness to verify that a Task Structure is sound.

Corollary 5.1 Let $\mathcal{W}$ be a Task Structure and let $\mathcal{N}_{\mathcal{W}}$ be the WF-net constructed using Definition 4.9. $\mathcal{W}$ is sound, if and only if, $(\overline{\mathcal{N}_{\mathcal{W}}}, i)$ is live.

Perhaps surprisingly, the verification of the soundness property boils down to checking whether the extended Petri net is live! This means that we can use standard Petri-net-based analysis tools to decide soundness. At the moment there are about 25 tools available for the analysis of liveness properties (see [Mor98]). Most of these tools construct the coverability graph [Pet81]. Although some of these tools are implemented very efficiently and use state-of-the-art state-space reduction techniques such as BDD's and stubborn sets, the complexity of the algorithm to construct the coverability graph can be worse than primitive recursive space. This is consistent with the observations in [HOR98], where it is shown that the problem of deciding whether a given Task Structure terminates is DSPACE(exp)-hard. Therefore, only 'brute-force' approaches to check soundness are possible to verify an arbitrary Task Structure. However, for many subclasses, see [Aal98c], it is possible to use more sophisticated techniques which exploit the structure of the WF-net. Some of the Petri-net-based tools (cf. [Mor98]) support structural techniques, i.e., for specific subclasses of Petri nets it is possible to avoid using a 'brute-force' approach. In fact many workflow management systems only allow for workflow processes which are in essence *free-choice* [DE95] and for free-choice nets the soundness property can be verified in polynomial time [Aal98c]. In free-choice nets it is not possible to mix choice and synchronization into one routing construct, i.e., either a choice is preceded by a synchronization or a synchronization is preceded by a choice. Since in many workflow management systems choices are only allowed *inside* tasks and synchronization is done *outside* tasks, the resulting workflow process definitions correspond to free-choice nets. See [Aal98c] for more details. Note that any Petri net constructed using Definition 4.7 is free-choice, i.e., Task Structures also correspond to free-choice nets. However, to verify soundness we need to introduce place S which in many cases violates the free-choice property.

It is interesting to observe that the value c in Definition 4.9 (i.e., the upperbound for the number of parallel streams) does not influence the efficiency of the verification process. If a 'brute-force' approach is used, the coverability graph is constructed and the number of reachable states should not be influenced by the value of c. (If the value of c influences the number of reachable states, then c was too small or the number of parallel streams is unbounded. Note that both situations can be detected by inspection of the coverability graph.) If more sophisticated techniques are used, the structure of the WF-net is exploited and the value of c is irrelevant.

In this section, we have shown that for the verification of Task Structures we can benefit from Petri-net theory and tools. This will be illustrated by the case study presented in Section 7. The starting point of this case study is a workflow process specified in Staffware. This process definition is automatically translated into a format readable by Woflan as is described in the next section.

6 Implementation based on Staffware and Woflan

To put the approach presented in this paper to work, we have developed a link between Staffware and Woflan. In this section we briefly describe both tools and the link between them.

Woflan (WOrkFLow ANalyzer, [Aal98c, AHV97, VBA99]) is an analysis tool which can be used to verify the correctness of a workflow process definition. The analysis tool uses state-of-the-art techniques to find potential errors in the definition of a workflow process. Woflan is designed as a WFMS-independent analysis tool. In principle it can interface with many workflow management systems. At the moment, Woflan can interface with the WFMS COSA (Software Ley [SL96]), the WFMS METEOR (LSDIS [SKM]), the WFMS Staffware (Staffware [Sta97]), and the BPR-tool Protos (Pallas Athena [Pal97]). In the future we hope to extend the set of workflow management systems which can interface with Woflan. Woflan uses Petri-net-based analysis routines to analyze the workflows at hand. One of the central issues which is analyzed by Woflan is the soundness property (see Definition 5.1). Woflan uses a brute force approach by constructing the coverability graph to decide soundness. This turns out to be satisfactory from a practical point of view. Even complex workflows contain less than 100 tasks and have less than 200.000 states. This is no problem for Woflan. The only way to deal with larger workflows from a managerial point of view, is to split the workflow into subflows which can be verified in a compositional way. Thus, the brute force approach is quite acceptable. However, deciding whether the workflow definition is sound is not sufficient. In many cases more requirements need to be satisfied. Moreover, if the workflow definition is not sound, then the user should be guided in detecting the source of the error and support should be given to repair the error. This is the reason Woflan offers a large selection of analysis methods:

- Syntactical checks, e.g., detection of tasks without input or output condition.
- Detection of potential errors by listing suspicious constructs, e.g., constructs violating the free-choice property, AND-split's complemented by OR-join's, OR-split's complemented by AND-join's, and parts of the net which are not S-coverable.
- Detection of dynamic errors by listing unbounded places, non-safe places, dead transitions and non-live transitions.
- Place and transition invariants. The absence or presence of certain invariants indicates the source of an error.
- Verification of the soundness property.

Woflan has an on-line help-facility which guides the user in using the tool and helps to understand the analysis results. A screenshot of Woflan showing the on-line help and some diagnostics is given in Figure 7. A detailed description of the analysis routines supported by

Woflan is outside the scope of this paper. For more information the reader is referred to [AHV97, VBA99, VA99].

Figure 7: A screenshot of Woflan.

Staffware is one of the leading workflow management systems. There are two reasons for using Staffware. First, we selected Staffware because the diagramming technique used is very close to Task Structures. Second, the large installed base makes Staffware an interesting platform to test our approach. We have used the current version of Staffware, i.e., Staffware 97 [Sta97]. This version of Staffware is used by more than 550,000 users worldwide and runs on more than 4500 servers. In 1998, it was estimated by the Gartner Group that Staffware has 25 percent of the global market [Cas98]. The routing elements used by Staffware are the Start, Step, Wait, Condition, and Stop. These routing elements correspond to respectively initial items, tasks, synchronizers, non-terminating decisions and terminating decisions. In the next section, we will give an example of a Staffware process definition and the corresponding Task Structure. For more information about the technical aspects of Staffware, the interested reader can download product information from http://www.staffware.com.

The link between Staffware and Woflan is realized as follows. The *Staffware Graphical Workflow Definer* (GWD) stores workflow process definitions in a file-based repository. For each workflow process definition, a so-called GWD file is created. This file contains all information relevant for the control-flow perspective. This file is converted into a format readable by Woflan (a so-called TPN file). The translation is essentially the same as the one described by Definition 4.9. Woflan

reads the resulting file and generates the diagnostics mentioned before. The translation is based on Staffware 97, i.e., the current version. The new version of Staffware, named Staffware 2000, is being rolled out throughout the year 1999. Staffware 2000 has essentially the same features with respect to the modeling of workflow processes. Therefore, we expect little problems in upgrading the link from Staffware 97 to Staffware 2000.

Both Woflan and the link with Staffware can be downloaded from http://www.win.tue.nl/~woflan.

7 Case: travel agency

To illustrate the approach presented in this paper, we consider a small workflow process in a travel agency. The workflow process is used to illustrate modeling of Task Structures and verification via WF-nets. Moreover, we will also use the example to present Staffware, Woflan, and the link between these two systems.

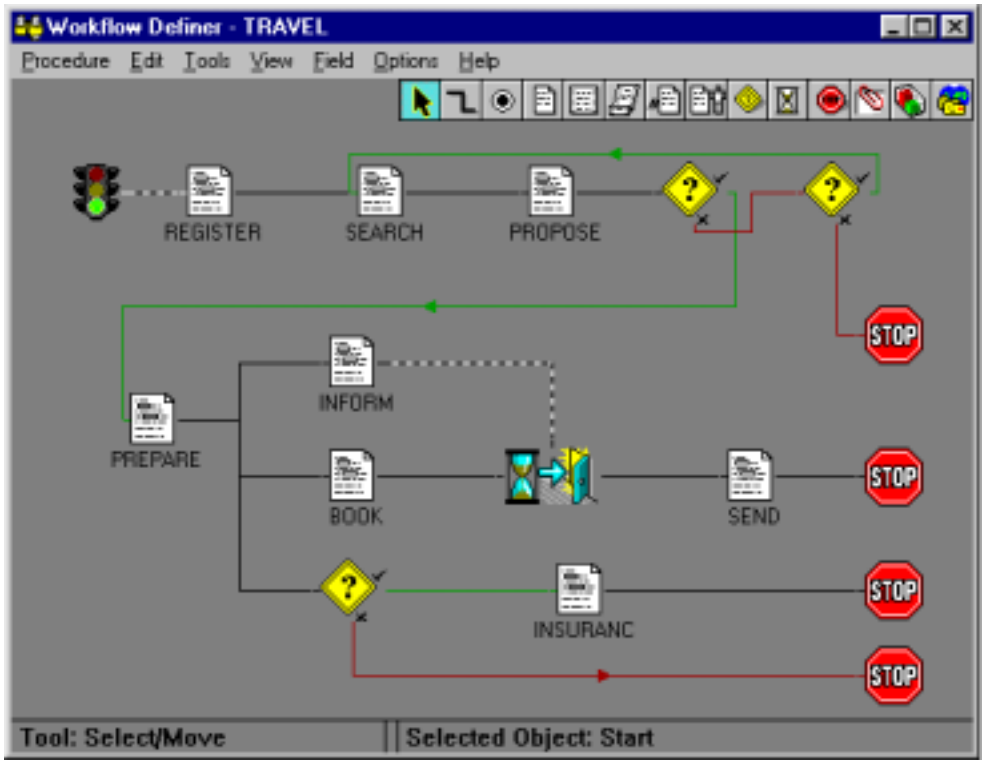

Figure 8: Screenshot of Staffware.

Example 7.1 A travel agency receives requests from customers interested in booking a trip. Each request is registered by an employee of the travel agency. During this first step in

the workflow process, the destination and the desired departure and arrival date of the planned journey are registered. Other constraints and preferences are also collected by the employee. All registered information is used to search for transport and accommodation. Then, a number of alternatives are proposed to the customer. There are three possibilities: (1) the customer selects a trip, (2) the customer requests for more alternatives, or (3) the customer is not interested anymore. If the customer requests for more alternatives, an employee will start looking for other alternatives which are again proposed to the customer. If the customer selects a trip, some preparations are made. After these preparations, the business partners (airline company, hotel, etc.) are informed and the trip is booked. During the preparation step, the customer indicates whether some insurance is needed. If the customer requests for insurance, the insurance is effected. Note that the tasks inform, book and insurance can be executed in parallel. After completing the tasks inform and book the appropriate documents are sent to the customer. □

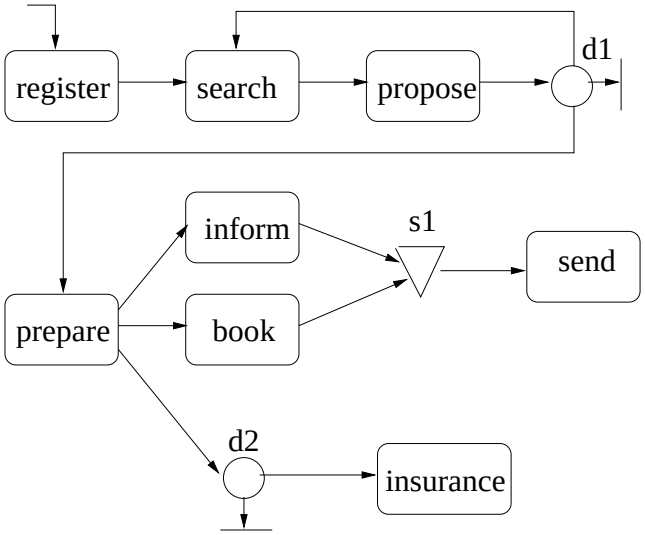

Figure 9: The Task Structure corresponding to the Staffware workflow process definition shown in Figure 8.

We have used the workflow management system *Staffware* ([Sta97]) to realize the workflow process. Figure 8 shows the workflow process in Graphical Workflow Definer (GWD) of Staffware. There is a one-to-one correspondence between the graphical diagramming technique used by Staffware and Task Structures. Figure 9 shows the workflow process of the travel agency in terms of a Task Structure. Comparing Figure 8 and Figure 9 shows that the diagramming techniques are very similar. Staffware uses the following workflow objects:

- *Start.*
  The start object, represented by a traffic light, corresponds to the initial state and points

to the initial item.

- *Step.*
  A step corresponds to a task and is represented by a form symbol. The input/output behavior is identical to tasks in a Task Structure.

- *Wait.*
  A wait is used to synchronize parallel flows and is represented by a sand timer. The semantics of a wait corresponds to a synchronizer in a Task Structure.

- *Condition.*
  A condition is represented by a diamond and is used for conditional routing. A condition in Staffware corresponds to a decision between two alternatives in a Task Structure (i.e. a decision with outdegree 2).

- *Stop.*
  A stop is represented by a stop sign and signifies that no further processing is required. The stop can be used to indicate that steps or conditions are terminating.

Although there is a one-to-one correspondence between the workflow objects used by Staffware and the objects present in Task Structures, there are some subtle differences. First of all, Staffware is more restrictive in the sense that there is just one initial item (i.e., $|\mathcal{I}| = 1$) and decisions can have just one preceding task and just two outcomes (i.e., $\mathsf{in}(d) = 1$ and $\mathsf{out}(d) = 2$ for any $d \in \mathcal{D}$). Note for example that decision $d1$ in Figure 9 corresponds to two conditions in Figure 8. Second, Staffware distinguishes between normal steps, automatic steps, steps with a deadline, priority steps, and event steps. In this paper, we do not consider these aspects and assume all steps to be normal steps. For the control-flow perspective, the different types of steps are not relevant. Similarly, we abstract from other, non-control-flow related aspects, such as work queues, form definitions, and application wrappers. Third, Staffware allows for a construct which withdraws work items from a work queue, i.e., it is possible to cancel an enabled task. This construct is not considered in this paper and is also not used in Figure 8. However, the link between Staffware and Woflan takes this additional control-flow-related feature into account. Finally, tasks (called steps in Staffware) have OR-join semantics rather than XOR-join semantics. This subtle difference is only relevant if multiple preceding tasks can enable a task *at the same time*. In a good design, such a situation should not occur because a design exploiting the difference between the OR-join semantics and the XOR-join semantics cannot be sound. Consider for example a task C with two preceding tasks A and B. Suppose that both A and B are enabled at a certain point in time. If one of these two tasks is executed, then C is enabled. If C is executed before the other preceding task is executed, then C will be executed twice (e.g., the occurrence sequence ACBC). However, if the execution of C is delayed until the other preceding task is executed, then C is only executed once (e.g., the occurrence sequence ABC). Since only the temporal ordering of tasks determines whether task C will be executed once or twice, the corresponding workflow process definition can never

satisfy the requirements stated in Definition 5.1. Either the environment expects C to fire once resulting in a potential dangling stream or the environment expects C to fire twice resulting in a potential deadlock. Therefore, it is reasonable to assume XOR-join semantics for the purpose of verification.

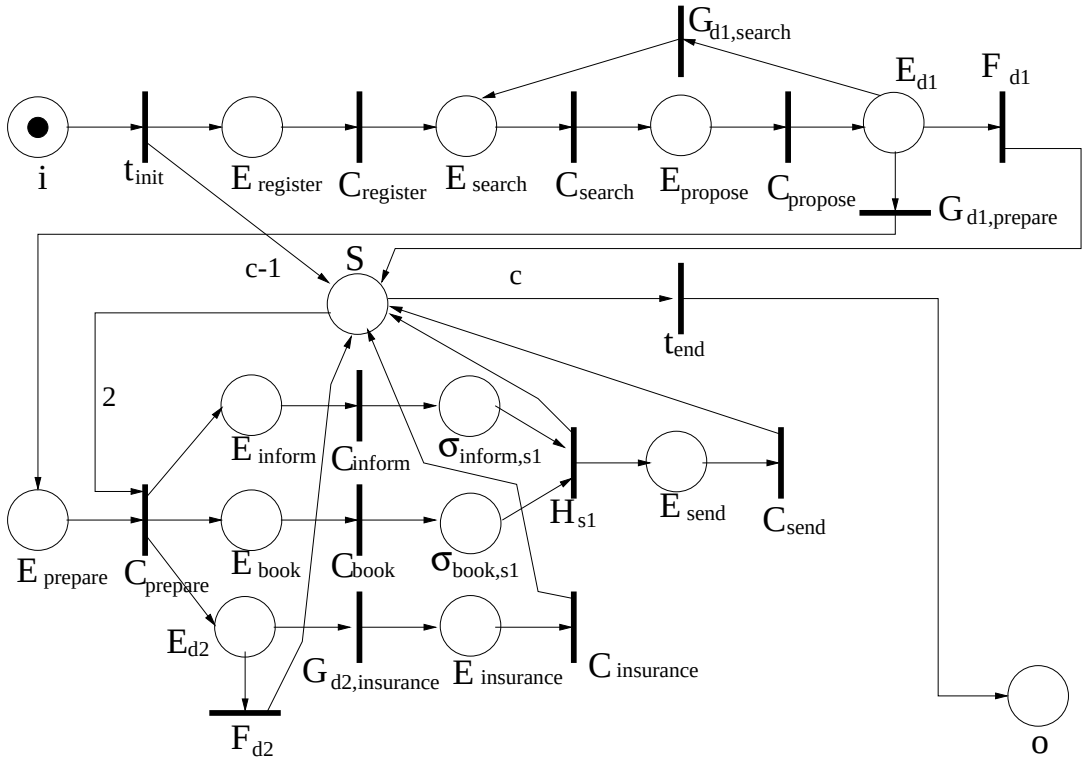

Figure 10: The workflow net corresponding to the Task Structure shown in Figure 9.

We can map the Task Structure shown in Figure 9 onto a WF-net using Definition 4.9. Figure 10 shows the result. Since the maximal number of parallel tasks is three we set c to three. We use the Petri-net-based analysis tool Woflan to verify the correctness of the Task Structure shown in Figure 9.

If we use Woflan to analyze the WF-net shown in Figure 10, Woflan will report that the WF-net is sound. Therefore, the corresponding Task Structure in Figure 9 is also sound. Figure 7 shows a screenshot of Woflan during the analysis of the WF-net shown in Figure 10. The screenshot shows two of the ten windows containing diagnostics generated by Wolfan. One of the windows visible in Figure 7 shows that the WF-net is live, bounded, and sound, has 15 places, 16 transitions, 42 connections, and that the coverability graph has 25 states. The other window gives information about invariants. The first invariant in this window is the invariant constructed in the proof of Lemma 5.4. The screenshot also shows a fragment of the on-line help-facility of Woflan.

If the Task Structure is sound, there is no need to look at all the diagnostics generated by Woflan. However, if the Task Structure is not sound, it is worthwhile to browse through the diagnostics provided by Woflan. To illustrate this we introduce the following error. We add a trigger connecting the task *insurance* and the synchronizer *s1* in Figure 9. In the corresponding WF-net shown in Figure 10 the connection between $C_{insurance}$ and S is replaced by a place and two arrows, connecting $C_{insurance}$ and H_{s1}. If we analyze this WF net, Woflan points out that the Task Structure is not sound because the WF-net is not live. The Task Structure deadlocks the moment the customer decides not to take insurance. Next, we introduce another error. A trigger connecting the task *send* and the task *search* is added to the Task Structure shown in Figure 9 and the decision *d1* is made non-terminating. In the corresponding WF-net, transition F_{d1} is removed and C_{send} is connected to E_{search} instead of S. If analyze the WF-net with Woflan, Woflan points out that the Task Structure is not sound because transition t_{end} is dead, i.e., it cannot fire. This indicates that the Task Structure in Figure 9 is not able to terminate, i.e., it contains an infinite loop. Figure 11 shows some of the diagnostics generated by Woflan for this Task Structure. Woflan reports that t_{end} is dead. Woflan also points out that there are three semi-positive transition invariants not containing t_{start} nor t_{end}. These invariants correspond to the three infinite loops cases cannot escape from.

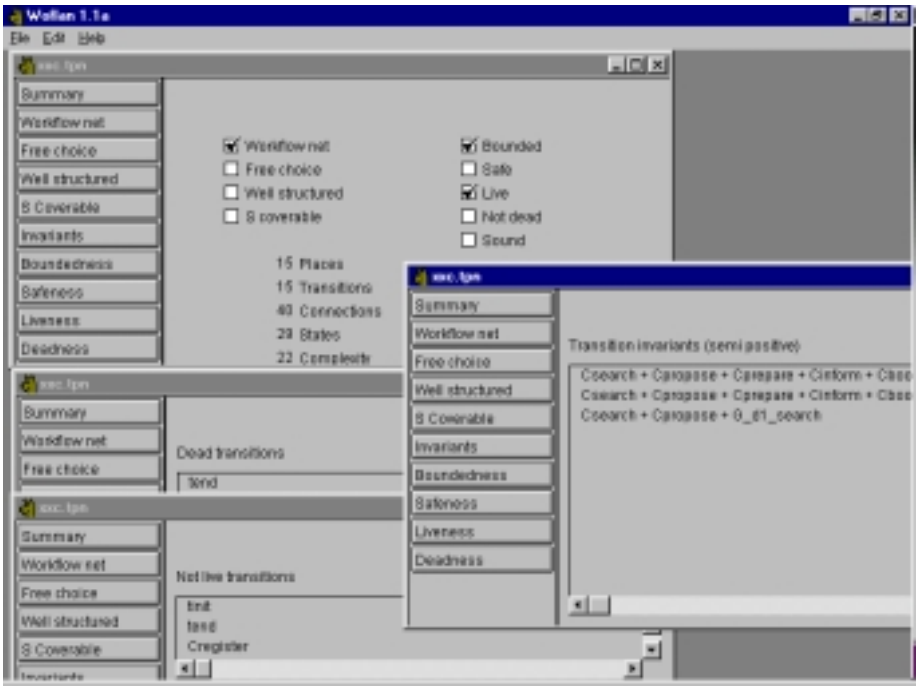

Figure 11: A screenshot showing some of the diagnostics generated by Woflan.

The examples given in this section show that the results obtained by the automatic translation from workflow scripts in Staffware to a format readable by Woflan and the subsequent

analysis using Woflan are valuable. The current problem is that the results are not presented in the Staffware Graphical Workflow Definer but in separate windows. Therefore, the most challenging problem is to translate the output generated by Woflan to diagnostic information understandable by Staffware users not familiar with Petri nets. At the moment, we are working on a new release of Woflan with a completely new user interface to overcome these (and other) problems.

8 Related work

Petri nets have been proposed for modeling workflow process definitions long before the term "workflow management" was coined and workflow management systems became readily available. Consider for example the work on Information Control Nets, a variant of the classical Petri nets, in the late seventies [Ell79, EN93]. For the reader interested in the application of Petri nets to workflow management, we refer to the two most recent workshops on workflow management held in conjunction with the annual International Conference on Application and Theory of Petri Nets [MEM94, AME98] and an elaborate paper on workflow modeling using Petri nets [Aal98c].

Many researchers have proposed languages specifically for workflow modeling. Task Structures are an example of such a language. Task Structures were introduced in the late eighties [Bot89] and have been extended in several ways [WH90, WHO92, HN93].

Only a few papers in the literature focus on the verification of workflow process definitions. In [HOR98] some verification issues have been examined and the complexity of selected correctness issues has been identified, but no concrete verification procedures have been suggested. In [Aal97] and [AAH98] concrete verification procedures based on Petri nets have been proposed. This paper builds upon the technique presented in [Aal97]. This technique was not immediately applicable since it assumes explicit termination in a given sink place and is restricted to Petri nets with arc weights equal to one (i.e., ordinary P/T nets). The technique presented in [AAH98] has been developed for checking the consistency of transactional workflows including temporal constraints. However, the technique is restricted to acyclic workflows and only gives necessary conditions (i.e., not sufficient conditions) for consistency. In [SO99a] a reduction technique has been proposed. This reduction technique uses a correctness criterion which corresponds to soundness and the class of workflow processes considered are in essence free-choice Petri nets. This approach is not applicable to Task Structures or Staffware, because the approach requires explicit termination and does not allow for iteration. Moreover, it is not possible to extend the approach presented in [SO99a] using the concept of the so-called "shadow place": Such an addition requires non-free-choice behavior (i.e., merging choice and synchronization) and the ability to handle multiple tokens in one place. Some researchers worked on the compositional verification of workflows [Aal98b, Bas98, Voo98] using well-known Petri-net results such as the refinement rules in [Val79]. In contrast to the approach presented in this paper, these compositional approaches also assume explicit termination.

As far as we know only two tools have been developed to verify workflows: *Woflan* [VA99] and *FlowMake* [SO99b]. Woflan is the tool used in this paper. FlowMake is a tool based on the reduction technique described in [SO99a] and can interface with the IBM MQSeries Workflow product. FlowMake requires explicit termination and can only handle acyclic workflows, therefore it can not be used to verify Staffware models.

The work presented in this paper builds on previous research reported by the authors [Aal97, HOR98]. The main contribution of this paper is a concrete technique for verifying workflows which do not have a uniquely identified point of termination. All other approaches reported in the literature assume a final node, i.e., a terminal state or an end task. The construction given in Section 4 is used to make implicit termination explicit. Some of the leading workflow management systems allow for implicit termination, because it simplifies the design (i.e., there is no need to join parallel and alternative paths at the end). Therefore, the results presented in this paper are very relevant. The paper also extends the results presented in [Aal97], because WF-nets with arbitrary arc weights are considered. To demonstrate the applicability of our approach we have established a link between Staffware and Woflan. Although Staffware represents 25 percent of the global workflow market, no such verification facility has been presented before.

9 Conclusion

In this paper, we have mapped Task Structures onto Petri nets. For verification purposes we added information about the number of parallel flows to the net by introducing a "shadow place" S and adding a start place/transition and an end place/transition. The resulting Petri net corresponds to the class of WF-nets defined in [Aal97, Aal98c] extended with arc weights. Since Task Structures are similar to diagramming techniques used in leading workflow management systems, the translation from Task Structures to Petri nets constitutes a basis for bridging the gap between the diagramming techniques in commercial systems and Petri-net-based analysis techniques. To illustrate this, we showed that Staffware, the worlds leading workflow management system, has a diagramming technique which is very similar to Task Structures.

The fact that we can map Task Structures onto WF-nets, allows for the verification of Task Structures using state-of-the-art Petri-net-based analysis techniques. In this paper, we showed that soundness property (a set of minimal requirements any Task Structure should satisfy) for Task Structures corresponds to liveness of the corresponding extended WF-net. Therefore, we can use standard Petri-net analysis tools to verify the correctness of a Task Structure. To illustrate the applicability of our approach, we developed a link between Staffware and Woflan, a Petri-net-based workflow analyzer, to enable the automatic verification of workflow processes specified in Staffware.

Acknowledgements

The authors would like to thank Eric Verbeek (scientific programmer, EUT) for implementing the link between Staffware and Woflan. Moreover, we would like to thank the anonymous referees for their useful comments.

References

[AAH98] N.R. Adam, V. Atluri, and W. Huang. Modeling and Analysis of Workflows using Petri Nets. *Journal of Intelligent Information Systems*, 10:131–158, 1998.

[Aal97] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azema 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.

[Aal98a] 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, 1998.

[Aal98b] W.M.P. van der Aalst. Finding Errors in the Design of a Workflow Process: A Petri-net-based Approach. In W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors, *Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98)*, volume 98/7 of *Computing Science Reports*, pages 60–81, Lisbon, Portugal, 1998. Eindhoven University of Technology, Eindhoven.

[Aal98c] 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.

[AHV97] 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, Sept 1997. University of Hamburg (FBI-HH-B-205/97).

[AME98] W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors. *Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98)*, Lisbon, Portugal, June 1998. UNINOVA, Lisbon.

[Bas98] T. Basten. *In Terms of Nets: Systems Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998.

[Ber86] G. Berthelot. Checking Properties of Nets Using Transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, Berlin, 1986.

[Ber87] G. Berthelot. Transformations and decompositions of nets. 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 360–376. Springer-Verlag, Berlin, 1987.

[BH97] A.P. Barros and A.H.M. ter Hofstede. Realizing the Full Potential of Workflow Modeling: A Practical Perspective. *Journal of Information Technology*, 3(2):59–86, December 1997.

[BHP97] A.P. Barros, A.H.M. ter Hofstede, and H.A. Proper. Towards Real-Scale Business Transaction Workflow Modelling. In A. Olivé and J.A. Pastor, editors, *Proceedings of the Ninth International Conference CAiSE'97 on Advanced Information Systems Engineering*, volume 1250 of *Lecture Notes in Computer Science*, pages 437–450, Barcelona, Spain, June 1997. Springer Verlag.

[Bot89] P.W.G. Bots. *An Environment to Support Problem Solving.* PhD thesis, Delft University of Technology, Delft, The Netherlands, 1989.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge University Press, Cambridge, United Kingdom, 1990.

[Cas98] R. Casonato. Gartner group research note 00057684, production-class workflow: A view of the market. http://www.gartner.com, 1998.

[CCPP98] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data & Knowledge Engineering*, 24(3):211–238, January 1998.

[CS90] J.M. Colom and M. Silva. Improving the Linearly Based Characterization of P/T Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–146. Springer-Verlag, Berlin, 1990.

[DE95] J. Desel and J. Esparza. *Free choice Petri nets*, volume 40 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1995.

[EKR95] C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock and C. Ellis, editors, *Conf. on Organizational Computing Systems*, pages 10 – 21. ACM SIGOIS, ACM, Aug 1995. Milpitas, CA.

[Ell79] 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.

[EN93] C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993.

[GCEV72] 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.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[HN93] A.H.M. ter Hofstede and E.R. Nieuwland. Task structure semantics through process algebra. *Software Engineering Journal*, 8(1):14–20, January 1993.

[HO99] A.H.M. ter Hofstede and M.E. Orlowska. On the Complexity of Some Verification Problems in Process Control Specifications. *Computer Journal*, 42(5), 1999. (in press).

[HOR98] 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.

[JB96] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, 1996.

[Jen96] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1996.

[KA99] E. Kindler and W.M.P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 70(6):269–274, 1999.

[Kou95] T.M. Koulopoulos. *The Workflow Imperative*. Van Nostrand Reinhold, New York, 1995.

[Law97] P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.

[MEM94] G. De Michelis, C. Ellis, and G. Memmi, editors. *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, Zaragoza, Spain, June 1994.

[Mor98] K. Mortensen. Petri nets tools database, petri net home page. http://www.daimi.au.dk/PetriNets/tools/db.html, 1998.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[Pal97] Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, The Netherlands, 1997.

[Pet62] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962. (In German).

[Pet81] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, 1981.

[Rei85] W. Reisig. *Petri nets: an introduction*, volume 4 of *Monographs in theoretical computer science: an EATCS series*. Springer-Verlag, Berlin, 1985.

[Sch96] T. Schäl. *Workflow Management for Process Organisations*, volume 1096 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1996.

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

[SL96] Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1996.

[SO99a] W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99)*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, Berlin, 1999.

[SO99b] W. Sadiq and M.E. Orlowska. FlowMake Product Information, Distributed Systems Technology Centre, Queensland, Australia. http://www.dstc.edu.au/Research/Projects/FlowMake/productinfo/index.html, 1999.

[Sta97] Staffware. *Staffware 97 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 1997.

[SV90] M. Silva and R. Valette. Petri Nets and Flexible Manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424 of *Lecture Notes in Computer Science*, pages 274–417. Springer-Verlag, Berlin, 1990.

[VA99] E. Verbeek and W.M.P. van der Aalst. Woflan Home Page, Eindhoven University of Technology, Eindhoven, The Netherlands. http://www.win.tue.nl/~woflan, 1999.

[Val79] R. Valette. Analysis of Petri Nets by Stepwise Refinements. *Journal of Computer and System Sciences*, 18:35–46, 1979.

[VBA99] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. Computing Science Report 99/02, Eindhoven University of Technology, Eindhoven, 1999.

[Voo98] M. Voorhoeve. Modeling and Verification of Workflow Nets. In W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors, *Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98)*, volume 98/7 of *Computing Science Reports*, pages 96–108, Lisbon, Portugal, 1998. Eindhoven University of Technology, Eindhoven.

[WFM96] WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.

[WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, 1990. Springer-Verlag.

[WHO92] G.M. Wijers, A.H.M. ter Hofstede, and N.E. van Oosterom. Representation of Information Modelling Knowledge. In V.-P. Tahvanainen and K. Lyytinen, editors, *Next Generation CASE Tools*, volume 3 of *Studies in Computer and Communication Systems*, pages 167–223. IOS Press, 1992.