Discovering Workflow Performance Models from Timed Logs

W.M.P. van der Aalst and B.F. van Dongen

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

Abstract. Contemporary workflow management systems are driven by explicit process models, i.e., a completely specified workflow design is required in order to enact a given workflow process. Creating a workflow design is a complicated time-consuming process and typically there are discrepancies between the actual workflow processes and the processes as perceived by the management. Therefore, we have developed techniques for discovering workflow models. Starting point for such techniques are so-called "workflow logs" containing information about the workflow process as it is actually being executed. In this paper, we extend our existing mining technique α [4] to incorporate time. We assume that events in workflow logs bear timestamps. This information is used to attribute timing such as queue times to the discovered workflow model. The approach is based on Petri nets and timing information is attached to places. This paper also presents our workflow-mining tool EMiT. This tool translates the workflow log of several commercial systems (e.g., Staffware) to an independent XML format. Based on this format the tool mines for causal relations and produces a graphical workflow model expressed in terms of Petri nets.

Key words: Workflow mining, workflow management, data mining, Petri nets.

1 Introduction

During the last decade workflow management concepts and technology [2, 3, 11, 17–19] have been applied in many enterprise information systems. Workflow management systems such as Staffware, IBM MQSeries, COSA, etc. offer generic modeling and enactment capabilities for structured business processes. By making graphical process definitions, i.e., models describing the life-cycle of a typical case (workflow instance) in isolation, one can configure these systems to support business processes. Besides pure workflow management systems many other software systems have adopted workflow technology. Consider for example ERP (Enterprise Resource Planning) systems such as SAP, PeopleSoft, Baan and Oracle, CRM (Customer Relationship Management) software, etc. Despite its promise, many problems are encountered when applying workflow technology. One of the problems is that these systems require a workflow design, i.e., a designer has to construct a detailed model accurately describing the routing of work. Modeling a workflow is far from trivial: It requires deep knowledge of the

workflow language and lengthy discussions with the workers and management involved.

Instead of starting with a workflow design, we start by gathering information about the workflow processes as they take place. We assume that it is possible to record events such that (i) each event refers to a task (i.e., a well-defined step in the workflow), (ii) each event refers to a case (i.e., a workflow instance), and (iii) events are totally ordered. Any information system using transactional systems such as ERP, CRM, or workflow management systems will offer this information in some form. Note that we do not assume the presence of a workflow management system. The only assumption we make, is that it is possible to collect workflow logs with event data. These workflow logs are used to construct a process specification which adequately models the behavior registered. We use the term *process mining* for the method of distilling a structured process description from a set of real executions.

case identifier	task identifier	timestamp (date:time)
case 1	A	08-05-2002 : 08:15
case 2	A	08-05-2002 : 08:24
case 3	A	08-05-2002 : 09:30
case 2	B	08-05-2002 : 10:24
case 5	A	08-05-2002 : 10:24
case 4	A	08-05-2002 : 10:25
case 3	B	08-05-2002 : 10:26
case 1	F	08-05-2002 : 11:45
case 4	B	08-05-2002 : 11:46
case 2	C	08-05-2002 : 12:23
case 2	D	08-05-2002 : 15:14
case 5	F	08-05-2002 : 15:17
case 3	D	08-05-2002 : 15:19
case 1	G	08-05-2002 : 16:26
case 4	C	08-05-2002 : 16:29
case 5	G	08-05-2002 : 16:43
case 3	C	09-05-2002 : 08:22
case 4	D	09-05-2002 : 08:45
case 3	E	09-05-2002 : 09:10
case 4	E	09-05-2002 : 10:05
case 2	E	09-05-2002 : 10:12
case 2	G	09-05-2002 : 10:46
case 3	G	09-05-2002 : 11:23
case 4	G	09-05-2002 : 11:25

Table 1. A workflow log.

To illustrate the principle of process mining, we consider the workflow log shown in Table 1. This log contains information about five cases (i.e., workflow instances). The log shows that for two cases (1 and 5) the tasks A, F and G

have been executed. For case 2 and case 4 the tasks A, B, C, D, E and G have been executed. For case 3 the same tasks have been executed. However, C and D are swapped. Each case starts with the execution of A and ends with the execution of G. If task C is executed, then also task D is executed. However, for some cases task C is executed before task D, and for some the other way around. Based on the information shown in Table 1 and by making some assumptions about the completeness of the log (i.e., assuming that the cases are representative and a sufficient large subset of possible behaviors is observed), we can deduce for example the process model shown in Figure 1. The model is represented in terms of a Petri net [23]. The Petri net starts with task A and finishes with task G. These tasks are represented by transitions. After executing A there is a choice between either executing B or executing F. After executing B, tasks C and D are executed in parallel, followed by E. Finally, after executing either E or F, G can be executed. To execute C and D in parallel, tasks B corresponds a so-called AND-split and task E corresponds a so-called AND-join. Note that for this example we assume that two tasks are in parallel if they appear in any order. By distinguishing between start events and end events for tasks it is possible to explicitly detect parallelism. However, for the moment we assume atomic actions.

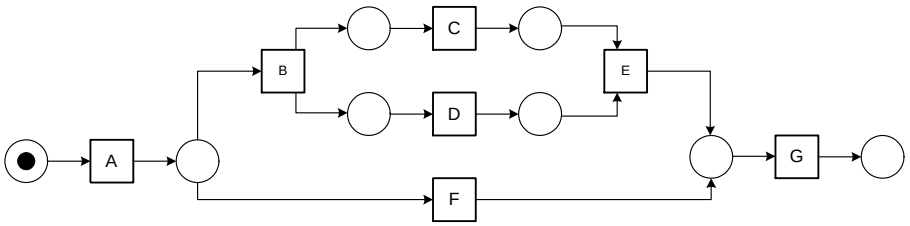

Fig. 1. A process model corresponding to the workflow log.

The first two columns of Table 1 contain the minimal information we assume to be present to provide process models such as the one shown in Figure 1. In [4] we have provided an algorithm (named α) which can be used to discover a large class for process models from logs containing this minimal information. However, in many applications the workflow log contains a *timestamp* for each event and this information can be used to extract information about the performance of the process, e.g., bottlenecks in the process. In this paper, we explore ways to mine timed workflow logs. In the log shown in Table 1 we can only see the completion time of tasks. In most logs we can also see when tasks are started. However, even using this minimal information we can calculate all kinds of performance measures. Figure 2 shows the minimal, maximal, and average time each case spends in a certain stage of the process. (The times indicated refer to the time a token spends in a place between production and consumption [23].) For example, the mean time between the completion of B and the completion of C is 573,

the minimum is 119, the maximum is 1316 minutes.[1] Similar information is given for all other places. Note that because the log shown in Table 1 only captures completion times, we cannot calculate service times and resource usage. Moreover, we do not know the exact arrival time of each case. The log only shows when the first step in the process (task A) is completed. Therefore, we can only calculate the flow time starting from the completion of A. As indicated the in Figure 2, the average flow time is 1101 minutes.

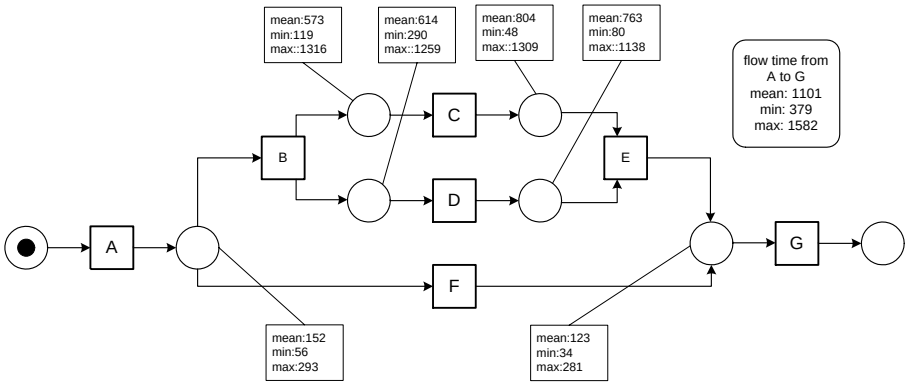

Fig. 2. Performance information (mean, min, and max) extracted from the workflow log is indicated in the process model.

For this simple example, it is quite easy to construct a process model that is able to regenerate the workflow log and attribute timing information to place. For larger workflow models this is much more difficult. For example, if the model exhibits alternative and parallel routing, then the workflow log will typically not contain all possible combinations. Consider 10 tasks which can be executed in parallel. The total number of interleavings is $10! = 3628800$. It is not realistic that each interleaving is present in the log. Moreover, certain paths through the process model may have a low probability and therefore remain undetected. Noisy data (i.e., logs containing exceptions) can further complicate matters.

In this paper, we do not focus on issues such as noise. We assume that there is no noise and that the workflow log contains "sufficient" information. Under these ideal circumstances we investigate whether it is possible to *discover* the workflow process and *extract timing information*, i.e., for which class of workflow models is it possible to accurately construct the model and performance information by merely looking at their logs. This is not as simple as it seems. Consider for example the process model shown in Figure 1. The corresponding

[1] B and C are executed for three cases: 2, 3, and 4. The time between the completion of B and C is respectively 119 (case 2), 1316 (case 3), and 283 (case 4) minutes. Therefore, the average time attached to the corresponding place is (119+1316+283)/3=573 minutes.

workflow log shown in Table 1 does not explicitly show any information about AND/XOR-splits and AND/XOR-joins. Nevertheless, this information is needed to accurately describe the process. These and other problems are addressed in this paper. For this purpose we use *workflow nets* (WF-nets) [1, 3]. WF-nets are a class of Petri nets specifically tailored towards workflow processes. The Petri net shown in figures 1 and 2 is an example of a WF-net.

The remainder of this paper is organized as follows. First, we introduce some preliminaries, i.e., Petri nets and WF-nets. In Section 3 we present an algorithm that discovers a large class of workflow processes. Section 4 extends this algorithm to also extract timing information. Section 5 presents the tool we have developed to mine timed workflow logs. The application of this tool to Staffware logs is demonstrated in Section 6. To conclude we provide pointers to related work and give some final remarks.

2 Preliminaries

This section introduces the techniques used in the remainder of this paper. First, we introduce standard Petri-net notations, then we define a subclass of Place/Transition nets tailored towards workflow modeling and analysis (i.e., WF-nets [1, 3]).

2.1 Petri nets

We use a variant of the classic Petri-net model, namely Place/Transition nets. For an elaborate introduction to Petri nets, the reader is referred to [10, 22, 23].

Definition 2.1. (P/T-nets) An Place/Transition net, or simply P/T-net, is a tuple (P, T, F) where:

1. P is a finite set of *places*,
2. T is a finite set of *transitions* such that $P \cap T = \emptyset$, and
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the *flow relation*.

A *marked* P/T-net is a pair (N, s), where $N = (P, T, F)$ is a P/T-net and $s \in \mathcal{B}(P)$, i.e., s is a bag over P, denoting the *marking* of the net. The set of all marked P/T-nets is denoted $\mathcal{N}$.

A marking is a *bag* over the set of places P, i.e., it is a function from P to the natural numbers. $\mathcal{B}(P)$ denotes the set of all bags over P. We use square brackets for the enumeration of a bag, e.g., $[a^2, b, c^3]$ denotes the bag with two a-s, one b, and three c-s. The sum of two bags $(X + Y)$, the difference $(X - Y)$, the presence of an element in a bag $(a \in X)$, and the notion of subbags $(X \leq Y)$ are defined in a straightforward way and they can handle a mixture of sets and bags.

Let $N = (P, T, F)$ be a P/T-net. Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y iff there is a directed arc from x to y (i.e., xFy). Node x is an *output node* of y iff yFx. For any $x \in P \cup T$, ${}^{N}\bullet x = \{y \mid yFx\}$ and $x^{N}\bullet = \{y \mid xFy\}$; the superscript N may be omitted if clear from the context.

Figure 1 shows a P/T-net consisting of 8 places and 7 transitions. Transition A has one input place and one output place, transition B has one input place and two output places, and transition E has two input places and one output place. The black dot in the input place of A represents a token. This token denotes the initial marking. The dynamic behavior of such a marked P/T-net is defined by a *firing rule*.

Definition 2.2. (Firing rule) Let $(N = (P, T, F), s)$ be a marked P/T-net. Transition $t \in T$ is *enabled*, denoted $(N, s)[t\rangle$, iff $\bullet t \leq s$. The *firing rule* $\_[\_\rangle\_ \subseteq \mathcal{N} \times T \times \mathcal{N}$ is the smallest relation satisfying for any $(N = (P, T, F), s) \in \mathcal{N}$ and any $t \in T$, $(N, s)[t\rangle \Rightarrow (N, s)\ [t\rangle\ (N, s - \bullet t + t\bullet)$.

In the marking shown in Figure 1 (i.e., one token in the source place), transition A is enabled and firing this transition removes the token for the input place and puts a token in the output place. In the resulting marking, two transitions are enabled: F and B. Although both are enabled only one can fire. If B fires, one token is consumed and two tokens are produced.

Definition 2.3. (Reachable markings) Let (N, s_0) be a marked P/T-net in $\mathcal{N}$. A marking s is *reachable* from the initial marking s_0 iff there exists a sequence of enabled transitions whose firing leads from s_0 to s. The set of reachable markings of (N, s_0) is denoted $[N, s_0\rangle$.

The marked P/T-net shown in Figure 1 has 8 reachable markings.

2.2 Workflow nets

Most workflow systems offer standard building blocks such as the AND-split, AND-join, OR-split, and OR-join [3, 11, 17, 18]. These are used to model sequential, conditional, parallel and iterative routing (WFMC [11]). Clearly, a Petri net can be used to specify the routing of cases. *Tasks* are modeled by transitions and causal dependencies are modeled by places and arcs. In fact, a place corresponds to a *condition* which can be used as pre- and/or post-condition for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs. Given the close relation between tasks and transitions we use these terms interchangeably.

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

Definition 2.4. (Workflow nets) Let $N = (P, T, F)$ be a P/T-net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a *workflow net* (WF-net) iff:

1. *object creation*: P contains an input place i such that $\bullet i = \emptyset$,
2. *object completion*: P contains an output place o such that $o\bullet = \emptyset$,
3. *connectedness*: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected,

The P/T-net shown in Figure 1 is a WF-net. Note that although the net is not strongly connected, the *short-circuited* net with transition $\bar{t}$ is strongly connected. Even if a net meets all the syntactical requirements stated in Definition 2.4, the corresponding process may exhibit errors such as deadlocks, tasks which can never become active, livelocks, garbage being left in the process after termination, etc. Therefore, we define the following correctness criterion.

Definition 2.5. (Sound) Let $N = (P, T, F)$ be a WF-net with input place i and output place o. N is *sound* iff:

1. *safeness*: $(N, [i])$ is safe,
2. *proper completion*: for any marking $s \in [N, [i]\rangle$, $o \in s$ implies $s = [o]$,
3. *option to complete*: for any marking $s \in [N, [i]\rangle$, $[o] \in [N, s\rangle$, and
4. *absence of dead tasks*: $(N, [i])$ contains no dead transitions.

The set of all sound WF-nets is denoted $\mathcal{W}$.

The WF-net shown in Figure 1 is sound. Soundness can be verified using standard Petri-net-based analysis techniques. In fact soundness corresponds to liveness and safeness of the corresponding short-circuited net [1, 3]. This way efficient algorithms and tools can be applied. An example of a tool tailored towards the analysis of WF-nets is Woflan [27].

3 Mining untimed workflow logs

After introducing some preliminaries we return to the topic of this paper: *workflow mining*. The goal of workflow mining is to find a workflow model (e.g., a WF-net) on the basis of a *workflow log*. In this section we first introduce the α mining algorithm which works on untimed logs. In the next section we focus on timed logs.

3.1 Definition of workflow logs and log-based ordering relations

Table 1 shows an example of a timed workflow log. In this section we only consider the first two columns. Note that the ordering of events within a case is relevant while the ordering of events amongst cases is of no importance. Therefore, we define a workflow log as follows.

Definition 3.1. (Workflow trace, Workflow log) Let T be a set of tasks. $\sigma \in T^*$ is a *workflow trace* and $W \in \mathcal{P}(T^*)$ is a *workflow log*.[2]

The workflow trace of case 1 in Table 1 is AFG. The workflow log corresponding to Table 1 is $\{ABCDEG, ABDCEG, AFG\}$. Note that in this paper we abstract from the identity of cases. Clearly the identity and the attributes of a case are relevant for workflow mining. However, for the algorithm presented in this section, we can abstract from this. For similar reasons, we abstract from the

[2] $\mathcal{P}(T^*)$ is the powerset of T^*, i.e., $W \subseteq T^*$.

frequency of workflow traces. In Table 1 workflow trace AFG appears twice (case 1 and case 5), workflow trace $ABCDEG$ also appears twice (case 2 and case 4), and workflow trace $ABDCEG$ (case 3) appears only once. These frequencies are not registered in the workflow log $\{ABCDEG, ABDCEG, AFG\}$. Note that when dealing with noise, frequencies are of the utmost importance. However, in this paper we do not deal with issues such as noise. Therefore, this abstraction is made to simplify notation.

To find a workflow model on the basis of a workflow log, the log should be analyzed for causal relations, e.g., if a task is always followed by another task it is likely that there is a causal relation between both tasks. To analyze these relations we introduce the following notations.

Definition 3.2. (Log-based ordering relations) Let W be a workflow log over T, i.e., $W \in \mathcal{P}(T^*)$. Let $a, b \in T$:

- $a >_W b$ if and only if there is a trace $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ and $i \in \{1, \ldots, n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,
- $a \rightarrow_W b$ if and only if $a >_W b$ and $b \not>_W a$,
- $a \#_W b$ if and only if $a \not>_W b$ and $b \not>_W a$, and
- $a \|_W b$ if and only if $a >_W b$ and $b >_W a$.

Consider the workflow log $W = \{ABCDEG, ABDCEG, AFG\}$ (i.e., the log shown in Table 1). Relation $>_W$ describes which tasks appeared in sequence (one directly following the other). Clearly, $A >_W B$, $A >_W F$, $B >_W C$, $B >_W D$, $C >_W E$, $C >_W D$, $D >_W C$, $D >_W E$, $E >_W G$, and $F >_W G$. Relation $\rightarrow_W$ can be computed from $>_W$ and is referred to as the *causal relation* derived from workflow log W. $A \rightarrow_W B$, $A \rightarrow_W F$, $B \rightarrow_W C$, $B \rightarrow_W D$, $C \rightarrow_W E$, $D \rightarrow_W E$, $E \rightarrow_W G$, and $F \rightarrow_W G$. Note that $C \not\rightarrow_W D$ because $D >_W C$. Relation $\|_W$ suggests potential parallelism. For log W tasks C and D seem to be in parallel, i.e., $C \|_W D$ and $D \|_W C$. If two tasks can follow each other directly in any order, then all possible interleavings are present and therefore they are likely to be in parallel. Relation $\#_W$ gives pairs of transitions that never follow each other directly. This means that there are no direct causal relations and parallelism is unlikely.

Property 3.3. Let W be a workflow log over T. For any $a, b \in T$: $a \rightarrow_W b$ or $b \rightarrow_W a$ or $a \#_W b$ or $a \|_W b$. Moreover, the relations $\rightarrow_W$, $\rightarrow_W^{-1}$, $\#_W$, and $\|_W$ are mutually exclusive and partition $T \times T$.[3]

This property can easy be verified. Note that $\rightarrow_W = (>_W \setminus >_W^{-1})$, $\rightarrow_W^{-1} = (>_W^{-1} \setminus >_W)$, $\#_W = (T \times T) \setminus (>_W \cup >_W^{-1})$, $\|_W = (>_W \cap >_W^{-1})$. Therefore, $T \times T = \rightarrow_W \cup \rightarrow_W^{-1} \cup \#_W \cup \|_W$. If no confusion is possible, the subscript W is omitted.

To simplify the use of logs and sequences we introduce some additional notations.

Definition 3.4. ($\in$, *first*, *last*) Let A be a set, $a \in A$, and $\sigma = a_1 a_2 \ldots a_n \in A^*$ a sequence over A of length n. $\in$, *first*, *last* are defined as follows:

[3] $\rightarrow_W^{-1}$ is the inverse of relation $\rightarrow_W$, i.e., $\rightarrow_W^{-1} = \{(y, x) \in T \times T \mid x \rightarrow_W y\}$.

1. $a \in \sigma$ if and only if $a \in \{a_1, a_2, \ldots a_n\}$,
2. $\mathit{first}(\sigma) = a_1$, and
3. $\mathit{last}(\sigma) = a_n$.

To reason about the quality of a workflow mining algorithm we need to make assumptions about the completeness of a log. For a complex process, a handful of traces will not suffice to discover the exact behavior of the process. Relations $\rightarrow_W$, $\rightarrow_W^{-1}$, $\#_W$, and $\|_W$ will be crucial information for any workflow-mining algorithm. Since these relations can be derived from $>_W$, we assume the log to be complete with respect to this relation.

Definition 3.5. (Complete workflow log) Let $N = (P, T, F)$ be a sound WF-net, i.e., $N \in \mathcal{W}$. W is a *workflow log of* N if and only if $W \in \mathcal{P}(T^*)$ and every trace $\sigma \in W$ is a firing sequence of N starting in state $[i]$, i.e., $(N, [i])[\sigma\rangle$. W is a *complete* workflow log of N if and only if (1) for any workflow log W' of N: $>_{W'} \subseteq >_W$, and (2) for any $t \in T$ there is a $\sigma \in W$ such that $t \in \sigma$.

A workflow log of a sound WF-net only contains behaviors that can be exhibited by the corresponding process. A workflow log is complete if all tasks that potentially directly follow each other in fact directly follow each other in some trace in the log. Note that transitions that connect the input place i of a WF-net to its output place o are "invisible" for $>_W$. Therefore, the second requirement has been added. If there are no such transitions, this requirement can be dropped.

3.2 Workflow mining algorithm

We now present an algorithm for mining processes. The algorithm uses the fact that for many WF-nets two tasks are connected if and only if their causality can be detected by inspecting the log.

Definition 3.6. (Mining algorithm α) Let W be a workflow log over T. $\alpha(W)$ is defined as follows.

1. $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{first}(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{last}(\sigma)\}$,
4. $X_W = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$,
5. $Y_W = \{(A, B) \in X_W \mid \forall_{(A', B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$,
6. $P_W = \{p_{(A,B)} \mid (A, B) \in Y_W\} \cup \{i_W, o_W\}$,
7. $F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y_W \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\}$, and
8. $\alpha(W) = (P_W, T_W, F_W)$.

The mining algorithm constructs a net (P_W, T_W, F_W). Clearly, the set of transitions T_W can be derived by inspecting the log. In fact, if there are no traces of length one, T_W can be derived from $>_W$. Since it is possible to find all initial

transitions T_I and all final transition T_O, it is easy to construct the connections between these transitions and i_W and o_W. Besides the source place i_W and the sink place o_W, places of the form $p_{(A,B)}$ are added. For such place, the subscript refers to the set of input and output transitions, i.e., $\bullet p_{(A,B)} = A$ and $p_{(A,B)}\bullet = B$. A place is added in-between a and b if and only if $a \rightarrow_W b$. However, some of these places need to be merged in case of OR-splits/joins rather than AND-splits/joins. For this purpose the relations X_W and Y_W are constructed. $(A, B) \in X_W$ if there is a causal relation from each member of A to each member of B and the members of A and B never occur next to one another. Note that if $a \rightarrow_W b$, $b \rightarrow_W a$, or $a \|_W b$, then a and b cannot be both in A (or B). Relation Y_W is derived from X_W by taking only the largest elements with respect to set inclusion.

If the α algorithm is applied to the log shown in Table 1, then the WF-net shown in Figure 1 is discovered. In fact the α algorithm will detect this WF-net from any complete workflow log of this workflow model. In [4] we prove the correctness of the mining algorithm for a large class of workflow processes. However, a precise description of this class and correctness proofs are beyond the scope of this paper because we focus on *timed* workflow logs. The interested reader is referred to [4].

4 Mining timed logs

The mining algorithm presented in the previous section ignores timing information. Therefore, we extend the algorithm to incorporate time information. Event logs typically add one timestamp to every line in the log. Therefore, we use the following definition.

Definition 4.1. (Timed workflow trace, Timed workflow log) Let T be a set of tasks and D a time domain (e.g., $D = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ or any totally ordered domain with $>$, $+$, and $-$ defined on it.). $\sigma \in (T \times D)^*$ is a *timed workflow trace* and $W \in \mathcal{B}((T \times D)^*)$ is a *timed workflow log*.[4]

Each line in the workflow log has a timestamp indicating at what time the corresponding event took place. A timed workflow trace is simply a sequence of such timed events. Note that a timed log is a bag of traces rather than a set of traces. In contrast to the untimed case (cf. Definition 3.1), we have to take into account the frequencies of traces. Without these frequencies we cannot calculate estimates for probabilities and averages because these depend to how many times a specific trace occurred.

Since each line in the workflow log has a single timestamp and no duration attached to it, we will associate time to places, i.e., the firing of a transition is an atomic action and tokens spend time in places. The time tokens spend in places is referred to as *sojourn time* or *holding time*. We distinguish between two kinds of sojourn time: *waiting time* and *synchronization time*. The waiting

[4] $T \times D$ is the Cartesian product of T and D.

time is the time that passes from the enabling of a transition until its firing. The synchronization time is the time that passes from the partial enabling of a transition (i.e., at least one input place marked) until full enabling (i.e., all input places are marked). Note that these times should be viewed from a token in a place, i.e., when a token arrives in a place p, the synchronization time is the time it takes to enable one of the output transitions in $p\bullet$ and the waiting time is the additional time it takes to fire the first transition in $p\bullet$.

Besides sojourn times we also want to analyze other metrics such as the probability of taking a specific path and the flow time (i.e., the time from the arrival of a case until its completion).

To calculate sojourn times, probabilities, flow times, and other metrics we first apply the α algorithm and then replay the log in the resulting WF-net. For each case in the log, we have a timed workflow trace $\sigma \in (T \times D)^*$. We know that the case starts in marking $[i]$. Therefore, we will start by putting a token with a timestamp equal to that of the first firing transition in place i. Then, one by one, each transition in the timed workflow trace may fire, thus collecting tokens from its input places and placing them in its output places. Every time a transition fires the waiting and synchronizing times will be calculated for the input places in the following way: (1) the maximum m of the timestamps of tokens in all input places is calculated, (2) if the transition has more than 1 input place, then for each place a "synchronization-time observation" will be added which will be the difference between the timestamp of the token in that place and m, and (3) for each of the input places a "waiting-time observation" is added. The latter observation is equal to the difference between m and the time the transition fires according to the log. This is repeated until the case reaches marking $[o]$. This analysis is done for each case, resulting in a number of synchronization-time observations and waiting-time observations per place. Based on these observations metrics such as average, variance, maximum and minimum synchronization/waiting time can be calculated.

By replaying the log in the discovered WF-net, also other metrics such as routing probabilities and flow times can be calculated. For example, if a place has multiple output transitions, then the probability that a specific transition will be chosen equals the number of occurrences of that transition in the log, divided by the total number of occurrences of all the enabled transitions.

To conclude this section, we point out legal issues relevant when mining timed workflow logs. Clearly, timed workflow logs can be used to systematically measure the performance of employees. The legislation with respect to issues such as privacy and protection of personal data differs from country to country. For example, Dutch companies are bound by the Personal Data Protection Act (Wet Bescherming Persoonsgegeven) which is based on a directive from the European Union. The practical implications of this for the Dutch situation are described in [6, 16, 24]. Timed workflow logs are not restricted by these laws as long as the information in the log cannot be traced back to individuals. If information in the log can be traced back to a specific employee, it is important that the employee is aware of the fact that her/his activities are logged and the fact that this logging

is used to monitor her/his performance. Note that in the timed workflow log as defined in Definition 4.1 there is no information about the workers executing tasks. Therefore, it is not possible to distill information on the productivity of individual workers and legislation such as the Personal Data Protection Act does not apply. Nevertheless, the logs of most workflow systems contain information about individual workers, and therefore, this issue should be considered carefully.

5 EMiT: A tool for mining timed workflow logs

This section introduces our tool *EMiT* (Enhanced Mining Tool). EMiT has been developed to mine timed workflow logs from a range of transactional systems including workflow management systems such as Staffware and ERP systems such as SAP.

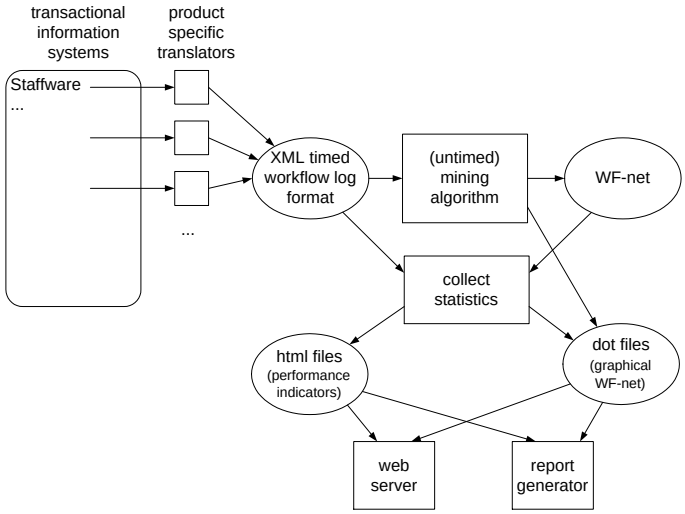

Fig. 3. The architecture of EMiT.

Figure 3 shows the architecture of EMiT. The mining starts from a tool-independent XML format. From any transactional information system recording event logs, we can export to this XML format. The DTD describing this format is as follows:

```
<!ELEMENT WorkFlow_log (source?,process+)>
  <!ELEMENT source EMPTY>
    <!ATTLIST source program (staffware|inconcert|pnet|IBM_MQ|
                              other) #REQUIRED>
  <!ELEMENT process (case*)>
    <!ATTLIST process id ID #REQUIRED>
```

```
<!ELEMENT case (log_line*)>
  <!ATTLIST case id ID #REQUIRED>
  <!ELEMENT log_line (task_name, event, date, time)>
    <!ELEMENT task_name (#PCDATA)>
    <!ELEMENT event EMPTY>
    <!ATTLIST event kind (normal|schedule|start|withdraw|
              suspend|resume|abort|complete) #REQUIRED>
    <!ELEMENT date (#PCDATA)>
    <!ELEMENT time (#PCDATA)>
```

Note that the XML file not only contains timed workflow traces as defined in Definition 4.1 but also information about the source of the information and the type of event recorded. This information can be used to filter and extract additional knowledge. Using the α algorithm, EMiT constructs a WF-net without considering timing information. Then the component "collect statistics" replays the timed traces in the discovered WF-net and outputs both HTML files and DOT files. EMiT exports WF-nets to the .DOT format to visualize the discovered model and performance indicators. There are two ways to view results. First, it is possible to generate a static report containing the graphical model and all performance indicators: probabilities, sojourn times (average, variance, minimum, and maximum), synchronization times, waiting times, etc. A more sophisticated way to view the results is through a combination of HTML, JPG, and MAP files. This requires the use of a web server, but allows for checkable models and a dynamic hypertext-like report.

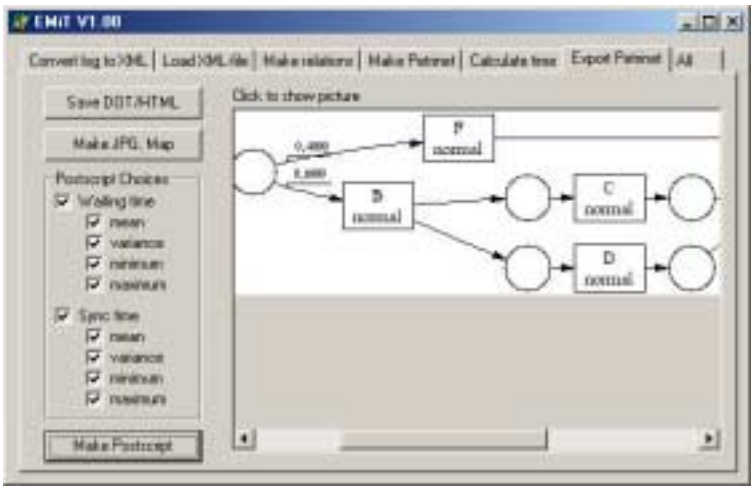

Fig. 4. EMiT screenshot.

EMiT has been developed using Delphi and provides an easy-to-use graphical user interface. Figure 4 shows one of the screens of EMiT while analyzing the log shown in Table 1. EMiT indeed discovers the correct sojourn times as indicated in Figure 2.

6 Application: Mining Staffware logs

Although EMiT and the underlying analysis routines are tool-independent, we focus on a concrete system to illustrate the applicability of the results presented in this paper. Staffware [26] is one of the leading workflow management systems. We have developed a translator from Staffware audit trails to the XML format described in the previous section. Staffware records the completion of each task in the log. (Note that in Staffware tasks are named steps.) However, it does not record the start of the execution of a task. Instead it records the scheduling of tasks. In the EMiT XML file, completion events are distinguished from schedule events. Other events recorded by Staffware and stored in the XML format are withdraw, suspend, and resume events. Using different profiles, EMiT either ignores or incorporates the various events.

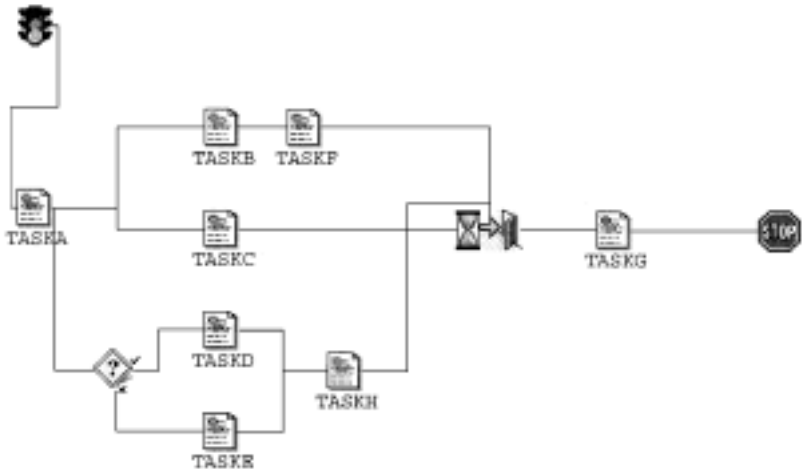

Fig. 5. Staffware process model.

We have tested EMiT on a wide variety of Staffware models. These tests demonstrate the applicability of the α algorithm extended with time. An example is shown in Figure 5. Note that in Staffware each step (i.e. task) is an OR-join/AND-split, and conditions (diamond symbol) and waits (sand-timer symbol) have been added to model respectively OR-splits and AND-joins. The workflow shown in Figure 5 starts with TASKA, followed by TASKD or TASKE,

TASKB followed by TASKF, and TASKC in parallel, and ends with TASKG. We have handled several cases using the Staffware model shown in Figure 5. By collecting the audit trails of this model and feeding this to EMiT, we obtained the WF-net shown in Figure 6. It is easy to verify that this WF-net indeed corresponds to the Staffware model of Figure 5. Figure 6 does not show metrics such as waiting times, etc. However, by clicking on the places one can obtain detailed information about these performance indicators.

Examples such as shown in figures 5 and 6 demonstrate the validity and applicability of our approach. It should be noted that it is not very useful to mine Staffware logs for discovering pre-specified workflow models. It is much more interesting to mine process models in the situation where the underlying model is unknown. However, even in the situation where the workflow specification is already available, it is interesting to compare the specified model with the discovered model. For example, it is useful to detect deviations between the actual workflow and the specified workflow. Moreover, EMiT also attributes performance indicators to a graphical representation of the real workflow. Clearly these features are not supported by contemporary workflow management systems.

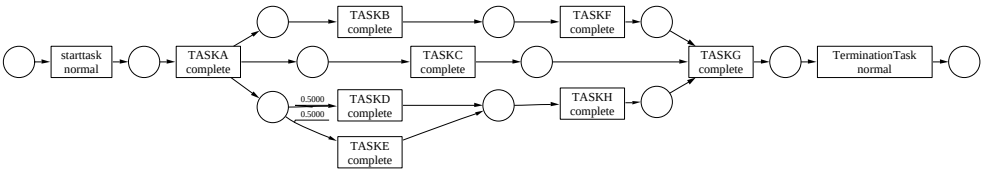

Fig. 6. The resulting model.

7 Related Work

The idea of process mining is not new [5, 7–9, 12–15, 21, 25]. Cook and Wolf have investigated similar issues in the context of software engineering processes. In [7] they describe three methods for process discovery: one using neural networks, one using a purely algorithmic approach, and one Markovian approach. The authors consider the latter two the most promising approaches. The purely algorithmic approach builds a finite state machine where states are fused if their futures (in terms of possible behavior in the next k steps) are identical. The Markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise. Note that the results presented in [6] are limited to sequential behavior. Cook and Wolf extend their work to concurrent processes in [8]. They propose specific metrics (entropy, event type counts, periodicity, and causality) and use these metrics to discover models out of event streams. However, they do not provide an approach to generate explicit process models. Recall that the final

goal of the approach presented in this paper is to find explicit representations for a broad range of process models, i.e., we want to be able to generate a concrete Petri net rather than a set of dependency relations between events. In [9] Cook and Wolf provide a measure to quantify discrepancies between a process model and the actual behavior as registered using event-based data. The idea of applying process mining in the context of workflow management was first introduced in [5]. This work is based on workflow graphs, which are inspired by workflow products such as IBM MQSeries workflow (formerly known as Flowmark) and InConcert. In this paper, two problems are defined. The first problem is to find a workflow graph generating events appearing in a given workflow log. The second problem is to find the definitions of edge conditions. A concrete algorithm is given for tackling the first problem. The approach is quite different from the approach envisioned in this proposal. Given the nature of workflow graphs there is no need to identify the nature (AND or OR) of joins and splits. Moreover, workflow graphs are acyclic. The only way to deal with iteration is to enumerate all occurrences of a given activity. In [21], a tool based on these algorithms is presented. Schimm [25] has developed a mining tool suitable for discovering hierarchically structured workflow processes. This requires all splits and joins to be balanced. Herbst and Karagiannis also address the issue of process mining in the context of workflow management [12–15]. The approach uses the ADONIS modeling language and is based on hidden Markov models where models are merged and split in order to discover the underlying process. The work presented in [12, 14, 15] is limited to sequential models. A notable difference with other approaches is that the same activity can appear multiple times in the workflow model. The result in [13] incorporates concurrency but also assumes that workflow logs contain explicit causal information. The latter technique is similar to [5, 21] and suffers from the drawback that the nature of splits and joins (i.e., AND or OR) is not discovered.

In contrast to existing work we addressed workflow processes with concurrent behavior right from the start (rather than adding ad-hoc mechanisms to capture parallelism), i.e., detecting concurrency is the prime concern of the α algorithm. Moreover, we focus on the mining of timed workflow logs to derive performance indicators such as sojourn times, probabilities, etc. Some preliminary results for untimed logs have been reported in [4, 20, 28, 29]. In [28, 29] a heuristic approach using rather simple metrics is used construct so-called "dependency/frequency tables" and "dependency/frequency graphs". In [20] another variant of this technique is presented using examples from the health-care domain. The preliminary results presented in [20, 28, 29] only provide heuristics and focus on issues such as noise. The approach described in [4] differs from these approaches in the sense that for the α algorithm is proven that for certain subclasses it is possible to find the right workflow model.

This paper builds on [4, 20, 28, 29]. The main contribution of this paper, compared to earlier work is the incorporation of time, practical experience with systems such as Staffware, and the introduction of EMiT.

8 Conclusion

This paper presented an approach to extract both a workflow model and performance indicators from timed workflow logs. The approach is supported by the EMiT tool also presented in this paper and has been validated using logs of transactional information systems such as Staffware.

It is important to see the results presented in this paper in the context of a larger effort [4, 20, 28, 29]. The overall goal is to be able to analyze any workflow log without any knowledge of the underlying process and in the presence of noise. At this point in time, we are applying our workflow mining techniques to two applications. The first application is in health-care where the flow of multi-disciplinary patients is analyzed. We have analyzed workflow logs (visits to different specialist) of patients with peripheral arterial vascular diseases of the Elizabeth Hospital in Tilburg and the Academic Hospital in Maastricht. Patients with peripheral arterial vascular diseases are a typical example of multi-disciplinary patients. The second application concerns the processing of fines by the CJIB (Centraal Justitieel Incasso Bureau), the Dutch Judicial Collection Agency located in Leeuwarden. For example fines with respect to traffic violations are processed by the CJIB. However, this government agency also takes care of the collection of administrative fines related to crimes, etc. Through workflow mining we try to get insight in the life-cycle of for example speeding tickets. Some preliminary results show that it is very difficult to mine the flow of multi-disciplinary patients given the large number of exceptions, incomplete data, etc. However, it is relatively easy to mine well-structured administrative processes such as the processes within the CJIB. In both applications we are also trying to take attributes of the cases being processed into account. This way we hope to find correlations between properties of the case and the route through the workflow process.

Acknowledgements The authors would like to thank Eric Verbeek, Ton Weijters, and Laura Maruster for contributing to this work.

References

1. 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.
2. W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Which Processes can be Rediscovered? BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven, 2002.
5. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

6. College Bescherming persoonsgegevens (CBP; Dutch Data Protection Authority). http://www.cbpweb.nl/index.htm.
7. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
8. J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. In *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6)*, pages 35–45, 1998.
9. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
10. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
11. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2001.
12. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
13. J. Herbst. Dealing with Concurrency in Workflow Induction. In U. Baake, R. Zobel, and M. Al-Akaidi, editors, *European Concurrent Engineering Conference*. SCS Europe, 2000.
14. J. Herbst and D. Karagiannis. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In M. Ibrahim and B. Drabble, editors, *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, Stockholm, Sweden, August 1999.
15. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 9:67–92, 2000.
16. B.J.P. Hulsman and P.C. Ippel. Personeelsinformatiesystemen: De Wet Persoonsregistraties toegepast. Registratiekamer, The Hague, 1994.
17. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
18. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
19. D.C. Marinescu. *Internet-Based Workflow Management: Towads a Semantic Web*, volume 40 of *Wiley Series on Parallel and Distributed Computing*. Wiley-Interscience, New York, 2002.
20. L. Maruster, W.M.P. van der Aalst, A.J.M.M. Weijters, A. van den Bosch, and W. Daelemans. Automated Discovery of Workflow Models from Hospital Data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 183–190, 2001.
21. M.K. Maxeiner, K. Küspert, and F. Leymann. Data Mining von Workflow-Protokollen zur teilautomatisierten Konstruktion von Prozemodellen. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 75–84. Informatik Aktuell Springer, Berlin, Germany, 2001.

22. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
23. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
24. L.B. Sauerwein and J.J. Linnemann. Guidelines for Personal Data Processors: Personal Data Protection Act. Ministry of Justice, The Hague, 2001.
25. G. Schimm. Process Mining. http://www.processmining.de/.
26. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 1999.
27. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
28. A.J.M.M. Weijters and W.M.P. van der Aalst. Process Mining: Discovering Workflow Models from Event-Based Data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 283–290, 2001.
29. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data. In V. Hoste and G. de Pauw, editors, *Proceedings of the 11th Dutch-Belgian Conference on Machine Learning (Benelearn 2001)*, pages 93–100, 2001.