Workflow Mining: A Survey of Issues and Approaches

W.M.P. van der Aalst[1], B.F. van Dongen[1], J. Herbst[2], L. Maruster[1], G. Schimm[3], and A.J.M.M. Weijters[1]

[1] 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
[2] DaimlerChrysler AG, Research & Technology, P.O. Box 2360, D-89013, Ulm, Germany. joachim.j.herbst@daimlerchrysler.com
[3] OFFIS, Escherweg 2, D-26121 Oldenburg, Germany. schimm@offis.de

Abstract. Many of today's information systems are driven by explicit process models. Workflow management systems, but also ERP, CRM, SCM, and B2B, are configured on the basis of a workflow model specifying the order in which tasks need to be executed. 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. To support the design of workflows, we propose the use of workflow mining. Starting point for workflow mining is a so-called "workflow log" containing information about the workflow process as it is actually being executed. In this paper, we introduce the concept of workflow mining and present a common format for workflow logs. Then we discuss the most challenging problems and present some of the workflow mining approaches available today.

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

1 Introduction

During the last decade workflow management technology [2, 4, 21, 35, 41] has become readily available. Workflow management systems such as Staffware, IBM MQSeries, COSA, etc. offer generic modeling and enactment capabilities for structured business processes. By making 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. These process definitions need to be executable and are typically graphical. 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, SCM (Supply Chain Management) systems, B2B (Business to Business) applications, etc. which embed workflow technology. 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 business process at hand (i.e., lengthy discussions with the workers and management are needed) and the workflow language being used.

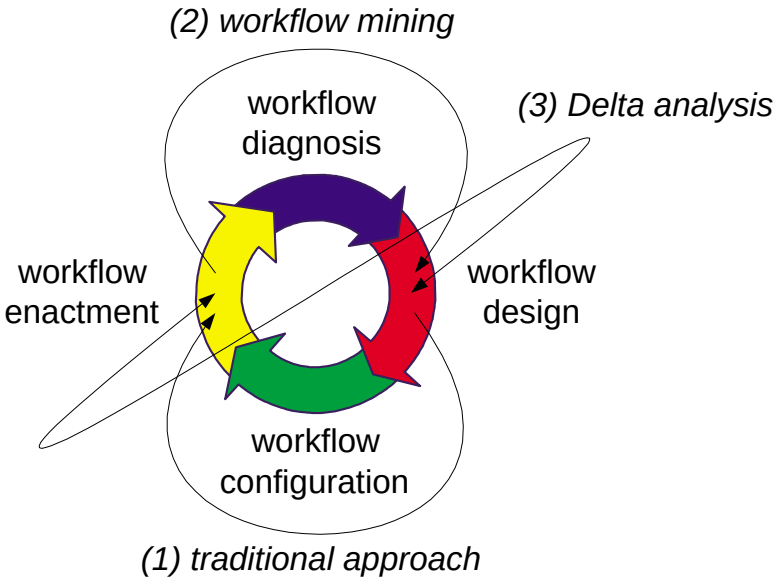

Fig. 1. The workflow life-cycle is used to illustrate workflow mining and Delta analysis in relation to traditional workflow design.

To compare workflow mining with the traditional approach towards workflow design and enactment, consider the *workflow life cycle* shown in Figure 1. The workflow life cycle consists of four phases: (A) *workflow design*, (B) *workflow configuration*, (C) *workflow enactment*, and (D) *workflow diagnosis*. In the *traditional approach* the design phase is used for constructing a workflow model. This is typically done by a business consultant and is driven by ideas of management on improving the business processes at hand. If the design is finished, the workflow system (or any other system that is "process aware") is configured as specified in the design phase. In the configuration phases one has to deal with limitation and particularities of the workflow management system being used (cf. [5, 65]). In the enactment phase, cases (i.e., workflow instances) are handled by the workflow system as specified in the design phase and realized in the configuration phase. Based on a running workflow, it is possible to collect diagnostic information which is analyzed in the diagnosis phase. The diagnosis phase can again provide input for the design phase thus completing the workflow life cycle. In the traditional approach the focus is on the design and configuration phases.

Less attention is paid to the enactment phase and few organizations systematically collect runtime data which is analyzed as input for redesign (i.e., the diagnosis phase is typically missing).

The goal of *workflow mining* is to reverse the process and collect data at runtime to support workflow design and analysis. Note that in most cases, prior to the deployment of a workflow system, the workflow was already there. Also note that in most information systems transactional data is registered (consider for example the transaction logs of ERP systems like SAP). The information collected at run-time can be used to derive a model explaining the events recorded. Such a model can be used in both the diagnosis phase and the (re)design phase.

Modeling an existing process is influenced by perceptions, e.g., models are often normative in the sense that they state what "should" be done rather than describing the actual process. As a result models tend to be rather subjective. A more objective way of modeling is to use data related to the actual events that took place. Note that workflow mining is not biased by perceptions or normative behavior. However, if people bypass the system doing things differently, the log can still deviate from the actual work being done. Nevertheless, it is useful to confront man-made models with models discovered through workflow mining.

Closely monitoring the events taking place at runtime also enables *Delta analysis*, i.e., detecting discrepancies between the design constructed in the design phase and the actual execution registered in the enactment phase. Workflow mining results in an "a posteriori" process model which can be compared with the "a priori" model. Workflow technology is moving into the direction of more operational flexibility to deal with workflow evolution and workflow exception handling [2, 7, 10, 13, 20, 30, 39, 40, 64]. As a result workers can deviate from the prespecified workflow design. Clearly one wants to monitor these deviations. For example, a deviation may become common practice rather than being a rare exception. In such a case, the added value of a workflow system becomes questionable and an adaptation is required. Clearly, workflow mining techniques can be used to create a feedback loop to adapt the workflow model to changing circumstances and detect imperfections of the design.

The topic of workflow mining is related to management trends such as Business Process Reengineering (BPR), Business Intelligence (BI), Business Process Analysis (BPA), Continuous Process Improvement (CPI), and Knowledge Management (KM). Workflow mining can be seen as part of the BI, BPA, and KM trends. Moreover, workflow mining can be used as input for BPR and CPI activities. Note that workflow mining seems to be more appropriate for BPR than for CPI. Recall that one of the basic elements of BPR is that it is radical and should not be restricted by the existing situation [23]. Also note that workflow mining is not a tool to (re)design processes. The goal is to understand what is really going on as indicated in Figure 1. Despite the fact that workflow mining is not a tool for designing processes, it is evident that a good understanding of the existing processes is vital for any redesign effort.

This paper is a joint effort of a number of researchers using different approaches to workflow mining and is a spin-off of the "Workflow Mining Work-

shop"[1]. The goal of this paper is to introduce the concept of workflow mining, to identify scientific and practical problems, to present a common format to store workflow logs, to provide an overview of existing approaches, and to present a number of mining techniques in more detail.

The remainder of this paper is organized as follows. First, we summarize related work. In Section 3 we define workflow mining and present some of the challenging problems. In Section 4 we propose a common XML-based format for storing and exchanging workflow logs. This format is used by the mining tools developed by the authors and interfaces with some of the leading workflow management systems (Staffware, MQSeries Workflow, and InConcert). Sections 5, 6, 7, 8, and 9 introduce five approaches to workflow mining focusing on different aspects. These sections give an overview of some of the ongoing work on workflow mining. Section 10 compares the various approaches and list a number of open problems. Section 11 concludes the paper.

2 Related work

The idea of process mining is not new [8, 11, 15–17, 24–29, 42–44, 53–57, 61–63]. Cook and Wolf have investigated similar issues in the context of software engineering processes. In [15] 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 [16]. 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 [17] 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 [11]. 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 other approaches: Because

[1] This workshop took place on May 22nd and 23rd 2002 in Eindhoven, The Netherlands.

the nature of workflow graphs there is no need to identify the nature (AND or OR) of joins and splits. As shown in [37], workflow graphs use true and false tokens which do not allow for cyclic graphs. Nevertheless, [11] partially deals with iteration by enumerating all occurrences of a given task and then folding the graph. However, the resulting conformal graph is not a complete model. In [44], a tool based on these algorithms is presented. Schimm [53, 54, 57] 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 [26, 24, 25, 28, 29, 27] using an inductive approach. The work presented in [27, 29] is limited to sequential models. The approach described in [26, 24, 25, 28] also allows for concurrency. It uses stochastic task graphs as an intermediate representation and it generates a workflow model described in the ADONIS modeling language. In the induction step task nodes are merged and split in order to discover the underlying process. A notable difference with other approaches is that the same task can appear multiple times in the workflow model. The graph generation technique is similar to the approach of [11, 44]. The nature of splits and joins (i.e., AND or OR) is discovered in the transformation step, where the stochastic task graph is transformed into an ADONIS workflow model with block-structured splits and joins. In contrast to the previous papers, the work in [8, 42, 43, 61, 62] is characterized by the focus on workflow processes with concurrent behavior (rather than adding ad-hoc mechanisms to capture parallelism). In [61, 62] a heuristic approach using rather simple metrics is used to construct so-called "dependency/frequency tables" and "dependency/frequency graphs". In [42] another variant of this technique is presented using examples from the health-care domain. The preliminary results presented in [42, 61, 62] only provide heuristics and focus on issues such as noise. The approach described in [8] differs from these approaches in the sense that for the α algorithm it is proven that for certain subclasses it is possible to find the right workflow model. In [3] the α algorithm is extended to incorporate timing information.

Process mining can be seen as a tool in the context of Business (Process) Intelligence (BPI). In [22, 52] a BPI toolset on top of HP's Process Manager is described. The BPI tools set includes a so-called "BPI Process Mining Engine". However, this engine does not provide any techniques as discussed before. Instead it uses generic mining tools such as SAS Enterprise Miner for the generation of decision trees relating attributes of cases to information about execution paths (e.g., duration). In order to do workflow mining it is convenient to have a so-called "process data warehouse" to store audit trails. Such as data warehouse simplifies and speeds up the queries needed to derive causal relations. In [19, 46–48] the design of such warehouse and related issues are discussed in the context of workflow logs. Moreover, [48] describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM) [34]. The later tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) [59] which is tailored towards mining Staffware logs.

Note that none of the latter tools is extracting the process model. The main focus is on clustering and performance analysis rather than causal relations as in [8, 11, 15–17, 24–29, 42–44, 53–57, 61–63].

Much of the work mentioned above will be discussed in more detail in sections 5, 6, 7, 8, and 9. Before doing so, we first look at workflow mining in general and introduce a common XML-based format for storing and exchanging workflow logs.

3 Workflow mining

The goal of workflow mining is to extract information about processes from transaction logs. 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. The term *process mining* refers to methods for distilling a structured process description from a set of real executions. Because these methods focus on so-called case-driven process that are supported by contemporary workflow management systems, we also use the term *workflow mining*.

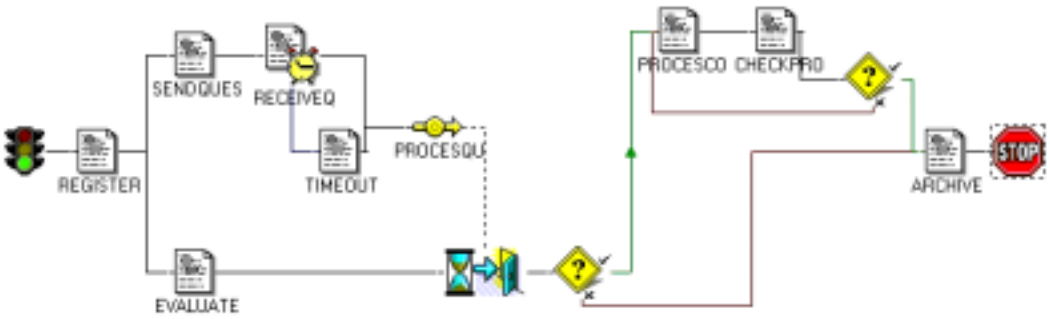

Fig. 2. The staffware model

Table 1 shows a fragment of a workflow log generated by the Staffware system. In Staffware events are grouped on a case-by-case basis. The first column refers to the task (description), the second to the type of event, the third to the user generating the event (if any), and the last column shows a time stamp. The

Case 10			
Directive Description	Event	User	yyyy/mm/dd hh:mm
	Start	bvdongen@staffw_e	2002/06/19 12:58
Register	Processed To	bvdongen@staffw_e	2002/06/19 12:58
Register	Released By	bvdongen@staffw_e	2002/06/19 12:58
Send questionnaire	Processed To	bvdongen@staffw_e	2002/06/19 12:58
Evaluate	Processed To	bvdongen@staffw_e	2002/06/19 12:58
Send questionnaire	Released By	bvdongen@staffw_e	2002/06/19 13:00
Receive questionnaire	Processed To	bvdongen@staffw_e	2002/06/19 13:00
Receive questionnaire	Released By	bvdongen@staffw_e	2002/06/19 13:00
Evaluate	Released By	bvdongen@staffw_e	2002/06/19 13:00
Archive	Processed To	bvdongen@staffw_e	2002/06/19 13:00
Archive	Released By	bvdongen@staffw_e	2002/06/19 13:00
	Terminated		2002/06/19 13:00
Case 9			
Directive Description	Event	User	yyyy/mm/dd hh:mm
	Start	bvdongen@staffw_e	2002/06/19 12:36
Register	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Register	Released By	bvdongen@staffw_e	2002/06/19 12:35
Send questionnaire	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Evaluate	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Send questionnaire	Released By	bvdongen@staffw_e	2002/06/19 12:36
Receive questionnaire	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Receive questionnaire	Released By	bvdongen@staffw_e	2002/06/19 12:36
Evaluate	Released By	bvdongen@staffw_e	2002/06/19 12:37
Process complaint	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Process complaint	Released By	bvdongen@staffw_e	2002/06/19 12:37
Check processing	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Check processing	Released By	bvdongen@staffw_e	2002/06/19 12:38
Archive	Processed To	bvdongen@staffw_e	2002/06/19 12:38
Archive	Released By	bvdongen@staffw_e	2002/06/19 12:38
	Terminated		2002/06/19 12:38
Case 8			
Directive Description	Event	User	yyyy/mm/dd hh:mm
	Start	bvdongen@staffw_e	2002/06/19 12:36
Register	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Register	Released By	bvdongen@staffw_e	2002/06/19 12:35
Send questionnaire	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Evaluate	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Send questionnaire	Released By	bvdongen@staffw_e	2002/06/19 12:36
Receive questionnaire	Processed To	bvdongen@staffw_e	2002/06/19 12:36
Receive questionnaire	Expired	bvdongen@staffw_e	2002/06/19 12:37
Receive questionnaire	Withdrawn	bvdongen@staffw_e	2002/06/19 12:37
Receive timeout	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Receive timeout	Released By	bvdongen@staffw_e	2002/06/19 12:37
Evaluate	Released By	bvdongen@staffw_e	2002/06/19 12:37
Process complaint	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Process complaint	Released By	bvdongen@staffw_e	2002/06/19 12:37
Check processing	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Check processing	Released By	bvdongen@staffw_e	2002/06/19 12:38
Process complaint	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Process complaint	Released By	bvdongen@staffw_e	2002/06/19 12:37
Check processing	Processed To	bvdongen@staffw_e	2002/06/19 12:37
Check processing	Released By	bvdongen@staffw_e	2002/06/19 12:38
Archive	Processed To	bvdongen@staffw_e	2002/06/19 12:38
Archive	Released By	bvdongen@staffw_e	2002/06/19 12:38
	Terminated		2002/06/19 12:38

Table 1. A Staffware log.

corresponding Staffware model is shown in Figure 2. Case 10 shown in Table 1 follows the scenario where first task *Register* is executed followed *Send questionnaire*, *Receive questionnaire*, and *Evaluate*. Based on the Evaluation, the decision is made to directly archive (task *Archive*) the case without further processing. For Case 9 further processing is needed, while Case 8 involves a timeout and the repeated execution of some tasks. Someone familiar with Staffware will be able to decide that the three cases indeed follow a scenario possible in the Staffware model shown in Figure 2. However, three cases are not sufficient to automatically derive the model of Figure 2. Note that there are more Staffware models enabling the three scenarios shown in Table 1. The challenge of workflow mining is to derive "good" workflow models with as little information as possible.

case identifier	task identifier
case 1	task A
case 2	task A
case 3	task A
case 3	task B
case 1	task B
case 1	task C
case 2	task C
case 4	task A
case 2	task B
case 2	task D
case 5	task A
case 4	task C
case 1	task D
case 3	task C
case 3	task D
case 4	task B
case 5	task E
case 5	task D
case 4	task D

Table 2. A workflow log.

To illustrate the principle of process mining in more detail, we consider the workflow log shown in Table 2. This log abstracts from the time, date, and event type, and limits the information to the order in which tasks are being executed. The log shown in Table 2 contains information about five cases (i.e., workflow instances). The log shows that for four cases (1, 2, 3, and 4) the tasks A, B, C, and D have been executed. For the fifth case only three tasks are executed: tasks A, E, and D. Each case starts with the execution of A and ends with the execution of D. If task B is executed, then also task C is executed. However, for some cases task C is executed before task B. Based on the information shown in Table 2 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 3. The model is represented in terms of a Petri net [50]. The Petri net starts with task A and finishes with task D. These tasks are represented by transitions. After executing A there is a choice between either executing B and C in parallel or just executing task E. To execute B and C in parallel two non-observable tasks (AND-split and AND-join) have been added. These tasks have been added for routing purposes only and are not present in the workflow log. 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 complete events for tasks it is possible to explicitly detect parallelism (cf. Section 4).

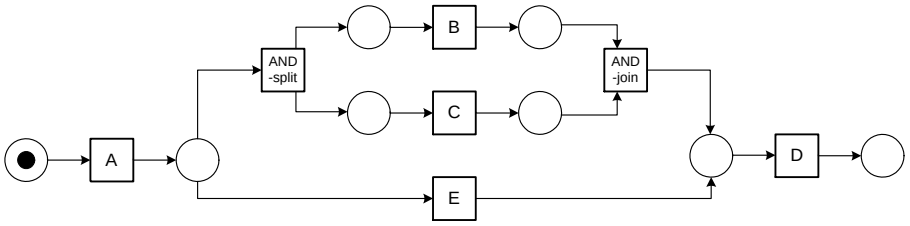

Fig. 3. A process model corresponding to the workflow log.

Table 2 contains the *minimal information* we assume to be present. In many applications, the workflow log contains a *time stamp* for each event and this information can be used to extract additional causality information. In addition, a typical log also contains information about the *type of event*, e.g., a start event (a person selecting an task from a worklist), a complete event (the completion of a task), a withdraw event (a scheduled task is removed), etc. Moreover, we are also interested in the relation between attributes of the case and the actual route taken by a particular case. For example, when handling traffic violations: Is the make of a car relevant for the routing of the corresponding traffic violation? (E.g., People driving a Ferrari always pay their fines in time.)

For this simple example (i.e., Table 2), it is quite easy to construct a process model that is able to regenerate the workflow log (e.g., Figure 3). For more realistic situations there are however a number of complicating factors:

- For larger workflow models mining 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.
- Workflow logs will typically contain *noise*, i.e., parts of the log can be incorrect, incomplete, or refer to exceptions. Events can be logged incorrectly

because of human or technical errors. Events can be missing in the log if some of the tasks are manual or handled by another system/organizational unit. Events can also refer to rare or undesired events. Consider for example the workflow in a hospital. If due to time pressure the order of two events (e.g., make X-ray and remove drain) is reversed, this does not imply that this would be part of the regular medical protocol and should be supported by the hospital's workflow system. Also two causally unrelated events (e.g., take blood sample and death of patient) may happen next to each other without implying a causal relation (i.e., taking a sample did not result in the death of the patient; it was sheer coincidence). Clearly, exceptions which are recorded only once should not automatically become part of the regular workflow.

- Table 2 only shows the order of events without giving information about the type of event, the time of the event, and attributes of the event (i.e., data about the case and/or task). Clearly, it is a challenge to exploit such additional information.

Sections 5, 6, 7, 8, and 9 will present different approaches to some of these problems.

To conclude this section, we point out legal issues relevant when mining (timed) workflow logs. Clearly, 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 [14, 31, 51]. 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 a timed workflow log we can abstract from information about the workers executing tasks and still mine the process. Therefore, it is possible to avoid collecting 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.

4 Workflow logs: A common XML format

In this section we focus on the syntax and semantics of the information stored in the workflow log. We will do this by presenting a tool independent XML format that is used by each of the mining approaches/tools described in the remainder. Figure 4 shows that this XML format connects transactional systems such as workflow management systems, ERP systems, CRM systems, and case handling systems. In principle, any system that registers events related to the execution of

tasks for cases can use this tool independent format to store and exchange logs. The XML format is used as input for the analysis tools presented in sections 5, 6, 7, 8, and 9. The goal of using a single format is to reduce the implementation effort and to promote the use of these mining techniques in multiple contexts.

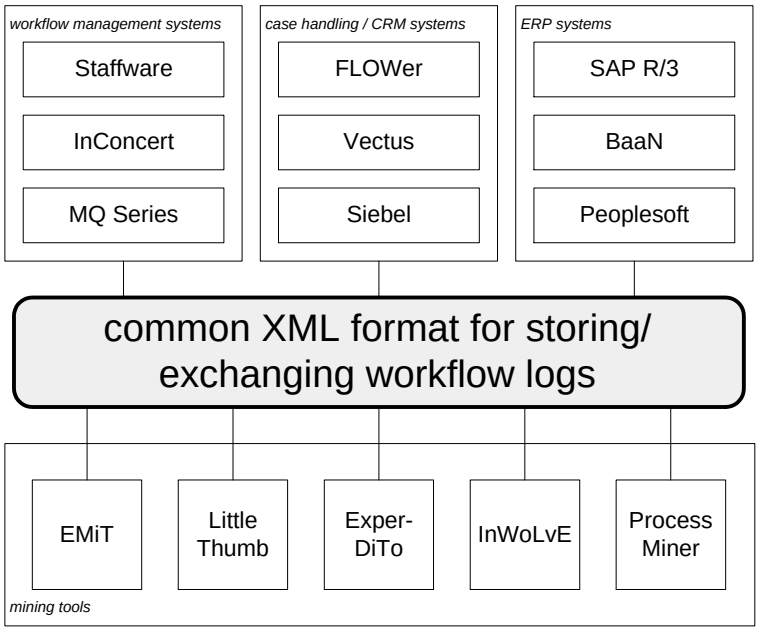

Fig. 4. The XML format as the solver/system independent medium .

Table 3 shows the *Document Type Definition* (DTD) [12] for workflow logs. This DTD specifies the syntax of a workflow log. A workflow log is a consistent XML document, i.e., a well-formed and valid XML file with top element *WorkFlow_log* (see Table 3). As shown, a workflow log consists of (optional) information about the source program and information about one or more workflow processes. Each workflow process (element *process*) consists of a sequence of cases (element *case*) and each case consists of a sequence of log lines (element *log_line*). Both processes and cases have an id and a description. Each line in the log contains the name of a task (element *task_name*). In addition, the line may contain information about the task instance (element *task_instance*), the type of event (element *event*), the date (element *date*), and the time of the event (element *time*).

It is advised to make sure that the process description and the case description are unique for each process or case respectively. The task name should be a unique identifier for a task within a process. If there are two or more tasks in a process with the same task name, they are assumed to refer to the same

```
<!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
    description CDATA "none"
>
<!ELEMENT case (log_line*)>
<!ATTLIST case
    id ID #REQUIRED
    description CDATA "none"
>
<!ELEMENT log_line (task_name, task_instance?, event?, date?, time?)>
<!ELEMENT task_name (#PCDATA)>
<!ELEMENT task_instance (#PCDATA)>
<!ELEMENT event EMPTY>
<!ATTLIST event
    kind (normal | schedule | start | withdraw | suspend |
    resume | abort | complete) #REQUIRED
>
<!ELEMENT date (#PCDATA)>
<!ELEMENT time (#PCDATA)>
```

Table 3. The XML DTD for storing and exchanging workflow logs.

task. (For example, in Staffware it is possible to have two tasks with different names, but the same description. Since the task description and not the task name appears in the log this can lead to confusion.) Although we assume tasks to have a unique name, there may be multiple instances of the same task. Consider for example a loop which causes a task to be executed multiple times for a given case. Therefore, one *can* add the element *task_instance* to a log line. This element will typically be a number, e.g., if task A is executed for the fifth time, element *task_name* is "A" and element *task_instance* is "5". The *date* and *time* elements are also optional. The *date* element must be in the following format: dd-mm-yyyy. So each date consists of exactly 10 characters of which there are 2 for the day, 2 for the month (i.e., 01 for January) and 4 for the year. The *time* element must be in the following format: HH:MM(:ss(:mmmmmm)). So each time-element consists of five, eight or seventeen characters of which there are two for the hour (00 - 23), two for the minutes (00 - 59) and optionally two for the seconds (00-59) and again optionally six for the fraction of a second that has passed. The complete log has to be sorted in the following way: Per case, all log entry's have to appear in the order in which they took place.

If information is not available, one can enter default values. For example, when storing the information shown in Table 2 the event type will be set to normal and the date and time will be set to some arbitrary value. Note that it is also fairly straightforward to map the Staffware log of Table 1 onto the XML format.

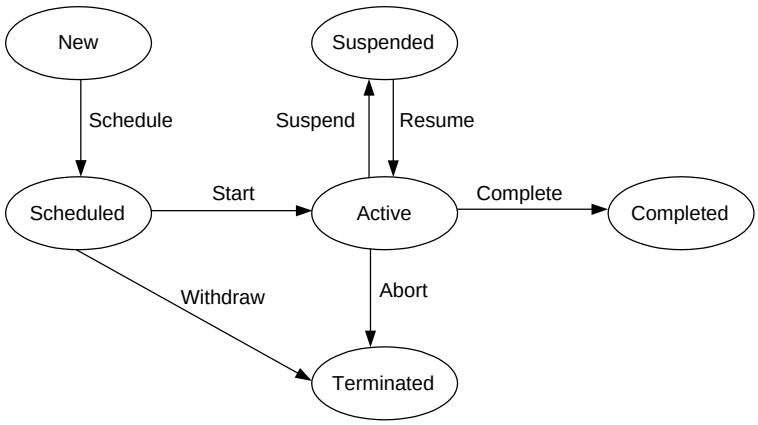

Fig. 5. A Finite State Machine describing the event types.

Table 3 specifies the syntax of the XML file. The semantics of most constructs are self-explaining except for the element *event*, i.e., the type of event. We identify eight event types: normal, schedule, start, withdraw, suspend, resume, abort, and complete. To explain these event types we use the Finite State Machine (FSM) shown in Figure 5. The FSM describes all possible states of a task from creation

to completion. The arrows in this figure describe all possible transitions between states and we assume these transitions to be atomic events (i.e., events that take no time). State *New* is the state in which the task starts. From this state only the event *Schedule* is possible. This event occurs when the task becomes ready to be executed (i.e., enabled or scheduled). The resulting state is the state *Scheduled*. In this state the task is typically in the worklist of one or more workers. From state *Scheduled* two events are possible: *Start* and *Withdraw*. If a task is withdrawn, it is deleted from the worklist and the resulting state is *Terminated*. If the task is started it is also removed from the worklist but the resulting state is *Active*. In state *Active* the actual processing of the task takes places. If the processing is successful, the case is moved to state *Completed* via event *Complete*. If for some reason it is not possible to complete, the task can be moved to state *Terminated* via an event of type *Abort*. In state *Active* it is also possible to suspend a task (event *Suspend*). Suspended tasks (i.e., tasks in state *Suspended*) can move back to state *Active* via the event *Resume*.

The events shown in Figure 5 are at a more fine grained level than the events shown in Table 2. Sometimes it is convenient to simply consider tasks as atomic events which do not take any time and always complete successfully. For this purpose, we use the event type *Normal* which is not shown in Figure 5. Events of type *Normal* can be considered as the execution of events *Schedule*, *Start*, and *Complete* in one atomic action.

Some systems log events at an even more fine-grained level than Figure 5. Other systems only log some of the events shown in Figure 5. Moreover, the naming of the event types is typically different. As an example, we consider the workflow management system Staffware and the log shown in Table 1. Staffware records the Schedule, Withdraw, and Complete events. These events are named respectively "Processed To", "Withdrawn", and "Released By" (see Table 1). Staffware does not record start, suspend, resume, and abort event. Moreover, it records event types not in Figure 5, e.g., "Start"[2], "Expired", and "Terminated". When mapping Staffware logs onto the XML format, one can choose to simply filter out these events or to map them on events of type *normal*.

The FSM representing the potential events orders recorded by *IBM MQSeries Workflow* is quite different from the common FSM shown in Figure 5. (For more details see [33].) Therefore, we need a mapping of events and event sequences originating from MQSeries into events of the common FSM. MQSeries records corresponding events for all events represented in the common FSM. Due to the fact that MQSeries FSM has more states and transitions than the common FSM, there are sets of events that must be mapped into a single event of the common FSM. The most frequent event sequence in MQSeries is *Activity ready - Activity started - Activity implementation completed*. This sequence is logged whenever an activity is executed without any exceptions or complications. It is mapped into *Schedule - Start - Complete*. Furthermore, there are a lot of different special cases. For example, an activity may be cancelled while being in state

[2] The start event in Staffware denotes the creation of a case and should not be confused with the start event in Figure 5.

Scheduled. The order of events in the common FSM is *Schedule* - *Withdraw*. The equivalent first event in MQSeries FSM is *Activity ready*. The second event could be *Activity inError*, *Activity expired*, *User issued a terminate command*, *Activity force-finished* or *Activity terminated*. So, a sequence with first part *Activity ready* and one of the five events mentioned before as second part is mapped into *Schedule* - *Withdraw*. Another difference is that an activity may be cancelled while running, i.e., it is in state *Active*. MQSeries will log this case in form of a sequence starting with *Activity ready*, proceeding with *Activity started*, and ending with one of the five events specified above. Such a sequence is mapped into *Schedule* - *Start* - *Abort*. Beside these examples, there are many more cases that have to be handled. The tool *QuaXMap* (M*Q*Series A*u*dit Tr*a*il *X*ML *Map*per, [53]) implements the complete mapping.

Let us return to Figure 4. At this moment, we have developed translations from the log files of workflow management systems Staffware (Staffware PLC, [58]), InConcert (TIBCO, [60]), and MQSeries Workflow (IBM, [32]) to our XML format. In the future, we plan to provide more translations from a wide range of systems (ERP, CRM, case handling, and B2B systems). Experience shows that it is also fairly simple to extract information from enterprise-specific information systems and translate this to the XML format (as long as the information is there). Figure 4 also shows some of the mining tools available. These tools support the approaches presented in the remainder of this paper and can all read the XML format.

5 Which class of workflow processes can be rediscovered? - An approach based on Petri net theory

The first approach we would like to discuss in more detail uses a specific class of Petri nets, named *workflow nets* (WF-nets), as a theoretical basis [1, 4]. Some of the results have been reported in [3, 8] and there are two tools to support this approach: EMiT [3] and MiMo [8]. Note that the tool Little Thumb (see Section 6) also support this approach but in addition is able to deal with noise.

In this more theoretical approach, 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 *rediscover* the workflow process, i.e., for which class of workflow models is it possible to accurately construct the model by merely looking at their logs. This is not as simple as it seems. Consider for example the process model shown in Figure 3. The corresponding workflow log shown in Table 2 does not show any information about the AND-split and the AND-join. Nevertheless, they are needed to accurately describe the process.

To illustrate the *rediscovery problem* we use Figure 6. Suppose we have a log based on many executions of the process described by a WF-net WF_1. Based on this workflow log and using a mining algorithm we construct a WF-net WF_2. An interesting question is whether $WF_1 = WF_2$. In this paper, we explore the class of WF-nets for which $WF_1 = WF_2$.

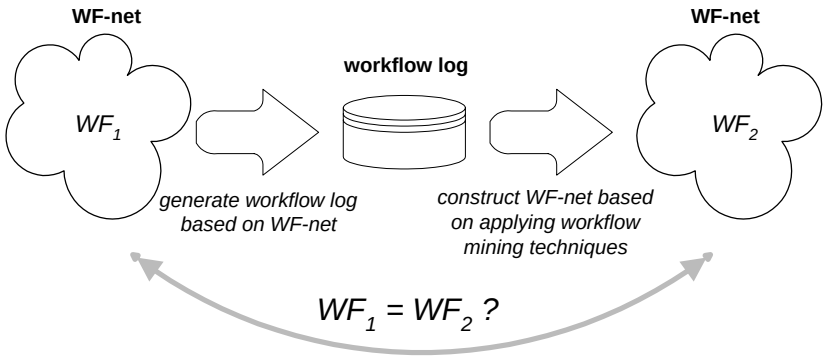

Fig. 6. The rediscovery problem: For which class of WF-nets is it guaranteed that WF_2 is equivalent to WF_1?

As shown in [8] it is impossible to rediscover the class of all WF-nets. However, the α algorithm described in [3, 8] can successfully rediscover a large class of practically relevant WF-nets. For this result, we assume logs to be complete in the sense that if two events can follow each other, they will follow each other at least once in the log. Note that this local criterion does not require the presence of all possible execution sequences.

The α algorithm is based on four ordering relations which can be derived from the log: $>_W$, $\rightarrow_W$, $\#_W$, and $\|_W$. Let W be a workflow log over a set of tasks T, i.e., $W \in \mathcal{P}(T^*)$. (The workflow log is simply a set or traces, one for each case, and we abstract from time, data, etc.). Let $a, b \in T$: (1) $a >_W b$ 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 $t_{i+1} = b$, (2) $a \rightarrow_W b$ if and only if $a >_W b$ and $b \not>_W a$, (3) $a \#_W b$ if and only if $a \not>_W b$ and $b \not>_W a$, and (4) $a \|_W b$ if and only if $a >_W b$ and $b >_W a$. $a >_W b$ if for at least one case a is directly followed by b. This does *not* imply that there is a causal relation between a and b, because a and b can be in parallel. Relation $\rightarrow_W$ suggests causality and relations $\|_W$ and $\#_W$ are used to differentiate between parallelism and choice. Since all relations can be derived from $>_W$, we assume the log to be complete with respect to $>_W$ (i.e., if one task can follow another task directly, then the log should have registered this potential behavior).

It is interesting to observe that classical limits in Petri-net theory also apply in the case of workflow mining. For example, the α algorithm has problems dealing with non-free-choice constructs [18]. It is well-known that many problems that are undecidable for general Petri nets are decidable for free-choice nets. This knowledge has been used to indicate the limits of workflow mining. Another interesting observation is that there are typically multiple WF-nets that match with a given workflow log. This is not surprising because two syntactically different WF-nets may have the same behavior. The α algorithm will construct

the "simplest" WF-net generating the desired behavior. Consider for example the log shown in Table 2. The α algorithm will construct a smaller WF-net (i.e., smaller than the WF-net shown in Figure 3) without explicitly representing the AND-split and AND-join transitions as they are not visible in the log. The resulting net is shown in Figure 7. Note that the behavior of the WF-net shown in Figure 7 is equivalent to the behavior of the WF-net shown in Figure 3 using trace equivalence and abstracting from the AND-split and AND-join.

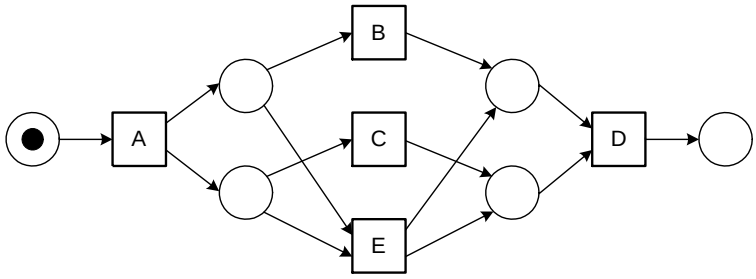

Fig. 7. The WF-net generated by the α algorithm using the log shown in Table 2.

A limitation of the α algorithm is that certain kinds of loops and multiple tasks having the same name cannot be detected. It seems that the problems related to loops can be resolved. Moreover, the α algorithm can also mine timed workflow logs and calculate all kinds to performance metrics. All of this is supported by the mining tool *EMiT* (Enhanced Mining Tool, [3]). EMiT can read the XML format and provides translators from Staffware and InConcert to this format. Moreover, EMiT fully supports the transactional task model shown in Figure 5. The output of EMiT is a graphical process model including all kinds of performance metrics. Figure 8 shows a screenshot of EMiT while analyzing a Staffware log.

6 How to deal with noise and incomplete logs: Heuristic approaches

The formal approach presented in the preceding section presupposes perfect information: (i) the log must be complete (i.e., if a task can follow another task directly, the log should contain an example of this behavior) and (ii) we assume that there is no noise in the log (i.e., everything that is registered in the log is correct). However, in practical situations logs are rarely complete and/or noise free. Therefore, in practical situations, it becomes more difficult to decide if between two events say a, b one of the three basic relations (i.e., $a \rightarrow_W b$, $a \#_W b$, and $a \|_W b$) holds. For instance the causality relation ($a \rightarrow_W b$) between two tasks a and b only holds if and only if in the log there is a trace in which a is directly followed by b (i.e., the relation $a >_W b$ holds) and there is no

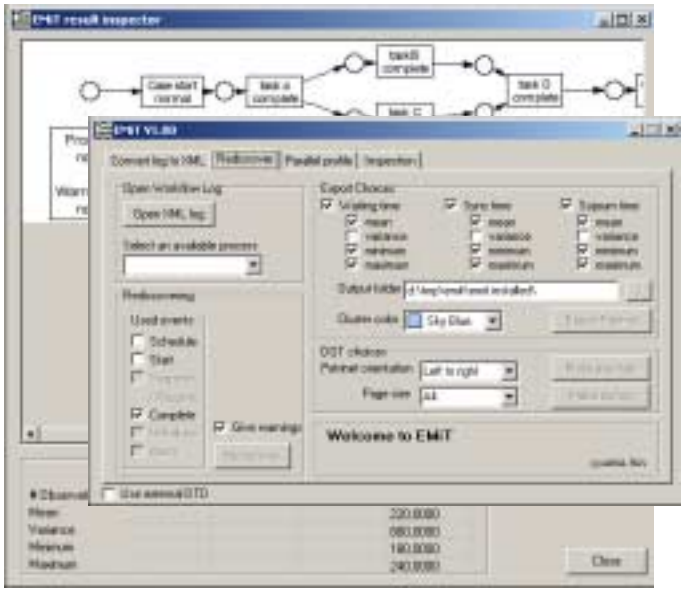

Fig. 8. A screenshot of the mining tool EMiT.

trace in which b is directly followed by a (i.e., not $b >_W a$). However, in a noisy situation one erroneous example can completely mess up the derivation of a right conclusion. For this reason we try to developed heuristic mining techniques which are less sensitive for noise and the incompleteness of logs. Moreover, we try to conquer some other limitations of the α algorithm (e.g., certain kinds of loops and non-free-choice constructs).

In our heuristic approaches [61, 62, 43] we distinguish three mining steps: Step (i) the construction of a dependency/frequency table (D/F-table), Step (ii) the mining of the basic relations out of the D/F-table (the mining of the R-table), and Step (iii) the reconstruction of the WF-net out of the R-table.

6.1 Construction of the dependency/frequency table

The starting point in our workflow mining techniques is the construction of a D/F-table. For each task a the following information is abstracted out of the workflow log: (i) the overall frequency of task a (notation $\#A$[3]), (ii) the frequency of task a directly preceded by task b (notation $\#B < A$), (iii) the frequency of a directly followed by task b (notation $\#A > B$), (iv) the frequency of a directly or indirectly preceded by task b but before the previous appearance of b (notation $\#B <<< A$), (v) the frequency of a directly or indirectly followed by task b but before the next appearance of a (notation $\#A >>> B$), and finally (vi)

[3] Note that we use a capital letter when referring to the number of occurrences of some task.

a metric that indicates the strength of the causal relation between task a and another task b (notation $\#A \rightarrow B$).

Metrics (i) through (v) seem clear without extra explanation. The underlying intuition of metric (vi) is as follows. If it is always the case that, when task a occurs, shortly later task b also occurs, then it is plausible that task a causes the occurrence of task b. On the other hand, if task b occurs (shortly) before task a, it is implausible that task a is the cause of task b. Below we define the formalization of this intuition. If, in an event stream, task a occurs before task b and n is the number of intermediary events between them, the $\#A \rightarrow B$-causality counter is incremented with a factor δ^n (δ is a causality fall factor and $\delta \in [0.0 \ldots 1.0]$). In our experiments δ is set to 0.8. The effect is that the contribution to the causality metric is maximal 1 (if task b appears directly after task a then $n = 0$ and $\delta^n = 1$ and decreases if the distance increases. The process of looking forward from task a to the occurrence of task b stops after the first occurrence of task a or task b. If task b occurs before task a and n is again the number of intermediary events between them, the $\#A \rightarrow B$-causality counter is decreased with a factor δ^n. After processing the whole workflow log the $\#A \rightarrow B$-causality counter is divided by the minimum overall frequency of task a and b (i.e., $min(\#A, \#B)$). Note that the value of $\#A \rightarrow B$ can be relatively high even when there is no trace in the log in which a is directly followed by b (i.e., the log is not complete).

6.2 The basic relations table (R-table) out of the D/F-table

Using relatively simple heuristics, we can determine the basic relations ($a \rightarrow_W b$, $a \#_W b$, and $a \|_W b$) out of the D/F-table. As an example we look at a heuristic rule for the $a \rightarrow_W b$-relation as presented in the previous section and [61].

IF $((\#A \rightarrow B \geq N)$ AND $(\#A > B \geq \theta)$ AND $(\#B < A \leq \theta))$
THEN $a \rightarrow_W b$

The first condition $(\#A \rightarrow B \geq N)$ uses the noise factor N (default value 0.05). If we expect more noise, we can increase this factor. The first condition calls for a higher positive causality between task a and b than the value of the noise factor. The second condition $(\#A > B \geq \theta)$ contains a threshold value θ. If we know that we have a workflow log that is totally noise free, then every task-pattern-occurrence is informative. However, to protect our induction process against inferences based on noise, only task-pattern-occurrences above a threshold frequency N are reliable enough for our induction process. To limit the number of parameters the value θ is automatically calculated using the following equation: $\theta = 1 + \frac{Round(N \times \#L)}{\#T}$. In this expression N is the noise factor, $\#L$ is the number of trace lines in the workflow log, and $\#T$ is the number of elements (different tasks). Using these heuristic rules we can build $a \rightarrow_W b$-relation and group the results in the so-called relations table (R-table).

6.3 The reconstruction of the WF-net out of the R-table

In step (iii) of our heuristic approaches, we can use the same α algorithm as in the formal approach. The result is a process model (i.e., Petri net). In a possible extra step, we use the task frequency to check if the number of task-occurrences is consistent with the resulting Petri-net.

To test the approach we use Petri-net-representations of different free-choice workflow models. All models contain concurrent processes and loops. For each model we generated three random workflow logs with 1000 event sequences: (i) a workflow log without noise, (ii) one with 5% noise, and (iii) a log with 10% noise. Below we explain what we mean with noise. To incorporate noise in our workflow logs we define four different types of noise generating operations: (i) delete the head of a event sequence, (ii) delete the tail of a sequence, (iii) delete a part of the body, and finally (iv) interchange two random chosen events. During the deletion-operations at least one event and at most one third of the sequence is deleted. The first step in generating a workflow log with 5% noise is a normal random generated workflow log. The next step is the random selection of 5% of the original event sequences and applying one of the four above described noise generating operations on it (each noise generation operation with an equal chance of 1/4). Applying the method presented in this section on the material without noise we found exact copies of the underlying WF-nets. If we add 5% noise to the workflow logs, the resulting WF-nets are still perfect. However, if we add 10% noise to the workflow logs the WF-nets contains errors. All errors are caused by the low threshold value. If we increase the noise factor value to a higher value ($N = 0.10$), all errors disappear. For more details we refer to [61].

The use of a threshold value is a disadvantage of the first approach. We are working on two possible solutions: (1) the use of machine learning techniques for automatically induction of an optimal threshold [43], and (2) the formulation of other measurements and rules without thresholds. Some of these heuristics are implemented in the heuristic workflow mining tool Little Thumb. (The tool is named after the fairy tail "Little Thumb" where a boy, not taller than a thumb, first leaves small stones to find his way back. The stones refer to mining using complete logs without noise. Then the boy leaves bread crusts that are partially eaten by birds. The latter situation refer to mining with incomplete logs with noise. Another analogy is the observation that the tool uses "rules of thumb" to extract causal relations.) Little Thumb follows the XML-input standard presented in Section 4.

7 How to measure the quality of a mined workflow model? - An experimental approach

As we already mentioned in Section 5, there are classes of Petri nets for which we can formally prove that the mined model is equivalent or has a behavior similar to the original Petri net. In this section we search for more general methods to measure the quality of mined workflow models.

An important criterion for the quality of a mined workflow model is the consistency between the mined model and the traces in the workflow log. Therefore, a standard check for a mined model, is to try to execute all traces of the workflow log in the discovered model. If the trace of a case cannot be executed in the Petri net, there is a discrepancy between the log and the model. This is a simple first check. However, for each workflow log it is possible to define a trivial model that is able to generate all traces of the workflow log (and many more). Another problem is the execution of traces with noise (i.e., the error is not in the model but in the log)

In our experimental setup, we assume that we know the workflow model that is used to generate the workflow log. In this subsection we will concentrate on methods to measure the quality of a mined workflow model by comparing it with the original model (i.e. the workflow model used for generating the workflow log used for mining). We will measure the quality of the mined model by specifying the amount of correctly detected basic relations, i.e. the correctness of the R-table descibed in the previous section.

The basic idea is to define a kind of a test bed to measure the performance of different workflow mining methods. In order to generate testing material that resembles real workflow logs, we identify some of the elements that vary from workflow to workflow and subsequently affect the workflow log. They are (i) the total number of events types, (ii) the amount of available information in the workflow log, (iii) the amount of noise and (iv) the imbalance in OR-splits and AND-splits. Therefore, we used a data generation procedure in which the four mentioned elements vary in the following way:

1. The number of task types: we generate Petri nets with 12, 22, 32 and 42 event types.
2. The amount of information in the process log or log size: the amount of information is expressed by varying the number of cases. We consider logs with 200, 400, 600, 800 and 1000 cases.
3. The amount of noise: we generate noise by performing four different operations on the event sequences representing individual cases: (i) delete the head of a event sequence, (ii) delete the tail of a sequence, (iii) delete a part of the body and (iv) interchange two randomly chosen events. During the noise generation process, minimally one event and maximally one third of the sequence is deleted. We generate five levels of noise: 0% noise (the initial workflow log without noise), 5% noise, 10%, 20% and 50% (we select 5%, 10%, 20% and respectively 50% of the original event sequences and we apply one of the four above described noise generation operations).
4. The imbalance of execution priorities: we assume that tasks can be executed with priorities between 0 and 2. In Figure 9 there is a choice after executing the event A (which is an OR-split). This choice may be balanced, i.e., task B and task F can have equal probabilities, or not. For example, task B can have an execution priority of 0.8 and task F 1.5 causing F to happen almost twice as often as B. The execution imbalance is produced on four levels:
   - Level 0, no imbalance: all tasks have the execution priority 1;

- Level 1, small imbalance: each task can be executed with a priority randomly chosen between 0.9 and 1.1;
- Level 2, medium imbalance: each task can be executed with a priority randomly chosen between 0.5 and 1.5;
- Level 3, high imbalance: each task can be executed with a priority randomly chosen between 0.1 and 1.9.

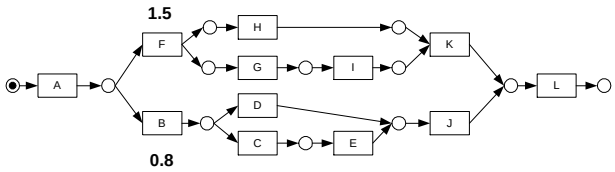

Fig. 9. Example of an unbalanced Petri net. Task B has an execution priority of 0.8 and task F an execution priority of 1.5.

The workflow logs produced with the proposed procedure allow the testing of different workflow mining methods, especially when it is desired to assess the method robustness against noise and incomplete data. We used the generated data for testing our heuristic approach discussed in the previous section.

The experiments show that our method is highly accurate when it comes to finding causal, exclusive and parallel relations. In fact we have been able to find almost all of them in the presence of incompleteness, imbalance and noise. Moreover, we gained the following insights:

- As expected, more noise, less balance and less cases, each have a negative effect on the quality of the result. The causal relations (i.e. $a \rightarrow_W b$) can be predicted more accurately if there is less noise, more balance and more cases.
- There is no clear evidence that the number of event types have an influence on the performance of predicting causal relations. However, causal relations in a structurally complex Petri net (e.g., non-free choice) can be more difficult to detect.
- Because the detection of exclusive/parallel relations ($a \#_W b$ and $a \|_W b$) depends on the detection of the causal relation, it is difficult to formulate specific conclusions for the quality of exclusive/parallel relations. It appears that noise is affecting exclusive and parallel relations in a similar way as the causal relation, e.g., if the level of noise is increasing, the accuracy of finding parallelism is decreasing.

When mining real workflow data, the above conclusions can play the role of useful recommendations. Usually it is difficult to know the level of noise and imbalance beforehand. However, during the mining process it is possible to collect more

data about these metrics. This information can be used to motivate additional efforts to collect more data.

The software supporting this experimental approach is called *ExperDiTo* (Experimental Discovery Tool). The generated data are available to be downloaded as benchmarks from `http://tmitwww.tm.tue.nl/staff/lmaruster/`.

8 How to mine workflow processes with duplicate tasks? - An inductive approach

The approaches presented in the preceding sections assume that a task name should be a unique identifier within a process, i.e., in the graphical models it is not possible to have multiple building blocks referring to the same task. For some processes this requirement does not hold. There may be more than one task sharing the same name. An example of such a process is the part release process for the development of passenger car from [25], which is shown in Figure 10. Although one may find unique names (e.g. "notifyEng-1", "notifyEng-2") even for these kind of processes, requiring unique names for the workflow mining procedure would be a tough requirement (compare [16]). Providing unique task names requires that the structure of the process is known at least to some extent. This is unrealistic if the workflow mining procedure is to be used to discover the structure.

In [26] we present a solution for mining workflow models with non-unique task names. It consists of two steps: the induction and the transformation step.

In the induction step a Stochastic Task Graph (also referred to as Stochastic Activity Graph, SAG [26]) is induced from the workflow log. The induction algorithm can be described as a graph generation algorithm (`InduceUniqueNodeSAG`) that is embedded into a search procedure.

The search procedure borrows ideas from machine learning and grammatical inference [49]. It searches for a mapping from task instances in the workflow log to task nodes in the workflow model. The search space can be described as a lattice of such mappings. Between the mappings there is a partial ordering (more general than/more specific than). The lattice is limited by a top or most general mapping (every task instance with name X is mapped to one single task node with name X) and a bottom or most specific element (the mapping is a bijection between task instances in the log and task nodes of the workflow model). Our search algorithm searches top down starting with the most general mapping for an optimal mapping. More specific mappings are created using a split operator. The split operator splits up all task instances mapped to the same task node of the model in two groups which are mapped two different task nodes by renaming task nodes. In the example shown in Figure 11 the task instances with names A and C of workflow log E_1 are split in A, A', C and C' using two split operations.

The `InduceUniqueNodeSAG` is called for a fixed mapping from instances to task nodes and it generates a stochastic task graph for this mapping as indicated

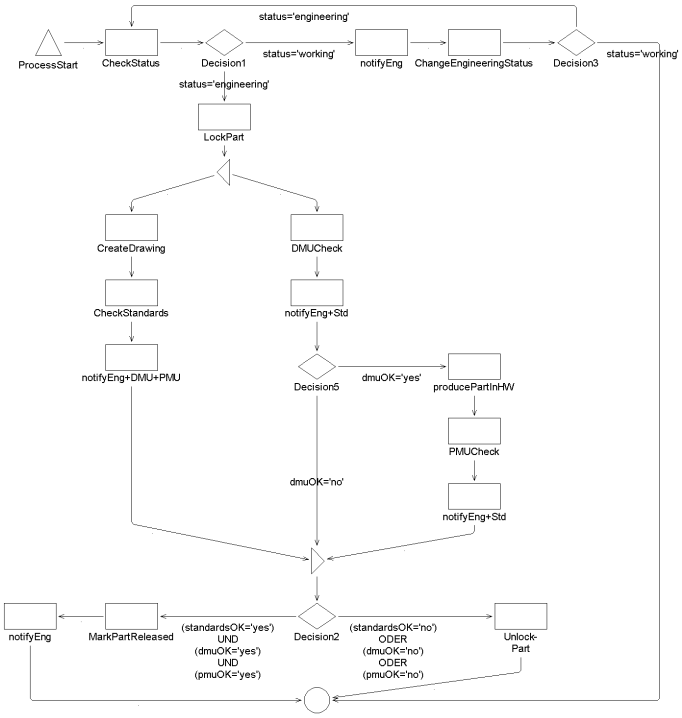

Fig. 10. The release process.

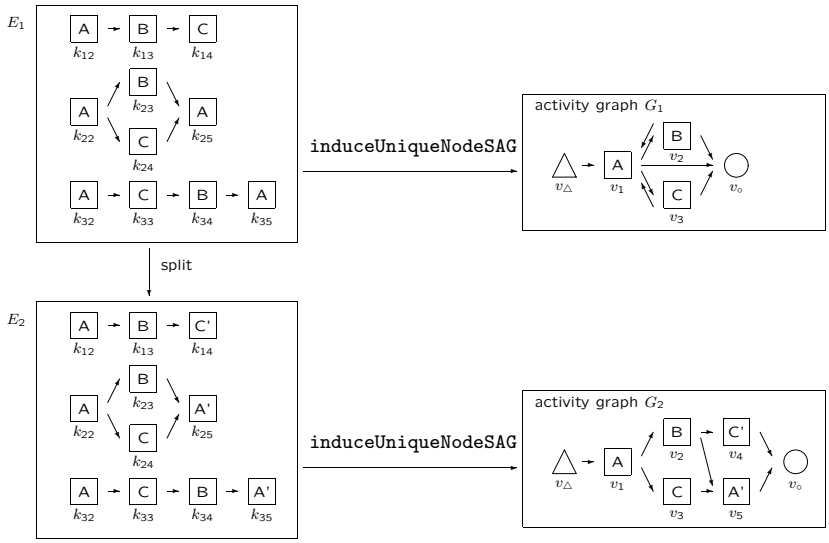

Fig. 11. The split operation.

in Figure 11. It is very similar to the approach presented in [11]. The main differences are a slightly different definition of the dependency relation and two additional steps for inserting copies of task nodes where required and for clustering task nodes sharing common predecessors. A notable difference to the formal approach in Section 5 is the determination of dependencies. `InduceUniqueNodeSAG` considers every pair of task instances occurring in the same instance - regardless of the number of task instances in between - for the determination of the dependency relation. The transitive reduction is used to identify direct successors. Note that the formal approach presented Section 5 considers only pairs of direct successors for determining the dependency relation.

The search algorithm applies beam-search. The search is guided by the log likelihood of the SAG per sample. The calculation of the log likelihood requires a stochastic sample. This means that the induction algorithm handles n workflow instances sharing exactly the same ordering of tasks as n different cases. For the formal approach (cf. Section 5) one instance for each ordering of tasks is enough. Using this information one is able to calculate not only the likelihood of the SAG but also the probability of tasks and edges. This information is useful to distinguish common from rare behavior.

In the transformation step the SAG is transformed into a block-structured workflow-model in the ADONIS format. This step is needed because the stochastic task graph provided by the induction phase does not explicitly distinguish alternative and parallel routing. The transformation phase can be decomposed into three main steps: (1) the analysis of the synchronization structures of the

workflow instances in the workflow log, (2) the generation of the synchronization structure of the workflow model, and (3) the generation of the model. Details of the transformation steps are given in [26].

The workflow mining tool InWoLvE (*In*ductive *Wo*rkflow *L*earning *v*ia *E*xamples) implements the described mining algorithm and two further induction algorithms, which are restricted to sequential workflow models. InWoLvE has an interface to the business process management system ADONIS [36] for interchanging workflow logs and process models. It has been successfully applied to workflow traces generated from real-life workflow models (such as the one shown in Figure 10) and from a large number of artificial workflow models.

Further details and additional aspects such as a transformation from the SAG to a well-behaved Petri net, an additional split operator for dealing with noise, and the results of the experimental evaluation are described in [26].

9 How to mine block-structured workflows? - A data mining approach

The last approach discussed in this paper is tailored towards mining block-structured workflows. There are two notable differences with the approaches presented in the preceding four sections. First of all, only block structured workflow patterns are considered. Second, the mining algorithm is based on rewriting techniques rather than graph-based techniques. In addition, the objective of this approach is to mine complete and minimal models: Complete in the sense that all recorded cases are covered by the extracted model, minimal in the sense that only recorded cases are covered. To achieve this goal the approach uses a stronger notion of completeness than e.g. the completeness notion based on direct successor (cf. Section 5).

Before we can mine a workflow model from event-based data it is necessary to determine what kind of model the output should be, i.e., the workflow language being used or the class of workflow models considered. Different languages/classes of models have different meta models. We distinguish two major groups of workflow meta-models: graph-oriented meta-models and block-oriented meta-models. This approach is based on a block-oriented meta-model. Models of this meta-model (i.e., block-structured workflows) are always well-formed and sound.

Block-structured models are made up from blocks which are nested. These building blocks of block-structured models can be differentiated into operators and constants. Operators build the process flow, while constants are the tasks or sub-workflows that are embedded inside the process flow. We build a block-structured model in a top-down fashion by setting one operator as starting point of the workflow and nest other operators as long as we get the desired flow structure. At the bottom of this structure we embed constants into operators which terminate the nesting process. A block-structured workflow model is a tree whose leafs are always operands.

Besides the tree representation of block-structured models we can specify them as a set of terms. Let S denote the operator *sequence*, P denote the operator *parallel*, and a, b, c denote three different tasks, the term $S(a, P(b, c))$, for example, represents a workflow performing task a completely before task b and task c are performed in parallel. Because of the model's block-structure each term is always well-formed. Further on, we can specify an algebra that consists of axioms for commutativity, distributivity, associativity, etc. These axioms form the basis for term rewriting systems we can use for mining workflows. A detailed description of the meta-model can be found in [54].

Based on the block-structured meta-model a process mining procedure extracts workflow models from event-based data. The procedure consists of the following five steps that are performed in sequential order.

First, the procedure reads event-based data that belongs to a certain process and builds a trace for each process instance from this data. A trace is a data structure that contains all *start* and *complete* events of a process instance in correct chronological order. After building traces, they are condensed on the basis of their sequence of *start* and *complete* events. Each trace group constitutes a path in the process schema.

Second, a time-forward algorithm constructs an initial process model from all trace groups. This model is in a special form called Disjunctive Normal Form (DNF). A process model in this form starts with an alternative operator and enumerates inside this block all possible paths of execution as blocks that are built up without any alternative operator. For each trace group such a block is constructed by the algorithm and added to the alternative operator that builds the root of the model.

The next step deals with relations between tasks that result from the random order of performing tasks without a real precedence relation between them. These pseudo precedence relations have to be identified and then removed from the model. In order to identify pseudo precedence relations the model is transformed by a term rewriting system into a form that enumerates all sequences of tasks inside parallel operators embedded into the overall alternative. Then, a searching algorithm determines which of these sequences are pseudo precedence relations. This is determined by finding the smallest subset of sequences that completely explains the corresponding blocks in the initial model. All sequences out of the subset are pseudo precedence relations and therefore removed. At the end of this step, the initial transformation is reversed by a term rewriting system.

Because the process model was built in DNF, it is necessary to split the model's overall alternative and to move the partial alternatives as near as possible to the point in time where a decision cannot be postponed any longer. This is done by a transformation step using another term rewriting system. It is based on distributivity axioms and merges blocks while shifting alternative operators towards later points in time. It also leads to a condensed form of the model.

The last step is an optional decision-mining step that is based on decision tree induction. In this step an induction is performed for each decision point of the model. In order to perform this step we need data about the workflow

context for each trace. From these data a tree induction algorithm builds decision trees. These trees are transformed into rules and then attached to the particular alternative operators.

After performing all steps, the output comes in form of a block-structured model that is complete and minimal. The process mining procedure is reported in more detail in [55, 56].

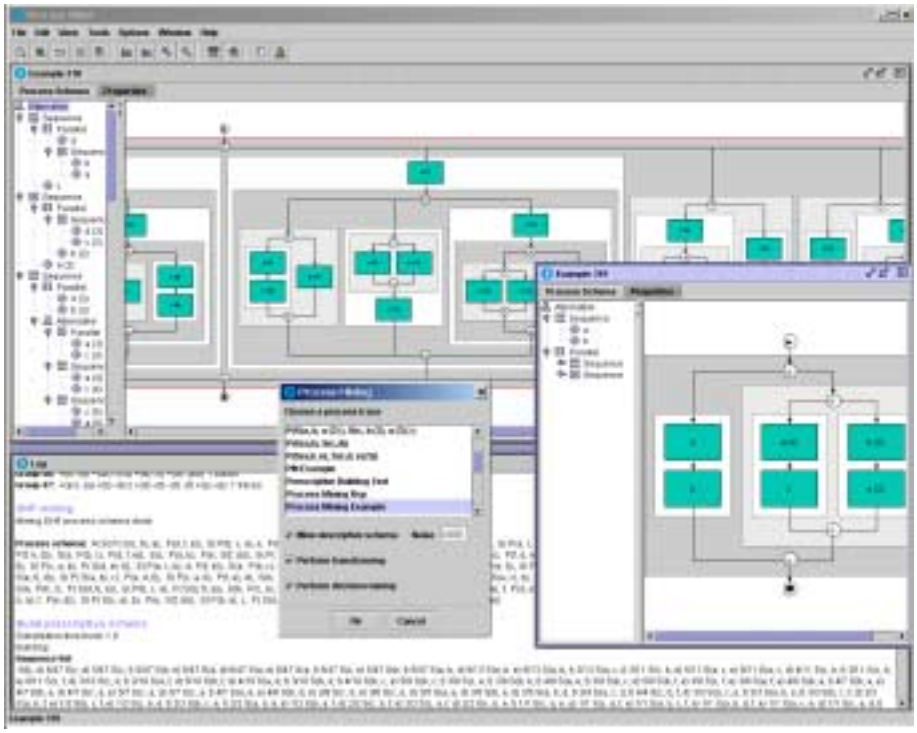

Fig. 12. The Tool *Process Miner*.

The approach on mining block-structured models is supported by a tool named *Process Miner*. This tool can read event-based workflow data from databases or from files in the XML format presented in Section 4. It then automatically performs the complete process mining procedure on this data. The decision-mining step is omitted if no context data are provided.

Process Miner comes with an graphical user interface (see Figure 12). It displays the output model in a graphical editor in form of a diagram and a tree. Additionally, it allows the user to edit a model and to export it for further use. It also contains a workflow simulation component. A description of *Process Miner* can be found in [57].

10 Comparison and open problems

As indicated in sections 5, 6, 7, 8, and 9 tools such as EMiT, Little Thumb, InWoLvE, and Process Miner are driven by different problems. In this section, we compare these approaches. To refer to each approach we use the name of the corresponding tool. *EMiT* [3] was introduced in Section 5 to explore the limits of mining (Which class of workflow processes can be rediscovered?). *Little Thumb* [9, 63] was introduced in Section 6 to illustrate how heuristics can be used to tackle the problem of noise. Section 8 presented the concepts the tool *InWoLvE* [26] is based on. One of the striking features of this tool is the ability to deal with duplicate tasks. Section 9 introduced the *Process Miner* [57], exploiting the properties of block-structured workflows through rewriting rules. In the remainder, we will compare the approaches represented by EMiT, Little Thumb, InWoLvE, and Process Miner. Note that we do not include the tool ExperDiTo (described in Section 7) in this comparison because it builds on EMiT and Little Thumb and does not offer alternative mining techniques.

To compare the approaches represented by EMiT, Little Thumb, InWoLvE, and Process Miner, we focus on nine aspects: Structure, Time, Basic parallelism, Non-free choice, Basic loops, Arbitrary loops, Hidden tasks, Duplicate tasks, and Noise. For each of these nine aspects we compare the four tools as indicated in Table 4 and described as follows.

Structure The first aspect refers to the structure of the target language. Languages such as Petri nets [50] are graph-based while textual languages such as π-calculus [45] are block-oriented. EMiT and Little Thumb are based on Petri nets and therefore graph-oriented. InWoLvE is also graph-based and Process Miner is the only block-oriented language.

Time Many logs also record time stamps of events. This information can be used to calculate performance indicators such as waiting/synchronization times, flow times, utilization, etc.

Basic parallelism All the tools are able to detect and handle parallelism. Simple processes where each AND-split corresponds to an AND-join can be mined by EMiT, Little Thumb, InWoLvE, and Process Miner. However, each of the four tools imposes requirements on the process in order to correctly extract the right model.

Non-free choice The Non-free choice (NFC) construct was mentioned in Section 5 as an example of a workflow pattern that is difficult to mine. NFC processes mix synchronization and choice in one construct as described in [18]. None of the four tools can deal with such constructs. Nevertheless, they are highly relevant as indicated in [6, 38].

Basic loops Each of the four tools can deal with loops. However, just like with parallelism, each of the tools imposes restrictions on the structure of these loops in order to guarantee the correctness of the discovered model.

Arbitrary loops None of the tools supports arbitrary loops. For example, the tool Process Miner can only have loops with a clear block structure. Note that not every loop can be modeled like this cf. [38]. EMiT and Little Thumb

initially had problems with loops of length 1 or 2. These problems have been (partially) solved by a preprocessing step. Note that to detect "short loops" more observations are required.

Hidden tasks Occurrences of specific tasks may not be recorded in the log. This is a fundamental problem since without this information processes are incomplete. Despite the fact that it will never be possible to detect task occurences that are not recorded, there could be facilities to indicate the presence of a so-called "hidden task". Suppose that a workflow language has special control tasks to model AND-splits and AND-joins. Even if these control tasks are not logged, one could still deduce their presence. None of the four tools supports the detection of hidden tasks in a structured manner.

Duplicate tasks EMiT, Little Thumb and Process Miner assume that each task appears only once in the workflow, i.e., the same task cannot be used in two different parts of the processes. (Note that this does *not* refer to loops. In a loop, the same part of the processes is repeatedly executed.) InWoLvE is the only tool dealing with this issue.

Noise The term noise is used to refer to the situation where the log is incomplete or contains errors. A similar situation occurs if a rare sequence of events takes place which is not representative for the typical flow of work (i.e., an exception). In both cases the resulting model can be incorrect (i.e., not representing the typical flow of work). EMiT and Process Miner do not offer features for dealing with noise. Little Thumb is able to deal with noise by using a set of heuristics which can be fine-tuned to tackle specific types of noise. InWoLvE uses a stochastic model which allows for the distinguishing common from rare behavior.

	EMiT	**Little Thumb**	**InWoLvE**	**Process Miner**
Structure	Graph	Graph	Graph	Block
Time	Yes	No	No	No
Basic parallelism	Yes	Yes	Yes	Yes
Non-free choice	No	No	No	No
Basic loops	Yes	Yes	Yes	Yes
Arbitrary loops	Yes	Yes	No	No
Hidden tasks	No	No	No	No
Duplicate tasks	No	No	Yes	No
Noise	No	Yes	Yes	No

Table 4. Comparing EMiT, Litte Thumb, InWoLvE, and Process Miner.

Table 4 shows that there are still a number of open problems. Few of the tools exploit timing information. Although EMiT extracts information on waiting times, flow times, and utilization from the log, time stamps are not used to improve the mining result. Similarly, other pieces of information like the data

objects being changed or the identity of the person executing a task are not exploited by the existing approaches. Existing approaches can deal with basic routing constructs such as basic parallelism and basic loops. However, these approaches fail when facing advanced routing constructs involving non-free choice constructs, hidden tasks, or duplicate tasks. Last but not least, there is the problem of noise. Although tools such as Little Thumb and InWoLvE can deal with some noise, empirical research is needed to evaluate and improve the heuristics being used.

The comparison shown in Table 4 can be used to position the various approaches. However, to truly compare the results there should be a number of *benchmark examples*. Section 7 discussed a number of small experiments that can be used for this purpose. However, for a real benchmark larger and more realistic examples are needed. Clearly, the XML format presented in Section 4 can be used to store these benchmark examples.

11 Conclusion

In this paper, we presented an overview of the various problems, techniques, tools, and approaches for workflow mining. It is quite interesting to see how the five approaches presented in sections 5, 6, 7, 8, and 9 differ and are driven by different problems. The more formal approach described in Section 5 uses Petri-net theory to characterize the class of workflow models that can be mined. The more heuristic approaches in sections 6 and 7 focus on issues such as noise and determining the quality of mining result. Unlike the other approaches, the approach in Section 8 takes into account the fact that there may be multiple tasks having the same label. Finally, the approach in Section 9 exploits the block structure (i.e., corresponding AND/XOR splits and AND/XOR joins) of many processes. Each of these approaches has its strengths and weaknesses.

Section 10 compared the approaches by focusing on nine aspects (Structure, Time, Basic parallelism, Non-free choice, Basic loops, Arbitrary loops, Hidden tasks, Duplicate tasks, and Noise). This comparison reveals differences and also points out problems that need to be tackled.

To join forces and to share knowledge and development efforts, we introduced a tool-independent XML format. This format was given in Section 4 and we would like to encourage other researchers/developers in this domain to use this format.

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 B.F. van Dongen. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer-Verlag, Berlin, 2002.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. QUT Technical report, FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002. (Also see http://www.tm.tue.nl/it/research/patterns.).
6. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. QUT Technical report, FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002. (Also see http://www.tm.tue.nl/it/research/patterns.) To appear in *Distributed and Parallel Databases*.
7. W.M.P. van der Aalst and S. Jablonski. Dealing with Workflow Change: Identification of Issues and Solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.
8. 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.
9. W.M.P. van der Aalst and T. Weijters. X-tra - KLeinduimpje in Workflowland: Op zoek naar procesdata. *Scope*, 10(12):38–40, 2002.
10. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 218–234. Springer-Verlag, Berlin, 2000.
11. 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.
12. T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. eXtensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, 2000.
13. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. In *Proceedings of ER '96*, pages 438–455, Cottubus, Germany, Oct 1996.
14. College Bescherming persoonsgegevens (CBP; Dutch Data Protection Authority). http://www.cbpweb.nl/index.htm.
15. 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.
16. 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.
17. 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.
18. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

19. J. Eder, G.E. Olivotto, and Wolfgang Gruber. A Data Warehouse for Workflow Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin, 2002.
20. C.A. Ellis and K. Keddara. A Workflow Change Is a Workflow. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 201–217. Springer-Verlag, Berlin, 2000.
21. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition.* Future Strategies, Lighthouse Point, Florida, 2001.
22. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
23. M. Hammer and J. Champy. *Reengineering the corporation.* Nicolas Brealey Publishing, London, 1993.
24. 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.
25. 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.
26. J. Herbst. *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen.* PhD thesis, Universität Ulm, November 2001.
27. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE, 1998.
28. 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.
29. 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.
30. T. Herrmann, M. Hoffmann, K.U. Loser, and K. Moysich. Semistructured models are surprisingly useful for user-centered design. In G. De Michelis, A. Giboin, L. Karsenty, and R. Dieng, editors, *Designing Cooperative Systems (Coop 2000)*, pages 159–174. IOS Press, Amsterdam, 2000.
31. B.J.P. Hulsman and P.C. Ippel. Personeelsinformatiesystemen: De Wet Persoonsregistraties toegepast. Registratiekamer, The Hague, 1994.
32. IBM. *IBM MQSeries Workflow - Getting Started With Buildtime.* IBM Deutschland Entwicklung GmbH, Boeblingen, Germany, 1999.
33. IBM. *IBM MQseries Workflow Programming Guide Version 3.3.* IBM Corporation, Armonk, USA, 2001.
34. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.

35. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
36. S. Junginger, H. Kühn, R. Strobl, and D. Karagiannis. Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation – ADONIS: Konzeption und Anwendungen. *Wirtschaftsinformatik*, 42(3):392–401, 2000.
37. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows (submitted)*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002. Available via http://www.tm.tue.nl/it/research/patterns.
38. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. QUT Technical report, FIT-TR-2002-03, Queensland University of Technology, Brisbane, 2002. (Also see http://www.tm.tue.nl/it/research/patterns.) To appear in *Acta Informatica*.
39. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems*, Seattle, Washington, November 1998.
40. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, volume 9 of *Special issue of the journal of Computer Supported Cooperative Work*, 2000.
41. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
42. 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.
43. L. Maruster, A.J.M.M. Weijters, W.M.P. van der Aalst, and A. van den Bosch. Process Mining: Discovering Direct Successors in Process Logs. In *Proceedings of the 5th International Conference on Discovery Science (Discovery Science 2002)*, volume 2534 of *Lecture Notes in Artificial Intelligence*, pages 364–373. Springer-Verlag, Berlin, 2002.
44. 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.
45. R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK, 1999.
46. M. zur Mühlen. Process-driven Management Information Systems Combining Data Warehouses and Workflow Technology. In B. Gavish, editor, *Proceedings of the International Conference on Electronic Commerce Research (ICECR-4)*, pages 550–566. IEEE Computer Society Press, Los Alamitos, California, 2001.
47. M. zur Mühlen. Workflow-based Process Controlling-Or: What You Can Measure You Can Control. In L. Fischer, editor, *Workflow Handbook 2001, Workflow Management Coalition*, pages 61–77. Future Strategies, Lighthouse Point, Florida, 2001.
48. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.

49. R. Parekh and V. Honavar. Automata Induction, Grammar Inference, and Language Acquisition. In Dale, Moisl, and Somers, editors, *Handbook of Natural Language Processing*. New York: Marcel Dekker, 2000.
50. 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.
51. L.B. Sauerwein and J.J. Linnemann. Guidelines for Personal Data Processors: Personal Data Protection Act. Ministry of Justice, The Hague, 2001.
52. M. Sayal, F. Casati, and M.C. Shan U. Dayal. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
53. G. Schimm. Process Mining. http://www.processmining.de/.
54. G. Schimm. Generic Linear Business Process Modeling. In S.W. Liddle, H.C. Mayr, and B. Thalheim, editors, *Proceedings of the ER 2000 Workshop on Conceptual Approaches for E-Business and The World Wide Web and Conceptual Modeling*, volume 1921 of *Lecture Notes in Computer Science*, pages 31–39. Springer-Verlag, Berlin, 2000.
55. G. Schimm. Process Mining elektronischer Geschäftsprozesse. In *Proceedings Elektronische Geschäftsprozesse*, 2001.
56. G. Schimm. Process Mining linearer Prozessmodelle - Ein Ansatz zur automatisierten Akquisition von Prozesswissen. In *Proceedings 1. Konferenz Professionelles Wissensmanagement*, 2001.
57. G. Schimm. Process Miner - A Tool for Mining Process Schemes from Event-based Data. In S. Flesca and G. Ianni, editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 525–528. Springer-Verlag, Berlin, 2002.
58. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 2000.
59. Staffware. Staffware Process Monitor (SPM). http://www.staffware.com, 2002.
60. Tibco. *TIB/InConcert Process Designer User's Guide*. Tibco Software Inc., Palo Alto, CA, USA, 2000.
61. 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.
62. 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.
63. A.J.M.M. Weijters and W.M.P. van der Aalst. Workflow Mining: Discovering Workflow Models from Event-Based Data. In C. Dousson, F. Höppner, and R. Quiniou, editors, *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 78–84, 2002.
64. M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In R. Sprague, editor, *Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-34)*. IEEE Computer Society Press, Los Alamitos, California, 2001.
65. Workflow Patterns Home Page. http://www.tm.tue.nl/it/research/patterns.

About the authors

Wil van der Aalst is a full professor of Information Systems and head of the section of Information and Technology of the Department of Technology Management at Eindhoven University of Technology. He is also a part-time full professor at the Computing Science faculty at the department of Mathematics and Computer Science at the same university. His research interests include information systems, simulation, Petri nets, process models, workflow management systems, verification techniques, enterprise resource planning systems, computer supported cooperative work, and interorganizational business processes.

Boudewijn van Dongen is a student at the Department of Computer Science and Mathematics at Eindhoven University of Technology, Eindhoven, The Netherlands. In 2002 he conducted a project on workflow mining and developed the workflow mining tool EMiT. Currently, he is doing his master thesis at the Department of Computer Science and Mathematics, after which he will become a Ph.D. candidate at the Department of Technology Management at Eindhoven University of Technology.

Joachim Herbst studied computer science at the University of Ulm, where he also did his Ph.D. in the area of workflow mining. Since 1995 he has been working for DaimlerChrysler Research and Technology. His research interests include machine learning, workflow management, enterprise application integration and concurrent engineering.

Laura Maruster received her B.S. degree in 1994 and M.S. in 1995, both in Computer Science Department at West University of Timisoara, Romania. At present she is a Ph.D candidate of the Department of Technology Management of Eindhoven University of Technology, Eindhoven, The Netherlands. Her research interests include induction of machine learning and statistical models, process mining and knowledge discovery.

Guido Schimm studied computer science and business economy at the University of Wernigerode, Germany. He joined the Oldenburg Institute for Computer Science Tools and Systems (OFFIS) in 1999 as a member of the business intelligence and knowledge management team. Guido has been engaged in many ERP and workflow projects. Currently, his research interest is focused on theoretical foundation and practical implementation of workflow mining technologies.

Ton Weijters is associate professor at the Department of Technology Management of the Eindhoven University of Technology (TUE), and member of the BETA research group. Currently he is working on (i) the application of Knowledge Engineering and Machine Learning techniques for planning, scheduling, and process mining (ii) fundamental research in the domain of Machine Learning and

Knowledge Discovering. He is the author of many scientific publications in the mentioned research field.