Discovering Workflow Patterns from Timed Logs

Walid Gaaloul, Sami Bhiri and Claude Godart

LORIA – INRIA CNRS (UMR 7503)
BP 239 F-54506 Vandoeuvre-lès-Nancy Cedex, France
gaaloul@loria.fr
bhiri@loria.fr
godart@loria.fr

Abstract: Traditionally, workflow modeling is typically done by interviews or questionnaires which stay quite limited for the acquisition of workflow models and their adaptation to changing requirements. In order to build such model, this paper proposes to "reverse the process". The approach, we propose, is able to acquire workflow model from workflow log, which contains information about execution events.

This paper presents a mining technique to discover workflows patterns (e.g. Sequence, Parallel split, Exclusive choice, etc.) from workflow event log in the objective to build a global workflow model. An event log is a view of a workflow execution at a given instant. The mining of workflow patterns is done by the combination of two complementary techniques: an algorithmic technique and a statistical analysis. The building of the global workflow model is done with bottom-up approach. The discovering technique proposed can deal with some ambiguities where the single use of an algorithmic technique method cannot resolve.

1 Introduction

One of the most time consuming activities for the process engineers is the acquisition of the initial workflow model [HB97]. This is due to the fact that the knowledge needed to define this workflow model is highly distributed within the organization. It is stored in the heads of the human actors, who are actively involved in the execution of the business process and have often conflicting interests. In besides, the modeling errors are commonly not detected until the process model is performed. In fact, it is difficult for process engineers to validate a formal process model using only visual representation of the process. They often agree visual representation, but when they are confronted with the WFMS implementing the process, it often turns out that the system has a different interpretation of the process model than they had expected. So the process model as it was modeled is rejected. We do believe that this "classical approach" is subjective and normative and pays less attention to the diagnosis phase. By closely monitoring the events taking place at run time, we can enable delta analysis, i.e., detecting discrepancies between the design constructed in the design phase and the actual execution registered in the enactment phase.

The workflow technology is moving into the direction of more operational flexibility to deal with workflow evolution and workflow exception handling [BDK99]. Trying to meet these requirements, we propose to "*reverse the process*". Instead of starting with a workflow design, we start by gathering information about the workflow processes as they took place. This approach is called *process mining* commonly called the *workflow mining*. We use this term for the method of distilling a structured process description from a set of real executions. This approach helps in the transformation of a tacit knowledge into an explicit knowledge. *Process mining* consists in the fact that all activities are traced and passed to the machine-learning component, which constantly adapts the provided workflow model. This adapted workflow model serves as input for the next optimization cycle. Such model can be used in both the diagnosis phase and the (re)design phase. Actually, *workflow mining* technique can be used to create a feedback loop to adapt the workflow model to changing circumstances and detect imperfections of the design.

The work described in this paper represents a technique to discover workflow patterns from traces of workflow events in the objective to build a global workflow model. Two problems are studied. The first problem is to discover workflow patterns, in a given workflow log, and identify the sequential or concurrent nature of joins and splits. The second problem is the global workflow build.

The remainder of this paper is as follows. First we introduce some prerequisites including the definition of a workflow log and an introduction of workflow patterns. Then we present a new technique for process mining by the discovery of workflow patterns. Finally, we conclude the paper by summarizing the main results and describing our future work.

2 Prerequisites: workflow patterns and event logs

2.1 Event log

Workflows are defined as *case-based*, i.e., every piece of work is executed for a specific *case*. Many cases can be handled by following the same workflow process definition. Routing elements are used to describe sequential, conditional, parallel, and iterative routing thus specifying the appropriate route of a case [JB96, La97]. To be complete, workflow log should cover all possible *cases* (i.e. if a specific routing element can appear in the mined workflow model, the log should contain an example of this behavior in at least one *case*). Thus, the completeness of mined workflow model depends on how much the log covers all possible dependencies between its tasks.

Because we are interested in extracting models that describe control flow within processes workflow, log abstracts only data relating to the life cycle of workflow task. We limit the information to the order in which a tasks are being executed. This presents a minimal information we assume to be present. Any information system using transactional systems such as ERP, CRM, or workflow management systems will offer this information in some form [VWM03]. Note that we do not assume the presence of a workflow management system. These workflow logs are used to construct a process model, which adequately conforms to the behavior registered.

We assume that it is possible to record events such that:

- Each event refers to a task (i.e., a welldefined step in the workflow),
- Each event refers to a case (i.e., a workflow instance),
- Events are totally ordered.

A workflow log is considered as a set of events streams. Each events stream represents an execution of one case. Each event is described by a task identifier and time of execution. Thus, an event is seen as a couple E : **event** (task_A, occurT_A), where (task_A : String) is the name of the task concerned with the event and (occurT_A : int) is the moment when it occurred.

An events stream is defined as a triplet **log** (beginTime, endTime, sequenceLog) where (beginTime: time) is the moment of log beginning, (endTime: time) is the moment of log end and (sequenceLog: sequence) is the sequence of the log events. For reasons of simplicity we assume that there is just one workflow process that has a log for each execution. An example of an events stream of the workflow of Figure 1 is given below:

L = log(5, 16, [**event**("A_1",5), **event**("A_2",8), **event**("A_4",9) , **event**("A_6",10) , **event**("A_8",11) , **event**("A_7",12), **event**("A_{10}",14), **event**("A_{11}",16)])

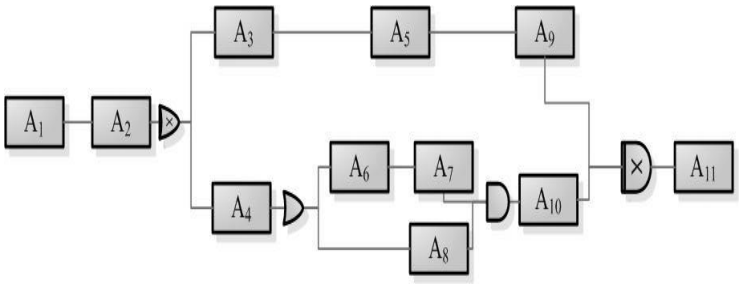


Figure 1. Example of workflow

One workflow log may contain information about thousands of events streams. Since there are no causal dependencies between events corresponding to different events streams, we can project the workflow log onto separate events streams for each instance of execution without loosing any information. Therefore, we can consider a workflow log as a set of events streams, where each event has an associate time of occurrence.

2.2 Workflow patterns

Workflows patterns can be used to examine the expressive power of a workflow that you intend to work with or they can serve as a set of ideas how to implement given business requirements [VHK03]. They identify comprehensive workflow functionality and provide the basis for an in-depth comparison of a number of commercially available workflow management systems. In this present work, we will exclusively interest to discover "elementary" workflow patterns: Sequence, Parallel split, Synchronization, Exclusive choice, Multiple choices, Simple merge and M-out-of-N Join.

Discovering a model of a system from event data basically involves determining the logical dependencies among events. *Direct dependence* is defined as an occurrence of one event type directly depending on another event type. We define three types of *direct dependence*. A *Sequential dependence* captures the sequencing of events (sequence pattern) where one event follows directly one other. A *Conditional dependence* captures selection, or a choice of one event from a set of events potentially following (*e.g.* exclusive choice pattern) or preceding (*e.g.* simple merge pattern) a given event. A *Concurrent dependence* captures concurrency in terms of "fork" (*e.g.* parallel split pattern) and "join" (*e.g.* synchronization pattern). Due to the *Concurrent dependence*, the events stream represents interleaved event sequences from all of concurrent threads. That's why we use "*concurrent windows*" (see section 3.1) to discover dependencies between events. The length of this window depends on the duration of concurrent sequence.

3 The workflow patterns mining technique

In this paper, two problems are studied. The first problem is to discover workflow patterns in a given workflow log and identify the sequential or concurrent nature of joins and splits. The second problem is to build the global workflow. A concrete algorithm is given for tackling the first problem. After that, a bottom-up technique allows to deal with the second problem.

In this section we present the details of our workflow mining technique. We can distinguish three mining steps: Step (i) the construction of a statistical dependency table, Step (ii) the discovery of frequent episodes in log, and Step (iii) the mining of workflow patterns and the reconstruction of the global workflow graph.

3.1 Construction of the statistical dependency table

Some numerical representations of characteristic sequencing behaviors are needed upon, which analysis can be performed to identify workflow patterns. The starting point of our workflow mining technique is the construction of a statistical dependency table. It is based on a notion of frequency table [CW95]. For each event *A* the following information is abstracted out of the workflow log: the overall frequency of task *A* (notation #*A*), and the frequency of *A* directly or indirectly preceded by another task *B* (notation *P(A/B)*). These information are recorded for each task in each events stream of the workflow log.

If we assume that events stream is exactly correct (i.e., contains no noise) and derives from a sequential workflow, as well as the zero entries are interpreted as signifying independence, the non-zero frequencies directly represent probabilistic dependence relations. The non-zero entry in the table would signify a correct event sequence, and so a causal dependency. But, in our case due to workflow patterns like Synchronization pattern, Parallel split pattern and Multiple choice pattern that produce concurrent events stream, some entries indicates spurious or false dependencies. For instance, an event might not, for some concurrency reasons, depend on the immediate predecessor, but on another "indirectly" preceding event. To unmask and correct this "dependency" frequency we calculate the frequency using a *concurrent window*, i.e. we will not only consider the events occurred immediately afterwards but also the events covered by the *concurrent window*.

Formally, a *concurrent window* defines a window of events over an events stream S:log(beginTime, endTime, SeqLog) as a triplet W:**window**(Ts, Te, wLog), where (Ts:time) is the moment of the window beginning (with beginTime≤Ts), (Te:time) is the moment of the window end (with Te≤endTime) and (wLog: sequence) is the sequence of the events which occurred between Ts and Te. We define a window as a slice of an event sequence, and we consider an events stream as a sequence of partially overlapping windows. The time span Te–Ts is called the *width* of the window, and it is denoted *width*(W). The *width* is the maximal duration that a concurrent execution can take. It depends on the studied workflow and is estimated by the user.

In others cases some entries in the statistical dependency table can indicate non-zero entries that do not correspond to dependencies. For example the events stream given in section 2.1 suggests a sequential dependency between A_6 and A_8, which is incorrect. To deal with this issue, we will use *episodes* to eliminate this noise and to identify correctly workflow patterns (see next section).

3.2 Discovering episodes in logs

Through the discovery of specific episodes in events stream, we can eliminate the confusion caused by the concurrence, which produces spurious non-zero entries in the statistical dependency table. For this reason we are interested in finding recurrent combinations of events, which we call *frequent episodes*. Our definition of *frequent episode* is a variation of the one from [MTI97]. Formally, an *episode* is a partially ordered collection of events occurring together. In our workflow mining technique we need to discover and identify parallel and serial episodes.

Parallel and serial episodes can be seen as two relations over workflow events. $P(A_1, A_2)$ denotes the parallel relation on events A_1 and A_2 and specifies that no ordering constraints are specified on these events. In practice, we will use an equivalency class of P to express parallel relation between more than two events. $S(A_3, A_4)$ denotes the serial relation on events A_3 and A_4 and specifies that these events occur in this order in the event stream. Note there can be other events occurring between them. The choice of parallel and serial episodes is important because they are easy to interpret and they can be discovered efficiently from long events stream [MTI97]. Moreover, any complex partially ordered episode could be seen as a recursive combination of parallel and serial episodes.

To deal with "noise" caused by the *concurrent dependence* in discovered episodes, we adapted an algorithm proposed in [MTI97] to find such class of episodes that are frequent enough and occur in *concurrent window*. To be considered interesting, the events of an episode must occur close each of the other enough in the time by belonging to the same *concurrent window*. The length of *concurrent window* defines how close are these events. In addition to the width of the window, we can specify in how many windows an episode has occurred to be considered frequent.

3.3 Mining of workflow patterns

After the compute of the statistical dependency table and the discovery of episodes, our last task will be the identification of workflow patterns and the construction of the workflow graph. In fact, each pattern will be identified by a particular episodes set and a test series that concern the frequency of the tasks included in it. Each pattern has its own features, which represents its unique identifier. Our algorithm allows, if the execution log is completed (*i.e.* having sufficient execution cases for the covering of all the possible different scenarios), the discovery of the whole workflow patterns included in the mined workflow.

We divided the workflows patterns in three categories: sequence, fork and join patterns. In the following we will present rules to discover the most interesting workflow patterns belonging to these three categories.

3.3.1 Sequence pattern

In this category we find only the sequence pattern (*c.f.* table 1). In this pattern, the enactment of the task B depends only on the completion of task A. So we need, in besides of the discovery of S(A,B) episode, statistics properties (P(B/A)=1 ∧ #B=#A) that ensure the exclusive dependency linked B to A.

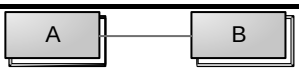
Rules : Episodes + Frequencies	**Mining Workflow pattern**
S (A,B)	
P(B/A)=1 ∧ #B= #A	Sequence pattern

Table 1. Rules of sequential workflow pattern mining

3.3.2 Fork patterns

This category (*c.f.* table 2) has a "fork" point where a single thread of control splits into multiple threads of control which can be, according to the used pattern, executed or not.

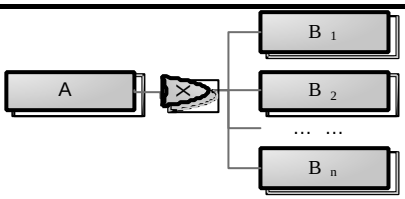
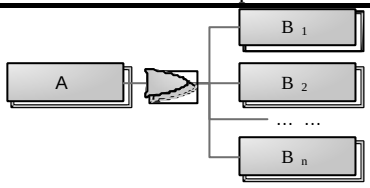
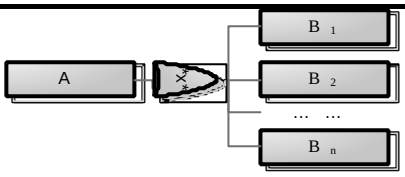
Rules : Episodes + Frequencies	**Mining Workflow pattern**
$\bigwedge_{i=1}^{n} (S(A, B_i))$	
$(\forall 0 \le i \le n; P(Bi/A)=1) \wedge (\forall 0 \le i,j \le n; P(Bi/Bj)=0) \wedge (\sum \#Bi = \#A)$	Exclusive choice pattern
$S(A, \overline{P}(B_1, B_2, .., B_n))$ *where* $\overline{P}$ *is an equivalency class of P containing* $\{B_i; 0 £ i £ n\}$	
$(\forall 0 \le i \le n; P(Bi/A)=1) \wedge (\forall 0 \le i,j < n; P(Bi/Bj) \neq 0) \wedge (\forall\ 0 \le i \le n \quad \#Bi = \#A)$	Parallel split pattern
$S(A, \overline{P}(B_1, B_2, .., B_n))$ *where* $\overline{P}$ *is an equivalency class of P containing* $\{B_i; 0 £ i £ n\}$	
$(\forall 0 \le i \le n; P(Bi/A)=1) \wedge (\forall 0 \le i,j < n; P(Bi/Bj) \neq 0) \wedge (\#A \le \sum \#Bi) \wedge (\#Bi \le \#A)$	Multi choice pattern

Table 2. Rules of fork workflow patterns mining

The causality between the tasks A and Bi before and after "fork" point is shared by Exclusive Choice, Parallel Split and Multi-choice, the three patterns of this category. This causality is ensured by the statistical property ($\forall 0 \le i \le n$; $P(B/Ai)=1$). The Exclusive choice pattern, where one of several branches is chosen after "fork" point, has an episode different from Parallel Split and Multi-choice patterns, which have the same episode. The non-parallelism between Bi, in the Exclusive choice pattern are ensured by ($\forall 0 \le i,j \le n$; $P(Bi/Bj)=0$). Parallel Split and Multi-choice patterns differentiate themselves by the frequencies relation between the task A and the tasks Bi. Effectively, only a part of tasks are executed in the Multi-choice pattern after "fork" point, while all the Bi tasks are executed in Parallel Split pattern.

3.3.3 Join patterns

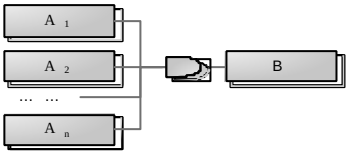
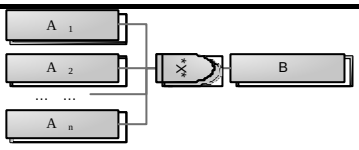
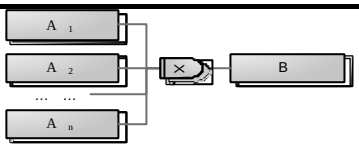
Rules : Episodes + Frequencies	**Mining Workflow pattern**
$S(\overline{P}(A_1, A_2, .., A_n), B)$ *where* $\overline{P}$ *is an equivalency class of P containing* $\{A_i; 0£i£n\}$ $(\forall 0 \le i \le n; P(B/Ai)=1) \wedge (\forall 0 \le i,j \le n\ P(Ai/Aj) \neq 0) \wedge (\forall 0 \le i \le n \quad \#B = \# Ai)$	 Synchronization pattern
$\bigwedge_{i=1}^{n} S(A_i, B)$ $(\sum P(B/Ai)=1) \wedge (\exists\ 0 \le i,j \le n\ P(Ai/Aj)=0) \wedge (\forall 0 \le i \le n \quad \sum \# Ai = \#B)$	 Simple merge pattern
$\bigwedge_{i=1}^{n} S(A_i, B)$ $(n \ge \sum P(B/Ai) \ge m) \wedge (\exists\ 0 \le i,j \le n\ P(Ai/Aj) \neq 0) \wedge (\forall 0 \le i \le n\ \ m*\#B \le \# Ai)$	 M-out-of-N Join pattern

Table 3. Rules of join workflow patterns mining

This category (*c.f.* table 3) has a "join" point where two or more alternative branches come together. These alternative branches come with or without synchronization according to the used pattern. The number of necessary branches for the activation of the task B after the "join" point depends on the used pattern.

The enactment of task B after the "join" point in the Synchronization pattern requires the execution of all the Ai tasks ($\forall 0 \leq i \leq n$; $P(B/Ai)=1$). In contrary of Simple Merge and M-out-of-N-join pattern that have the same episodes different from the Synchronization pattern and where the parallelism between the Ai tasks can be only seen in the M-out-of-N-join pattern ($\exists$ $0 \leq i,j \leq n$ $P(Ai/Aj) \neq 0$). Furthermore, the enactment of task B in M-out-of-N-join pattern requires the execution of (n) Ai tasks at least ($n \geq \sum P(B/Ai) \geq m$).

3.4 Reconstruction of global workflow graph

The mined workflows are based on pattern-oriented model. It defines that each workflow model consists of an arbitrary number of nested and linked workflow patterns (see Figure 2). After the discovery of all the workflow patterns, the construction of the complete graph of the global workflow will be done following the bottom-up technique by linking one by one the discovered workflow patterns. Discovering a pattern-oriented model ensures a sound and well-formed mined workflow. Therefore, by using this kind of model we are sure that the extracted workflow models do not contain any deadlocks or other anomalies.

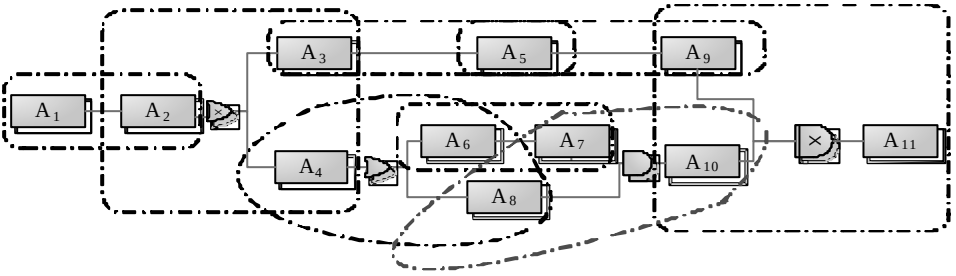


Figure 2. Discovered workflow

As a working example, consider the workflow shown in Figure 2. We will focus on the discovery of the synchronization pattern (A_7,A_8,A_{10}). The width of the *concurrence window* infers the inclusion of tasks A_6 in our computing statistical dependency table and the discovery of episodes. This inclusion will allow us to remove any confusion or erroneous deductions. Table 4 presents a fraction of the statistical dependency table. Values in bold are used to check statistical dependencies for this pattern :

$P(A_i/A_j)$	A_6	A_7	A_8	A_{10}
# A_6=100	0	0	0.36	0
# A_7=100	1	0	**0.41**	0
# A_8=100	0.43	**0.29**	0	0
#A_{10}=100	1	**1**	**1**	0

Table 4. Fraction of the statistical dependency table

The episodes set discovered in the log and allowing us to identify the synchronization pattern are:

$$S(\overline{P}(A_7,A_8), A_{10})$$

Note that the frequency $P(A_7/A_6)=1$ lets us think about the sequential pattern which can give an indication about the episodes class that we must find in order to identify this pattern.

4 Conclusion

In this paper we (i) introduced the context of workflow processes and process mining, (ii) some preliminaries including an introduction of workflow patterns, and (iii) a definition of a workflow log. After that, we presented the details of the three steps of our process mining technique: Step (i) the construction of the statistical dependency table, Step (ii) the discovery of frequent episodes in log, Step (iii) the mining of workflow patterns, and Step (iv) the reconstruction of global workflow graph.

Our algorithm is distinguished vis-à-vis others proposed techniques in *workflow mining* [AGL98, CW98, He00, VWM03] :

- It assumes a new approach never stated until now, it is characterized by a partial discovery of the workflow at the initial phase. Therefore, we can recover results of mining patterns workflows even if our log is incomplete;
- It discovers more complex transactional features with a better specification of "fork" point (Exclusive choice, Parallel split and Multi choice patterns) and "join" point (Synchronization, Simple merge and M-out-of-N Join patterns);
- It deals better with concurrency through the introduction *"concurrent window"*;
- It seems to be more simple in computing. This simplicity will not affect its efficiency in treating the concurrent aspect of workflow.

However, the work described in this paper represents an initial investigation. In our future works, we hope to discover more complex patterns by using more metrics (*e.g.* entropy, periodicity, etc.) and enriching the workflow log. We are also interested in the modeling and the discovery of the transactional characteristics of cooperative workflows (*e.g.*, workflows composition, compensate task, roll-back, etc).

References

[AGL98] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In the proceedings of the Sixth International Conference on Extending Database Technology, pages 469-483, 1998.

[BDK99] Abraham Bernstein and Chrysanthos Dellarocas and Mark Klein, Towards adaptive workflow system, CSCW-98 workshop report, SIGGROUP Bull, pages 54--56. 1999.

[CW95] J.E. Cook and A.L. Wolf. Automating Process Discovery through Event-Data Analysis. In proceeding of the 17th International Conference of Software Engineering, pages 73-82. Association for Computer Machinery, April 1995.

[CW98] 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.

[HB97] J. Herbst and J. Bumiller. Towards Engineering Process Management Systems. In Proceedings of the Concurrent Engineering Europe Conference, pages 109–115. Society for Computer Simulation (SCS), 1997.

[He00] J. Herbst. A Machine Learning Approach to Workflow Management. In 11th European Conference on Machine Learning, volume 1810 of Lecture Notes in Computer Science, pages 183-194, Springer, Berlin, Germany, 2000.

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

[La97] P. Lawrence (editor). Workflow Handbook 1997, Workflow Management Coalition. John Wiley and Sons, New York, 1997.

[MTI97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo: Discovery of frequent episodes in event sequences. Report G-1997-15, Department of Computer Science, University of Helsinki, February 1997. 45 pages.

[VHK03] W.M.P. van der Aalst, A.H.M. Ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Distributed and Parallel Databases, 14(3), pages 5-51, July 2003.

[VWM03] W.M.P. van der Aalst , A.J.M.M. Weijters, and L. Maruster, Workflow Mining: Discovering Process Models from Event Logs. QUT Technical report, FIT-TR-2003-03, Queensland University of Technology, Brisbane, 2003.