WorkflowMiner: a New Workflow Patterns and Performance Analysis tool

Karim Baïna[1], Walid Gaaloul[2], Reda El Khattabi[1], and Abdellah Mouhou[1]

[1] ENSIAS, Université Mohammed V - Souissi,
BP 713 Agdal - Rabat, Morocco
[2] LORIA - INRIA - CNRS - UMR 7503
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
baina@ensias.ma,gaaloul@loria.fr

1 Introduction

Engineering workflow applications is becoming more and more complex, involving numerous interacting business objects within considerable processes. Analysing the interaction structure of those complex applications will enable them to be well understood, controlled, and redesigned. Our contribution to workflow patterns analysis is a statistical technique to discover workflow patterns from event-based log. Our approach is characterised by a "local" workflow patterns discovery that allows to cover partial results through a dynamic programming algorithm. Those local discovered workflow patterns are then composed iteratively until discovering the global workflow model. Our approach has been implemented within our prototype WorkflowMiner[3] that we present through this demonstration paper as follows: section 2 describes detailed WorkflowMiner design architecture, and section 3 presents a demonstration case study on which we base our tool demonstration. Theoretical aspects are to be seen in [1].

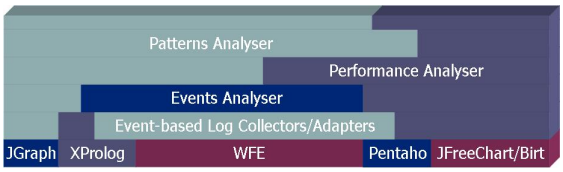

Fig. 1. WorkflowMiner Architecture

2 WorkflowMiner Design Overview

Figure 1 shows the general architecture of WorkflowMiner upon four main components: *Event-based Log Collectors/Adapters*, *Events Analyser*, *Patterns Analyser*, and *Performance Analyser*. WorkflowMiner components spirit inherits from 1^{st} order logic predicates based reasoning, multidimensional database based business intelligence, and rich visual reporting. WorkflowMiner components are built on a panel of libraries and packages which the authors have either developed or integrated into WorkflowMiner. Data flow between WorkflowMiner components is described in the figure 2.

[3] The authors wish to thank all other ENSIAS engineers contributors within WorkflowMiner.

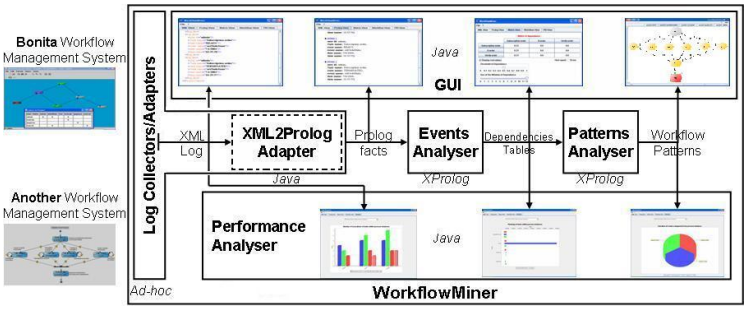

Fig. 2. WorkflowMiner Pipes and Filters Data Flow

2.1 Event-based log Collectors/Adapters

WorkflowMiner *Event-based log Collectors/Adapters* collect events and adapt them to WorkflowMiner required XML format as an ETL (Event-Transform-Load). Events come from existing Workflow engine (WFE) logs (for instance Bonita [2, 3] After collecting ad-hoc heterogeneous events (textual log lines, exchanged network messages), those events are adapted into XML structured format, and then into 1^{st} order logic Predicates Prolog form. WorkflowMiner event-based log collectors/adapters are developed using java xml parsers, ad-hoc adapters, and XProlog [4].

2.2 Events Analyser

WorkflowMiner *Events Analyser* infers, through statistical techniques, *causal dependencies* over event-based log which are based on a notion of frequency table. A *causal dependency* between two events expresses that the occurrence of an event involves the activation of an other event. While a non-causal dependency specifies other events behavioural dependency. Basically, for each event we extract from the log the following information: (i) The *overall occurrence number* and (ii) The *elementary dependencies* to previous events. Concurrent behaviour, as in patterns (like and-split, and-join, etc.), may produce interleaved events sequences. As consequence, some dependencies can indicate non-zero entries that do not correspond to ”real” dependencies. These entries are erroneous because there are no causal dependencies between these events. WorkflowMiner *Events Analyser* discover parallelism that marks these erroneous dependencies. Moreover, for concurrency reasons, an event might not depend on its immediate predecessor in the events stream, but it might depend on another ”indirectly” preceding event. To discover these indirect dependencies, WorkflowMiner *Events Analyser* uses the mechanism of *concurrent window*. A *concurrent window* is related to its last event covering its directly and indirectly preceding events. WorkflowMiner partitions the workflow log as a set of partially overlapping *window*s, then, it computes final dependencies, and adjusts *dynamically*, through the width of the *concurrent window*, the process calculating event dependencies. WorkflowMiner events analyser is developed using java xml parsers, and XProlog.

2.3 Patterns Analyser

WorkflowMiner *Patterns Analyser* uses 1^{st} order logic predicates rules to discover a set of the most useful patterns which are divided into three categories: sequence pattern, split patterns (xor-split, and-split, or-split patterns) and join patterns (xor-join, and-join and M-out-of-N-Join patterns). Workflow patterns analysis is expressed using statistical properties rules that tailor the main behaviour features of the chosen discovered patterns. We define three types of properties: sequential, concurrent and choice. Sequential and concurrent properties inherit from causal dependency. These patterns rules specify an indicator function defining as a unique manner a pattern. Indeed, each pattern has its own statistical rules which abstract statistically its causal or/and non-causal dependencies. These rules are characterised by a "local" patterns discovery. They proceed through a *local log analysing* that allows to *recover partial results* of structural workflow patterns. In fact, to discover a particular pattern we need only events relating to pattern's elements. Thus, even using only fractions of log, we can discover correctly corresponding patterns (which their events belong to these fractions). WorkflowMiner patterns analyser is developed using XProlog, and JGraph [5].

2.4 Performance Analyser

WorkflowMiner *Performance Analyser* uses adaped event-based log, discovered causal dependencies, and discovered partial and global workflow patterns to measure workflow performance metrics. Those metrics (aka key performance indicators -KPI-) measure the performance of a workflow in a transverse vision of its activities instances. The performance analyser proposes strategical versus operational indicators insuring the coherence, the coordination and the balance of the piloting at the operational level. Indicators seek to appreciate a workflow process performance according to three complementary axes: *process instance*, *activity instances*, and the *execution time*. WorkflowMiner performance analyser is developed using Pentaho [6] for OLAP multidimensional storage and browsing, and JFreeChart [7] and Birt [8] for visual reporting.

3 Demonstration Case Study

To demonstrate WorkflowMiner, we have used a process application in the banking domain. Figure 3 shows loan request processing workflow that covers 9 process activities, starting when a customer contacts the bank and finishing when the customer receives the appropriate response from the bank, either a denial or a granting of the loan. (1) The customer enters the amount and loan terms through the *Loan Terms* activity. Then, (2) the workflow instance retrieves customer information and assesses the credit worthiness through the *Credit Worthness* activity. After that, (3) the bank makes its decision choosing exclusively between these three options: either (3.1) the bank evaluates the credit risk through (3.1.1) *Risk Evaluation* activity and the value of the banks total involvement through (3.1.2) *Risk Update* activity which is performed only if the *Risk Evaluation Risk Evaluation* activity succeeds. It also implicitly assumes that the loan request will eventually be granted which is not necessarily the case ; or (3.2)

the customer may be important and have a convincing argument enbaling the loan to be granted, without risk evaluation or despite the evaluation failing. Under these circumstances an executive officer of the loan department would have to accept the risk through *Risk Acceptance* activity ; or (3.3) the bank rejects the loan through *Loan Rejection*. Then (4) the *Enter Decision* activity states the decision to either grant or reject the loan request and records the relevant information on the agreed terms of the loan in bank database. The (6) *Decision Delivery* activity (loan contract or rejection notification) cannot be executed towards the customer without the bank direction decision. Indeed, (5) a *Direction Decision* activity supervises the whole process throughout the decision process. Thus, the loan can only be granted if the supervisor agent agrees. This agent may reject the loan request even if we have a positive decision in *Enter Decision* activity. The supervisor can give freely his decision at any time during the loan process.

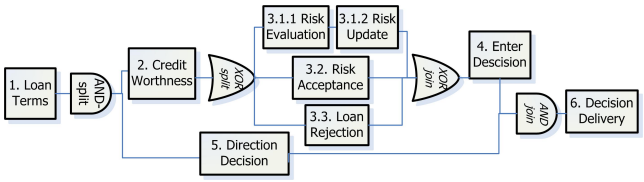

Fig. 3. Demonstration workflow example

Scenarios on complex use of WorkflowMiner will be presented in the demo to illustrate how WorkflowMiner analyses workflow patterns, and computes workflow performance KPI. We show how, from a workflow data log, WorkflowMiner can extract precious hidden intelligent knowledge in a graphical smart manner. Complete WorkflowMiner screen shots are not shown due to space reasons.

References

1. Walid Gaaloul, Karim Baïna, and Claude Godart. Towards mining structural workflow patterns. In Kim Viborg Andersen, John K. Debenham, and Roland Wagner, editors, *DEXA*, volume 3588 of *Lecture Notes in Computer Science*, pages 24–33. Springer, 2005.
2. Daniela Grigori, François Charoy, and Claude Godart. Coo-flow: A process technology to support cooperative processes. *International Journal of Software Engineering and Knowledge Engineering*, 14(1):61–78, 2004.
3. Bonita: Workflow Cooperative System. http://bonita.objectweb.org.
4. XProlog. http://www.iro.umontreal.ca/∼vaucher/XProlog/.
5. Java Graph. http://www.jgraph.com/.
6. Pentaho open source business intelligence. http://www.pentaho.org/.
7. Java Free Chart. http://www.jfree.org/jfreechart/.
8. Eclipse BIRT (Business Intelligence and Reporting Tools). http://www.eclipse.org/birt.