Incremental Workflow Mining for Process Flexibility

Ekkart Kindler, Vladimir Rubin, Wilhelm Schäfer

Software Engineering Group, University of Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany
`[kindler, vroubine, wilhelm]@uni-paderborn.de`

Abstract. *Incremental workflow mining* is a technique for automatically deriving a process model from the on-going executions of a process. This way, the process model becomes more and more accurate, and is automatically adapted when the process is being changed. Therefore, incremental workflow mining could help in flexible workflow support: In this paper, we describe a setting that combines incremental workflow mining with *gradually increasing* the control of a workflow system.
Process mining is an active research area. Most of the existing process mining algorithms need a log of the activities resp. the tasks of the process executions. In practice, however, many systems used for executing the processes are not aware of the activities – they see only the documents and how they are changed. Therefore, they do not provide activity logs. In order to make process mining available for this kind of systems, this paper improves an algorithm that identifies the activities from the accesses to documents. We call this algorithm *task mining*. In combination with our incremental workflow mining approach, task mining makes process mining available for many areas that use document management systems, version management systems or product data management systems.
Originally, our incremental workflow mining approach aimed at increasing the maturity level of software enterprises. But, the methods can be applied to any kind of processes that are supported by some kind of system that is aware of documents and document changes during the execution of the processes.

1 Introduction

Process mining [1, 2] denotes a bunch of techniques that automatically extract a process model (process type) from the execution of one or more process instances of the corresponding process. The information on the actual execution, typically, comes from the logs of some workflow management system or of some standard software. By using logs of more and more executions, the models become more and more accurate and reliable. In combination with increasing the control of the workflow system, workflow mining allows to gradually move from ad-hoc workflows without any process model to strongly structured workflows.

Some process mining techniques do not require that all executions of the processes need to be there right from the beginning. Rather, the process model is changed, once new information on the execution of some of its instances is available. This way, a process model will incrementally be changed when its executions change. Therefore, process mining is one technique for automatically achieving *process flexibility* and, in particular, incremental *process type evolution* [3, 4].

Most process mining techniques rely on the fact that there are logs of the tasks that are executed while a process is running. In our terminology, a task denotes a unit of works represented in a process model. We call an execution of a task, i. e. its instance, an activity. Therefore, we call a log of executed tasks an *activity log*. A process mining algorithm extracts the order in which the activities have been executed from the activity log and reconstructs a process model from this information. The problem, however, is that many systems that are used for executing the processes do not provide activity logs. For example, *document management systems* such as software configuration management (SCM) systems or product data management (PDM) systems or version management systems integrated in enterprise resource planning (ERP) systems are aware of the documents and the changes made; but, they are not aware of the tasks resp. activities of the underlying processes. Therefore, the *event logs* or audit information of these systems do not provide the information on activities that is needed for traditional process mining techniques.

In order to improve this situation, we had a fresh look to process mining and devised a set of algorithms for mining processes from *document versioning logs*, which we called *incremental workflow mining* [5, 6]. These algorithms allow us to mine process models from *audit information* of document management systems. Here, we will not go into the details of our approach. Rather, we will show how to obtain the information necessary for mining tasks. This information is derived from the versioning logs in combination with the model that needs to be defined anyway in order to obtain a quality certificate, i.e. the model of dependencies between documents.

In this paper, we present the task mining technique – which is the missing ingredient for completing our incremental workflow mining approach. More importantly, however, we discuss how the incremental workflow mining approach in combination with gradual workflow support can be exploited for achieving flexible workflow support.

The paper is structured as follows: We start with an overview of related work. In Sect. 3, we discuss the relation of process mining and flexible workflow support and how flexible workflow support could benefit from workflow mining. Then, we discuss which information is available in typical document management systems (Sect. 4) and how this information could be exploited for identifying the activities (Sect. 5).

2 Related Work

A lot of work has been done in the area of *process flexibility* since the 90ties [7–9]. Flexibility, dynamic process change and process evolution belong to the major research topics both in the area of business process management [4] and in the area of software processes [3]. In these areas, people distinguish between process model flexibility and process instance flexibility. This difference is also crucial for the other important research domains, such as process mining.

The research in the area of *process mining* started in the mid 90ties with new approaches to the grammar inference problem proposed by Cook and Wolf [1]. The first application of "process mining" to the *workflow domain* was presented by Agrawal in 1998 [10]. The approach of Herbst and Karagiannis [11] uses machine learning techniques for deriving the workflow models. The foundational approach to workflow mining was presented by van der Aalst et al. [12]. Within this approach, the α-mining algorithm for discovering workflow models, and its improvements are presented.

The mining-based approach for process evolution and aligning the model and the instance flexibility is researched by Weber et al. [13] and is based on adaptive process management and case-base reasoning paradigms. This approach uses activity logs and, thus, implicitly assumes the existence of a process management system, which produces such logs. In contrast to this approach, in the present paper, we start with the logs of document management system and gradually introduce the workflow management system to the company.

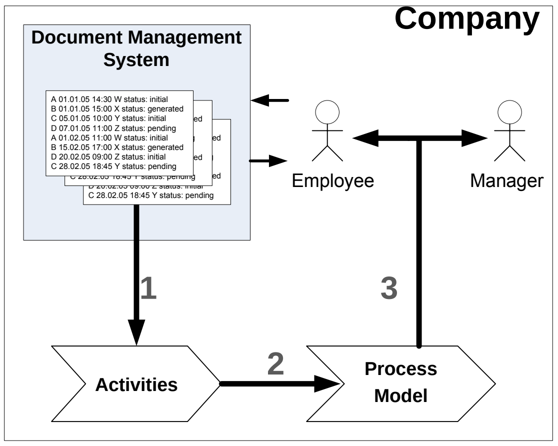
So, current process mining approaches start dealing with rich and complex sources of information about process instances. In this context, the problem of type discovering discussed in Sect. 5 becomes more and more important. Similar problems were also discovered in the other research areas, such as reverse engineering and clustering. In *reverse engineering*, people use a program runtime information (execution history) to assist building the class diagram that reflects the real implementation [14]. *Clustering* data mining techniques [15], for example, divide data into groups of similar objects (types, in our context).

3 Incremental Process Change

In this section, we describe our *incremental workflow mining* approach (for further details, see [5]) and its advantages concerning flexibility in business processes. This semi-automatic approach is used for constructing process models from the information about process instances and, thus, for keeping the models compliant to the rapidly changing instances.

The incremental workflow mining is useful for companies that utilize document management systems for a collaborative work of their employees. We use *versioning logs* of these document management systems (see Sect. 4) as a source of information about process instances. So, the approach consists of the following steps, see Fig. 1:

- First, we do *activity mining* from the versioning logs. As a result, we get a set of activities.
- Second, we take the set of discovered activities and do *reverse engineering* - derive the overall process model in a system internal formalism (Petri nets). This model contains behavioural, informational and organizational perspectives of the process.
- Third, we make the transformation from the system internal model to the external model (UML2.0 Activity Diagrams [16]). This external process model is shown to managers and employees in the company.



```
A 01.01.05 14:30 W status: initial
B 01.01.05 15:00 X status: generated
C 05.01.05 10:00 Y status: initial
D 07.01.05 11:00 Z status: pending
A 01.02.05 11:00 W status: initial
B 15.02.05 17:00 X status: generated
D 20.02.05 09:00 Z status: initial
C 28.02.05 18:45 Y status: pending
```

Fig. 1. Incremental Workflow Mining Schema

So, in the context of changing processes, the approach is aimed to fill the gap between *process type evolution* and *process instance evolution*. Our activity mining and reverse engineering algorithms provide an automatic support for redesigning the process. They use the information about deviation in the process instances for this redesign.

The approach works incrementally, i.e. as soon as new process instance is executed and, consequently, new records are added to the versioning log, we refine the set of activities and the process model derived on the previous step. Thus, starting with *revolutionary changes* in the first step, when there is no process model at all, in a consecutive manner we come to *incremental changes* in the further steps.

Following our approach, after the process models are discovered, they can be inserted to the *Workflow Management System* (WfMS), where they are maintained and executed. But the role of the WfMS and its user support evolves with the time. Thus, on the first steps, it is utilized only for storing the newly discovered models; after further refinements, when process models become more faithful, the WfMS starts advising and guiding the users in the company. This increasingly changing control of the WfMS in the company is called *gradual workflow support*. So, introducing incremental workflow mining and gradual workflow support in the company enables dealing with the flexibility in a formal and documented manner.

4 Document Management Systems and their Logs

In this section, we deal with the sources, from which the audit information on the process instances can be obtained. It can be obtained from different types of Document Management Systems: Software Configuration Management systems (SCM), Product Data Management systems (PDM), Version Management systems integrated in Enterprise Resource Planning systems (ERP). All these systems are widely used in companies working in the areas of software engineering, mechanical engineering, electrical engineering, telecommunications and others. For example, SCM systems are an essential part of modern software engineering environments and their use is stipulated in widely accepted software process improvement frameworks, such as Capability Maturity Model (CMM).

However, the problem is that these systems do not provide logs of activities, but data about the informational and the organizational perspectives of the process. In the rest of the paper, we focus on the area of software processes; but our approach is general and applicable to the other areas, where document management systems are used for collaborative work of employees in the product development process.

In spite of the fact that there is such a variety of different document management systems, their typical auditing capabilities, such as history, logging and traceability [17] produce similar results. The results differ only syntactically and since they are obtained from different systems, people are using different commands and utilities. We call these results *versioning logs* and present a general format of these logs in this section.

In order to extract this format we have looked at the versioning logs containing information about commits of documents in several SCM and PDM systems, such as: CVS [18] and Subversion (open-source file-based version management systems), Visual SourceSafe [19] (commercial filed-based version management system for small developer teams), ClearCase [20] (SCM system for large developer teams), Metaphase and Teamcenter (PDM systems).

An example of a versioning log with the information which is generally available in these systems is presented in Table 1. The log consists of *records* (rows). Each record contains information about the name of the committed document, e.g. “A”, the timestamp of the commit – “01.01.05 14:30”, the name of the user who did it – “W” and the comment of the user – “status: initial”. The versioning log contains many records related to different executions of different processes. For example, the versioning log in Table 1 contains two executions of one process, they are separated with a double line. The set of records, that belong to one execution of one process is called the *execution log*.

Table 1. Versioning Log

Document	Date	Author	Comment
A	01.01.05 14:30	W	status: initial
B	01.01.05 15:00	X	status: generated
C	05.01.05 10:00	Y	status: initial, type: manual
D	07.01.05 11:00	Z	status: pending
A	01.02.05 11:00	W	status: initial
B	15.02.05 17:00	X	status: generated
D	20.02.05 09:00	Z	status: initial, type: manual
C	28.02.05 18:45	Y	status: pending

When dealing with the versioning logs the following problems occur: which records belong to which execution logs and which execution logs belong to which process; do all the execution logs use the same naming conventions; how big are differences between timestamps of two records when they belong to the same commit, etc. In this paper, we assume that we know the structure of execution logs, naming conventions are the same in the whole log and records with different timestamps belong to different commits.

Here, we deal with the following *problem*: versioning logs do not contain information about document types, but only information about concrete names of concrete documents and sequence of their commits. The incremental workflow mining algorithms discover the behaviour of the process from such versioning logs and combine it with informational and organizational perspectives. But deriving the input and the output of activities is not fully covered by these algorithms because of the missing type information. Thus, the process model, which can be mined from the logs, can be neither reused in the other projects, nor changed by the people that were not involved in the concrete process.

5 Discovering the Types

In this section, we propose a solution to the problem described above and raise the following questions: 1. how do the informational models (document type models) look like? 2. is there an algorithm for assigning the types to concrete documents using these models?

Informational Model

One of the most important requirements for modern SCM and PDM systems is the capability of informational modelling. In the area of PDM, for example, there is a STEP (Standard for the Exchange of Product Model Data) ISO standard (ISO DIS 10303), which includes the EXPRESS language for defining the product models.

Like for versioning logs, different systems have different informational models. But there are typical relationships used in most of these models [21], for example *dependency relationship*. This relationship implies that the contents of the dependent document must be consistent with the contents of the master document.

An example of the informational model is shown in Fig. 2 as a UML class diagram. In the example, $Code$ depends on the $Design$. In our case, this is also a lifecycle dependency – the document $Code$ can appear in the system only after the $Design$ document.

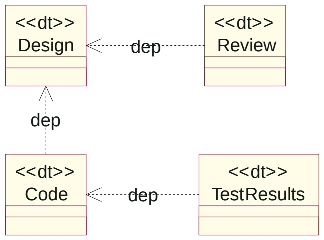


Fig. 2. Informational Model

Discovering the Types

Now, besides the versioning log with document names, we also have a set of document types and a dependency relationship on this set. Here, we propose a principal algorithm for assigning the types to the documents in the log.

Let D be a set of documents in the log with cardinality $k = |D|$, DT be a set of document types with cardinality $n = |DT|$ and $dep \subseteq DT \times DT$ be a dependency relationship on the set of document types. The type discovering algorithm is the following:

- If $k <= n$, find all possible assignments of types to the documents, i.e. find all the k-subsets on DT.
- If $k > n$, find all possible assignments of types to the documents so, that every type is assigned at least once. In this case, we will get several documents of the same type.
- For each assignment, for each execution log, check whether dependency relationship is fulfilled (so that dependent document does not appear earlier than the document it depends on). If at least one execution log exists, where dependency relationship is not fulfilled, the assignment is wrong and should be deleted.

Thus, as a result, we assign the types, which do not contradict with the dependencies, to the documents.

For the log in Table 1 and document types in Fig. 2, if we take the 4-subset {*Design, Review, TestResults, Code*} and add this type information to the document names (we use ":" notation to separate the name and the type), we get the following orders of documents in the logs: *(A:Design, B:Review, C:TestResults, D:Code)* for the first execution log and *(A:Design, B:Review, D:Code, C:TestResults)* for the second.

In the first execution log, the document of type $TestResults$ appears earlier than the $Code$ document, but it contradicts the dependency *(TestResults,Code)*, which shows that $TestResults$ is dependent on $Code$ and not vice versa. So, the selected subset is not correct. The same way, we can check all the other subsets and detect, that the only correct subsets are {*Design, Code, TestResults, Review*} and {*Design, Code, Review, TestResults*}. If we had another execution log containing additional order, for example {*A, D, B, C*}, we could see, that the only correct assignment of types is {*Design, Code, TestResults, Review*}.

The success of the type detection algorithm is dependent on the number of execution logs and the number of dependencies. If the numbers of logs and dependencies are not sufficient, we do not come to an unambiguous set of types in spite of the fact that we are checking all the possible type permutations. In this case, we need interaction with a user, who has to give us the types of some documents.

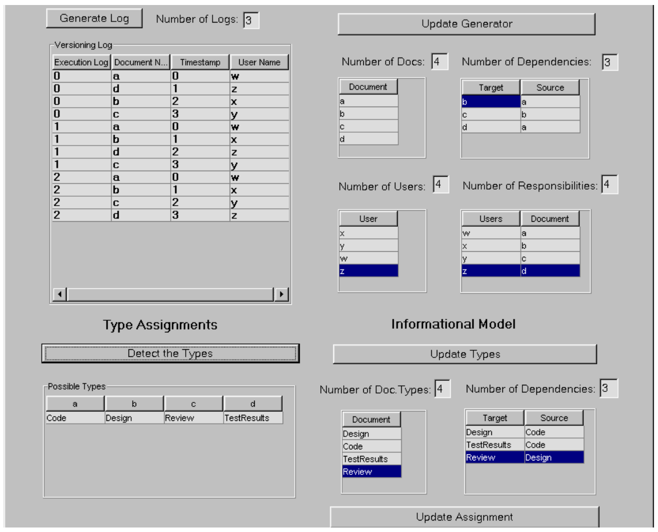
6 Conclusion and Future Work

In this paper, we have presented the advantages of the *incremental workflow mining* approach and *gradual workflow support* in the context of flexible processes. This semi-automatic approach uses information about the changing process instances and derives the process model compliant to them. The information about instances is obtained from the logs of document management systems. We have also looked at different document management systems and proposed a uniform format for these logs.

In the second part of the paper, we have presented the problem of *discovering the types* of documents in the logs of document management systems and proposed a solution for it. This solution is used to augment the incremental workflow mining approach with information about tasks and, therefore, to enable *task mining.*

In order to deal with this problem, we need other information in addition to the versioning logs. This information contains the types of documents and dependencies between them. After executing the type discovering algorithm and possibly interacting with the user, we get the assignment of types to the documents in the log. The type discovering algorithm was implemented in our research prototype and tried out on several examples from the software process domain, see Fig. 3.

Neither the advantages of incremental process mining techniques in the context of changing processes, nor the problem of discovering the types were discussed in details in the areas of process flexibility and process mining. But using



Number of Logs: 3

Versioning Log

Execution Log	Document N...	Timestamp	User Name
0	a	0	w
0	d	1	z
0	b	2	x
0	c	3	y
1	a	0	w
1	b	1	x
1	d	2	z
1	c	3	y
2	a	0	w
2	b	1	x
2	c	2	y
2	d	3	z

Number of Docs: 4

Document
a
b
c
d

Number of Dependencies: 3

Target	Source
b	a
c	b
d	a

Number of Users: 4

User
x
y
w
z

Number of Responsibilities: 4

Users	Document
w	a
x	b
y	c
z	d

Type Assignments

Possible Types

a	b	c	d
Code	Design	Review	TestResults

Informational Model

Number of Doc.Types: 4

Document
Design
Code
TestResults
Review

Number of Dependencies: 3

Target	Source
Design	Code
TestResults	Code
Review	Design

Fig. 3. Research Prototype

new process management and new support systems and obtaining more and more complex multi perspective log information there makes the stated problems more and more prominent.

So, the future research in this area must deal with the user role in incremental workflow mining and in guiding process flexibility in the company. Since the input information for the algorithms is defined manually and the logs can contain noise or be incomplete, the role of interactive type detection and mining, which lead to unambiguous solutions, will become more and more important.

References

1. Cook, J.E., Wolf, A.L.: Discovering Models of Software Processes from Event-Based Data. ACM Trans. Softw. Eng. Methodol. **7** (1998) 215–249
2. van der Aalst, W., van Dongena, B.F., Herbst, J., Marustera, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering **47** (2003) 237–267
3. Bandinelli, S.C., Fugetta, A., Ghezzi, C.: Software Process Model Evolution in the SPADE Environment. IEEE Transactions on Software Engineering **19** (1993) 1128–1144
4. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow Evolution. In: International Conference on Conceptual Modeling / the Entity Relationship Approach. (1996) 438–455
5. Kindler, E., Rubin, V., Schäfer, W.: Incremental Workflow mining based on Document Versioning Information. In Li, M., Boehm, B., Osterweil, L.J., eds.: Proc. of the Software Process Workshop 2005, Beijing, China. Volume 3840 of LNCS., Springer (2005) 287–301

6. Kindler, E., Rubin, V., Schäfer, W.: Activity mining for discovering software process models. In Biel, B., Book, M., Gruhn, V., eds.: Proc. of the Software Engineering 2006 Conference, Leipzig, Germany. Volume P-79 of LNI., Gesellschaft für Informatik (2006) 175–180
7. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COCS '95: Proceedings of conference on Organizational computing systems, New York, NY, USA, ACM Press (1995) 10–21
8. Reichert, M., Dadam, P.: A framework for dynamic changes in workflow management systems. In: DEXA '97: Proceedings of the 8th International Workshop on Database and Expert Systems Applications, Washington, DC, USA, IEEE Computer Society (1997) 42–48
9. Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., Teschke, M.: A comprehensive approach to flexibility in workflow management systems. In: WACC '99: Proceedings of the international joint conference on Work activities coordination and collaboration, New York, NY, USA, ACM Press (1999) 79–88
10. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Proceedings of the 6th International Conference on Extending Database Technology, Springer-Verlag (1998) 469–483
11. Herbst, J., Karagiannis, D.: An Inductive approach to the Acquisition and Adaptation of Workflow Models. citeseer.ist.psu.edu/herbst99inductive.html (1999)
12. Weijters, A., van der Aalst, W.: Process mining: discovering workflow models from event-based data. In: Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001). (2001) 283–290
13. Weber, B., Reichert, M., Rinderle, S., Wild, W.: Towards a framework for the agile mining of business processes. In Bussler, C., Haller, A., eds.: Business Process Management Workshops: BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005. Revised Selected Papers. Volume 3812 of LNCS., Springer (2006) 192–202
14. Guéhéneuc, Y.G., Douence, R., Jussien, N.: No Java without Caffeine: A Tool for Dynamic Analysis of Java Programs. In: ASE '02: Proceedings of the 17th IEEE international conference on Automated software engineering, Washington, DC, USA, IEEE Computer Society (2002) 117
15. Berkhin, P.: Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA (2002)
16. OMG: UML 2.0 Superstructure Specification. Version 2.0 ptc/03-08-02, Object Management Group (2003) Final Adopted Specification.
17. Frauf, K., Zeller, A.: Software configuration management: State of the art, state of the practice. In: 9th International Symposium on System Configuration Management (SCM-9). (1999)
18. Fogel, K.F.: Open Source Development with CVS. Coriolis Group Books (1999)
19. Microsoft: Visual SourceSafe. Web: http://msdn.microsoft.com/vstudio/previous/ssafe/ (2003)
20. Rational Software Corporation: Rational ClearCase Rational ClearCase LT. Technical Report 800-026160-000, Rational Software Corporation (2003) Version: 2003.06.00 and later.
21. Conradi, R., Westfechtel, B.: Version models for software configuration management. ACM Comput. Surv. **30** (1998) 232–282