Towards Process Instances Building for Spaghetti Processes

Claudia Diamantini[1], Laura Genga[1], Domenico Potena[1], and Wil M.P. van der Aalst[2]

[1] Information Engineering Department
Università Politecnica delle Marche
via Brecce Bianche, 60131 Ancona, Italy
{c.diamantini,l.genga,d.potena}@univpm.it
[2] Faculty of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{w.m.p.v.d.aalst}@tue.nl

Discussion Paper

Abstract. *Process Mining* techniques aim at building a process model starting from an event log generated during the execution of the process. Classical process mining approaches have problems when dealing with *Spaghetti Processes*, i.e. processes with little or no structure, since they obtain very chaotic models. As a remedy, in previous works we proposed a methodology aimed at supporting the analysis of a spaghetti process by means of its most relevant subprocesses. Such approach exploits graph-mining techniques, thus requiring to reconstruct the set of process instances starting from the sequential traces stored in the event log. In the present work, we discuss the main problems related to process instances building in spaghetti contexts, and introduce a proposal for extending a process instance building technique to address such issues.

Keywords: Process Mining; Spaghetti Processes; Process Instances Building

1 Introduction

Process Mining (PM) discipline involves a set of techniques aimed at extracting a process schema from an event log generated during process execution [1]. Such techniques turned out to have several problems when addressing the so-called *Spaghetti Processes*, i.e. processes with little or no structure, where the flow of the activities mostly depends on the decisions taken by the involved people; indeed, for these processes PM techniques usually obtain very messy and intricate models, called *spaghetti models*, almost incomprehensible for a human analyst.

As a remedy, in previous works [2], [3] we introduced a methodology aimed at supporting the analysis of spaghetti processes by extracting their most relevant patterns, i.e. subprocesses, since we expect them to provide some interesting insights about the overall process. Such approach exploits a graph-based technique; more precisely, it requires to transform the event log of the process in a set of graphs, each of them describing the execution of a certain process instance, then extracting the subprocesses from these graphs. However, such conversion is not trivial, since events in the log are stored according to their temporal order of occurrence, thus hiding possible parallelisms, which instead should be explicitly represented when modeling a process instance. Few approaches have been developed to face the process instances building issue; furthermore, they present some limits when addressing spaghetti processes, either because they require specific domain knowledge or because they obtain too general models. In the present work, we discuss the problems related to process instances building in spaghetti contexts, and outline a proposal for extending a process instance building technique to deal with such issues.

The rest of this work is organized as follows. Section 2 provides a brief overview of related works; section 3 introduces some definitions used throughout the paper; section 4 describes the process instances building procedure, with its extension; finally, section 5 draws some conclusions and delineates future works.

2 Related Works

Previous works addressing process instances building can be grouped in two main classes. For the first group, we can mention for example tools like ARIS Process Performance Manager [5], which allows for the visualization, the analysis and the aggregation of process instances, represented as *instance EPCs*, where *EPC* is the "Event Driven Process Chain", a common process modeling formalism (see e.g. [1]). Such techniques, however, require a well-defined process model, from which the instances are derived. Instead, techniques of the second group use only the event log, trying to derive the ordering relations of the events, and then using such relations to build every single instance from the corresponding trace. For example, [6] proposes a distinction between *Data Annotated Logs* (DALs) and not annotated event logs. A DAL is an event log in which each event has a set of "input" and a set of "output" attributes, i.e. attributes which are respectively read or written by the corresponding activity. Ordering relations can be derived by the data-flow; in fact, if an event e_j has as input an attribute a generated by e_i, clearly e_i has to precede e_j in every process instance. However, the generation of a DAL requires either a dedicated event log generation system, or the log preprocessing by a domain expert. In the case of a not annotated event log, [6] suggests to exploit domain knowledge about the process to establish its ordering relations. A more general approach is proposed in [4], which derives the ordering relations by means of a PM technique, without requiring neither special data stored in the log, nor any additional knowledge about the process. However, it is known to have several issues when applied to unstructured processes.

Another work related with our proposal is [7], which deals with the *process repair* topic. Given an event log and a process model which does not conform to the log (i.e., which is not able to describe all the traces in the log), the goal is generating a new model which conforms the event log. In [7], however, the focus is on repairing the overall process model, while our work is aimed at repairing every single process instance.

3 Definitions

In this section, some definitions used throughout the paper are provided, starting with those related to the notion of "process".

Process, Process Instance. A *process* consists in a set of activities which have to be performed by some actors to reach a certain goal. Every time a process is executed, a *process instance* is generated.

As an example, the management of a loan application in a financial institute is a process involving activities like the *submission* of an application performed by a customer, the *approval* of the request decided by some manager and so on. Different applications are managed in different process instances. A process can be represented by means of a *Process Schema*, defined as:

Process Schema. A *process schema* is a schema representing the control-flow of the performed activities, i.e. their ordering relations.

The main elements of a process schema are activities, sequences of activities (SEQ), parallelization (SPLIT) and merging (JOIN) operators. SPLIT and JOIN operators are characterized by the kind of synchronization of flows; hence, for example, a SPLIT-AND means that the end of an activity starts all the linked activities, while in a SPLIT-XOR only one will be executed. Many different process modeling formalisms exist; as an example, Figure 1 shows a simple process schema describing a possible loan process exploiting the BPMN standard[3]. The process starts when a customer does a *Submit Application* (SA) activity, followed by both the activities *Check Financial Situation* (CFS) of the applicant and *Contact the applicant* (CA) for additional information, performed by one or more employees. When both the activities have been completed, a manger can proceed with the *Analyze documents* (AD) activity, to decide whether refusing or approving the request, thus terminating the process. Note that the last two activities of the model (i.e., *Refuse Application* (RA) and *Approve Application* (AA)) are involved in a XOR construct, since only one of them can be executed for each process instance.

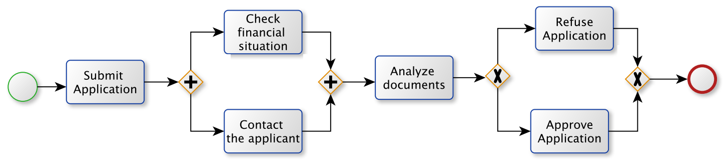


Fig. 1: A BPMN model for a loan process.

[3] http://www.bpmn.org/

Figure 2a shows a possible process instance for the loan process, where the application has been approved. Note that the choice constructs, i.e. in this example the XOR operator, are not represented. Indeed, also if the process involves alternative paths, only some of these paths will be executed within an instance.

Event, Trace, Log. An *event* is the execution of a certain activity, described by a set of event *attibutes*. A *trace* is a finite sequence of events $\sigma \in \Sigma$, generated during the execution of a certain process instance. An *event log* is a collection of traces, such that each event appears at most once in the entire event log.

An example of trace which can be generated by the process of Figure 1 is $t_1 = \{SA, CFS, CA, AD, AA\}$[4]. It is noteworthy that in the spaghetti processes cases, usually one does not have a schema as simple as the one represented in Figure 1. In these contexts, it is hard to detect the main process behavior; moreover, also if one is able to actually detect a main behavior, typically a lot of exceptions in the event log exist. As an example, let us assume that the process in Figure 1 actually represents the main behavior for a loan process, but the actors are not strictly forced to follow the proposed structure, and can deviate on the basis of their needs. Table 1 represents a possible log for our example.

InstanceId	**Events**
1	*<SA, CFS, CA, AD, AA>*
2	*<SA, CA, CFS, AD, AA>*
3	*<SA, CFS, AD, CA, AA>*
4	*<SA, CFS, CA, AA>*
5	*<SA, RA>*

Table 1: Event log for the loan process of Figure 1.

While the first two traces follow the proposed schema, the others show process instances where the applicant has been contacted only shortly before approving the application (i.e., line 3), instances in which a manager approved an application without analyzing the relative documents (i.e., line 4) and where an application is rejected by default, without actually performing the process (i.e., line 5). These traces are called *anomalous* with respect to our model, since they represent an ordering of occurrence of events which is not allowed by such model.

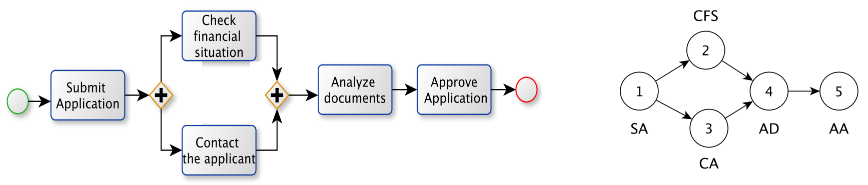
4 From Event Log to Process Instances

Our goal consists in building for each trace in an event log the corresponding process instance, describing the execution path of the performed activities. Such a task is actually not trivial, since possible parallelisms between events are hidden in the sequential traces. As an example, if we consider only the first trace in Table

[4] For the sake of simplicity, attributes which uniquely identify the events are omitted.

1 we cannot derive that the activities CFS, CA have been executed in parallel, unless having at disposal the process model, which is generally not the case. To address this issue, here we refer to the approach proposed by [4], which is developed in two steps. First, it extracts the set of causal dependencies from the event log by means of a PM technique, i.e. the α-*algorithm* [10]. Then, it uses such dependencies to define for each trace a so-called *Instance Graph* (IG), that is a graph representing a process instance. An IG involves a node for each entry in the event log; two nodes are linked by an edge only if a causal dependency between their corresponding activities is defined. [4] proposes an iterative procedure for building the IGs set. For each trace, it analyzes each pairs of events $(e_i,\, e_j)$ with $i < j$, such that a causal dependency exists between them (i.e., between the activities they represent), and links the corresponding nodes in the graph only if between these events there is not neither another causal successor of e_i, nor a causal predecessor of e_j. In other words, this procedure links each event only with its closest causal successors. As an example, let us assume we want to build the IG for the first trace in Table 1, by using the dependencies shown in Figure 1 (e.g. $SA \rightarrow CFS$, $CFS \rightarrow AD$, etc.). We start from the pair (SA, CFS); a causal dependency exists and there is not any other event between them. Therefore, we can link the corresponding nodes in the graph. Then, we consider (SA, CA); also in this case a causal dependency $SA \rightarrow CA$ exists, but we have CFS between them, that is a causal successor of SA. However, there is not any causal predecessor of CA, so we can anyway link the nodes corresponding to SA and CA. By analyzing in this way the entire trace, we obtain the corresponding instance graph, shown in Figure 2b. Note that, since an IG explicitly represents parallelisms, it can describe a set of traces, rather than a single one; as an example, the IG we built can correspond both to the first and the second trace in the event log.

The discussed procedure presents some limits when applied to a spaghetti process; such problems are discussed in the following subsection, where it is also outlined our proposed extension to overcome them.

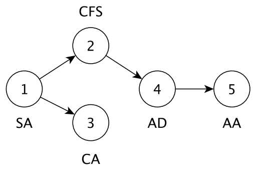


(a) (b)

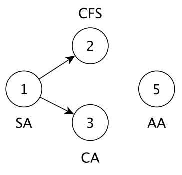
Fig. 2: Example of an instance for the loan application process (a) and its corresponding instance graph (b).

4.1 Dealing with noise

The α-algorithm exploited by [4] is known to have problems in dealing with spaghetti processes. This technique recognizes a causal dependency between two events a, b if and only if a occurred before b, and b never occurred before a in the log; otherwise, they are considered parallels. In the presence of exceptional behaviors, some problems arise. As an example, from the log in Table 1 it returns a parallelism between CA and AD that is clearly undesired. In a spaghetti context, with such a logic we can expect to derive a very imprecise model, with the result that the IGs built by using its CR set will likely involve too many parallelisms, thus significantly overgeneralizing the process behavior. To limit overgeneralization, some authors [8] proposed the generation of several schemas, each representing a process variant, by clustering log traces. Here we adopt a different approach, based on deriving the CR set using filtering techniques. Filtering techniques are a kind of PM techniques aimed at evaluating the relevance of each detected dependency on the basis on some heuristic (e.g. frequency), to build the final model by using only the most relevant dependencies. A well-known example of filtering technique is the *Heuristic Miner* [9], aimed at extracting only the most frequent dependencies; clearly, the graphs built by using only such dependencies provide a much more precise description of the process than the ones built by the α-algorithm. However, the model obtained by a filtering technique cannot describe all the traces in the event log. Hence, we will have a subset of anomalous traces, for which the IG building procedure obtains again overgeneralizing graphs. As an example, let us consider the third and fourth traces in Table 1, hereafter indicated as t_3 , t_4 respectively. In t_3, the activity CA occurred in a not allowed position, while in t_4 the activity AD did not occur. Figure 3 shows the IGs obtained for such traces, i.e. IG_3 and IG_4.



(a) IG_3



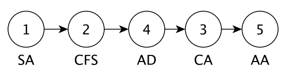
(b) IG_4

Fig. 3: Instance graphs for the anomalous traces t_3 and t_4.

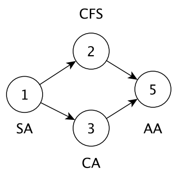
Let us consider first IG_3. Such graph introduces a significant overgeneralization, since the only execution constraint for CA is to be performed after SA; hence, it can occur in every order with respect to all the other events. Consequently, IG_3 can generate many other traces besides t_3, providing a poor representation for the corresponding process instance. The reason for such an outcome is that the IG building procedure exploits the regular CR set to build

IG_3, while, being CA occurred in a not allowed position, those relations do not hold for it. As regards IG_4, due to the deletion of the activity AD the node representing the activity AA is not linked to anything, which means that such activity is considered parallel to all the others.

To deal with mentioned issues, we propose an extension of [4], aimed at *repairing* IGs corresponding to anomalous traces, i.e. transforming them in graphs capable of representing also the anomalous traces without overgeneralizing. To this end, first we need to recognize which are the anomalous traces in the event log. This can be done by means of a *conformance checking* technique, i.e. a technique devoted to check whether or not an event log fits with a given process model [11]. More precisely, we use the approach proposed in [12], which checks if it is possible to *replay* every trace t of the event log in a given process model (typically a Petri Net), i.e. if exists a mapping between every event occurred in t and a transition fired in the net. If t involves one or more anomalies, the reply fails. Among the several possible mappings between t and the net, [12] returns the one involving as few anomalies as possible. Such a mapping allows us to identify the set of IGs to repair, with precise indications about the positions and the kinds of the anomalies. Then, we can proceed with the repairing, applying tailored rules for deleted and inserted events. For deleted events, the repairing consists in identifying the disjunction points of the graph and properly connecting them. For the insertion repairing, we have to change the edges connecting the nodes corresponding to the event(s) before and the event(s) after the inserted event, to connect such nodes with the node corresponding to the inserted event in the graph. Figure 4 shows the repaired graphs corresponding to IG_3, IG_4.



(a) Repaired IG_3



(b) Repaired IG_4

Fig. 4: Repaired graphs for anomalous traces t_3, t_4

We performed some preliminary tests on real event logs to validate our approach, which cannot be presented here for the lack of space. Preliminary results are promising, since we were actually able to build process instances also for less structured processes, whose event logs involved quite heterogenous traces.

5 Conclusion and Future Works

In the present work we addressed the issue of building process instances from sequential traces stored in the event log in complex spaghetti contexts, where it is hard to detect the ordering relations of the activities in the process. We discussed about limits of previous approaches, and introduced an extension of an IG building technique, aimed at overcoming such limits. Preliminary results suggest to explore further in this direction. In particular, we plan to complete and extend the present experimentation.

In future works, we also plan to test other filtering PM approaches, to evaluate how the application of different techniques can affect results. Furthermore, we intend to apply the extended IG building procedure to the methodology proposed in [2], to exploit such instances to extract meaningful subprocesses.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Diamantini, C., Genga, L., Potena D., Storti, E.: Patterns discovery from innovation processes. In: 2013th International Conference of Collaboration Technologies and Systems, pp. 457-464. IEEE (2013)
3. Diamantini, C., Genga, L., Potena, D., Storti, E.: Discovering Behavioural Patterns in Knowledge-Intensive Collaborative Processes. In: Appice, A., Ceci, M., Loglisci, C., Manco, G., Masciari E., Zbigniew W. (eds.) New Frontiers in Mining Complex Patterns, LNCS, vol. 8983, pp. 149-163. Springer International Publishing (2015)
4. van Dongen, B., van der Aalst, W. : Multi-phase Process mining : Building Instance Graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T. (eds.) Conceptual Modelling-ER 2004, LNCS, vol 3288, pp. 362-376. Springer Berlin Heidelberg (2004)
5. Markus Fischer: Aris process performance manager. In: 14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems. (2008)
6. Lu, X., Fahland, D., van der Aalst, W.: Conformance checking based on partially ordered event data. In: 10th International Workshop on Business Process Intelligence. (2014)
7. Fahland, D., van der Aalst, W.: Model repair - aligning process models to reality. Information Systems 47, 220-243. (2015)
8. De Medeiros, A. , Guzzo, A., Greco, G., van der Aalst, W., Weijters, A., van Dongen, B., Saccà, D.: Process mining based on clustering: A quest for precision. In: ter Hofstede, A., Benatallah, B., Paik, H. (eds.) Business Process Management Workshops, LNCS, vol 4928, pp. 17-29. Springer Berlin Heidelberg (2008)
9. Weijters, A., van der Aalst, W., De Medeiros, A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP 166, pp 1-34. (2006)
10. van der Aalst, W., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16, 1128-1142. (2004)
11. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems 33 (1), 64-95. (2008)
12. Adriansyah, A., Boudewijn, F., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC), pp. 55-64. IEEE (2011)