Applying graph grammars for the generation of process models and their logs

Valeriia Kataeva
School of Software Engineering
Software Management Department
NRU – Higher School of Economics
Moscow, Russian Federation
lerileri25@gmail.com

Dr. Anna A. Kalenkova
School of Software Engineering
Software Management Department
NRU – Higher School of Economics
Moscow, Russian Federation
akalenkova@hse.ru

***Abstract* - This work is dedicated to one of the most urgent problems in the field of process mining. Process mining is a technique that offers plenty of methods for the discovery and analysis of business processes based on event logs. However, there is a lack of real process models and event logs, which can be used to verify the methods developed to achieve process mining goals. Hence, there is a need in an instrument that would generate process models and logs, thus allowing verification of the process mining discovery algorithms. This aim can be reached by the creation of a model and log generator.**

In this paper a possible solution for the creation of such a generator will be proposed. Namely, it is the generation of process models and event logs using the rules of graph grammars on the example of structured workflow nets. The approach proposed is based on the creation of grammar rules to generate a model and an event log, which fits this model. The evaluation of the process discovery algorithms will be available due to the presence of initial models and event logs generated on the basis of these models. The tools used to perform this work are publicly available.

This paper is the research-in-progress, which is conducted in frame of master's thesis in the field of software engineering.

Keyword: process mining, discovery algorithms, conformance checking, graph rewriting rules, graph grammar, event logs.

I. INTRODUCTION

Process mining [1] is a process management technique that allows the analysis of business processes based on event logs. The basic idea is to extract knowledge from event logs recorded by an information system. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs. Moreover, process mining is an approach to compare the analyzed events with preferred or predefined models or rules. The key point here is that the model has to be evaluated according to the criteria of how well it matches to the real-life process. This evaluation requires as many as it is possible event logs.

Event logs can be used to conduct three types of process mining:

- *Discovery*. A discovery algorithm takes an event log and produces a model. This can be demonstrated on the example of α-algorithm [1]. The algorithm takes an event log and produces a Petri net explaining the behavior recorded in the log.
- *Conformance checking*. In this method, an existing process model is compared with an event log (or with a model) of the same process. The conformance checking is used to check whether information recorded in the log (or in the model) corresponds to the model discovered.
- *Enhancement.* The idea here is to extend or improve existing process model using additional information about the process recorded in the event log.

The area of our research is presented in Figure 1. First, we will generate an initial model, as far as there is a lack of real examples from the business. After, we will extract logs from the model. The logs, further, will be used for applying discovery algorithms and hence a creation of a new model. Finally, the initial and a new model will serve as an input data for conformance checking.

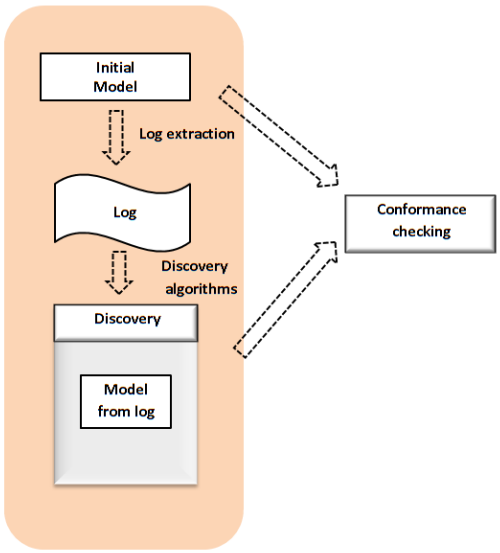


Figure 1. The graphical representation of research area.

We will start the research with the problem definition. The problem is that only a small amount of logs are available. This is caused by the fact that many industries are uncomplying to make their private data public. And this appears to be a serious obstacle for the reconstruction and developing more effective process discovery algorithms.

In this paper, we will present a possible solution to the problem stated above. The solution is based on the GROOVE – a graph transformation tool set, which allows for creating and applying graph grammars [7].

In this work we will use workflow nets (WF-nets). WF-nets are the subclass of Petri nets. *A Petri net is a triple* $(\mathrm{P}, \mathrm{T}, \mathrm{F})$ *:*

- P *is a finite set of places,*
- T *is a finite set of transitions, such that* $\mathrm{P} \cup \mathrm{T} = \varnothing$ *,*
- $\mathrm{F} \subseteq (\mathrm{P} \times \mathrm{T}) \cup (\mathrm{T} \times \mathrm{P})$ *is a set of arcs (flow relation).*

A place p is called an input place of a transition t iff there exists a directed arc from p to t. Place p is called an output place of transition t iff there exists a directed arc from t to p. $\bullet t$ is used to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing $p\bullet$ as an input place.

At any time a place contains zero or more tokens, drawn as black dots. The state, often referred to as marking, is the distribution of tokens over places, i.e. $M \in P \to N$. The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following firing rule:

(1) A transition t is said to be enabled iff each input place p of t contains at least one token.

(2) An enabled transition may fire. If transition t fires, then t consumes one token from each input place p of t and produces one token for each output place p of t [3].

A Petri net $\mathrm{PN} = (\mathrm{P}, \mathrm{T}, \mathrm{F})$ *is a WF-net (Workflow net) iff:*

(i) PN *has two special places: i and o. Place i is a source place, such that* $\bullet i = \varnothing$ *. Place o is a sink place, such that* $o\bullet = \varnothing$ *.*

(ii) If the transition t^* *is added to* PN *, which connects place o with i (i.e.* $\bullet t = \{o\}$ *and* $t\bullet = \{\mathrm{i}\}$ *), then the resulting Petri net is strongly connected.*

The second requirement (ii) (the Petri net extended with t^* should be strongly connected), states that for each transition t (place p) there should be directed path from place i to o via t (p). This requirement has been added to avoid dangling nodes, i.e. tasks and conditions which do not contribute to the processing of cases [2].

Business processes in the particular sphere or a company can be formalized via WF-nets, which define their semantics. The WF-net specifies a set of tasks required to process the business cases. Also, it defines the order in which these tasks have to be executed. However, as it was already mentioned, there is an urgent lack of the models and event logs that can be analyzed according to the reluctance of the companies.

To piece out the lack of such model graph grammar rules can be applied. Graph rewriting technique is one that allows creating a new graph out of an original graph algorithmically. The definition of grammar is based on well-known process patterns, particularly in this case, patterns for WF-nets [8]. The general idea is to use the basic patterns for the generation of the process via grammar. Note that an approach for generating models using grammars was already presented in [6]. The main advantage of the approach presented in this paper is that we use an external tool to generate models, which allows working with arbitrary graph grammars. Thus, we are not bounded to the concrete processes models. Moreover, we propose an approach for a log generation based on graph grammars as well.

II. GRAPH GRAMMAR

Graph grammars are used for graphs generation. The grammar is specified by a start graph and a set of production rules. The aim of production rules is to replace one part of a graph by another (these parts of graphs are highlighted in blue and green respectively in the figures below) [5]. Moreover, as it will be seen from the examples below, each production rule is applicable under the specified conditions, which take into account the types of nodes. These conditions could be also formalized and each node can modify its attribute value according to the rules [4]. Here we would like to show an example of the generation of a structured WF-net [3], which could be defined as a hierarchy of subprocesses, based on a graph grammar. First, an initial or start graph was set and a transition counter was initialized. This is demonstrated in Figure 2.

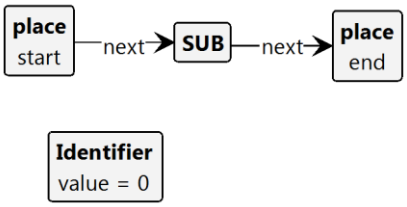


Figure 2. Start graph for applying graph grammar.

According to this image, there are two nodes of type *place* that denote the beginning and the end of the model. The node of type *SUB(* the *SUB block*) is defined as the subgraph that has to be modified according to the grammar rules. Below the graph in Figure 2 there is a node of type *Identifier* that is used for a transition identifier generation. Initially, we've put zero number.

Further, the rules applying for the graph generation were created. They contain four rules. They are:

- Transition
- Sequence

- AND-joint
- XOR-joint

R1. The rule removes the *SUB block* and sets the transition. Meanwhile, the number for the transition is incremented. The identifier for the transition is put into the newly created container which is connected with the particular, newly created transition (Figure 3).

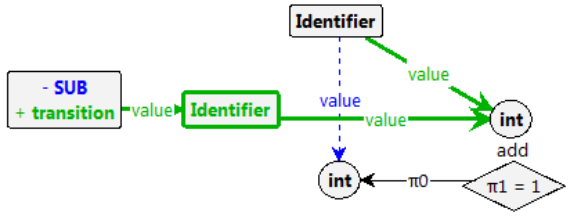


Figure 3. Rule#1 for generation WF-net.

R2. This rule replaces the *SUB block* according to the following rule that is demonstrated in Fig.2. The rule creates the sequence of nodes with the types *SUB*, *place*, *SUB*. Further, other rules can be applied to the *SUB block* .

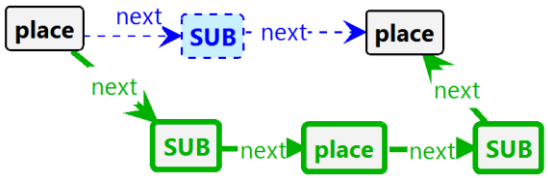


Figure 4. Rule#2 for generation WF-net.

R3. The rule is used for the replacement of *SUB block* with AND-joint combination (Figure 5).

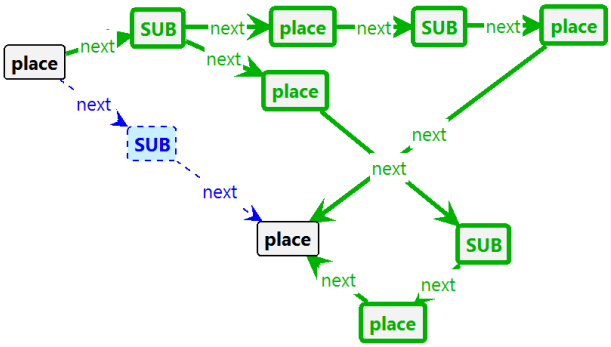


Figure 5. Rule#3 for generation WF-net.

R4. The last rule is used for the creation of OR-joint (Figure 6).

In this chapter, we have demonstrated the key principles of applying graph grammar for model creation. Note that more rules for the expansion of nodes with type *SUB* can be added, such as loop, inclusive join and others.

However, our main aim is not only about to create a model. Model is just a raw material. Further we have to extract the execution log from this model in order to apply a discovery algorithm.

The idea of a log generation is presented in the next chapter.

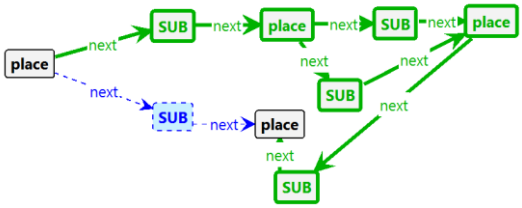


Figure 6. Rule#4 for generation WF-net.

III. LOG GENERATION

As it was already mentioned event logs allow analyzing, detecting problems and finding the solutions for process optimization.

Let *A be the set of activities, which could be recorded in the event log, then the set of pairs (or records)* $\sigma \subseteq A \times T$ *, such that there are no two pairs with the same timestamp, where T is a set of timestamps, denotes a trace. An event log* $L \in \mathrm{B}(A^*)$ *is a multiset of such traces.*

So, every trace in a real-life event log is considered a set of event identifiers and corresponding timestamps.

After creating a model, containing no nodes, which could be expanded, as it was demonstrated above, we use this model as a start graph for the log generation. The start graph is pictured in Figure 7.

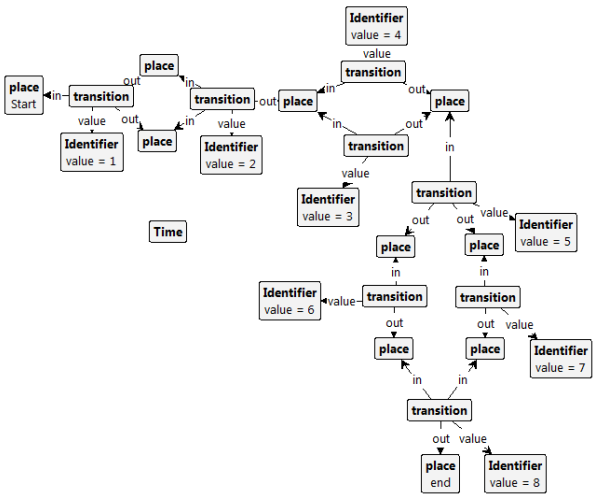


Figure 7. Start graph for the Petri net.

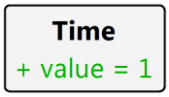
By moving a token through the model each trace is to be built. Turning to the same example of a WF-net, we will put a token on the start position in an attempt to keep track on what transition is executed, when we move.

In order to capture the log a time node is created, it serves as a counter, so that after the execution of every transition the value of its attribute is incremented.

For this realization the following rules were created:

- Time initialization
- Putting token on start position
- Firing rule

Time initialization. This rule as a default is executed first. It sets the value of the counter as 1 (Figure 8).



Figire 8. Time initialization rule.

Putting token on start position. The next rule puts an initial token on the start position. This is needed because the start graph does not contain it (Figure 9).

Figure 9. Initialization of token on start position.

Firing rule. This rule moves tokens according to their positions and quantity, meanwhile creating nodes with the time and identifier of the executed transition (Figure 10).

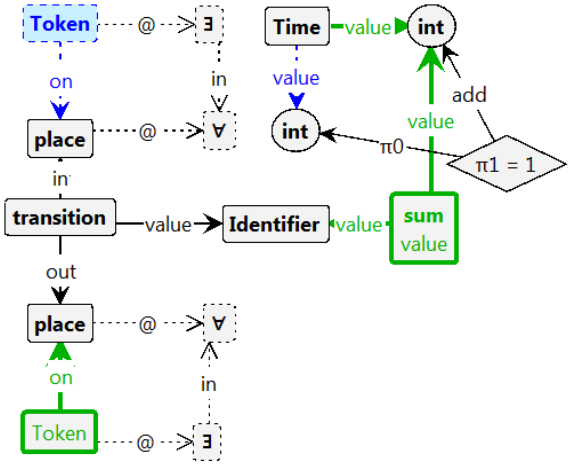


Figure 10. Firing rule for Petri Net.

After applying of firing rules and creating the final, already-executed model we can obtain the set of elements with the time and identifier of every transition fired. Namely this set can further be used as a trace.

The log created due to applying the rules has to be converted into the proper format. The proper format depends on the software that is planned to be used for the log analysis. According to this, the next chapter will describe the destination software and give additional information on the realization of the general idea.

IV. Applying the Approach in Frames of Process Mining

In this section, we will put the main technical principles that were used for the creation of the graph grammars and the description of software. Then, we will present an idea of how these grammars can be applied for the possible solution for extracting logs [6] from the models generated and how this approach can be applied for the integration with another framework that supports a wide variety of process mining techniques in a form of plug-ins.

First we will start with the technology and software that were used for graph grammar creation. For this purpose we have chosen the tool called Groove Simulator [7].

The GROOVE tool is an instrument that is aimed on the use of simple graphs for modelling different structures of object-oriented systems and graph transformations as a basis for model transformation and operational semantics. The GROOVE tool contains:

• an editor for creating graph production rules;

• a simulator for visualization of the graph transformations;

• a generator for automatic search of state spaces and a model checker.

The GROOVE tool set was used to create WF-net generation rules, which were demonstrated above. Using the generator it became possible to produce a numerous quantity of models for the further log extraction. The WF-nets were obtained using different exploration strategies that are predefined in GROOVE and rules priorities as well. However, in the majority of cases the Random strategy was used. This strategy allows applying all the rules with an equal probability of 50 %. Other strategies were tested as well in an experimental mode.

A grammar for a log generation from a given WF-net was created. The generation of a log was constructed in such a way that after the execution of every firing rule a new node, containing time and event identifier was created.

All the generated WF-nets and corresponding event logs were saved in one of the XML-formats for the further integration with ProM tool [1].

ProM is an open source Java- framework that offers a variety of process mining techniques, which are represented by the plug-ins. Currently, ProM supports import of Petri nets and event logs in specially developed XML-formats.

It is planned to implement the integration with ProM on the basis of XML-documents conversion. One possible and more probable solution for the integration is the utilization of Extensible Stylesheet Language Transformations (XSLT) - language XML-documents transformations. Based on this the plug-in for ProM that will allow importing of models and logs will be created and namely now the work is in progress.

V. Conclusion

The basic idea of this paper was to investigate and present the possible usage of graph grammars in solving the problem of shortage of models and logs for verifying process mining methods.

In frames of this study graph grammars for structured WF-nets and log generation were developed. We have started the research with only these types of models; however, it is important to notice that these grammars can be adapted to other more difficult and advanced process models such as causal nets, process trees and BPMN models.

Now it is possible to generate a variety of models and logs for applying discovery and further conformance checking algorithms. The generation of models can be organized via built-in extraction algorithms of GROOVE Simulator.

For the further research we are planning to develop an algorithm for conversion and a convertor itself, that will allow integrate the GROOVE models with ProM tool. The integration will be investigated due to the development of model and log import plug-in for ProM.

This paper is considered to be a part of more complex research that will be conducted further and be expressed in the master thesis. This research will be dedicated to the development of model and process generator based on the appliances of graph grammars. Now it is planned to create a program application that will allow creating various process models and rules for logs generation. However, it needs an additional research and work to be conducted.

Acknoledgment

This study was carried out within the National Research University Higher School of Economics' Academic Fund.

References

[1] W. M. P. van der Aalst. Process Mining. Discovery, Conformance and enhancement of Business Processes. Department Mathematics & Computer Science Eindhoven University of Technology. Eindhoven, the Netherlands, 2011.

[2] W.M.P. van der Aalst. Verification of Workflow Nets, Eindhoven University of Technology, Eindhoven, The Netherlands, 21 pages.

[3] Irina A. Lomazova, Ivan, V. Romanov. Analyzing Web Service Resource Compatibility, Moscow, Russia, 12 pages.

[4] Frank Hermann, Harmen Kastenberg, Tony ModicaK. Proceedings of the Second International Workshop on Graph and Model Transformation (GraMoT 2006): Towards Translating Graph Transformation Approaches by Model Transformations, 14 pages.

[5] Prof. Dr. Leila Ribeiro, Prof. Dr. Antonio Carlos da Rocha Costa. Rational Approach of Graph Grammars.Porto Alegre, July 2010, 137 pages.

[6] Andrea Burattin and Alessandro Sperduti. PLG: a Framework for the Generation of usiness Process Models and their Execution Logs, Department of Pure and Applied Mathematics University of Padua, Italy, 6 pages.

[7] Arend Rensink, Iovka Boneva, Harmen Kastenberg and Tom Staijen. User Manual for the GROOVE Tool Set, Department of Computer Science, University of Twente, The Netherland, November 2012, 23 pages.

[8] Nick Russell1, Arthur H. M. ter Hofstedel, Wil M.P. van der Aalst, Nataliya Mulyar. Work-flow control patterns, Brisbane Australia, Eindhoven, The Netherlands, 134 pages.