DOI: 10.15514/ISPRAS-2019-31(4)-9

Method for Building UML Activity Diagrams from Event Logs

N.S. Zubkova, ORCID: 0000-0002-0123-7689 <nszubkova@edu.hse.ru>
S.A. Shershakov, ORCID: 0000-0001-8173-5970 <sshershakov@hse.ru>
Laboratory of Process-Aware Information Systems (PAIS Lab),
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow 101000, Russia

Abstract. UML Activity Diagrams are widely used models for representing software processes. Models built from event logs, recorded by information systems, can provide valuable insights into real flows in processes and suggest ways of improving those systems. This paper proposes a novel method for mining UML Activity Diagrams from event logs. The method is based on a framework that consists of three nested stages involving a set of model transformations. The initial model is inferred from an event log using one of the existing mining algorithms. Then the model, if necessary, is transformed into an intermediate form and, finally, converted into the target UML Activity Diagram by the newly proposed algorithm. The transforming algorithms, except one used at the last stage, are parameters of the framework and can be adjusted based on needed or available models. The paper provides examples of the approach application on real life event logs.

Keywords: process mining; Petri nets; UML activity diagrams; process discovery

For citation: Zubkova N.S., Shershakov S.A. Method for Building UML Activity Diagrams from Event Logs. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 4, 2019. pp. 139-150. DOI: 10.15514/ISPRAS-2019-31(4)-9

Acknowledgements. This work is supported by the Basic Research Program of the National Research University Higher School of Economics.

Метод построения UML диаграмм деятельности по журналам событий

Н.С. Зубкова, ORCID: 0000-0002-0123-7689 <nszubkova@edu.hse.ru>
С.А. Шершаков, ORCID: 0000-0001-8173-5970 <sshershakov@hse.ru>
Лаборатория процессно-ориентированных информационных систем (ПОИС),
Национальный исследовательский университет «Высшая школа экономики»,
101000, Россия, Москва, ул. Мясницкая, д.20

Аннотация. UML диаграммы деятельности широко используются для представления процессов в программной инженерии. Модели, построенные по журналам событий, могут предоставить ценную информацию о реальные процессах в информационных системах, на основании которой эти процессы можно улучшить. Данная статья представляет новый метод майнинга UML диаграмм деятельности по журналам событий. Метод основам на параметрической схеме, которая состоит из трех вложенных ступеней, включающих в себя набор преобразований над моделями. Начальная модель извлекается из журнала событий один из существующих алгоритмов синтеза. Эта модель, если требуется, преобразуется в промежуточную форму, и затем конвертируется в целевую диаграмму деятельности с помощью предлагаемого алгоритма. Алгоритмы синтеза, помимо используемого на последней стадии, являются параметрами схемы и могут быть изменены, исходя из имеющихся или требуемых моделей. В статье представлены примеры применения подхода к журналам событий из реальной жизни.

Ключевые слова: анализ процессов; сети Петри; UML диаграммы деятельности; process discovery

Для цитирования: Зубкова Н.С., Шершаков С.А. Метод построения UML диаграмм деятельности по журналам событий. Труды ИСП РАН, том 31, вып. 4, 2019 г., стр. 139-150 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(4)-9

Благодарности. Работа выполнена при поддержке Программы фундаментальных исследований Национального исследовательского университета «Высшая школа экономики».

1. Introduction

Process mining techniques [1] aim at analyzing and improving real-life processes by taking information from event logs. Event logs are generally produced by process-aware information systems (PAIS) that support these processes. One particular problem of process mining is *process discovery*; its goal is to build a model of a process, based on the data in an event log. Such models can be expressed in different notations. For example, transition systems (TS) naturally represent sequences of events (*traces*) as they are recorded in event logs. However, if a process contains concurrent behavior, transition systems tend to be very complex and as large as the event log itself. This occurs due to the fact that similar patterns are not joined together and concurrency is expressed in the form of interleaving.

There are other types of models that allow to represent concurrency patterns, namely choice and parallelism, and that are widely used in the field of process mining. *Petri nets (PN), BPMN, Fuzzy maps* and *UML Activity Diagrams (AD)* are examples of such models. Unified Modeling Language (UML) [18] is a standard for defining, documenting and visualizing artifacts, especially in the software engineering domain. Particularly, UML Activity Diagrams are used, among other, to represent and analyze actual or expected behavior of software systems. AD is not the only UML class that allows to represent concurrency [9]. For instance, *UML State Machine Diagrams* have their own semantics to illustrate concurrency. However, they reflect different *states* of a system that are not explicitly represented in event logs. These states, therefore, have to be mined using different techniques, i.e. encoding states with trace prefixes. Given that event logs contain information representing *activities* performed by process participants and supporting systems, we regard Activity Diagrams as the de- sired class of target models in this paper.

In our work, we propose a framework for building UML Activity Diagrams from event logs, consisting of a number of steps. The frameworks *essential* part is the algorithm for converting Petri nets into UML ADs. Other intermediate models (namely, TS and PN) can be synthesized using different algorithms which are parameters of the framework. Here we consider the algorithm of regions [8] as a means to generate Petri nets which are consequently converted into target UML ADs. ADs are usually more compact than Petri nets and are more easily interpretable. Moreover, generated diagrams can be imported and used in different visual modeling and design tools used in the software engineering domain, i.e. Sparx Enterprise Architect, and be later included as part of bigger software models.

The contributions of this paper are as follows: (1) a framework for generating UML AD from event logs, (2) a novel method for UML AD synthesis from a Petri nets as intermediate models, (3) implementation of the proposed framework specified by a particular set of synthesis algorithms. The rest of the paper is organized as follows. Section II gives a brief overview of related work. Section III defines necessary concepts needed for the explanation of the proposed approach. The framework is described in Section IV and the PN-to-UML AD conversion algorithm is presented in Section V. Section VI contains models derived from real-life event logs. Finally, Section VII concludes the paper and outlines possible directions for future work.

2. Related work

There exist many approaches to construct Petri nets from event logs [1], [2], [19]. The algorithm of regions and its extensions are described, particularly, in [6], [8], [10]. The algorithm produces a Petri

net from a given TS that serves as an input of the algorithm. The behavior of the derived PN is guaranteed to be equivalent to the TS. Previously, Petri nets have also been used as intermediate models for constructing other types of target models, such as BPMN in [14].

The similarity between UML Activity Diagrams and Petri nets are studied in numerous works. Arlow et al. present UML specification in application to Unified Process including UML AD structural elements, and also mention that UML AD are based on the Petri Net techniques [5]. In [12] authors formalize AD semantics and compare them to semantics of Petri Nets. There are many works dedicated to the transformation of UML Activity Diagrams into Petri nets; the reverse transformation is studied scantily. In [13] the author describes an approach to translate UML AD into Petri nets. Agarwal [4] developed a method for transforming AD into Petri nets for verification purposes. The author considers a set of UML patterns and indicates corresponding Petri net instances.

3. Preliminaries

$B(X)$ is the set of all multisets over some set X. For a given set X, X^+ is the set of all non-empty finite sequences over X.

3.1 Trace, Event Log

Let Z be a set of activities. A *trace* is a finite sequence $\sigma = (a_1, a_2, \dots, a_i, \dots, a_n) \in Z^+$. By $\sigma(i) = a_i$ we denote i-th element. $L \in B(Z^+)$, such that $|L| > 0$, is an event log. Here, $|L|$ is the number of all traces.

3.2 Labeled Petri Net, Well-structured Labeled Petri Net

A *labeled Petri net* (*PN*) is a tuple $= \{P, T, F, l\}$, where P is a set of places, T is a set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, l is the labeling function $l : T \to \Lambda$, and Λ is a set of labels. In process mining, labels of transition represent events.

Given $p \in P$, the set $p^\bullet = \{y | (p, y) \in F\}$ is the postset of p.

In this paper, we denote by Petri net a well-structured Petri net, i.e., a hierarchical Petri net that can be recursively divided into parts having single entry and exit points [15].

3.2 UML Activity Diagram

A *UML Activity Diagram* is a tuple $AD = \{N, E, NT\}$, where

- NT is a set of node types, $NT = \{control, object, executable\}$;
- N is a set of nodes. $n \in N : n = (\lambda, type), \lambda \in \Lambda, type \in NT$;
- E is a set of edges. $e \in E : e = (n1, n2), n1, n2 \in N$.

Similar definition was used in [11]. In this paper we mainly focus on the following elements of the UML AD (see fig. 1):

1) A is a set of activity nodes, $a \in A : a = (\lambda, executable)$;
2) F is a set of parallel nodes, $f \in F : f = (control)$;
3) D is a set of decision nodes, $d \in D : d = (control)$;
4) initial and final are initial and final nodes, both of type control.

UML decision nodes should be equipped with guards that indicate the conditions under which the decision is made. In this paper, we regard non-deterministic Petri nets as intermediate models[1]. The

[1] There exists an extension to Petri nets that adds guards to its semantics. However, most of the process mining algorithms consider Petri nets without guards. Here we follow the same approach.

proposed conversion algorithm does not assume the presence of guard information in the event log and uses only the input Petri net. Thus, the produced Activity Diagram is non-deterministic as well.

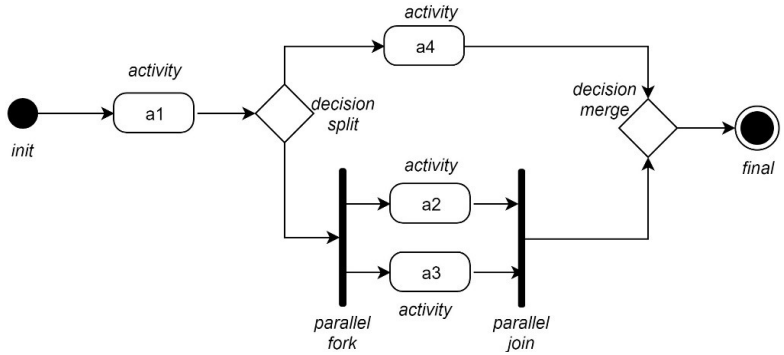

Fig. 1. An Activity Diagram example

4. Framework

The proposed framework is illustrated in fig. 2. The framework consists of a number of nested stages related to individual steps of the proposed method. At every step a transformation from one entity, event log or process model, into another is made. There exist numerous approaches to construct both Petri nets and transition systems. Models obtained from the same event log, but using different algorithms, represent the same process. However, they vary in details that are usually represented by quality metrics [7]. Depending on the task, specific combinations of quality metrics can be considered.

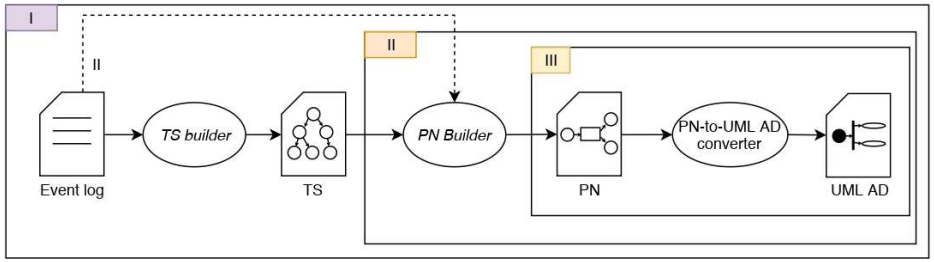

Fig. 2. Proposed framework: I = II + TS construction; II = III + PN synthesis; III = PN-to-UML AD conversion

The long path of the framework (I) includes first building a TS needed for the algorithm of regions. Here, various techniques for TS construction can be used, for instance, prefix tree synthesis [3], frequency based reduction [16], neural approach [17] etc. However, the TS synthesis can be bypassed and a Petri net can be generated directly from the event log (II). There are many algorithms for that, i.e., Inductive miner [15], α-algorithm [2], ILP-miner [20] and other. Finally, in III the generated Petri net is converted into a UML Activity Diagram.

4.1 Proposed implementation

In this paper we consider the full version of the framework with the following parameters.

1) Prefix tree builder, unlimited window, for TS construction.
2) The algorithm of regions for converting the TS into a PN.

3) The PN-to-UML AD converter described in the following section.

The following paragraph gives a brief overview of the algorithms used in steps 1 and 2.

Prefix tree builder [3] is an algorithm for TS synthesis. Event logs usually do not explicitly contain states that are needed for the construction of a transition system. A *state function* is introduced in order to infer such states. This function maps events in an event log onto states of a TS. Let E be a set of events in an event log, and S be a set of states in a transition system. For each event $e \in E$ the state function produces a state $s \in S$ regarding either pre- or posthistory of the state s. A prefix tree is a special type of transition systems, for which the state function consideres prehistory (prefix) of the states. Informally, a transition (s, e, s') appears if $prehistory_{s'} = prehistory_s + e$. If the prefix size is unlimited, the size of the generated TS can be equivalent to the size of the event log.

The algorithm of regions [8] used is based on finding equivalent behaviors in a given transition system. These behavioral fragments are grouped into so-called *regions*. Intuitively a region is equivalent to a place in a Petri net. Placing a token in such a place means allowing such a behavior to appear – via activating a post-transition. In UML Activity Diagrams transitions are translated into *activities*. Thus, considering a transitive dependence between an initial TS and an AD, one can ascertain a link between equivalent behavioral fragments in TS (regions) and corresponding nodes in AD.

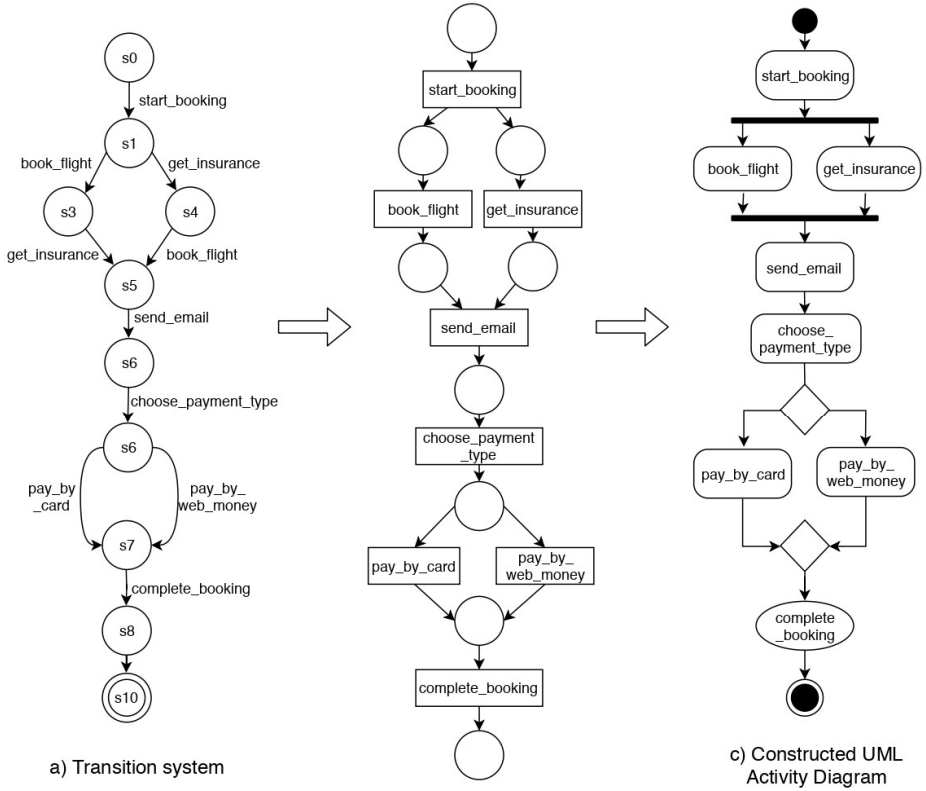

Fig.3. Example models

3. Petri Net to UML Activity Diagram conversion algorithm

The PN-to-UML AD conversion algorithm is based on the idea of converting places and transitions of a given Petri net into corresponding elements of the target UML Activity Diagram. UML AD specification notes that an activity diagram can only have a single entry point, whereas the inception of a process modeled by a Petri net can be determined by placing tokens in multiple places (an initial marking). Here, we consider all places without incoming edges as a potential starting place. Then a single starting point (initial node) in an Activity Diagram is constructed and connected to the following activities. Final places are also not explicitly indicated in Petri nets, however it is sensible to regard those without outgoing edges as such, corresponding final nodes are inserted in the AD.

While translating a Petri net into a UML activity diagram the algorithm considers special patterns, namely parallelisms and decisions. Such patterns can be translated into equivalent patterns in an Activity Diagram. A similar approach was used in [4], [13] for the reverse transformation.

In order to describe the proposed transformation we illustrate it on a running example (fig. 3).

We consider different types of AD nodes and describe the according transformations as follows.

5.1 Transformation functions

Let $\alpha: (T, l) \rightarrow (A, l)$ be a function transforming transitions of the Petri net into *activities* of the constructed UML AD, tagged by the same labels;

Let $\varphi: P \rightarrow D$ be a function transforming appropriate positions of the PN into decision nodes of the UML AD;

Let $\xi: T \rightarrow F$ and $\psi: P \rightarrow F$ be functions transforming PN transitions and sets of PN places into UML parallel nodes accordingly.

5.2 Building a UML Activity Diagram

UML Activity Diagram construction includes the following procedures.

5.2.1 Constructing activity nodes

The semantics of Petri nets suggests that transitions, which model events in Petri nets, correspond to activities in Activity Diagrams. So the first transformation step of the algorithm is turning transitions of a given Petri net into UML AD activities, i.e. for each transition $t \in T$ we create an activity $a = \alpha(t)$ in the AD.

5.2.2 Detecting parallel forks

We now need to connect nodes and identify more complex behaviors. In a Petri net a concurrent pattern occurs if a transition has multiple outgoing edges, allowing tokens to appear in *all* of the following places when the transition is fired (see Fig. 4). Considering a transition $t \in T$ of a Petri net, let T^* be a set of transitions reachable from t in one step. For each transition $t \in T$, if t has:

a) 0 outgoing edges, then activity $\boldsymbol{\alpha(t)}$ is connected to a final node;

b) 1 outgoing edge, then activity $\boldsymbol{\alpha(t)}$ is connected to $\boldsymbol{\alpha(t^*)}$, for each $\boldsymbol{t^* \in T^*}$;

c) > 1 outgoing edge, activity $\boldsymbol{\alpha(t)}$ is connected to a fork node $\boldsymbol{\xi(t)}$, and $\boldsymbol{\xi(t)}$ is then connected to $\boldsymbol{\alpha(t^*)}$, for each $\boldsymbol{t^* \in T^*}$.

5.2.3 Detecting parallel join

In order for the model to be more interpretable, for each parallel fork there should be a reciprocal parallel join. So for each fork, described in 2) we need to find the corresponding join. This is done according to the following steps.

a) For each maximum set of places $\boldsymbol{S} = \{\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}\} \subseteq \boldsymbol{P}$ that have coinciding postsets ($\boldsymbol{p_1^\bullet} = \ldots = \boldsymbol{p_n^\bullet}$) and $\boldsymbol{n > 1}$, a $\boldsymbol{\psi(P)}$ join node is inserted in the $\boldsymbol{AD}$.

b) For each transition $\boldsymbol{t}$ immediately preceding each place from $\boldsymbol{S}$, the activity $\boldsymbol{\alpha(t)}$ is connected to $\boldsymbol{\psi(P)}$.

c) Join node $\boldsymbol{\psi(P)}$ is then connected to $\boldsymbol{\alpha(t')}$, for all $\boldsymbol{t'} \in \boldsymbol{T'}$, where $\boldsymbol{T'}$ is a set of transitions immediately following places $\{\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}\}$.

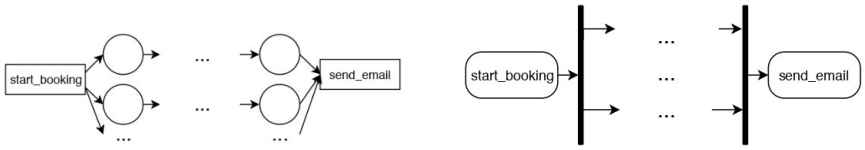

Fig. 4. Concurrency pattern in Petri Net and UML AD

5.2.4 Detecting decision splits and merges

A decision pattern in a Petri net occurs if a place has multiple outgoing edges allowing *only one* consecutive transition to fire (see fig. 5). So for each place $\in P$, that has more than one outgoing edge a decision node $\varphi(p)$ is inserted into the AD and is connected to $\alpha(\tilde{t})$, for all $\tilde{t} \in \tilde{T}$, $\tilde{T}$ are PN transitions connected to p (both before and after). Likewise, if the place p has multiple incoming edges, a reciprocal *merge* node $\varphi(p)$ is inserted into the AD.

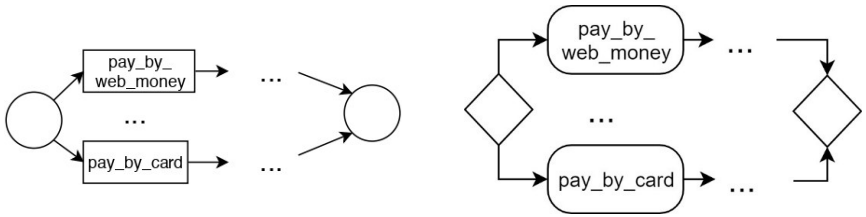

Fig. 5. Choice pattern in Petri Net and UML AD

Applying the steps 5.2.1 – 5.2.4 to an input Petri net, the target UML Activity Diagram is constructed.

6. Application

In this section, we provide examples of models obtained from real logs. Log1 and Log2 consist of 243 and 1132 traces respectively. For observability purposes, intermediate transition systems were reduced using a frequency reduction algorithm described in [16].

In fig. 6 models were generated with window size 1 and frequency reduction parameter 0.04. Log1 contains information about bank operations.

In fig. 7 models were generated on a log containing information about building permit applications from five Dutch municipalities. Transition system was built with unlimited window parameter and reduced with frequency reduction parameter of 0.15.

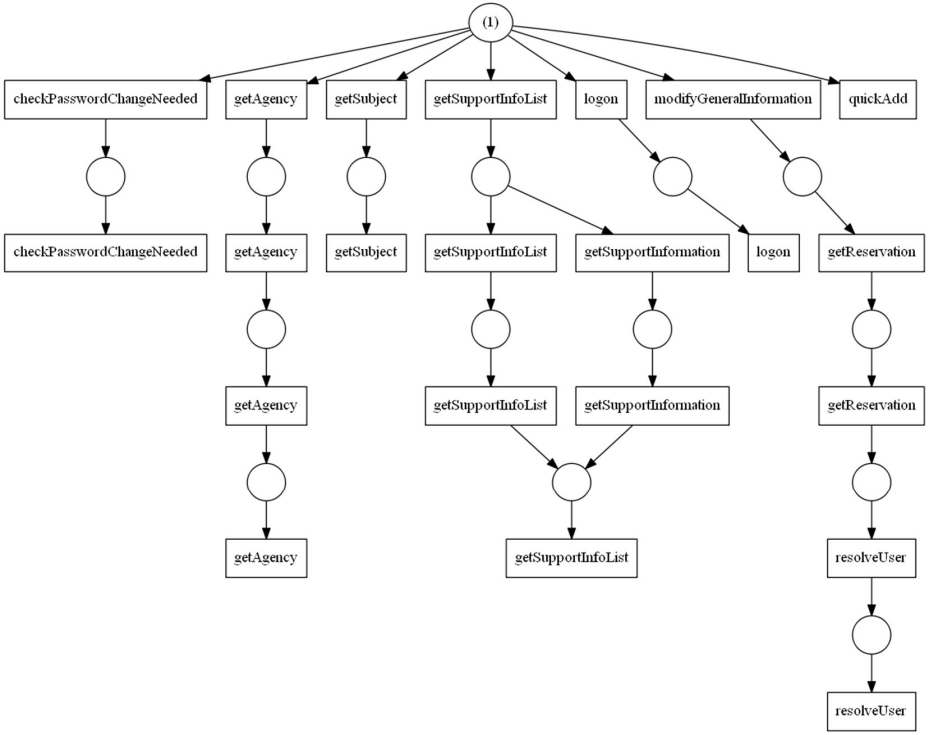

Fig. 6a. Log1: Petri net

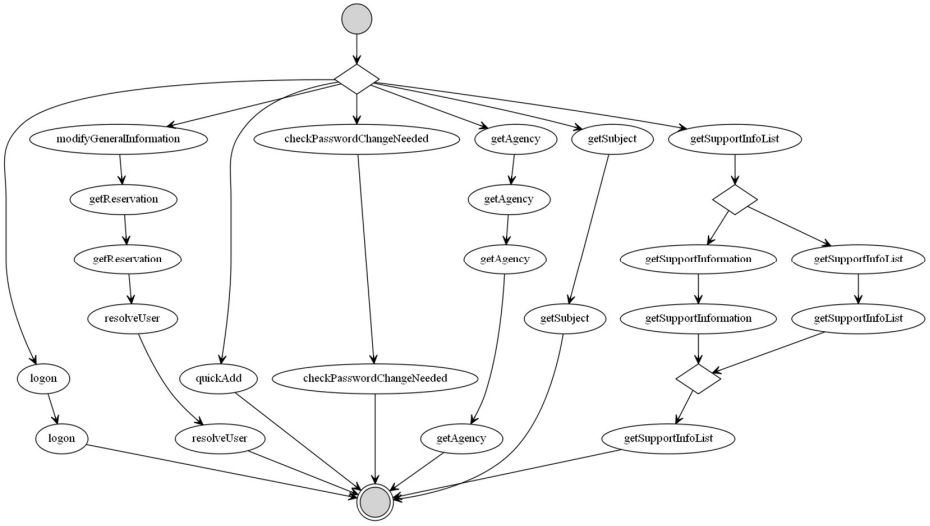

Fig. 6b. Log1: UML Activity Diagram

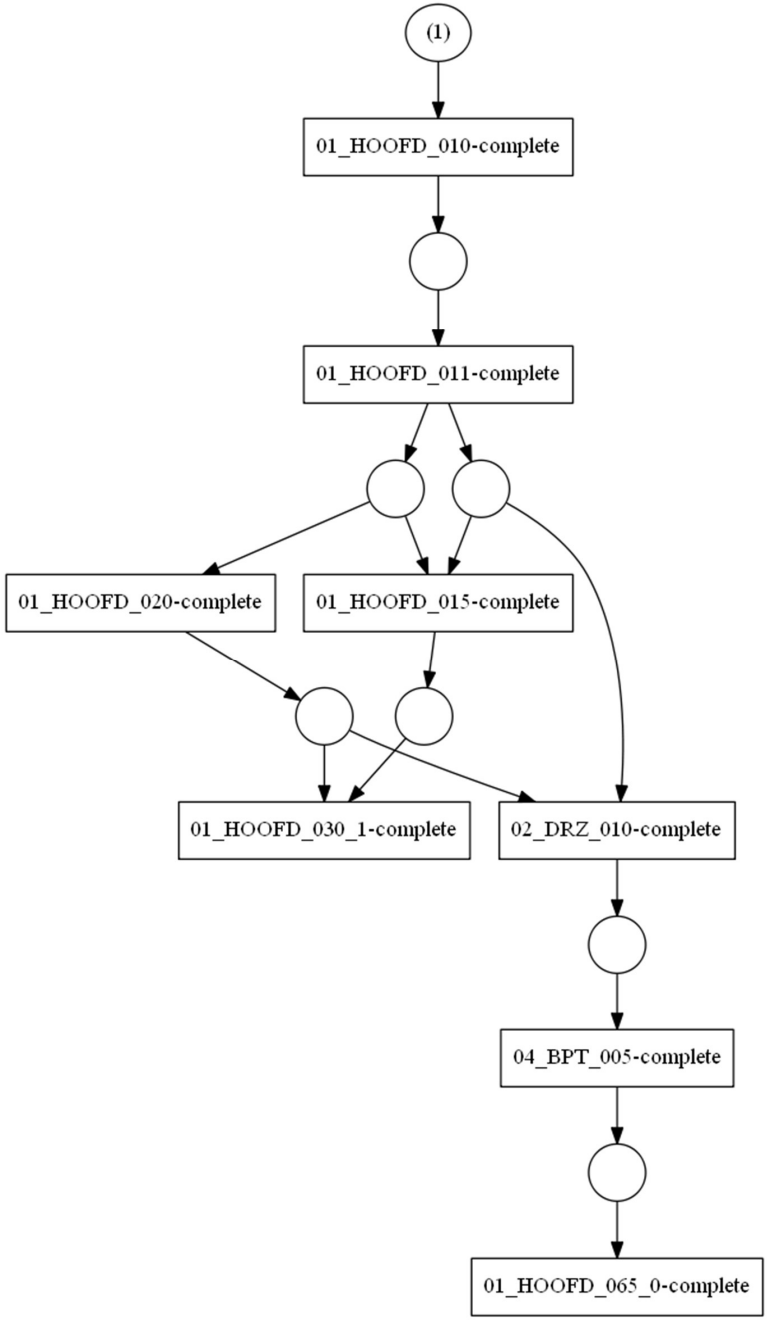

Fig. 7a. Log2: Petri net

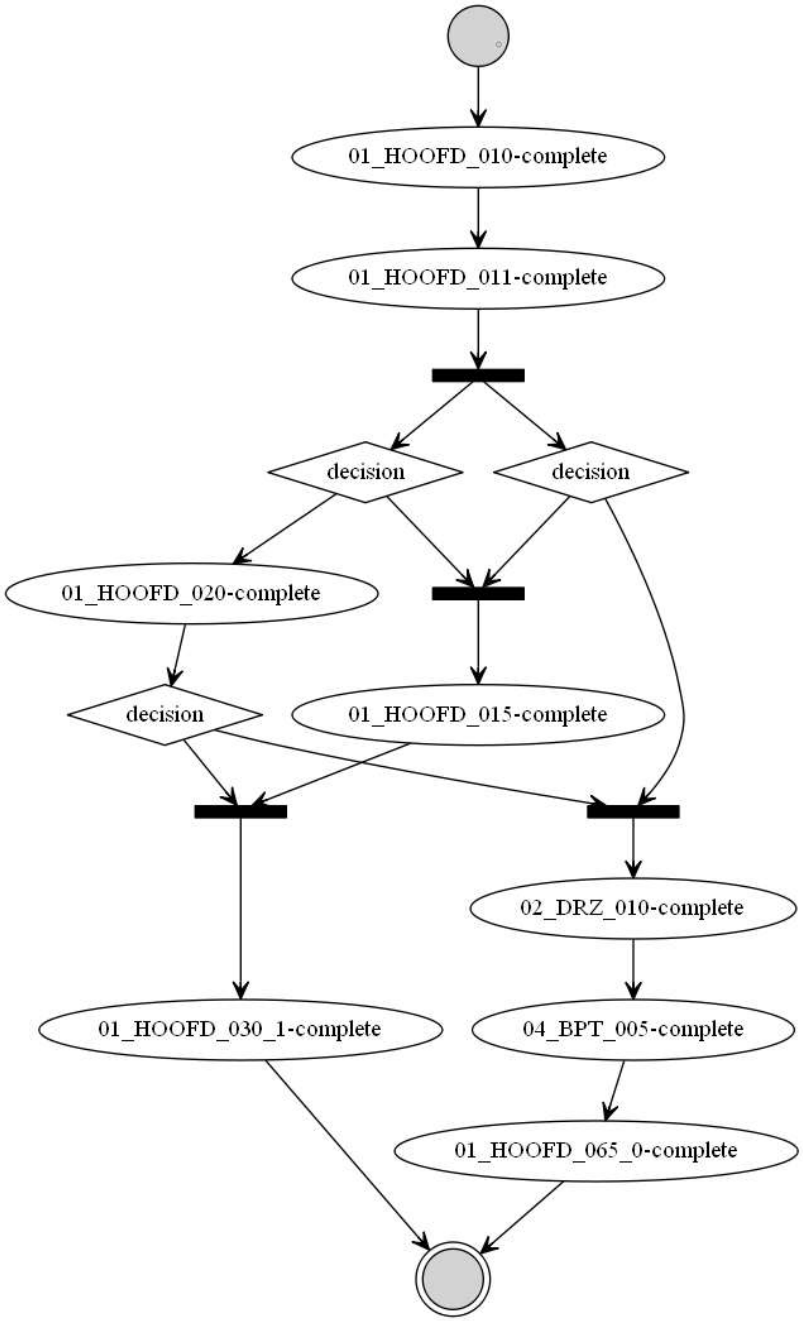

Fig. 7b. Log2: UML Activity Diagram

7. Conclusion

In this paper, we proposed a method based on a framework to build UML Activity Diagrams from event logs and introduced a novel algorithm for converting a well-structured Petri net into a UML Activity Diagram. The method is implemented as a part of the LDOPA[2] library. Future work includes studying the execution semantics of Petri nets with guards, mining dependencies and adding guards to Activity Diagrams. Moreover, the framework can be further investigated by implementing different TS and PN synthesis algorithms.

References

[1]. Van der Aalst W. Data science in action. In Process Mining, Springer, 2016, pp. 3-23.
[2]. Van der Aalst W.M.P., Van Dongen B.F. Discovering petri nets from event logs. Lecture Notes in Computer Science, vol. 7480, 2013, pp. 372-422.
[3]. Van der Aalst W., Rubin V., Verbeek H., van Dongen B., Kindler E., Günther C. Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling, vol. 9, no. 1, 2010, pp. 87-111.
[4]. Agarwal B. Transformation of UML activity diagrams into Petri nets for verification purposes. International Journal of Engineering and Computer Science, vol. 2, no. 3, 2013, pp. 798-805.
[5]. Arlow J., Neustadt I. UML 2 and the unified process: practical object-oriented analysis and design. Pearson Education, 2005, 624 p.
[6]. Badouel E., Bernardinello L., Darondeau P. Polynomial algorithms for the synthesis of bounded nets. Lecture Notes in Computer Science, vol. 915, 1995, pp. 364-378.
[7]. Buijs J.C.A.M., van Dongen B.F., van der Aalst W. M. P. On the role of fitness, precision, generalization and simplicity in process discovery. Lecture Notes in Computer Science, vol. 7565, 2012, pp. 305-322.
[8]. Carmona J., Cortadella J., Kishinevsky M. A region-based algorithm for discovering Petri nets from event logs. Lecture Notes in Computer Science, vol. 5240, 2008, pp. 358-373.
[9]. Concurrency in UML. Available at: https://www.omg.org/ocup-2/documents/concurrency_in_uml_version_2.6.pdf. Accessed: 2019-03-05.
[10]. Cortadella J. et al. Deriving Petri nets from finite transition systems. IEEE Transactions on Computers, vol. 47, no. 8, 1998, pp. 859-882.
[11]. Davydova K.V., Shershakov S.A. Mining hybrid UML models from event logs of SOA systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 155-174. DOI: 10.15514/ISPRAS-2017-29(4)-10.
[12]. Eshuis R., Wieringa R. A comparison of Petri net and activity diagram variants. In Proc. of the 2nd Int. Coll. on Petri Net Technologies for Modelling Communication Based Systems, 2001, pp. 93-104.
[13]. Fahland D. Translating uml2 activity diagrams to petri nets. Informatik-Berichte 226, Humboldt-Universitat zu Berlin, 2008.
[14]. Kalenkova A., van der Aalst W., Lomazova I., Rubin V. Process mining using BPMN: relating event logs and process models, Software and Systems Modeling, 2017, vol. 16, no. 4, pp. 1019-1048.
[15]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Discovering block-structured process models from event logs-a constructive approach. Lecture Notes in Computer Science, vol. 2472, 2013, pp. 311-329.
[16]. Shershakov S.A., Kalenkova A.A., Lomazova I.A. Transition systems reduction: balancing between precision and simplicity. Lecture Notes in Computer Science, vol. 10470, 2017, pp. 119-139.
[17]. Shunin T., Zubkova N., Shershakov S. Neural Approach to the Discovery Problem in Process Mining. Lecture Notes in Computer Science, vol. 11179, 2018, pp. 261-273.
[18]. UML specification. Available at: https://www.omg.org/spec/UML/About-UML/. Accessed: 2019-03-01.
[19]. Weijters A., van Der Aalst W., De Medeiros A.K.A. Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP, 2006, 34 p.
[20]. Van der Werf J. M. E. M., van Dongen B. F., Hurkens C. A., Serebrenik A. Process Discovery Using Integer Linear Programming. Lecture Notes in Computer Science, vol. 5062, 2008, pp. 368-387.

Информация об авторах / Information about the authors

Наталья Сергеевна ЗУБКОВА в настоящее время является студенткой бакалаврской программы «Программная инженерия» факультета компьютерных наук. Область ее научных интересов включает анализ и моделирование процессов, интеллектуальный анализ данных и машинное обучение.

Natalia Sergeyevna ZUBKOVA is currently a student enrolled in the «Software Engineering» bachelor's program, faculty of Computer Science. Her research interests include process modelling and analysis, data mining and machine learning.

Сергей Андреевич ШЕРШАКОВ получил степень магистра в области программной инженерии в НИУ ВШЭ (Москва) в 2012 году. В настоящее время он является научным сотрудником научно-учебной лаборатории процессно-ориентированных информационных систем факультета компьютерных наук. В число научных интересов входят извлечение и анализ процессов (process mining), верификация программного обеспечения, архитектуры информационных систем и преподавание программной инженерии.

Sergey Andreevitch SHERSHAKOV received the MS degree in software engineering from HSE (Moscow, Russia) in 2012. He is currently a research fellow at PAIS Lab of the Faculty of Computer Science. His research interests include process mining, software verification, information systems architectures and teaching software engineering.

[2] Available at https://prj.xiart.ru/projects/ldopa