Mining Hybrid UML Models from Event Logs of SOA Systems

K.V. Davydova <kvdavydova@edu.hse.ru>
S.A. Shershakov <sshershakov@hse.ru>
National Research University Higher School of Economics,
PAIS Lab at the Faculty of Computer Science,
20 Myasnitskaya st., Moscow, 101000, Russia

Abstract. In the paper we consider a method for mining so-called "hybrid" UML models, that refers to software process mining. Models are built from execution traces of information systems with *service-oriented architecture* (SOA), given in the form of event logs. While common reverse engineering techniques usually require the source code, which is often unavailable, our approach deals with event logs which are produced by a lot of information systems, and some heuristic parameters. Since an individual type of UML diagrams shows only one perspective of a system's model, we propose to mine a combination of various types of UML diagrams (namely, *sequence* and *activity*), which are considered together with *communication* diagrams. This allows us to increase the expressive power of the individual diagram. Each type of diagram correlates with one of three levels of abstraction (workflow, interaction and operation), which are commonly used while considering web-service interaction. The proposed algorithm consists of four tasks. They include splitting an event log into several parts and building UML sequence, activity and communication diagrams. We also propose to encapsulate some insignificant or low-level implementation details (such as internal service operations) into activity diagrams and connect them with a more general sequence diagram by using *interaction use* semantics. To cope with a problem of immense size of synthesized UML sequence diagrams, we propose an abstraction technique based on regular expressions. The approach is evaluated by using a developed software tool as a Windows-application in C#. It produces UML models in the form of XML-files. The latter are compatible with well-known Sparx Enterprise Architect and can be further visualized and utilized by that tool.

Keywords: event log, process mining, hybrid UML model, UML sequence diagram, UML activity diagram, reverse engineering.

DOI: 10.15514/ISPRAS-2017-29(4)-10

For citation: 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

1. Introduction

Nowadays we use information systems everywhere. They are used not only at home to increase the comfort of our life but also to support business processes. The complexity of the systems is growing together with the complexity of processes and tasks. Moreover, a lot of systems interact with each other. There is an increasing chance of error as the complexity of the system increases. If the system finds these errors, they are written into so-called event logs together with other information about system execution. The logs store a lot of information during the work of the system. On the one hand, manual processing of the logs is almost impossible because of their size and lack of structure. On the other hand, the event logs are an inestimable source of knowledge about real-life system behavior. Tools, which help to obtain this knowledge in suitable form for analytics are extremely useful.

Different approaches, such as modeling, development within the standardized life cycle, testing, quality assurance (QA), verification, etc., are applied to improve the system quality and error correction. Using combinations of these instruments (for example, testing and verification, modeling and reverse engineering with continuous delivery) gives good results. New tools, modeling tools in particular, help to make the process more convenient and more effective.

Models are built on different life cycle stages. In the classic approach, an architect models an information system based on the customer‘s requirements. However, the implemented system often differs from previously developed models because the system is developed faster than its models. Developers may sometimes make mistakes and may need to spend additional time on critical situations and deadlines. This means that the design and implementation of some components is not completed properly.

When there is no complete model of a system, reverse engineering techniques can be applied to extract the necessary information from the system and build an appropriate model. It allows us to obtain models of a real-life system automatically or semi-automatically. These models correspond to a developed system rather than to an initial plan and initial models. Such models aim both to understand a structure/behavior of a real system and to eliminate any inadequacy of a real model as compared to the initial model. This also makes it easier to fix errors in the system. There are a number of approaches and tools aimed for this purpose. Most of them require the source code of a system to perform analysis. It is not always possible because of different reasons: the source code may not be available to analysts; it is impossible to get the last copy of code or it can be lost. Moreover, different work groups can develop different system components which complicates centralized collection of source code.

Unlike existing reverse engineering approaches that use source code, we propose an approach that works with system execution traces which can be extracted from event logs. Our approach can be considered as a particular implementation of Process Mining [1], a discipline aimed to discover, analyze and improve business processes and their models. Our approach also includes features that are relevant to software engineering. Hence, we refer to it as *software process mining* [2].

Process mining usually uses process models such as Petri nets, BPMN, Fuzzy maps, etc. which are produced by applying different algorithms such as α-algorithm [1], [3], [4], NLP-algorithm [5] or fuzzy miner [6] respectively. However, these models are not perfectly suitable for software developers. In the software engineering area, more specific approaches such as the Unified Modeling Language (UML) [7] are more common. The most common approaches deal with *static class diagrams*, *statecharts*, *sequence* and *activity* diagrams considering them as more descriptive than other. According to UML 2.5, there are two groups of diagrams: *structural* and *behavioral*. In this work we primarily focus on the behavioral group, in particular, on *sequence*, *activity* and *communication* diagrams.

Modern approaches to the development of information systems make out small reusable well-defined pieces of code, which are commonly refered to as *services*. Systems, using services as a main component, are based on *service-oriented architecture* (SOA) [8]. Services from heterogeneous SOA-systems are developed using different languages, environments and tools, but they work in a single *information space*. Mining unified models of those systems is a challenge and has some difficulties. For example, none of the popular reverse engineering tools works with all languages used for web-service development [9]. As almost all systems produce event logs which contain information about interesting system components, it is possible to build models including all of these components. It simplifies the process of reverse engineering and allows us to expand its application area.

In the paper, we consider event logs written by SOA-systems. Our goal is to expand the applicability of UML-based models for SOA-systems by developing new approaches and tools for mining such models from event logs. UML standard describes different types of models which suit different modeling aspects of an information system. Nevertheless, there are situations when analysts would like to use expressive opportunities of several diagram types. UML 2.5 does not describe such diagrams, and it does not forbid them either. In our paper, we propose a new approach to UML-modeling, which includes mining a so-called hybrid diagram that comprises elements of UML *sequence* and UML *activity diagrams*.

To illustrate the proposed approach, consider the following example.

1.1. Motivating example

We consider an event log (Table I) produced by an online banking information system with service-oriented architecture. The log contains a number of traces corresponding to individual instances of a business process maintained by the information system. Our goal is to obtain a UML model that represents some behavioral aspects of the system from different perspectives [9].

Each row of Table I represents a single event. Columns represent attributes of the log. Events are grouped in cases (by CaseID attribute); then, cases are represented in the log by traces. Events are ordered by Timestamp attribute. Different components of SOA are represented by other attributes such as Domain, Service/Process and Operation. Domains contain services and processes while the latter consist of operations [10].

Table 1. Log fragment L1. Banking SOA-system

CaseID	Domain	Service/Process	Operation	Action	Payload	Timestamp
23	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client=Maria, manager=Julia	17:32:15 135
23	Account	CardInfo	GetCardID	REQ	user=a, num=0	17:32:15 250
23	Account	CardInfo	GetCardInfo	REQ	num=0	17:32:15 260
23	Account	CardInfo	GetCardInfo	RES	date=07/16, name=MARIA GRISHINA, id=15674839	17:32:15 267
23	Account	CardInfo	GetCardID	RES	res=15674839	17:32:15 297
23	Card	Operations	GetOperations	REQ	days=30	17:32:15 378
23	Utils	Calendar	GetDate	REQ	days=30	17:32:15 409
23	Utils	Calendar	GetDate	RES	res=23.06.2015	17:32:15 478
23	Card	Operations	GetOperations	RES	res={BP Billing Transfer}	17:32:15 513
23	Card	OperationData	GetPlaceAndDate	REQ	op=BP Billing Transfer	17:32:15 559
23	Card	OperationData	GetPlace	REQ	op=BP Billing Transfer	17:32:15 563
23	Card	OperationData	GetPlace	RES	res=RUS SBERBANK ONLAIN PLATEZH	17:32:15 571
23	Card	OperationData	GetDate	REQ	op=BP Billing Transfer	17:32:15 575
23	Card	OperationData	GetDate	RES	res=20.07.2015	17:32:15 589
23	Card	OperationData	GetPlaceAndDate	RES	res=RUS SBERBANK ONLAIN PLATEZH, date=20.07.2015	17:32:15 601
23	Account	Operations	GetLastOperations	RES	res=succ	17:32:15 822
25	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client= Maxim, manager=Julia	17:40:18 345
25	Account	CardInfo	GetCardID	REQ	user=a	17:40:18 408
25	Account	CardInfo	GetCard	REQ	num=0	17:40:18 422
25	Account	CardInfo	GetCard	RES	res=no cards	17:40:18 434
25	Account	CardInfo	GetCardID	RES	res=error	17:40:18 489
25	Account	Operations	GetLastOperations	RES	res=no bounded cards	17:40:18 523

By applying a method [9] to the example log, we obtain a UML sequence diagram as depicted in Figure 1 representing the overall process. The diagram contains all possible details (excluding operation parameters) of the behavior of the system as it is represented in the event log. Along with regular messages which connect two different lifelines (depicted as vertical dash lines), the diagram also contains a number of self-calls represented as labeled loop arrows, e.g. `GetCardInfo`, `GetCard`. These self-calls are not important for studying the model from a more abstract perspective. In contrast, they are important when modeling the process of the individual service or another SOA component.

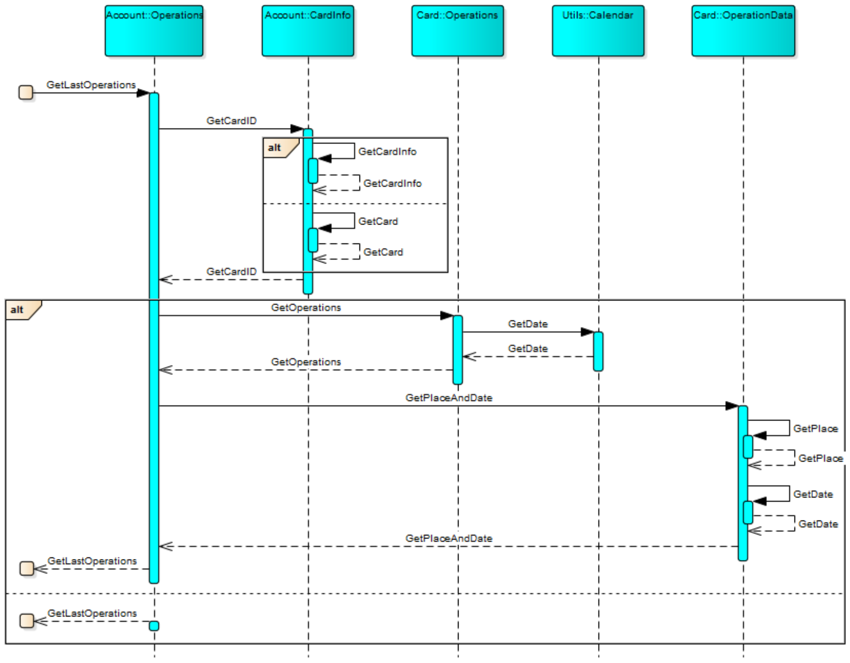


Fig. 1. Usual UML sequence diagram mined from event log L1.

Thus, we propose to hide these calls on the general model with giving a reference to another diagram. Note, that the hidden calls are restricted by one lifeline only. So, using UML sequence diagram here loses its meaning, since only one agent is involved. Therefore, it is convenient to model such behavior by using *UML activity diagrams*, another type of UML diagram. Figures 2, 3 and 4 illustrate this idea and represent a *hybrid UML diagram* combining the best features of two different model types.

A distinctive feature of SOA, which is considered, is that processes call other processes and services while services do not call other participants. To demonstrate this feature, it is important to show the interaction between one selected service and its direct services-neighbors which the service communicates with. A UML communication diagram suits this purpose. Example diagrams for `Card::Operations` and `Card::OperationData` processes from example event log are depicted in Figures 5 and 6 respectively. We can see that these processes are called by other processes and call both different services and themselves.

We developed a tool that builds hybrid diagrams of UML sequence and activity diagrams automatically. Moreover, the tool is able to build a UML communication diagram for a selected SOA component.

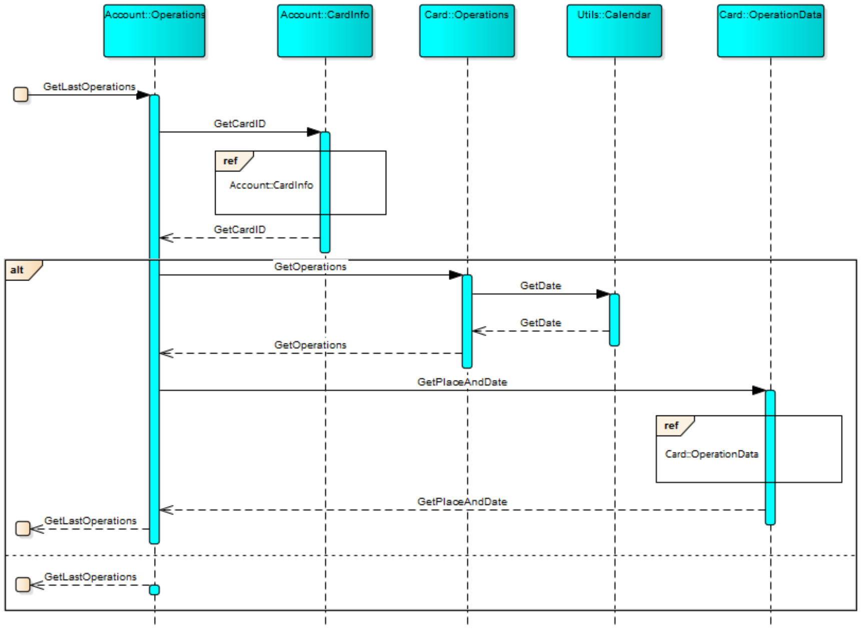


Fig. 2. UML sequence diagram with hidden self calls. High-level diagram of a hybrid UML diagram.

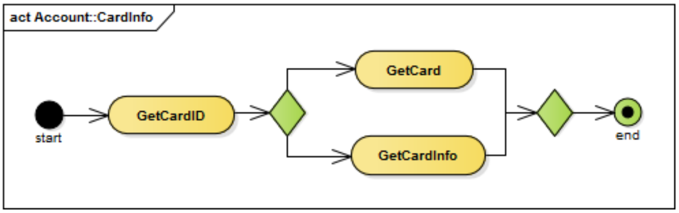


Fig. 3. UML activity diagram with an activity inside Account::CardInfo service.

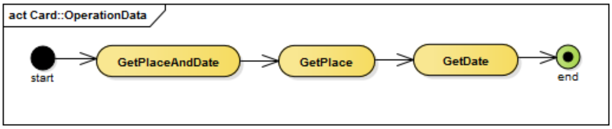


Fig. 4. UML activity diagram with an activity inside Card::OperationData service.

1.2. Related work

Reverse engineering of behavioral UML diagrams is not a new area. There are a number of works [11], [12], [13], [14], about building the UML diagrams based on static source code analysis. Besides, there are some CASE tools [15], [16], [17], [18], which can be used for reverse engineering of sequence and activity UML diagrams. There is also a plug-in [19] for NetBeans development environment that is able to build different types of behavioral models from Java source code.

However, all of the methods and tools mentioned above use static program analysis (getting models from source code without execution) for their work. As it was considered earlier, source code and all of its versions are not always available for analysis. Hence, these tools and methods are useless in this case. Furthermore, none of these tools is able to infer models from the code written in most popular languages used for developing SOA information systems. Moreover, SOA architectures are often developed with various programming languages. For example, some modules can be written in C#, whereas others can be developed in Java; they can interact with LAMP service, so a single CASE tool cannot produce models for that system. Mining diagrams from event logs solves this problem.

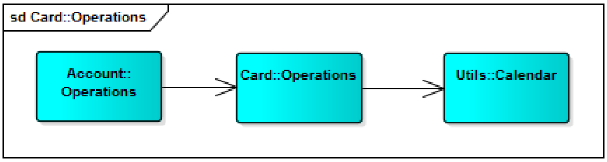


Fig. 5. UML communication diagram for Card::Operations service.

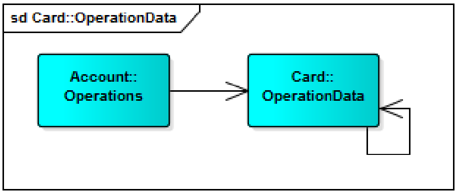


Fig. 6. UML communication diagram for Card::OperationData service.

In [20], [21], [22], approaches to building models based on execution traces are proposed. One related work [20] analyzes a single trace using meta-models of an event log trace and a UML sequence diagram (UML SD). The trace includes information not only about invocation of methods but also about loops and conditions, which makes easier recognition of fragments such as iteration, alternatives and options. However, logs of information systems do not usually include this information, so it is necessary to modify the source code to apply this approach.

There is a description of the mining UML sequence diagrams method based on several execution traces in [22]. The authors propose to use a labeled transition system (LTS) as an intermediate model to present one trace and an algorithm to merge LTSs built by several traces. After that, the LTS is transformed into a UML sequence diagram. Moreover, LTS can be used to build a Petri net that can then be converted into a UML activity diagram [23]. This conversion possibility can be used to apply different process mining algorithms for receiving a UML activity diagram. The approach to mining hierarchical UML sequence diagrams is proposed in [9] (see Section III-D).

In [24], the authors describe a framework which allows not only behavioral but also static UML diagrams to be built. Their framework generates execution traces by itself from Java source code. After that, the framework is able to build UML activity diagrams from traces, but it requires source code for its work.

Process mining proposes to use three abstraction levels for mining models for web services interaction [25]: workflow, interaction and operation. At the operation level, only one service is considered in order to look at its internal behavior and functionality. At the interaction level, they consider not only one selected service but also its direct callers and callees. Finally, the overall services interaction is covered at the workflow level. We apply all of these levels to service-oriented architecture in the paper.

Furthermore, research on service mining was described in [26]. The author builds different Petri nets for different services (considered at the operation level) and then combines them by places. Thus, he builds a generalized model which refers to the workflow level.

The rest of the paper is organized as follows. Section II gives definitions. Section III introduces our approach to mining hybrid UML models. Section IV contains a description of tool implementation. Section V concludes the paper and gives directions for further research.

2. Preliminaries

$\mathcal{P}(X)$ is the powerset over some set X; Λ is a set of all possible string labels.

Definition 1. (Event log) Let $e = (a_1, a_2, \dots, a_n)$ be an event, where a_i is an i-th attribute and n is a number of them. E is a set of events. $\sigma =< e_1, e_2, \dots, e_k >$ is an event trace where $e_1, e_2, \dots, e_k$ is an ordered set of events. $Log = \mathcal{P}(E)$ is an event log which is a powerset of traces.

Definition 2. (UML Sequence Diagram) A UML sequence diagram is a tuple $U_{SD} = (L, T, A, P, M, Ref, F)$, where:

- T is a set of moments of discrete time, which determine a partial order over diagram components.
- L is a set of named lifelines. $L = \{l = (\lambda, t) | \lambda \in \Lambda, t \in T\}$
- A is a set of activations mapped onto lifelines. $a \in A: a = (l, t_b, t_e)$, where $l \in L, t_b, t_e \in T, t_b < t_e$
- $P \subset \Lambda$ is a set of message parameters.
- *Ref* is a set of interaction use (ref fragments) which group lifelines and hide them interaction. $ref \in Ref : ref = (L', \lambda)$, where $L' \subset L, \lambda \in \Lambda$
- M is a set of messages. $m \in M: m = (a_1, t, \lambda, a_2, type)$, where $a_1, a_2 \in A \cup Ref, t \in T, \lambda \in P, type \in \{call, return\}. a_1 = (l_1, t_{11}, t_{12}), a_2 = (l_2, l_{21}, l_{22}): t_{11} \leq t_{21}, t_{11} < t_{12}, t_{21} < t_{22}$
- F is a set of combined fragments of the diagram. $F = \{(frag, M') | M' \subseteq M, frag \in \{alt, loop, opt, par\}\}$

Figure 1 represents an example of UML sequence diagram. A *lifeline* is represented as a vertical dashed line with its name at the top. An *activation* is represented as a rectangle on a lifeline, which takes and emits messages (represented as arrows). Message can be *call* and *return* and they contain text parameters. Messages inside one fragment are ordered by time. *Fragments* contain a number of messages and can contain other combined fragments. They are able to show alternatives, loops, parallelisms and other control structures. Another type of fragment, *ref* fragments, refer to other diagrams. Such diagrams can be both UML sequence diagrams and UML activity ones.

Definition 3. (UML Activity Diagram) A UML activity diagram is a tuple $U_{AD} = (N, E, NT)$, where:

- NT is a set of node types. $NT = \{control, object, executable\}$
- N is a set of nodes. $n \in N: n = (\lambda, type)$, where $\lambda \in \Lambda, type \in NT$
- E is a set of edges. $e \in E: e = (n_1, n_2)$, where $n_1, n_2 \in N$

Figure 3 represents an example of a UML activity diagram for `Account::CardInfo` service. Different node types have different meanings. Control nodes represent different behavioral elements such as start, fork and decision. Object nodes represent data (input and output) of an action. Executable nodes represent steps (actions) of the modeling activity. There are three named executable nodes and four control nodes (start, end, decision and merge) in Figure 3. Different control nodes can impose limitations. For instance, start nodes cannot have incoming edges, end nodes cannot have outgoing edges, decision and fork nodes can have only one incoming edge but several outgoing ones; the opposite is true for merge and join.

$\mathfrak{U}_{AD}$ is a set of all possible UML activity diagrams U_{AD}.

Definition 4. (Hybrid UML Diagram) A hybrid UML diagram is a tuple $U_{HD} = (U_{SD}, AD, f)$, where:

- $U_{SD} = (L, T, A, P, M, Ref, F)$ is a UML sequence diagram.

- $AD \subset \mathfrak{U}_{AD}$
- $f: Ref \rightarrow AD$ is a function which maps ref fragments from a UML sequence diagram onto corresponding activity diagram.

Figures 2, 3 and 4 illustrate an example of a hybrid UML diagram. Figure 2 is a UML sequence diagram and represents a high-level diagram. It refers to UML activity diagrams (Figures 3 and 4) using *ref* fragments.

Definition 5. (UML Communication Diagram) A UML communication diagram is a tuple $U_{CD} = (L_{CD}, M_{CD})$, where:

- $L_{CD} \subset \Lambda$ is a set of named lifelines which represent interaction participants.
- M_{CD} is a set of messages. $m_{CD} \in M_{CD}: m_{CD} = (l_1, l_2, \lambda)$, where $l_1, l_2 \in L_{CD}, \lambda \in \Lambda$.

Figures 5 and 6 provide examples of UML communication diagrams for two different services.

$\mathfrak{U}_{CD}$ is a set of all possible UML communication diagrams U_{CD}.

Definition 6. (Hybrid UML Model) A hybrid UML model is a tuple $U_{CD} = (U_{HD}, \mathrm{CD})$, where:

- U_{HD} is a hybrid UML diagram.
- $\mathrm{CD} \subset \mathfrak{U}_{CD}$.

Figures 2, 3, 4, 5 and 6 represent a hybrid UML model built for example event log L1.

3. Mining Hybrid UML Models

The authors in [25] propose definitions of three levels of abstraction: *operation*, *interaction* and *workflow*. The levels are used for consideration of web service interaction. It motivated us to use different types of UML diagrams which demonstrate features of these levels. In the following sections, we consider which UML diagrams suit each abstraction level and why.

3.1. Operation and workflow abstraction levels

Operation level of abstraction shows what is happening inside one isolated service. Activities outside the service are not considered at the operation level; the only process participants are services. Using a UML sequence diagram leads to a large number of self-calls and *“snowball models”*. It makes the diagram less readable and less understandable. A UML activity diagram suits this purpose since it allows us to demonstrate the complex relationships between operations inside a single participant. Figure 3 shows an example of a UML activity diagram for service `Card::OperationData`.

A business process, provided by services, is represented at a *workflow* abstraction level. There are a lot of participants, so it is useful to use a UML sequence diagram for this level. The diagram is suitable to present not only a sequence of business process actions but also participants of this process and their interaction. An example for event log L1 is depicted in Figure 1.

To bind different abstraction levels, it is necessary to connect them. Our proposal is to use *hybrid UML diagrams* to represent and connect *operation* and *workflow* abstraction levels together. A UML sequence diagram is used to represent a business process at a workflow abstraction level. The diagram contains special objects, *ref* fragments, which make a connection to corresponding UML activity diagram. Every such activity diagram models the behavior of a single service. An example of considered hybrid diagram is presented in Figures 2, 3 and 4.

```
Input  : an event log $Log$;
an attribute name with REQ/RES value $A_{RR}$;
a set of attributes for mapping onto lifelines $A_L$;
a set of attributes for mapping onto message parameters
$A_M$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
a set of regular expressions for merging diagram
components $L_{RE}$ ;
Output: $U_{HM} = (U_{HD}, CD)$ — hybrid UML model;
begin
    /* Split event log into several
    parts                                        */
    $Log_w, Log_o \leftarrow$ splitEventLog($Log, A_L, A_{RR}$);
    /* Build activity diagrams using
    $\alpha$-algorithm [3]                        */
    $AD \leftarrow$ buildADsAlpha($Log_o$);
    $U_{SD} \leftarrow$
    buildSD($Log_w, AD, L_{RE}, A_L, A_M, A_{RR}, caseId$);
    $CD \leftarrow$ buildCDs($Log_w, A_L, ARR$);
    return $U_{HM}$;
```

Algorithm 1. Building a hybrid UML model U_{HM}

3.2. Interaction abstraction level

This level shows interaction of one selected service or process with its nearest neighbors. For a given service, its nearest neighbors are caller and callee services. A UML sequence diagram does not fully suit for representing this level as well as an activity diagram. In the former case, a UML sequence diagram contains a time perspective on which no relation can be mapped. Thus, this leads us to have a "blind" diagram. In the latter case, it does not support multiple participants which is important for this abstraction level.

We propose to use UML communication diagrams for depicting processes occurring in SOA system at interaction abstraction level. An example of such a diagram for

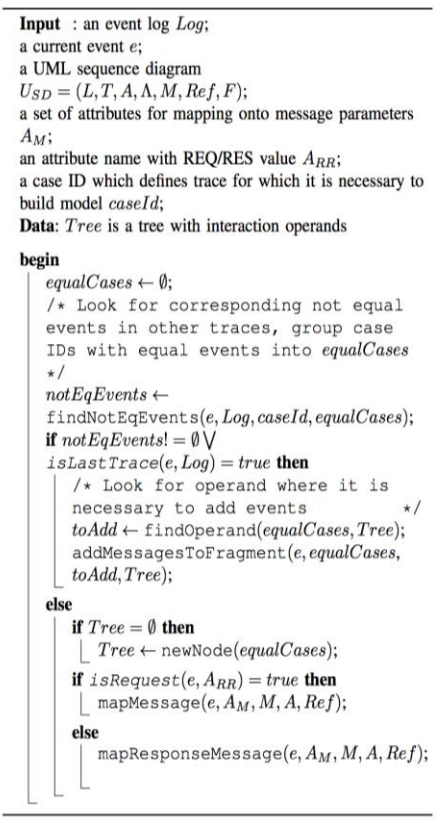
`Card::Operations` and `Card::OperationData` from an event log example is presented in Figures 5 and 6.

```
Input  : an event log $Log$;
a set of attributes for mapping onto lifelines $A_L$;
an attribute name with REQ/RES value $A_{RR}$;
Output: $Log_w$ — a part of an event log which contains
        interaction between different services;
$Log_o$ — a set of event logs (parts of initial event log).
Each of them contains events related to an individual
service;
Data:  $f : K \to \mathcal{P}(V)$, where $K$ is a set of keys and
       $\mathcal{P}(V)$ is a set of value sets;

begin
    /* Get lifeline names from an event log */
    $K \leftarrow$ getLifelineNames($Log, A_L$);
    for $\sigma \in Log$ do
        $\sigma' \leftarrow \emptyset$;
        /* stack - stack with nested events */
        $stack \leftarrow \emptyset$;
        for $i \leftarrow 1$ to $|\sigma|$ do
            $e \leftarrow \sigma[i]$;
            $f$(getLifelineName($e, A_L$)) $\leftarrow$
            $f$(getLifelineName($e, A_L$)) $\bigcup\{e\}$;
            if isRequest($e, A_{RR}$) $= true$ then
                $e_{prev} \leftarrow stack.peek()$;
                if
                $i = 0 \bigvee$ getLifelineName($e, A_L$)! =
                getLifelineName($e_{prev}, A_L$) then
                    $\sigma' \leftarrow \sigma' \bigcup\{e\}$;
                $stack.push(e)$;
            if isRequest($e, A_{RR}$) $= false$ then
                $stack.pop()$;
        $Log_w \leftarrow Log_w \bigcup\{\sigma'\}$;
    for $k \in K$ do
        $Log_o \leftarrow Log_o \bigcup\{f(k)\}$;
    return $Log_w, Log_o$;
```

Algorigm 2. Splitting of an event log into several parts `splitEventLog`

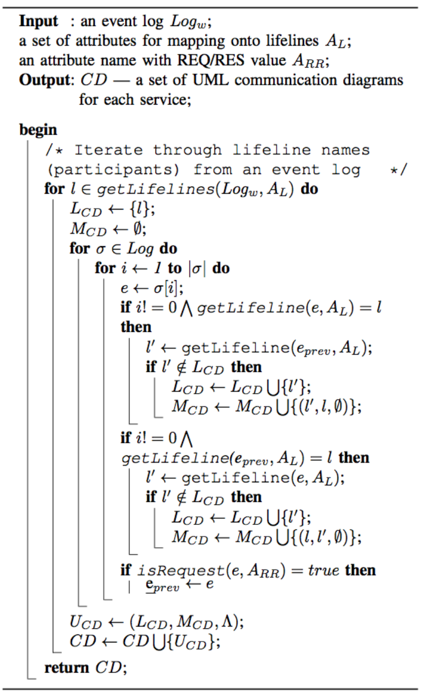
```
Input  : an event log $Log$; a set of UML activity
         diagrams $AD$ which $U_{SD}$ will be refer to;
a set of regular expressions for merging diagram
components $L_{RE}$ a set of attributes for mapping onto
lifelines $A_L$; a set of attributes for mapping onto
message parameters $A_M$; an attribute name with
REQ/RES value $A_{RR}$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
Output: $U_{SD} = (L, T, A, \Lambda, M, Ref, F)$ — UML
         sequence diagram referring to UML activity
         diagrams;

begin
    /* Get lifelines from event log */
    $L \leftarrow$ mapLifelines($Log, A_L$) ;
    if $caseId = \emptyset$ then
        $isAlt \leftarrow true$;
        $caseId \leftarrow$
        getCaseIdOfLongestTrace($Log$);
    else
        $isAlt \leftarrow false$;
    /* Get trace with case ID which is equal to $caseId$ */
    $\sigma \leftarrow Log[caseId]$;
    for $i \leftarrow 1$ to $|\sigma|$ do
        $e \leftarrow \sigma[i]$;
        while isRequest($e, A_{RR}$) $= true$ do
            if $isAlt = true$ then
                /* Look for differences
                between corresponding events
                in other traces, add found
                events to diagram using
                combined fragments */
                findFrames($Log, caseId, e, U_{SD}, A_M, A_{RR}$);
            else
                /* Get a message parameter
                and add its message to
                diagram */
                mapMessage($e, A_M, M, A, Ref$);
            $i \leftarrow i + 1$;
        while isRequest($e, A_{RR}$) $= false$ do
            if $isAlt = true$ then
                findFrames($Log, caseId, e, U_{SD}$);
            else
                mapResponseMessage($e, A_M, M, A, Ref$);
            $i \leftarrow i + 1$;
    if $L_{RE}! = \emptyset$ then
        /* Merge components of the diagram
        using regular expressions */
        changeDiagramUsingREs($U_{SD}, L_{RE}$)
    return $U_{SD}$;
```

Algorithm 3. Building a UML sequence diagram `buildSD`



```
Input  : an event log $Log$;
a current event $e$;
a UML sequence diagram
$U_{SD} = (L, T, A, \Lambda, M, Ref, F)$;
a set of attributes for mapping onto message parameters
$A_M$;
an attribute name with REQ/RES value $A_{RR}$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
Data: $Tree$ is a tree with interaction operands

begin
    $equalCases \leftarrow \emptyset$;
    /* Look for corresponding not equal
    events in other traces, group case
    IDs with equal events into equalCases
    */
    $notEqEvents \leftarrow$
    findNotEqEvents($e, Log, caseId, equalCases$);
    if $notEqEvents! = \emptyset \bigvee$
    isLastTrace($e, Log$) $= true$ then
        /* Look for operand where it is
        necessary to add events                  */
        $toAdd \leftarrow$ findOperand($equalCases, Tree$);
        addMessagesToFragment($e, equalCases,$
        $toAdd, Tree$);
    else
        if $Tree = \emptyset$ then
            $Tree \leftarrow$ newNode($equalCases$);
        if isRequest($e, A_{RR}$) $= true$ then
            mapMessage($e, A_M, M, A, Ref$);
        else
            mapResponseMessage($e, A_M, M, A, Ref$);
```

Algorithm 4. Looking for differences between corresponding events in other traces `findFrames`



```
Input  : an event log $Log_w$;
a set of attributes for mapping onto lifelines $A_L$;
an attribute name with REQ/RES value $A_{RR}$;
Output: $CD$ — a set of UML communication diagrams
        for each service;

begin
    /* Iterate through lifeline names
    (participants) from an event log   */
    for $l \in$ getLifelines($Log_w, A_L$) do
        $L_{CD} \leftarrow \{l\}$;
        $M_{CD} \leftarrow \emptyset$;
        for $\sigma \in Log$ do
            for $i \leftarrow 1$ to $|\sigma|$ do
                $e \leftarrow \sigma[i]$;
                if $i! = 0 \bigwedge$ getLifeline($e, A_L$) $= l$
                then
                    $l' \leftarrow$ getLifeline($e_{prev}, A_L$);
                    if $l' \notin L_{CD}$ then
                        $L_{CD} \leftarrow L_{CD} \bigcup \{l'\}$;
                        $M_{CD} \leftarrow M_{CD} \bigcup \{(l', l, \emptyset)\}$;
                if $i! = 0 \bigwedge$
                getLifeline($e_{prev}, A_L$) $= l$ then
                    $l' \leftarrow$ getLifeline($e, A_L$);
                    if $l' \notin L_{CD}$ then
                        $L_{CD} \leftarrow L_{CD} \bigcup \{l'\}$;
                        $M_{CD} \leftarrow M_{CD} \bigcup \{(l, l', \emptyset)\}$;
                if isRequest($e, A_{RR}$) $= true$ then
                    $e_{prev} \leftarrow e$
        $U_{CD} \leftarrow (L_{CD}, M_{CD}, \Lambda)$;
        $CD \leftarrow CD \bigcup \{U_{CD}\}$;
    return $CD$;
```

Algorithm 5. Building UML communication diagrams for each service `buildCDs`

3.3. Building process

Figure 7 represents a workflow diagram of a hybrid mining process. The scheme contains the following tasks (see Algorithm 1):

- An event log is split into several parts. The workflow part of the log refers to services communication. Such communication is represented on a UML sequence diagram at workflow level. The operation parts consist of events referred to activity only inside a specific service.
- A UML sequence diagram is built from a workflow part of an event log using the method proposed in [9] (see Section III-D) extended by a number of necessary *ref* fragments used for connecting with corresponding activity diagrams.

- UML activity diagrams are built from the operation parts of the log independently using one of the process mining algorithms which produces a Petri net. For instance, α-algorithm [4] or inductive miner [27] can be considered here. Then, Petri nets are converted into activity diagrams by a simple translation routine. This conversion is rather trivial since UML activity diagrams are initially based on Petri nets [7], [23].

3.4. Mining UML sequence diagrams

To mine a UML sequence diagram we use a method proposed in [9]. There, we propose an approach to mining UML sequence diagrams with different levels of abstraction. It consists of three steps. The first step of the approach is mapping event log attributes onto UML sequence diagram components. There are two functions for mapping attributes onto lifelines and message parameters. The smaller the SOA element we choose for mapping onto lifelines, the lower the abstraction level we receive.

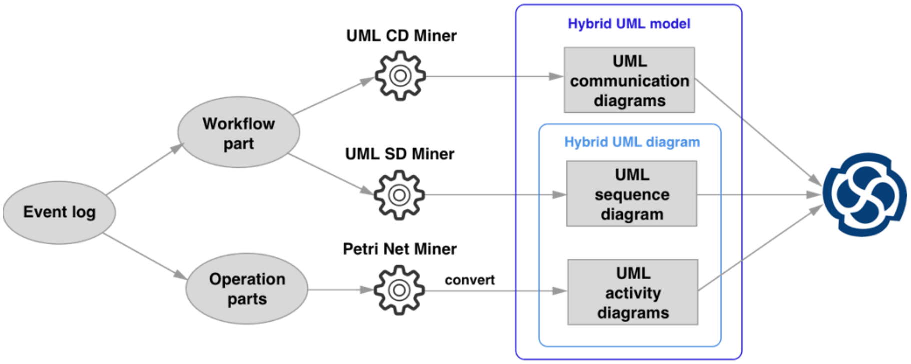


Fig. 7. The workflow diagram of a hybrid mining process.

The second step is set to build a smaller model by applying regular expressions for merging similar messages and lifelines on a diagram. For example, we have two messages with the following parameters: `GetPlaseAndDate, op=BP Billing Transfer` and `GetPlaseAndDate, op=Retail`. They differ in op value, thus, these messages can be combined into one message with the following parameter: `GetPlaseAndDate, op=.*`. After the merging, a derived model becomes more generalized and its size decreases in width and height.

To demonstrate the hierarchy of calls, which is important for SOA, a hierarchical diagram can be applied. Thus, the third step of our approach contains a way to present a complex model by using hierarchical UML diagrams. UML standard [7] allows us to divide the model into some parts and connect them by means of *interaction use* (*ref* fragment) and *gates*.

4. *Tool Overview*

This section presents a brief overview of the software tool implementing the proposed algorithm.

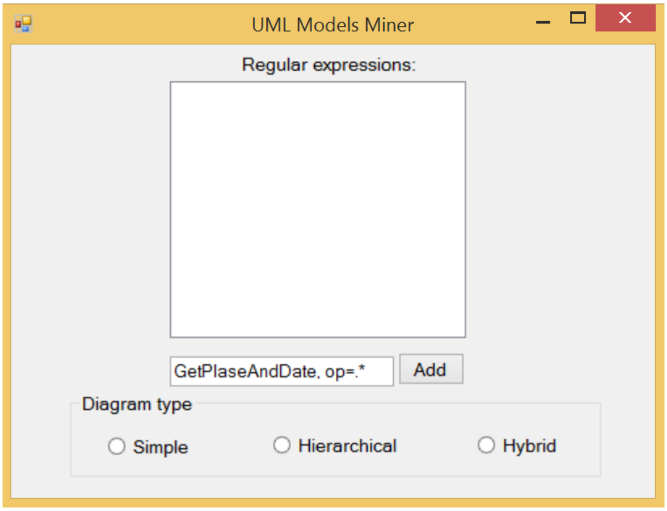
4.1. Event log

The tool requires an input event log to be presented in definite format. We use simple CSV text files to represent event logs. An event log should contain a number of fields that are mapped onto mandatory attributes, namely *CaseID*, *Timestamp* and *Activity*.

4.2. Tool implementation

The tool is implemented as a Windows application written in C# programming language. The tool allows users to configure main parameters such as regular expressions, hierarchy and type of output diagram (regular UML, hierarchical or hybrid). Regular expressions are applied for merging diagram components. It is implemented as shown in Figure 8. The GUI allows the user to set the type of diagram. The *perspective* of the diagram (a mapping attributes onto diagram lifelines and messages) is set as it described in [9].

The output of the tool is an XMI-file containing a model and a description of diagrams. It can be visualized by Sparx Enterprise Architect [15].



GetPlaseAndDate, op=.*

Fig. 8. GUI to set a type of the diagram and regular expressions for merging its components.

5. Conclusion

This paper introduced a new concept of hybrid UML models and proposed a method of mining them from event logs of SOA information systems using a service mining approach. Our method can also be applied to other types of UML diagrams. The paper discussed approaches to mining diagrams at different abstraction levels.

Our method builds models by using only event logs. This is an advantage over some reverse engineering techniques because the source code is not always available. The proposed method includes mining hybrid UML diagrams that represent workflow abstraction level on UML sequence diagrams and operation level on UML activity diagrams. Moreover, we proposed to build UML communication diagrams to show interaction abstraction level with regards to the service mining approach.

Generally, control structures in system‘s behavior lead to a presence of a big number of nested combined fragments within a UML sequence diagram. It makes the diagram less readable and less understandable. Although UML activity diagrams have no time perspective in contradistinction to sequence diagrams, the former show alternatives, loops and parallelism more clearly. Since there are also a lot of event logs which are not produced by SOA systems, we are going to expand our approach to mining hybrid UML diagrams from event logs of more broad types of software architecture in the future.

Acknowledgement

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

References

[1]. W. M. P. van der Aalst. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edition, 2011.

[2]. V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer. Process Mining Framework for Software Processes, pages 169– 181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3]. A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the α-algorithm to mine short loops. In Eindhoven University of Technology, Eindhoven, 2004.

[4]. W. M. P. van der Aalst, A. J. M. M. Weijter, and L. Maruster. Workflow Mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16:2004, 2003.

[5]. F. Friedrich, J. Mendling, and F. Puhlmann. Process Model Generation from Natural Language Text, pages 482–496. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6]. C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining – Adaptive Process Simplification Based on Multiperspective Metrics, pages 328–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[7]. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, August 2015.

[8]. T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9]. K. V. Davydova and S. A. Shershakov. Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA systems while Balancing between Abstracted and Detailed Models. 28(3):85–102, 2016.

[10]. S. A. Shershakov and V. A. Rubin. System runs analysis with process mining. In Modeling and Analysis of Information Systems, pages 818–833, 2015.

[11]. A. Rountev and B. H. Connell. Object Naming Analysis for Reverse-engineered Sequence Diagrams. In Proceedings of the 27th International Conference on Software Engineering, ICSE ’05, pages 254–263, New York, NY, USA, 2005. ACM.

[12]. A. Rountev, O. Volgin, and M. Reddoch. Static Control-flow Analysis for Reverse Engineering of UML Sequence Diagrams. SIGSOFT Softw. Eng. Notes, 31(1):96–102, September 2005.

[13]. P. Tonella and A. Potrich. Reverse engineering of the interaction diagrams from C++ code. In International Conference on Software Maintenance, pages 159–168. IEEE Computer Society, 2003.

[14]. E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In WCRE, pages 297–298. IEEE Computer Society, 2006.

[15]. Sparx Systems’ Enterprise Architect. http://www.sparxsystems.com.au/products/ea/.

[16]. IBM Rational Software Architect. https://www.ibm.com/ developerworks/downloads/r/architect/.

[17]. Visual Paradigm. https://www.visual-paradigm.com/ features/.

[18]. Altova UModel. http://www.altova.com/umodel.html.

[19]. NetBeans UML. http://plugins.netbeans.org/plugin/1801/netbeans-uml.

[20]. L. C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. IEEE Trans. Softw. Eng., 32(9):642–663, September 2006.

[21]. R. Delamare, B. Baudry, and Y. Le Traon. Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces. In Proceedings of the workshop on Object-Oriented Reengineering at ECOOP 06, Nantes, France, July 2006.

[22]. T. Ziadi, M. A. A. da Silva, L. M. Hillah, and M. Ziane. A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams. In Isabelle Perseil, Karin Breitman, and Roy Sterritt, editors, ICECCS, pages 107– 116. IEEE Computer Society, 2011.

[23]. B. Agarwal. Transformation of UML Activity Diagrams into Petri Nets for Verification Purposes. 2(3):798–805, 2013.

[24]. A. Bergmayr, H. Bruneliere, J. Cabot, J. García, T. Mayerhofer, and M. Wimmer. fREX: FUML-based Reverse Engineering of Executable Behavior for Software Dynamic Analysis. In Proceedings of the 8th International Workshop on Modeling in Software Engineering, MiSE ’16, pages 20–26, New York, NY, USA, 2016. ACM.

[25]. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Tech. Rep. TUV-1841-2004-16. 2004.

[26]. W. M. P. van der Aalst. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. IEEE Transactions on Services Computing, 6(4):525–535, 2013.

[27]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, pages 66–78. Springer International Publishing, Cham, 2014.

Метод автоматического построения гибридных UML-моделей на основе журналов событий систем с сервис-ориентированной архитектурой

К.В. Давыдова <kvdavydova@edu.hse.ru>
С.А. Шершаков <sshershakov@hse.ru>
Национальный исследовательский университет Высшая школа экономики, лаборатория ПОИС факультета компьютерных наук,
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. В данной статье мы предлагаем метод автоматического построения так называемых «гибридных» UML-моделей, что относится к области извлечения и анализа процессов ПО. Модели строятся на основе трасс исполнения, представленных в виде журналов событий, систем с сервис-ориентированной архитектурой (СОА). В то время как известные техники обратной разработки обычно используют исходный программный код, который часто недоступен, наш подход работает с журналами событий, записываемыми большинством информационных систем, и некоторыми эвристическими параметрами. Так как отдельный класс UML-диаграмм представляет только одну перспективу модели системы, мы предлагаем синтезировать комбинацию нескольких классов UML-диаграмм (последовательности и деятельности), которые рассматриваются совместно с диаграммами коммуникаций. Это позволяет повысить выразительную силу отдельной «гибридной» диаграммы. Каждый класс диаграмм представляет один из уровней абстракции (workflow, operation и interaction), которые обычно используются при рассмотрении взаимодействия web-сервисов. Предлагаемый алгоритм состоит из четырех этапов: разделение журнала событий на несколько частей, построение UML диаграмм последовательности, деятельности и коммуникаций. Мы также предлагаем инкапсулировать некоторые незначительные или низкоуровневые имплементационные детали (например, внутренние операции сервисов) в диаграммы деятельности и соединять их с более высокоуровневыми диаграммами последовательности с использованием «interaction use» фрагментов. Чтобы решить проблему больших размеров синтезируемых UML диаграмм последовательности, мы предлагаем обобщающую технику, основанную на регулярных выражениях. Предложенный подход оценен с использованием разработанного программного инструмента в виде Windows-приложения, написанного на языке C#. Этот инструмент строит UML модели и сохраняет их в виде XML-файлов. Такие файлы совместимы с хорошо известным интрументом проектирования программной архитектуры Sparx Enterprise Architect, в котором синтезированные модели могут быть визуализированы и отредактированы.

Ключевые слова: журнал событий; извлечение и анализ процессов (process mining); гибридные UML модели; диаграмма последовательности UML; диаграмма деятельности UML; обратная разработка.

DOI: 10.15514/ISPRAS-2017-29(4)-10

Для цитирования: Давыдова К.В., Шершаков С.А. Метод автоматического построения гибридных UML моделей на основе журналов событий СОА-систем. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 155-174 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-10

Список литературы

[1]. W. M. P. van der Aalst. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edition, 2011.

[2]. V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer. Process Mining Framework for Software Processes, pages 169– 181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3]. A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the α-algorithm to mine short loops. In Eindhoven University of Technology, Eindhoven, 2004.

[4]. W. M. P. van der Aalst, A. J. M. M. Weijter, and L. Maruster. Workflow Mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16:2004, 2003.

[5]. F. Friedrich, J. Mendling, and F. Puhlmann. Process Model Generation from Natural Language Text, pages 482–496. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6]. C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining – Adaptive Process Simplification Based on Multiperspective Metrics, pages 328–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[7]. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, August 2015.

[8]. T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9]. K. V. Davydova and S. A. Shershakov. Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA systems while Balancing between Abstracted and Detailed Models. 28(3):85–102, 2016.

[10]. S. A. Shershakov and V. A. Rubin. System runs analysis with process mining. In Modeling and Analysis of Information Systems, pages 818–833, 2015.

[11]. A. Rountev and B. H. Connell. Object Naming Analysis for Reverse-engineered Sequence Diagrams. In Proceedings of the 27th International Conference on Software Engineering, ICSE '05, pages 254–263, New York, NY, USA, 2005. ACM.

[12]. A. Rountev, O. Volgin, and M. Reddoch. Static Control-flow Analysis for Reverse Engineering of UML Sequence Diagrams. SIGSOFT Softw. Eng. Notes, 31(1):96–102, September 2005.

[13]. P. Tonella and A. Potrich. Reverse engineering of the interaction diagrams from C++ code. In International Conference on Software Maintenance, pages 159–168. IEEE Computer Society, 2003.

[14]. E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In WCRE, pages 297–298. IEEE Computer Society, 2006.

[15]. Sparx Systems' Enterprise Architect. http://www.sparxsystems.com.au/products/ea/.

[16]. IBM Rational Software Architect. https://www.ibm.com/developerworks/downloads/r/architect/.

[17]. Visual Paradigm. https://www.visual-paradigm.com/ features/.

[18]. Altova UModel. http://www.altova.com/umodel.html.

[19]. NetBeans UML. http://plugins.netbeans.org/plugin/1801/netbeans-uml.

[20]. L. C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. IEEE Trans. Softw. Eng., 32(9):642–663, September 2006.

[21]. R. Delamare, B. Baudry, and Y. Le Traon. Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces. In Proceedings of the workshop on Object-Oriented Reengineering at ECOOP 06, Nantes, France, July 2006.

[22]. T. Ziadi, M. A. A. da Silva, L. M. Hillah, and M. Ziane. A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams. In Isabelle Perseil, Karin Breitman, and Roy Sterritt, editors, ICECCS, pages 107– 116. IEEE Computer Society, 2011.

[23]. B. Agarwal. Transformation of UML Activity Diagrams into Petri Nets for Verification Purposes. 2(3):798–805, 2013.

[24]. A. Bergmayr, H. Bruneliere, J. Cabot, J. García, T. Mayerhofer, and M. Wimmer. fREX: FUML-based Reverse Engineering of Executable Behavior for Software Dynamic Analysis. In Proceedings of the 8th International Workshop on Modeling in Software Engineering, MiSE '16, pages 20–26, New York, NY, USA, 2016. ACM.

[25]. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Tech. Rep. TUV-1841-2004-16. 2004.

[26]. W. M. P. van der Aalst. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. IEEE Transactions on Services Computing, 6(4):525–535, 2013.

[27]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, pages 66–78. Springer International Publishing, Cham, 2014.