Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA Systems while Balancing between Abstracted and Detailed Models

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 this paper, we consider an approach to reverse engineering of UML sequence diagrams from event logs of information systems with a service-oriented architecture (SOA). UML sequence diagrams are graphical models quite suitable for representing interactions in heterogeneous component systems; in particular, the latter include increasingly popular SOA-based information systems. The approach deals with execution traces of SOA systems, represented in the form of event logs. Event logs are created by almost all modern information systems primarily for debug purposes. In contrast with conventional reverse engineering techniques that require source code for analysis, our approach for inferring UML sequence diagrams deals only with available logs and some heuristic knowledge. Our method consists of several stages of building UML sequence diagrams according to different perspectives set by the analyst. They include mapping log attributes to diagram elements, thereby determining a level of abstraction, grouping several components of a diagram and building hierarchical diagrams. We propose to group some of diagram components (messages and lifelines) based on regular expressions and build hierarchical diagrams using nested fragments. The approach is evaluated in a software prototype implemented as a Microsoft Visio add-in. The add-in builds a UML sequence diagram from a given event log according to a set of customizable settings.

Keywords: event logs; UML sequence diagram; reverse engineering; process mining.

DOI: 10.15514/ISPRAS-2016-28(3)-6

For citation: DavydovaK.V., ShershakovS.A. Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA systems while Balancing between Abstracted and Detailed Models. *Trudy ISP RAN / [Proc. ISP RAS]*, vol.28, issue 3, 2016. pp. 85-102. DOI: 10.15514/ISPRAS-2016-28(3)-6

1. Introduction

Nowadays there are a lot of information systems. They are developed by people, which are error-prone. Systems also can have a structure which is difficult to understand. Thus, models are necessary to understand systems and find errors. When there is no complete model of a system, reverse engineering techniques can be applied to extract necessary information from the system and build an appropriate model. There are a number of tools for this purpose, they analyze source code of the system and build a model.

There are some types of models, which are useful to analyze in software engineering. For example, state machines are able to model a large number of software problems. However, they have a weakness in describing an abstract model of computation. Another example of a software model is Petri nets which can describe processes with concurrent execution. Furthermore, there are a number of models described by a standard of Unified Modeling Language (UML) for visualizing design of information systems. UML 2.4.1 [1] has two groups of diagrams, structural and behavioral ones. In particular, such kind of UML diagrams as *state class diagrams, statecharts* and *sequence diagrams* are widely applied to reverse engineering domain.

Almost every information system has an ability to write results of its execution to event logs. We propose approaches to mine UML sequence diagrams (UML SD) from these logs. Event logs of information systems with a service-oriented architecture (SOA) are considered and UML SDs are applied to modeling interaction between SOA information system components.

In contrast to existing reverse engineering tools, which use source code, we work with *system execution traces* in the form of event logs. A technique that allows analysis of business processes based on event logs is called process mining [2]. It uses specialized algorithms for extracting knowledge from event logs recorded by an information system. Moreover, process mining helps to check the conformance of a derived model with its earlier specification. Using execution traces works even if there is no access to the source code of an information systems. Also, not all code versions are normally stored. Moreover, large information systems tend to be distributed. Different components of a system are often implemented in different programming languages. Such a problem is solved by considering event logs instead of source code.

1.1. Motivating example

There is an event log written by a SOA-based banking information system (Table 1). We are interested in building a model in the form of a UML sequence diagram reflecting processes in the system. We have only some of the runs of the process, so one of the problems is to build an as feasible model as possible. The log contains a number of execution traces. Each trace consists of a sequence of events ordered by Timestamp attribute. Columns represent attributes of the log and rows represent its events. System executions are maintained by different components of the system. They are grouped in attributes such as *Domain*, *Service/Process* and *Operation*. *Domains* group *Services* and *Processes*, and the latter consist of *Operations* [3].

Interaction between program system components can be represented at different abstraction levels. For example, by mapping some log attributes onto structural elements of UML SDs, such as lifelines and messages, one can get a UML SD diagram such as on Fig. 1. Specific values of these attributes appear with head names such as “Domain::Service/Process”. Similarly, values of *Operation* and *Payload* attributes, which are mapped onto messages parameters appear with message arrows. Timestamp attribute sets an order of calls (time goes from the top to the bottom of a diagram).

It can also be useful to merge some messages or lifelines in order to reduce the size of a diagram and avoid “spaghetti-like” models. A regular expression suits it and an example of their usage is depicted on Fig. 2.

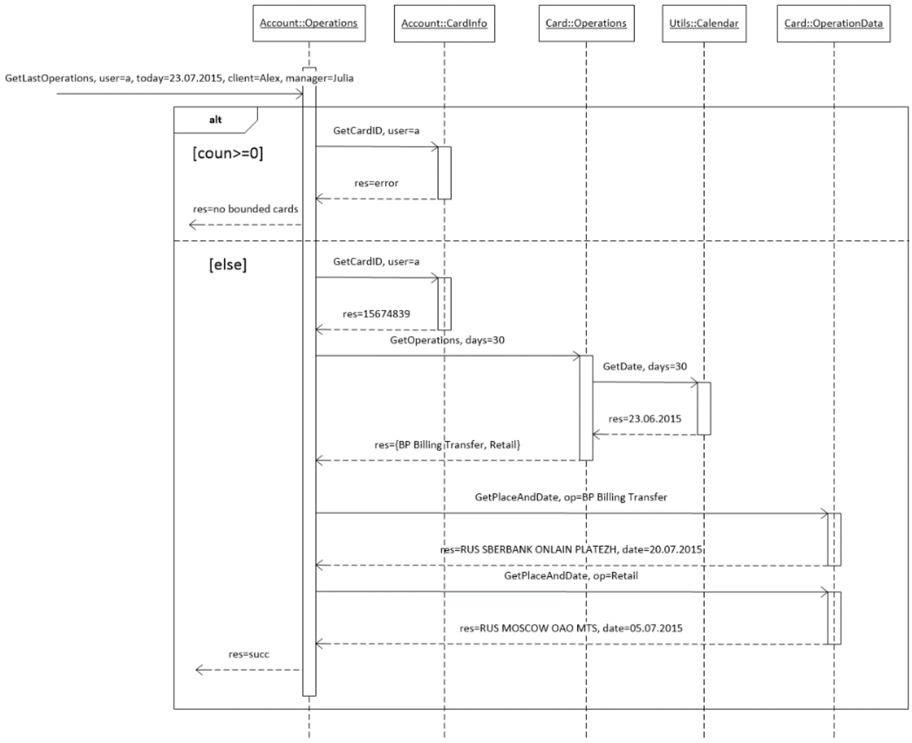

Account::Operations
Account::CardInfo
Card::Operations
Utils::Calendar
Card::OperationData
GetLastOperations, user=a, today=23.07.2015, client=Alex, manager=Julia
alt
[coun>=0]
GetCardID, user=a
res=error
res=no bounded cards
[else]
GetCardID, user=a
res=15674839
GetOperations, days=30
GetDate, days=30
res=23.06.2015
res={BP Billing Transfer, Retail}
GetPlaceAndDate, op=BP Billing Transfer
res=RUS SBERBANK ONLAIN PLATEZH, date=20.07.2015
GetPlaceAndDate, op=Retail
res=RUS MOSCOW OAO MTS, date=05.07.2015
res=succ

Fig. 1. Mapping log attributes onto UML sequence diagram components

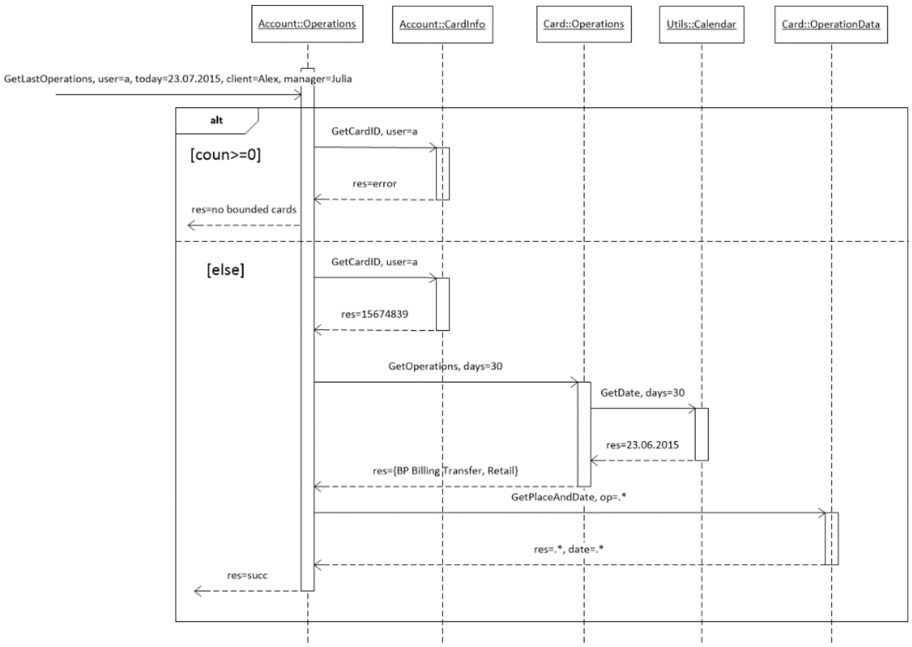


Fig. 2. Merge of diagram components based on a regular expression

Some interaction sometimes can be useful to represent on one diagram and other interactions on a nested diagram. Those both diagrams use an interaction fragment labeled *ref*. An example of a hierarchical diagram is on Fig. 3.

It would be good to have a tool which can do mapping of event log attributes on UML sequence diagram elements with ability to set an abstraction level for seeing different perspectives of the system execution. An approach approved in VTM4Visio framework is applied, which allows building these diagrams.

1.2. Related work

Reverse engineering of UML sequence diagrams is not a new problem. There are a number of works such as [4], [5], [6], [7] applied static approaches (getting models from source code without execution) for solving this problem. Moreover, there are a number of CASE tools for reverse engineering of UML sequence diagrams and other types of UML diagrams. However, most of them use static program analysis without execution of a program. Static program analysis usually uses source code or object code (a result of source code compilation). Some of these tools analyze source code, some of these tools analyze both source code and object code.

However, event logs are execution traces of source code. Thus, we do not need access to source code.

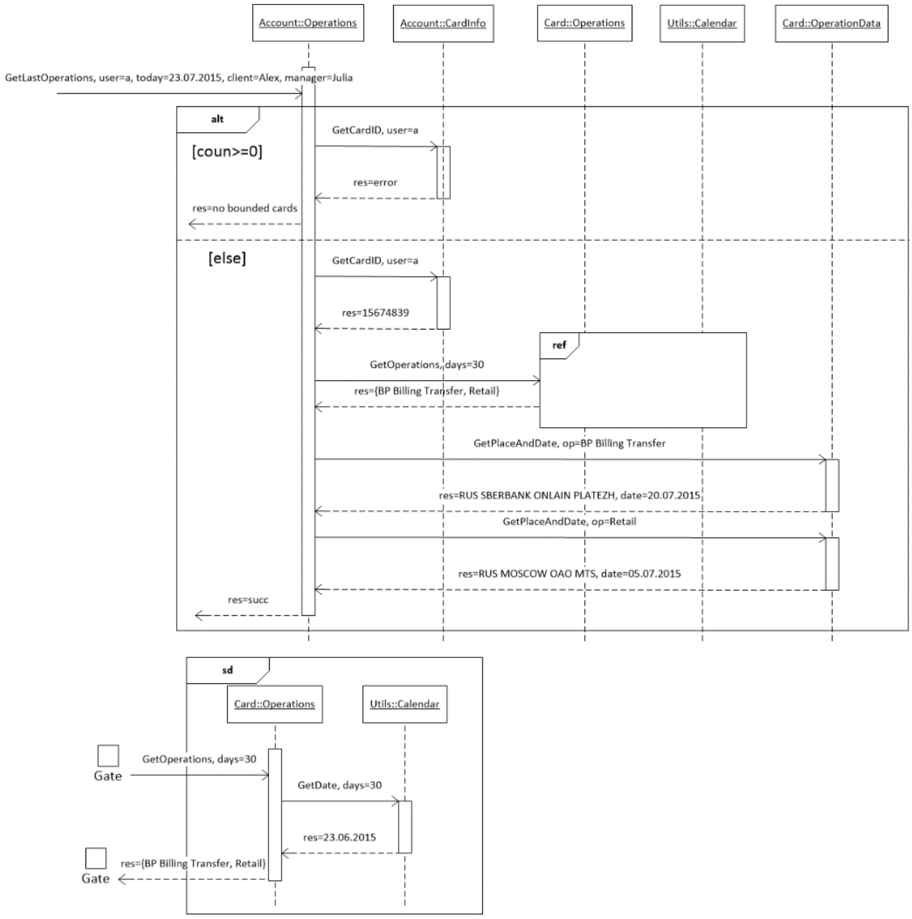


Fig. 3. Hierarchical UML sequence diagram using nested fragments

The most popular CASE tools are Sparx Systems' Enterprise Architect [8], IBM Rational Software Architect [9], Visual Paradigm [10], Altova UModel [11], MagicDraw [12], StarUML [13], ArgoUML [14]. There are both tools for end-to-end design and simple UML editors. The former include Sparx Systems' Enterprise Architect, IBM Rational Software Architect, Visual Paradigm, Altova UModel and MagicDraw, the latter include StarUML and ArgoUML. Beside that, the main aim of these tools is to get models from source code. Table 2 [15] contains CASE tools and program languages, for which models can be built. As we can see, none of these tools is able to infer models from the most popular languages used for developing

SOA information systems. Moreover, a SOA architecture can be developed with various programming languages. For example, some modules can be written in C#, 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.

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=Alex, manager=Julia	17:32:15 135
23	Account	CardInfo	GetCardID	REQ	user=a	17:32:15 250
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, Retail}	17:32:15 513
23	Card	OperationData	GetPlaceAndDate	REQ	op=BP Billing Transfer	17:32:15 589
23	Card	OperationData	GetPlaceAndDate	RES	res=RUS SBERBANK ONLAIN PLATEZH, date=20.07.2015	17:32:15 601
23	Card	OperationData	GetPlaceAndDate	REQ	op=Retail	17:32:15 638
23	Card	OperationData	GetPlaceAndDate	RES	res=RUS MOSCOW OAO MTS, date=05.07.2015	17:32:15 735
23	Account	Operations	GetLastOperations	RES	res=succ	17:32:15 822
25	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client=Alex, manager=Julia	17:40:18 345
25	Account	CardInfo	GetCardID	REQ	user=a	17:40:18 408
25	Account	CardInfo	GetCardID	RES	res=error	17:40:18 489
25	Account	Operations	GetLastOperations	RES	res=no bounded cards	17:40:18 523

Table 2. Programming languages of reverse engineering tools

Tools	Programming languages					
	PHP	C++	Java	Ruby	Python	VB	C#
Sparx Systems' Enterprise Architect	+	+	+	-	+	+	+
IBM Rational Software Architect	-	+	-	-	-	+	+
Visual Paradigm	+	+	+	+	+	-	+
Altova UModel	-	-	+	-	-	+	+
MagicDraw	-	+	+	-	-	-	+
StarUML	-	+	+	-	-	-	+
ArgoUML	-	+	+	-	-	-	+

There are some works, such as [16], [17], [18], [19], where approaches are applied for building UML sequence diagrams from program system execution traces (dynamic approaches). One of related works [16] analyzes one trace using a meta-model of the trace and a 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 option. However, program systemsloggingdoesnotusuallyincludethisinformation,so it is necessary to change source code to apply this approach. In opposite to this approach, our approach recognizes fragments as conditions based on traces' difference.

There is a dynamic approach to build a UML sequence diagram based on multiple execution traces in [18]. The authors apply an approach to build a Labeled Transition System (LTS) from a trace and an algorithm to merge some LTSs into one. After that, the LTS is transformed into a UML sequence diagram. In opposite to this approach, we propose not to use other data structures to represent traces and merge them. We propose to map traces onto a UML sequence diagram directly without intermediate models, which is more efficient.

In [19] the authors pay more attention to analysis of derived models. They describe an approach briefly, without details. They mention that diagrams of one trace are merged into one UML sequence diagram. However, there is no mathematically strict definition of a trace or a UML sequence diagram and it is not clear how they merge several diagrams.

The rest of the current paper is organized as follows. Section 2 gives definitions. Section 3 introduces our approach to mining UML sequence diagrams. Section 4 discusses results of some experiments on deriving models with the help of the developed tool. Section 5 concludes the paper and gives directions for further research.

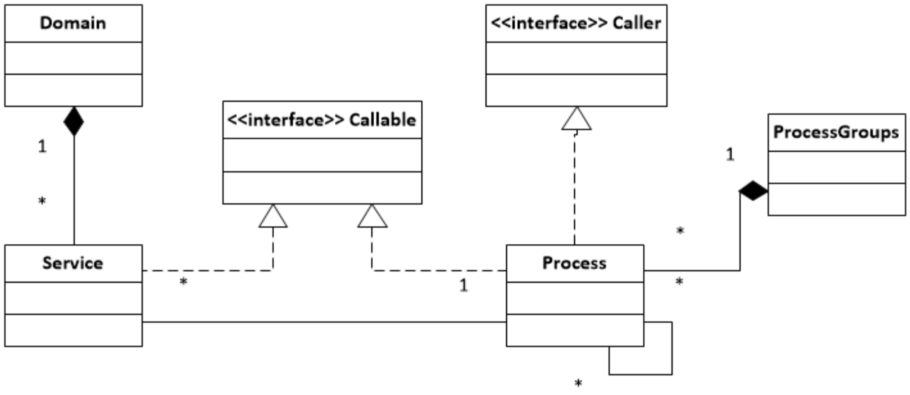


Fig. 4. Meta-model of a SOA system

2. Preliminaries

Definition 1. (Event log) Let E be a set of events. An event is a tuple $e = (a_1, a_2, \ldots, a_n)$, where n is a number of attributes. $\sigma =< e_1, e_2, \ldots, e_k >$ is an event trace (i.e. an ordered set of events which normally belongs to one case). $Log = \mathrm{P}(E)$ is an event log which is a multi-set of traces.

In the paper, we consider primarily event logs written by SOA information systems. The logs have a structure according to a SOA system standard. A meta-model of such a system is depicted on Fig. 4. The model complies with a Service Oriented Architecture standard (Fig. 5) proposed by Object Management Group [20].

We introduce a formal definition of a UML sequence diagram as follows.

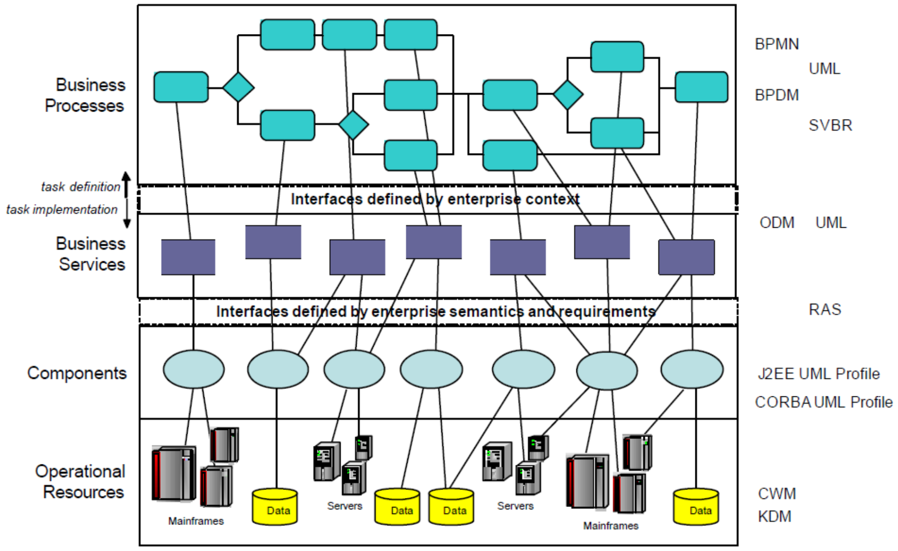


Fig. 5. Service-Oriented Architecture structure

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

- L is a set of lifelines, they represent objects whose interaction is shown on the diagram.
- A is a set of activations (emit and take messages) mapped onto lifelines. $A \subseteq (L \times T \times T)$
- T is time, it goes from the top of the diagram to the bottom. $\forall t \in T$, $\tau(t) = y$, where $y \in \mathbb{Z}$

- M is a set of messages (call and return) with its parameters and is ordered by time. $M \subseteq \left((A \cup Ref) \times T \times P \times (A \cup Ref)\right), m \in M: m = (a_1, t, p, a_2)$, where $a_1 \in A \cup Ref, t \in T, p \in P, a_2 \in A \cup Ref$
  $$a_1 = (l_1, t_{11}, t_{12}), a_2 = (l_2, t_{21}, t_{22}): t_{11} < t_{21}, t_{11} < t_{12}, t_{21} < t_{22}$$
- P is a set of parameters of messages.
- Ref is a set of *ref* fragments which group lifelines and hide their interaction. The interaction is shown on another diagram.
- $\delta: U_{SD} = (U_{SD1}, U_{HSD} | L^{'} \subseteq L, L_1 \subseteq L$
  $A^{'} \subseteq A, A_1 \subseteq A, A_1 \cap A^{'} = \emptyset$
  $M^{'} \subseteq M, M_1 \subseteq M, M_1 \cap M^{'} = \{m = (a_1, t, p, a_2) | a_1 \in A_1, a_2 \in A'\}$
  $P^{'} \subseteq P, P_1 \subseteq P, P_1 \cap P^{'} = \{p | m = (a_1, t, p, a_2),$
  $a_1 \in A_1, a_2 \in A^{'}, p \in P_1, p \in P'\}$
  $Ref^{'} \subseteq Ref, Ref_1 \subseteq Ref, Ref_1 \cap Ref^{'} = \emptyset)$,
  where:
  $U_{SD} = (L, A, M, T, P, Ref)$ – a detailed diagram.
  $U_{SD1} = (L_1, A_1, M_1, T, P_1, Ref_1)$ – a diagram with *ref* fragment.
  $U_{HSD} = (L^{'}, A^{'}, M^{'}, T, P^{'}, Ref^{'})$ – a nested diagram.

3. Approach to balance between abstraction and detalization

We propose an approach to mining UML sequence diagrams from an event log with a various degree of detalization. The approach consists of three steps derived one from another. It is necessary to map attributes of the log onto elements of a diagram prior to begining a mining procedure. Some mapping functions are therefore needed. First, it is necessary to define which interaction of SOA components (*Services*, *Processes*, *Domains* etc.) must be depicted on the diagram. Function α (1) maps events of the log with their attributes onto lifelines of diagrams. It allows choosing attributes to be represented on the diagram as lifelines.

$$\begin{gathered} E = (e_1, e_2, \ldots, e_k), k - a\ number\ of\ events \\ \alpha: U(E) \to L \end{gathered} \quad (1)$$

3.1. Mapping log attributes onto UML sequence diagram components

The first step allows getting diagrams with different abstraction levels by choosing log attributes for mapping onto lifelines and attributes for mapping onto parameters. To map attributes onto lifelines function α is used. Values of attributes *Domain* and *Service* are mapped onto composite lifeline objects with head names such as “Domain::Service/Process” on Figure 1. Also, function γ (2) is introduced for mapping attributes onto message parameters. *Operation* and *Payload* attributes are mapped onto messages parameters on Figure 1 such as “Operation, Payload”.

$$\gamma: U(E) \to P \tag{2}$$

The diagram depicted on Fig. 1 demonstrates interaction of services. The model represents one of the possible configurations of abstraction for the event log in table 1. For example, another possible configuration includes *Service/Process* and *Operation* attributes as diagram objects. Choosing such attributes allows inferring diagrams with different abstraction levels.

3.2. Merge of diagram components

On Figure 1 we see that the last two invocations of *GetPlaceAndDate* function are almost equal except for operation parameters. The second step of our approach performs merging some parts of a diagram. We propose to merge similar parts by using regular expressions. A regular expression contains a common part of a number of merged parts. The approach allows reducing the size of a model by merging similar parts. It increases generalization of the model.The approach involves a Cartesian square of a log with filtering. Function β (3) is used to map a filtered Cartesian square of the log on the set {1, 0} so that the element of the square is a pair “event” - “event from a set of next events”. If the pair satisfies a regular expression then it is marked as 1, otherwise as 0. We introduce η (4) to compare elements of the square. η considers events as equal to each other if their corresponding attributes are equal. In this case, attributes are equal if they can be matched as a single regular expression. Functions α and γ are used in this approach for mapping event attributes onto UML sequence diagram elements. There is also introduced function ξ (5) which determines a family of messages that are satisfied with pair event attributes. A message can be just a value of attributes or a regular expression applicable to single event attributes.

$$\beta: E \times E \to \{0,1\} \tag{3}$$

$$\begin{gathered} e_i = (a_{i,1}, a_{i,2}, \dots a_{i,n}) - an\ event\ with\ n\ attributes \\ (e_1, e_2) \in E \times E \\ \widetilde{e_1} = (a_{1,1}, a_{1,3}, \dots, a_{1,p}) - an\ event\ with\ p\ sample\ attributes, p < n \\ \widetilde{e_2} = (a_{2,1}, a_{2,3}, \dots, a_{2,p}) - an\ event\ with\ p\ sample\ attributes, p < n \\ \eta: \widetilde{e_1} = \widetilde{e_2} \Rightarrow a_{1,1} = a_{2,1} \& a_{1,3} = a_{2,3} \& \dots \& a_{1,p} = a_{2,p} \end{gathered} \tag{4}$$

$$\begin{gathered} \forall m \in M \exists \tilde{e} \in E \times E: \xi(\tilde{e}) = m \ \& \ \beta(\tilde{e}) = 1, \\ M - set\ of\ messages \end{gathered} \tag{5}$$

If one looks at the example introduced above on Figure 2, the diagram is obtained through applying this function and regular expressions. It is noticeable that two invocations of operation *GetPlaceAndDate* are merged in one invocation with regular expressions in message parameters. Regular expression “.*” means that any sequence of symbols can be inserted instead of this expression. It is also possible to merge lifelines by using regular expressions. It can be useful if class A is invoked only by class B; so, these classes can be merged into one lifeline.

3.3. Mining a hierarchical UML sequence diagram using nested fragments

One of the ways to represent a complex model is creating a hierarchical model. The UML standard [1] allows us to divide a complex diagram into more abstract and detailed models interacting through *gates*.

In order to define a hierarchy in a UML sequence diagram we introduce a definition of a selection criterion as follows. The definition of a hierarchical UML sequence diagram is given in Definition 2.

Definition 3. (Selection criterion) Let k be a number of hierarchical levels and RE be a regular expression defined in [21] with an added symbol “.” as an any symbol designation. Then, $c =< c_i | c_i \in RE >$, $c_i$is a selection criterion of events for i-hierarchical level. $c = c_1 \cup c_2 \cup \ldots \cup c_k$and $c_1 \cap c_2 \cap \ldots \cap c_k = \emptyset$. The regular expressions defined in [21] as selection criteria are Boolean expressions because their abstract syntax includes Boolean operations.

The components of SOA systems described by a meta-model depicted on Figure 4 have hierarchical relationship with each other. According to the SOA model there is a hierarchy in L1 event log because processes invoke different subprocesses or services.

It is also possible to distinguish some technical sublevels from main level by applying regular expressions. We propose a previously defined step with regular expressions to group elements.

Each hierarchical level is able to be encapsulated into another level on a UML sequence diagram. We propose to use nested fragments labeled as *ref*,which is defined in [1]. It allows combining high-level and detailed views of diagrams at the same time.

For applying the approach, a number of hierarchical levels and selection criteria, which are defined in Definition 3, need to be specified. Function β defines whether two events can be grouped into a single sublevel. If events match a selection criterion then they are moved to a nested diagram. For this case, values of some attributes must be equal or match a single regular expression. Function δ (Definition 2) maps some part of a UML sequence diagram considered as nested on a separate UML SD. The mapping uses *interaction use*which is shown as a *combined fragment* with operator *ref* [1]. This fragment hides some details of a high-level diagram moved to a nested diagram while the referred diagram allows seeing the details.

On Figure 3, a hierarchical UML sequence diagram for event log L1 is depicted there. There is some elements' interaction on the high-level diagram and some interaction is abstracted as *ref* fragment and depicted on the nested one. A selection criterion used for building the diagrams is *"Operation=GetDate"* which defines a part to be abstracted.

4. Evaluation

This section discusses our evaluation of the approach presented in this paper.

4.1. VTM4Visio Framework

Microsoft Visio is a professional drawing tool for making business charts and diagrams. It also supports some of UML diagrams. Besides, Visio has reverse engineering of databases, but it does not support UML reverse engineering. One of its flexible features is that it can be expanded by add-ins. It is possible to use Visio SDK [22] for having access to a Visio object model. Thus, it is a good solution to implement our tool for visualizing results (UML sequence diagrams) of our mining algorithm.

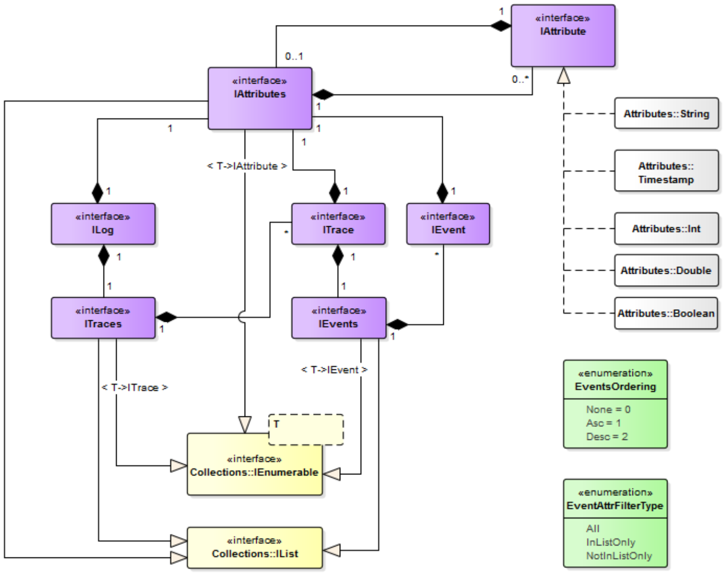


Fig. 6. Class diagram of Event log object model library

VTM4Visio is an extensible framework aimed at process mining purposing. It is implemented as an add-in for Microsoft Visio 2010. Our tool is implemented as a plug-in, which is supported by one of the VTM4Visio components called Plugin Manager.

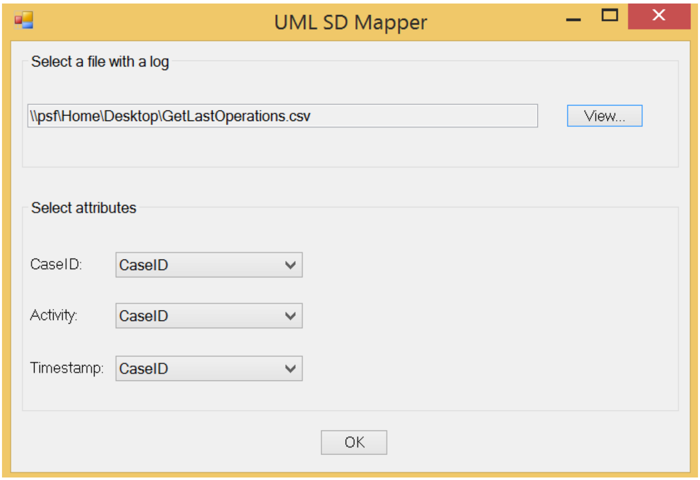
This framework was chosen because it provides useful instruments for accessing Microsoft Visio object models. It also has a convenient GUI.

4.2. Log pre-processing

It is necessary to have an event log in a definite format to apply our algorithm. A lot of information systems write logs in their own format. Our algorithm requires the event log to contain attributes which can be used as a case ID, timestamp and activity attributes. It is necessary to format and validate the event log before applying the algorithm.

4.3. Log library

Our algorithm requires an event log for mining a UML SD to be in some definite format. That is why it is necessary to have a library for working with event logs. We made the library and called it “Event Log Object Model Library”. Its UML class diagram is depicted on Fig. 6. The structure of our library is inspired by XES format [23]. It is not based on it but main components are taken from XES standard. We introduce special types such as *EvntsOrdering* and *EventAttrFilterType* for CSV and RDBMS-based event logs [24] because XML-based XES format is excessive. The library is written in C#. It is extensible, which allows working with different event log formats.


UML SD Mapper

Select a file with a log

\\psf\Home\Desktop\GetLastOperations.csv

Select attributes

CaseID: CaseID

Activity: CaseID

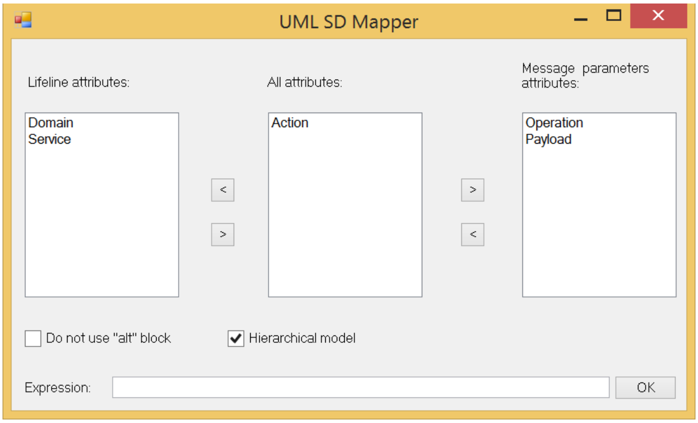
Timestamp: CaseID

Fig. 7. Event log configuration

4.4. Prototype implementation

Our prototype was written in C# programming language as a plug-in for VTM4Visio framework. The prototype allows configuring parameters for our approaches as CaseID, Timestamp and Activity, names of lifelines and messages' parameters, a regular expression through some GUI forms (Fig. 7 and8). The configuration for reading of event logs from a file is implemented as shown on Figure 7. The configuration of the diagram is implemented as shown on Figure 8. This GUI form allows setting different perspectives and a regular expression for merging diagram elements and, hence, specifying hierarchy.

The processing result of the event log in Table 1 is depicted on Fig. 1, 2, 3.


UML SD Mapper

Lifeline attributes: Domain, Service

All attributes: Action

Message parameters attributes: Operation, Payload

[ ] Do not use "alt" block
[✓] Hierarchical model

Expression:

Fig. 8. Diagram configuration

5. *Conclusion*

This paper proposes a method of reverse engineering of UML sequence diagrams from event logs of SOA information systems. It contains three approaches to balance high-level diagrams and low-level ones.

Our method is a dynamic analysis of software because it uses only event logs. This is an advantage since source code is not always available. In addition, our approaches do not use intermediate models of an event log representation. The proposed method 1) maps log attributes onto diagram components, 2) merges diagram elements based on regular expressions and 3) builds hierarchical UML diagrams using a *ref* fragment.

Work with event logs of real-life SOA information systems shows that it is necessary to mine diagrams not only from single- threaded event logs but also from multi-threaded ones. Thus, it is a direction of our future work. UML sequence diagrams do not always show parallel interactions properly. Thus, we are going to mine hybrid diagrams as UML sequence diagrams with a *ref* fragment, which abstracts parallel interactions and refers to UML activity diagramsillustrating parallel processes.

Acknowledgement

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and the Russian Foundation for Basic Research, project No. 15-37- 21103.

References

[1]. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, August 2011.

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

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

[4]. A. Rountev, 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.

[5]. A. Rountev. Static control-flow analysis for reverse engineering of uml sequence diagrams. In Proc. 6th Workshop on Program Analysis for Software Tools and Engineering (PASTE’ 05), pages 96–102. ACM Press, 2005.

[6]. P. Tonella, A. Potrich. Reverse engineering of the interaction diagrams from C++ code. pages 159–168. IEEE Computer Society, 2003.

[7]. E. Korshunova, M. Petkovic, M. G. J. van den Brand, 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.

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

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

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

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

[12]. MagicDraw. http://www.nomagic.com/products/magicdraw.html.

[13]. StarUML. http://staruml.io.

[14]. ArgoUML. http://argouml.tigris.org.

[15]. H. Osman, M. R. V. Chaudron. Correctness and completeness of CASE tools in reverse engineering source code into UML model. The GSTF Journal on Computing (JoC), 2(1), 2012.

[16]. L. C. Briand, Y. Labiche, J. Leduc. Toward the reverse engineering of uml sequence diagrams for distributed java software. IEEE Trans. Softw. Eng., 32(9):642–663, September 2006.

[17]. R. Delamare, B. Baudry, Y.L. 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.

[18]. T. Ziadi, M.A.A. da Silva, L.M. Hillah, 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.
[19]. Y.-G. Guéhéneuc. Automated reverse-engineering of UML v2.0 dynamic models. In Proceedings of the 6 th ECOOP Workshop on Object-Oriented Reengineering. http://smallwiki.unibe.ch/WOOR, 2005.
[20]. OMG. The OMG and Service Oriented Architecture, 2006
[21]. S. Owens, J. Reppy, A. Turon. Regular-expression derivatives re-examined. J. Funct. Program., 19(2):173–190, March 2009.
[22]. Visio 2010: Software Development Kit, 2010. https://www.microsoft.com/en-us/download/details.aspx?id=12365.
[23]. C. W. Günther and E. Verbeek. XES Standart Definition version 2.0, 2014.
[24]. S.A. Shershakov. VTMine framework as applied to process mining modeling, International Journal of Computer and Communication Engineering vol. 4, no. 3, pp. 166-179, 2015.

Метод автоматического построения иерархических UML-диаграмм последовательности с задаваемым уровнем детализации на основе журналов событий

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 диаграмм последовательности UML на основе журналов событий информационных систем с сервис-ориентированной архитектурой (SOA). Диаграммы последовательности UML — графические модели, подходящие для представления взаимодействий в гетерогенных компонентных системах, в частности, в набирающих сейчас популярность информационных SOA-системах. Описываемый метод использует трассы исполнения SOA-систем, представленные в виде журналов событий. Почти все современные информационные системы имеют возможность записывать результаты своей работы в журналы событий, которые используются в основном для процесса отладки. По сравнению с традиционными техниками автоматического синтеза моделей, которые требуют не всегда имеющийся исходный код для своей работы, наш метод для автоматического построения диаграмм последовательности UML работает только с доступными журналами событий и некоторыми эвристическими данными. Метод состоит из нескольких этапов построения диаграмм последовательности UML в зависимости от разной перспективы, заданной аналитиком.Они включают отображение атрибутов журнала событий на элементы диаграммы с возможностью задать уровень абстракции через параметры, группировку некоторых компонент диаграммы и построение иерархических диаграмм последовательности. Мы предлагаем группировать некоторые компоненты (сообщения и линии жизни) на основе регулярных выражений и строить иерархические диаграммы, используя вложенные фрагменты. Мы апробировали данный метод при помощи разработанного в виде плагина Microsoft Visio прототипа. Плагин строит диаграмму последовательности UML на основе заданного журнала событий в соответствии с набором настраиваемых параметров.

Ключевые слова: журнал событий; диаграмма последовательности UML; автоматическое выведение моделей; извлечение процессов.

DOI: 10.15514/ISPRAS-2016-28(3)-6

Для цитирования:Давыдова К.В., Шершаков С.А.Метод автоматического построения иерархических UML-диаграмм последовательности с задаваемым уровнем детализации на основе журналов событий.*Труды ИСП РАН,* том 28, вып. 3, 2016. стр. 85-102 (на английском). DOI: 10.15514/ISPRAS-2016-28(3)-6

Список литературы

[1]. OMG. OMGUnifiedModelingLanguage (OMGUML), Superstructure, Version 2.4.1, August 2011.

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

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

[4]. A. Rountev, 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.

[5]. A. Rountev. Static control-flow analysis for reverse engineering of uml sequence diagrams. In Proc. 6th Workshop on Program Analysis for Software Tools and Engineering (PASTE’ 05), pages 96–102. ACM Press, 2005.

[6]. P. Tonella, A. Potrich. Reverse engineering of the interaction diagrams from C++ code. pages 159–168. IEEE Computer Society, 2003.

[7]. E. Korshunova, M. Petkovic, M. G. J. van den Brand, 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.

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

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

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

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

[12]. MagicDraw. http://www.nomagic.com/products/magicdraw.html.

[13]. StarUML. http://staruml.io.

[14]. ArgoUML. http://argouml.tigris.org.

[15]. H. Osman, M. R. V. Chaudron. Correctness and completeness of CASE tools in reverse engineering source code into UML model. The GSTF Journal on Computing (JoC), 2(1), 2012.

[16]. L. C. Briand, Y. Labiche, J. Leduc. Toward the reverse engineering of uml sequence diagrams for distributed java software. IEEE Trans. Softw. Eng., 32(9):642–663, September 2006.

[17]. R. Delamare, B. Baudry, Y. L. 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.

[18]. T. Ziadi, M. A. A. da Silva, L. M. Hillah, 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.

[19]. Y.-G. Guéhéneuc. Automated reverse-engineering of UML v2.0 dynamic models. In Proceedings of the 6 th ECOOP Workshop on Object-Oriented Reengineering. http://smallwiki.unibe.ch/WOOR, 2005.

[20]. OMG. The OMG and Service Oriented Architecture, 2006

[21]. S. Owens, J. Reppy, A. Turon. Regular-expression derivatives re-examined. J. Funct. Program., 19(2):173–190, March 2009.

[22]. Visio 2010: Software Development Kit, 2010. https://www.microsoft.com/en-us/download/details.aspx?id=12365.

[23]. C. W. Günther and E. Verbeek. XES Standart Definition version 2.0, 2014.

[24]. S.A. Shershakov. VTMine framework as applied to process mining modeling, International Journal of Computer and Communication Engineering vol. 4, no. 3, pp. 166-179, 2015.