Моделирование и анализ информационных систем. Т. 22, № 2 (2015), с. 175–190
Modeling and Analysis of Information Systems. Vol. 22, No 2 (2015), pp. 175–190

©Shershakov S. A., Rubin V. A., 2013

DOI: 2015\mais\22\1\1\175–190
UDC 519.682.6+004.43

System Runs Analysis with Process Mining

Shershakov S. A.[1], Rubin V. A.

Received April 15, 2013

Information systems (IS) produce numerous traces and logs at runtime. In context of SOA-based (service-oriented architecture) IS, these logs contain details about sequences of process and service calls. Modern application monitoring and error tracking tools provide only rather straightforward log search and filtering functionality. However, "clever" analysis of the logs is highly useful, since it can provide valuable insights into the system architecture, interaction of business domains and services. Here we took runs event logs (trace data) of a big booking system and discovered architectural guidelines violations and common anti-patterns. We applied mature process mining techniques for discovery and analysis of these logs. Process mining aims to discover, analyze, and improve processes on the basis of IS behavior recorded as event logs. In several specific examples, we show successful applications of process mining to system runtime analysis and motivate further research in this area.

Keywords: process mining, software process, software runtime analysis

For citation: Shershakov S. A., Rubin V. A., "System Runs Analysis with Process Mining", *Modeling and Analysis of Information Systems*, **22**:2 (2015), 175–190.

On the authors:
Shershakov Sergey Anreevich, orcid.org/0000-0001-8173-5970, research fellow,
National Research University Higher School of Economics,
20 Myasnitskaya str., Moscow, 101000, Russia, e-mail: sshershakov@hse.ru

Rubin Vladimir Aleksandrovich, orcid.org/0000-0001-8176-2426, PhD, CEO,
Dr. Rubin IT Consulting,
60599, Frankfurt am Main, Germany, e-mail: vroubine@gmail.com

Acknowledgments:
[1]This work is supported by the Basic Research Program of the National Research University Higher School of Economics.

Introduction

Processes are all around us. Processes accompany data and are accompanied by data. As processes becomes more complex, the information systems accompanying them become more complex too. Thus, the complexity of modern software systems containing millions of lines of code and thousands dependencies among components is extremely high. Supporting such systems requires involving new techniques and tools responding to the challenge of scale and complexity of modern information systems.

Almost all modern software systems trace data at runtime. Information about failures and exceptions is always traced, but also particular data about system execution, system state, called services and so on. In most cases, traces are the only possibility to understand the behavior of a productive system, which usually runs in a separate production environment and can not be debugged.

Process mining is a discipline, basic research and practical purpose of which is to extract process models from data of a special type, that is *event logs* [1]. The traditional areas of the process mining application include business processes (management), social processes, such as medicine or management of municipalities, technological processes. The Process Mining Manifesto released by the IEEE Task Force on Process Mining [2] in 2011 is supported by more than 50 organizations, more than 70 experts contributed to it. One particularly interesting research area is Software Process Mining (SPM), that deals with extracting models of processes related to design, development, debugging and support of software from event logs containing data that software systems trace at runtime [3, 4, 5].

Speaking of ISs, one can distinguish a separate class of component-based information systems, the main feature of which is the structure in which the expansion of functionality of the IS is achieved by adding special components. One of the most growing and rather young approaches to component design is Service-oriented architecture (SOA) [6]. For such systems, logged data can represent traces of interconnection of their components such as *processes* and *services*.

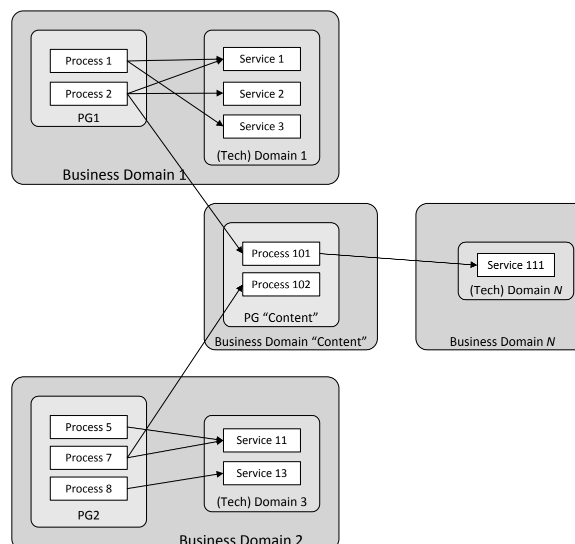

Fig 1. "Domains-Processes-Services" architecture scheme.

A large European touristic Computer Reservation System (CRS) will be considered as an example of process-driven system based on SOA architecture. A CRS system we are studying is a distributed SOA-based software containing different client (thin and rich clients) and server components. The system contains *processes* as a basic task of the underlying business logic. Each instance of these processes can be considered as a start point for yet another use case studied with using process mining technics. Processes orchestate *services*. There are a lot of services that can be considered as basic working elements of the system.

There are several *business domains*. Each of them consists of processes and technical services. Each process and service belongs to only one *business domain* according to their business purposes. For example, there are *"booking"* and *"accounting"* business domains. The processes are combined into *process groups* (PG) and the services are grouped into *technical domains* (TD)[1] (Fig. 1). Coming back to the earlier example, there are PG and TD named "booking". Technically all the processes and services are implemented as Java interfaces (correspond to *InterfaceName*) organized as Java-packages (correspond to *PackageName*). Each process and service send and receive messages (correspond to *OperationName*), and each specific message is logged as an individual trace to a log.

We have a set of logs tracing interconnections of those processes and services and containing a lot of different aspects of data. By examining certain *views* of the data we can look at the system from different perspectives. Each specific view can represent some aspect of the system. We distinguish such aspects as *Control Flow Aspect*, *Data Perspective* (also can be treated as *Informational Aspect*), *Organizational Aspect* and *Infrastructural context*. Here we are primarily focusing on the control flow aspect, hovewer at subsequent steps the data aspect is also considered.

Architecture teams normally define a set of rules about how individual processes can invoke other (sub)processes and services from different domains. As an example there are a lot of restrictions about invoking services or processes from other services. These rules and restrictions are also known as *Architectural Guidelines or Architectural Conventions*. Such rules can be complicated enough in order to be simply tested at compile-time, or during the module testing or at runtime. This paper deals with detecting some architectural violations in the model discovered from the event logs derived from a running system.

In the area of software engineering there are a set of well-known *architectural principles*, such as loose coupling, separation of concerns, etc. Our goal here is not only to detect the violations of architectural conventions of the company, but also of the violations of these common architectural principles. We consider the ability to make some kinds of models of a given software system by using process mining approaches.

The rest of this paper is organized as follows. Section 1. presents data logs and tools used for logs analysis. It also contains three examples of violations of architectural conventions and principles detected by using process mining techniques. Section 2. discusses some related work and section 3. summarizes the work done and discusses future work.

[1]We especially refer to them as *technical domains* in order not to mix them with the *business domains*.

1. Experience report

1.1. Log and Tools

An *event log* is the starting point for almost any process mining research.

The subsystems of the CRM system maintains an ability for tracing of all the necessary processes and services communication. The initial invocation of any process, e.g. made by a rich-client application, is accompanied by allocation of a special *invocation id* (also reffered to as `InvID`).

Both processes (PR) and services (SV) receive a request message (RQ) as input and a return response message (RS) or an exception (SE) as output. These messages are logged. On the log level the traces are written in an XML format. A sample of such XML log is presented in the Listing 1. Each trace is included into a `tracingevent` element containing several sections that describe the trace and contain additional data that could be used for deeper analysis. For the first step we are interested in the following data: an *invocation id*, a *message recipient* (given by its full-qualified name including *business domain* (we also refer to it as `UnitName`), *package name*, *interface name*, and operation name).

Then, we also consider two very important fields. An *event timestamp* is first. Describing the second one we have to mention that a trace event contains a *payload*, given in the form used by processes and services to exchange data between each other. Payloads are presented in the form of an XML-based piece of data and can be used to analyze a model made with this log from a data perspective. During this work we are considering only two attributes of the payload: size of its data and its hash sum to identify whether the payload is changed from call to call.

Listing 1. Event trace for resolveLocationByAlias service call

```
<tracingevent>
<InvocationIdentifier>
<id>6390414390044799821</id>
<time>Fri Dec 20 00:11:48 CET 2013</time>
...
</InvocationIdentifier>
<TransactionContext>
...
</TransactionContext>
<log4j:event logger="tracer.de.der.pu.domains.geo.
location.LocationQuery.resolveLocationsByAlias"
timestamp="1387494715759" level="INFO"
thread="WorkerThread#8[10.10.10.42:57387]">
<log4j:message><![CDATA[Request]]></log4j:message>
</log4j:event>
<jboss>
...
</jboss>
<payload>
<![CDATA[
```

```
<ServiceRequest>
...
</ServiceRequest>
]]>
</payload>
</tracingevent>
```

In this paper we are considering a log produced by the system during a period of 24 hours. The size of all XML files of the log is approximately 10 GB.

1.1.1. Log traces "normalization" with an RDBMS approach

The event log has a complex structure and contains heterogenous data covering different aspects, such as control flow, organizational aspect (including user information, authentication and authorisation info), infrastructural context and other resources. It is necessary to represent them in a formalized form.

We decided to use RDBMS as a log representation because it provides the possibility to make various data views for different types of analysis depending on each specific aspect containing in our log. Also, it is an instrument for effective manipulation of big amounts of log data.

Creating a view on data involves two operations. First, *projection* of a data set (that could be a regular table as well as a joint of a number of tables) performs selecting only a particular subset of all *attributes* (table columns) that correspond to specific aspects. Second, *filtering* of the dataset provides only those records (table rows) which match some selection criteria.

Converting a text-based logs into a well-defined RDBMS allows us to obtain a desirable data projection on a specific aspect in a very natural way just by pointing all the necessary atributes out. At the same time it is very simple to obtain a filtered subset of events by specifying arbitrary filtering criteria [1, 261]), with subsequent export as frequently as it is necessary. Using indexes prepared in a proper way allows performing such operations quickly enough even on a very big amount of data. Similar approach was effectively used in other case studies [7].

We decided to use SQLite database engine to store and manipulate the data. A part of a relation diagram for the log database is depicted on Fig. 2. Here, we distinguish two main parts of physical storage. One part consists of declarations of interfaces including PR/SV type, business domain unit name, package name, and interface name. The part is represented as a table with `Interfaces` name. All other data corresponding to the log's trace events are represented by `TracingEvent` table. For our cases we used joint records with `Interfaces.ID` and `TracingEvents.Interface_ID` fields as keys. We refer to those as `FullData3` view below.

An example of records corresponding to the trace described by Listing 1 is given on Fig. 3.

We developed a tool performing parsing of XML source files and adding parsed data to a database. For the given 10 GB log (discussed in sect. 1.1.), it takes at least an hour to convert all the XML data to the DB format using a laptop with an Intel® i3 @ 2,4 GHz comparable processor on the board[2].

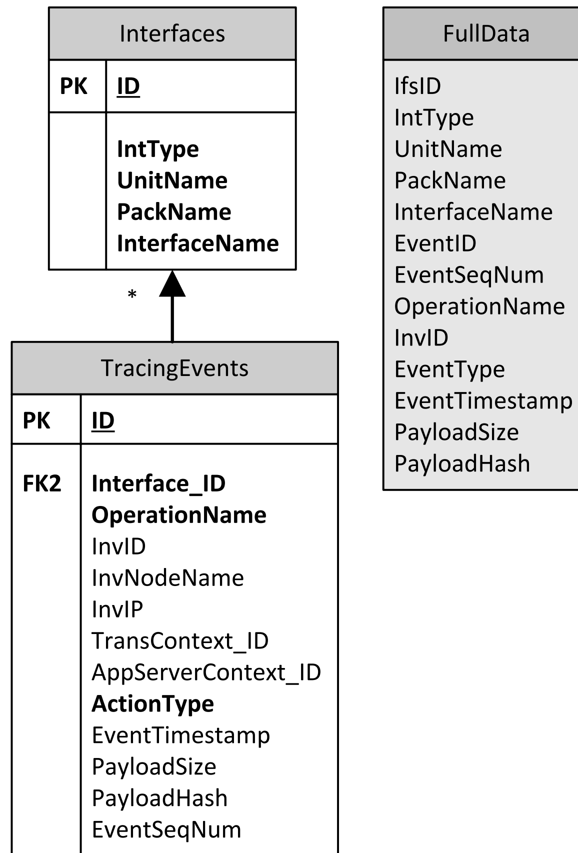

Fig 2. A relation diagram of a log DB.

RecNo	ID	PDType	UnitName	PackName	InterfaceName
1	110	SV	geo	location	LocationQuery

RecNo	ID	Interfa-ce_ID	Operation Name	InvID	EventType	Event Timestamp	Payload Size	Event Seq Num
1	170097	110	resolve Locations ByAlias	639041439 044799000	RQ	1387494715759	393	11

Fig 3. DB records for a trace from Listing 1 (table `Interfaces` at top and table `TracingEvents` at bottom).

"Normalizing" the log by converting it to the DB allows making a rather compact representation of source data. Thus, for the 10 GB containing approximately 500000 traces there is just 60 MB of data given in SQLite DB ver.3 format. Moreover, reexporting a full set of (merged) data from the DB to an external CSV-text file takes just a couple of seconds.

As shown further, using SQL queries for extraction of a precisely needed data projection is significantly efficient. Thus, the SQL-based approach is one of the basic tools used in this work.

[2]Among the factors significantly affecting the converting speed, one can distinguish a hashing algorithm.

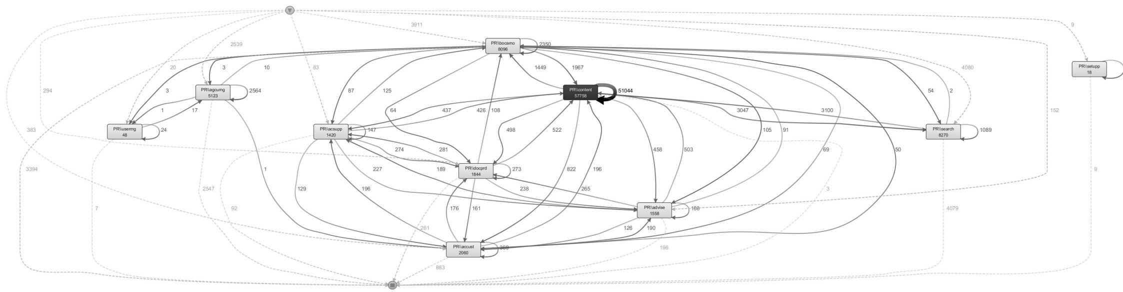

Fig 4. Fuzzy diagram representing relations between processes, $AP(100, 100)$.

1.1.2. Process mining specific tools

Today, there is a set of freeware and commercial process mining tools. Examples of these tools include ProM [8], which is a widely used research workbench containing more then 600 plugins. Disco is another well known process mining tool [9].Disco is based on Fuzzy miner which was initially implemeted in ProM [10].

In [11] we show an RDBMS-based approach to automation process mining experiments with rapid creation of datasets using VTMine Framework, which is our own tool for modeling and conducting process mining experiments. DPModel [12, 13], a graphical language for automation of experiments in Process Mining, underlies the tool.

We decided to mine a process model of our system as a *fuzzy model* that can be discovered with Disco. It has a good usability and good performance in processing large event logs. Moreover, it can perform advanced filtering on the basis of Fuzzy miner.

In the further sections, with the help of process mining techniques we analyze the logs in order to find software architectural violations and inconsistencies.

1.2. Example 1: Architectural Violations

In Introduction we have already discussed Architectural Conventions and Architectural Violations. Invoking some processes from other processes can be an example of such violation. In our CRS the only processes from “content” PG (and, respectively, business domain) can be invoked from other processes. Calling any other processes from different domains are not allowed. In this section we investigate the presence of exactly this violation.

We can detect such forbidden calls from different domains by investigating a fuzzy diagram.

The very important assumption we have to make is that there are no asynchronous messages calls between processes and services in the scope of the set of events related to one specific case.

1.2.1. Model in Disco

Now we are looking at Control Flow Aspect for discovering forbidden services calls. In order to discover process calls violations we build a Disco model depicting PG/Domain relationships.

Although all event attributes can be used for process mining, we focus on the two attributes that are mandatory for process mining. Any event should refer to a *case* (i.e., a process instance) and an *activity*. Moreover, events related to a particular case should be ordered. Thus, for performing data import to Disco from a RDBMS one need to point out which attributes are used for indicating *case*, *activity* and *timestamp* (for ordering reasons). Other attributes are indicated for Disco as *resources* that can be used for filtering.

In this work, we are using *invocation identifier* given as `InvID` attribute as a *case* ID and `EventTimestamp` attribute as a *timestamp*. Choosing attributes that play a role of *activity ID* is closely related to the studying case. For this case, we use a pair of attributes `IntType` (which can be either `PR` or `SV` for processes/PG and services/domains respectively) + `UnitName` (representing the name of PG or domain and corresponding to business domain) as an *activity ID*.

There are two parameters of Disco's Fuzzy Miner: (1) number of activities, and (2) number of paths shown on a fuzzy map. They are used to make a quick filtration by the criteria of frequency of activities and paths being met while making the map. We refer to both of these parameters as a pair, e.g. $AP(100, 100)$, where the numbers in the brackets are the percentage of activities and paths, respectively.

There is a fuzzy map with $AP(100, 100)$ produced by Disco depicted on Fig. 4. This map is a graph that demonstrates the relations between both PGs and domains transparently given through the messages sent by their interfaces. The map was built from a filtered dataset in order to restrict the model to view only **processes**. For this very purpose we can apply Disco *attribute filter* selecting only the activities containing `PR\\` we would like to observe.

The map contains only 10 vertices and a number of arcs which is not so big so we are able to track individual relations between each pair of processes. Vertices color coding shows us how many messages are sent to individual vertices. Thus, `PR\\content` is represented as a dark blue vertex showing us that it has many more incoming messages (57758) than the others. This is because process `content` plays a special role in the whole system (as it was mentioned above). In other words, it contains common "routines".

At the same time we can remark the presence of pairs of income/outcome edges between other PGs as well as those which contain only few traces. The direct communication of the PGs with each other represent examples of violations of the architectural conventions.

To investigate this violation more precisely we will first make a more detailed filtration by using Disco. As a concrete example one can consider inappropriate direct relations between two PGs — `bocamo` and `search`. We use the so-called *Follower Filter* that lets us define a couple of activities with a restriction of how they should follow one another. Thus we set a filter a way that `search` activity follows `bocamo` activity (Fig. 5).

We can see that there are 54 messages from `bocamo` to `search`. Here, we understand a message as a pair of events, first of which corresponds to `PR\\bocamo` activity and is directly followed by second one corresponding to `PR\\search` activity. Such pairs are found in a number of cases, and some of the cases contain such pairs several times. Depending on event activities and their order in the cases the latter are grouped into so-called *variants*.

By using Disco case statistics we can see there are 10 different variants of traces

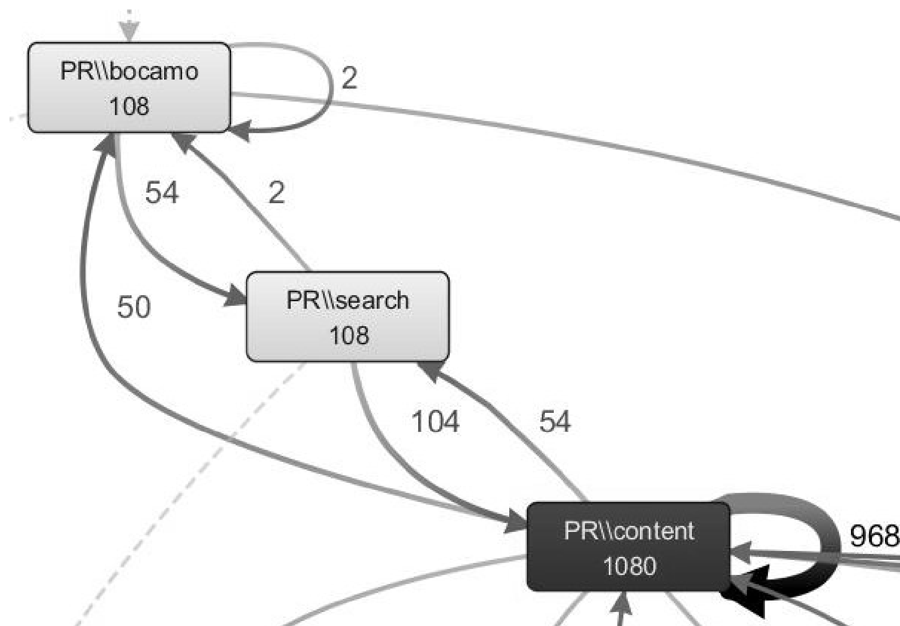

Fig 5. Correlation between activities `search`, `bocamo`, and `content`.

including from 1 to 40 cases per variant. For example, variant 2 contains 5 cases, from which we have detected one (with invocation id 190248972438317148) when bocamo process `quoteItemFromSearch` calls search process `getPricesForRouting` and it is forbidden. See a sequence diagram in Fig. 7.

PG	Operation	Type
bocamo	quoteItemFromSearch	RQ
search	getPricesForRouting	RQ
content	priceAvailability	RQ
content	priceAvailability	RS
content	calculateFees	RQ
content	calculateFees	RS
search	getPricesForRouting	RS
content	determineBestProduct	RQ
content	determineBestProduct	RS
content	checkBookingUnitRestriction	RQ
content	checkBookingUnitRestriction	RS
content	getExternalReferences	RQ
content	getExternalReferences	RS
content	performQuote	RQ
content	performQuote	SE
bocamo	quoteItemFromSearch	SE

Fig 6. Subset of events of case 190248972438317148 related to PGs `bocamo`, `search` and `content`.

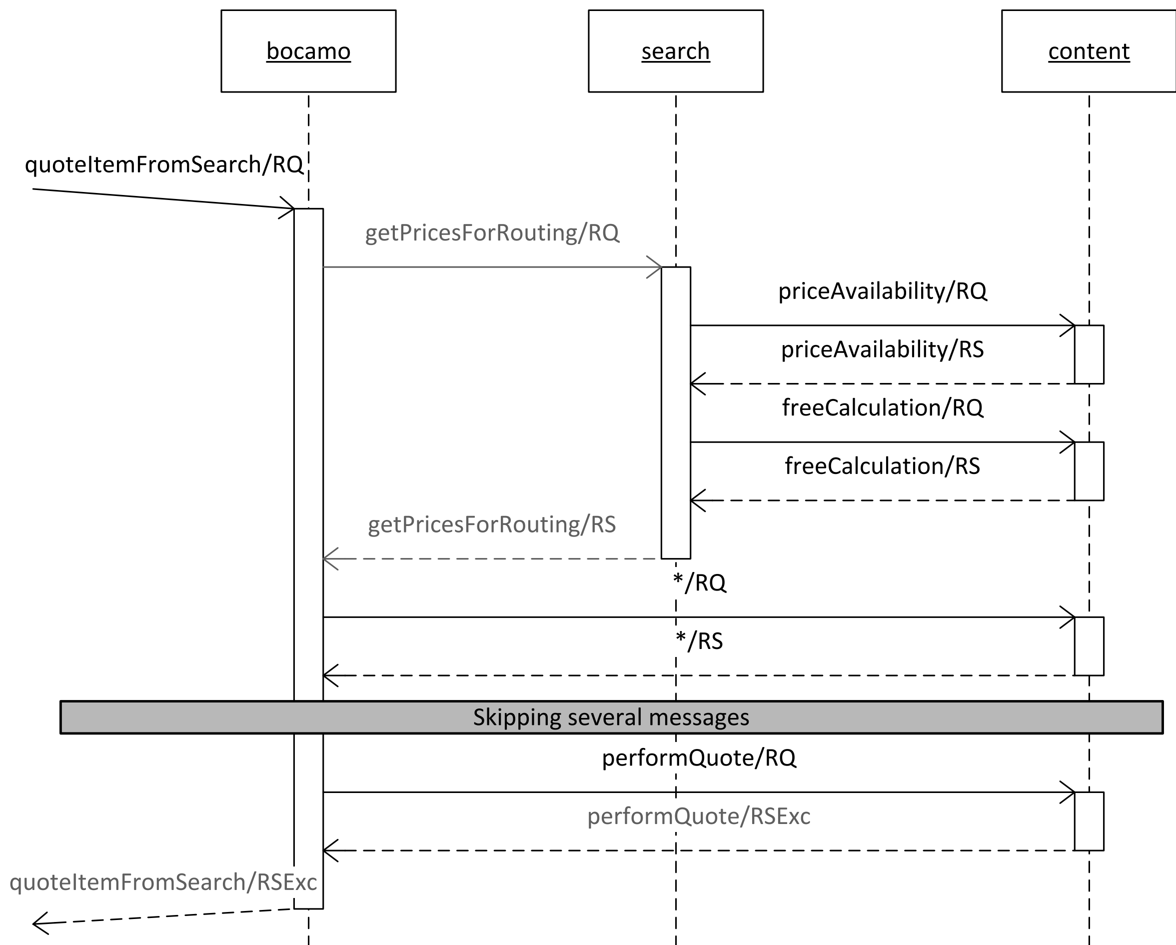

Fig 7. UML diagram corresponding to case 190248972438317148.

On Fig. 7, a manually created UML sequence diagram based on the data of the 16-event subset above is depicted. It can be used as a convenient tool for reporting detected violations to developers.

1.2.2. Generalization in SQL

In order to obtain a collection of all the similar violations we created a set of SQL queries. These queries are based on the idea of a square Cartesian product of the set of all the traces. As a result of such product and futher filtering by the same case ID for both parts of the product, we obtained a collection of so-called N-step relations between all the events. Each N-metric is calculated as a difference between relative time positions of each events in one given case. Among the pairs from this relation there are some pairs that contain 1-step relations. Every such relation corresponds to a couple of events among which the first event is directly followed by the second one. Finally, we have to filter the resulting dataset by setting desirable restrictions, such as:

1. both parts of a pair contain different PGs;
2. no part of a pair contains `content` PG;

3. both parts of a pair contain RQ messages, so we are interested only in “request $\rightarrow$ request” processes with different PGs that is actually forbidden;

As a result of excecuting the SQL query we obtain a set of 2031 traces that produce system architectural violations. The set is mapped to a set of cases containing these events. By constructing one more query grouping cases by `InvID` attribute, we obtain a set of cases containing at least one violation. So, we have found exactly 1789 cases that violate discussed architectural violations. The maximum number of forbidden 1-step relations per case is 3 (for 34 cases). Typical number is just 1 per a case (1581 cases).

1.2.3. Benefits

We have found the violation and reported an appropriate bug to the developers with a view to creating a ticket in a bug-tracking system. Using a notion of UML sequence diagrams may help showing to developers an explicit fragment of the system where the violations are detected.

1.3. Example 2: Antipattern “Unnecessary repeating calls”

Let us now look for more oddities by applying the same techniques.

Thus, we set `OperatinName` in conjuction with `IfsID` and `EventType` attributes as activity. Looking at the statistics panel in Disco one can conclude that among the most often used activities there is `getAgency` activity in `agcumg` domain: there are 7642 RQs and RSs instances. Let us consider it more precisely. Applying an attribute filter for this activity name and marking it as “mandatory” we obtain a fuzzy map which, depending on $AP()$ value, contains a small or large number of activities, but under the condition that all these activities are related to `getAgency`. Thus, there is a number of cases containing `getAgency` as a repetitive activity. Among the latter, `getInvoiceListByReservationNumber` in `accust` PG can be identified.

Such cases are characterized by the fact that they have repetitive messages `getInvoiceListByReservationNumber` $\leftrightarrow$ `getAgency` with the same payload (given by its size and hash sum), that can lead to the fact of presence of multiple excessive process invocations. This can signify to us a **bad implementation example** where payload data must be cached or stored instead of their repititive obtaining.

Let us consider a technique for detecting such patterns. The first step is creating of an auxilary view `ActivitiesCountPerCases1` that for each case contains a set of activities (identified separately for RQ/RS) with a number of their repetitions:

```
CREATE VIEW [ActivitiesCountPerCases1] AS
SELECT *, COUNT(*) AS ActsNum
FROM FullData3
GROUP BY InvID, IfsID, OperationName, EventType;
```

Then we can fix a certain threshold number of activities' repetitions and select only the instances of repetitive activities (make another auxilary view `Activities3PerCase`):

```
CREATE VIEW [Activities3PerCase] AS
SELECT InvID, IfsID, OperationName
```

```
FROM ActivitiesCountPerCases1
WHERE ActsNum >= 3;
```

Next step: retrieving full attributed data for all the activities satisfying the condition above (another view EventsBy3ActivitiesPerCase1):

```
CREATE VIEW [EventsBy3ActivitiesPerCase1] AS
SELECT *
FROM Activities3PerCase AS L INNER JOIN FullData3 AS R
ON (L.InvID = R.InvID AND L.IfsID = R.IfsID AND
L.OperationName = R.OperationName)
GROUP BY ID;
```

Finally, we form a resulting set according to identical values of PayloadSize and PayloadCache:

```
SELECT *, COUNT(*) AS NumOfRepeatedPayload
FROM EventsBy3ActivitiesPerCase1
GROUP BY InvID, IfsID, OperationName, EventType,
PayloadSize, PayloadCache
```

By filtering NumOfRepeatedPayload attribute by number of maximum allowable repetitions we obtain all the events (and consequently cases) with excessive payload transmitting.

1.4. Example 3: Antipattern "Cross-cutting concern"

According to the statistics, getConfiguration in smerge domain is the most often invoked service (aprx. 12 % of all traces). Let us observe how smerge domain (both as business and techical means) is related to other PGs.

First, we set a pair of attributes IntType and UnitName as activity. In order to eliminate irrelevant cases we add some filters: (1) selecting only RQ traces, (2) selecting only processes from all domains and services in smerge domain, (3) marking getConfiguration operation name as mandatory. The resulting fuzzy map is depicted on Fig. 8.

As we can conclude, smerge domain is actively "invoked" by 7 other PGs. Precisely, operation getConfiguration is invoked by the processes contained in these PGs. So, this operation can be considered as a so-called "Cross-cutting concern" and must belong to a dedicated domain (but not to smerge domain) or should be moved to context domain.

2. Releated work

There are various works on *software execution traces* and *runtime analysis*. In [14], two runtime analysis algorithms, a data race detection algorithm and a deadlock detection algorithm, are introduced to analyze Java programs. The concerned approach is based on the idea of *single* program execution and observing the generated run to extract various kinds of information. In contrast with this approach, process mining works with a set of traces, but not with only one trace, despite using algorithms.

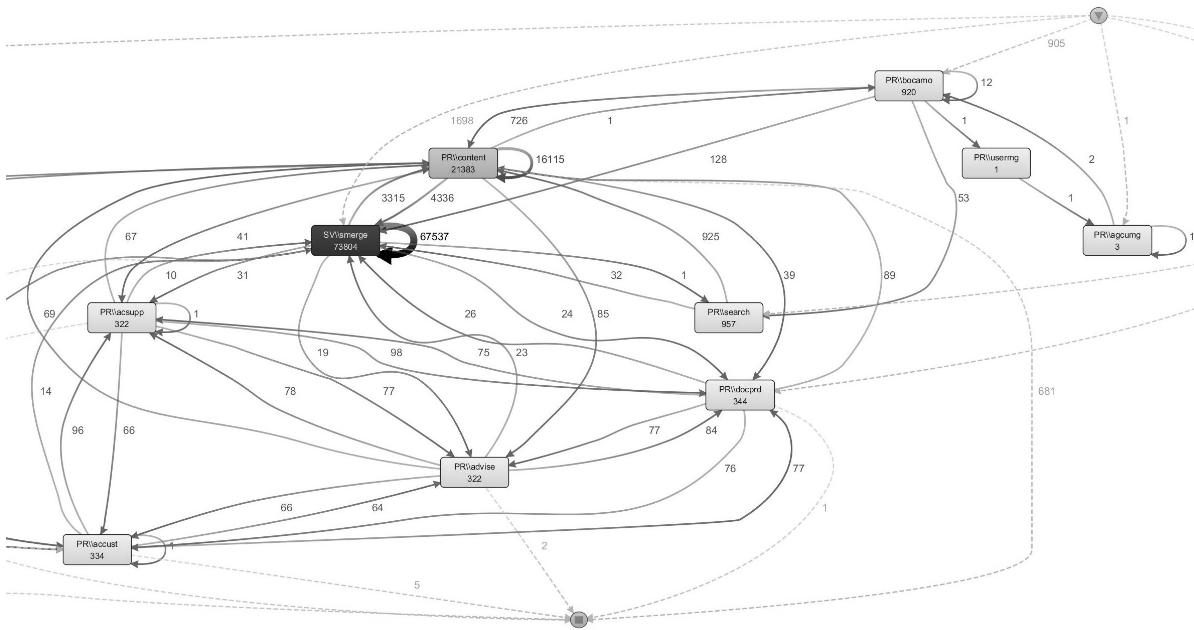

Fig 8. Fuzzy diagram representing relations between PG and a `smerge` domain, $AP(100, 100)$.

An approach to recover interaction patterns between different entities such as methods, files, or modules, based on analysis and comparision of execution traces of different versions of a software system, is proposed in [15]. One of the goals is to track the evolution of particular modules and to visualize the findings. Like in our paper, the authors use a standard database technology for maintaining analyzed data.

Both of the approaches above and a number of other “classical” approaches to runtime analysis is based on an idea of code instrumenting [16]. Process mining does not require any specific code instrumenting and utilize any traces that software can produce during its execution.

A *trace summarization* technique for manipulating traces, based on metrics for measuring various properties of an execution trace, was introduced in [17]. It is proposed to use trace summaries to enable top-down analysis of traces as well as recovery of system behavioural models. There was proposed a trace summarization algorithm that is based on successive filtering of implementation details from traces.

An idea to apply process mining to services, so-called *service mining*, was proposed in [18, 19]. Finally, based on the process mining discipline, a comprehensive approach to diagnostic information in compliance checking was proposed in [20]. We suppose that using a similar approach for Petri net models of the processes discussed in our paper can introduce some new ideas for achieving the objectives of SRA.

One of the most novel approaches to reverse engineering for obtaining real-life event logs from distributed systems is presented in [21]. The approach allows to analyze operational processes of software systems under real-life conditions and use process mining techniques to obtain precise and formal models.

3. Future work and conclusion

The results obtained during the first practical experiments show us several ways for future work.

First, taking into account specificity of the subject domain, which is software architecture and engineering, introduction of convenient and accustomed tools particularly for model representation is desirable. As an example, the UML sequence diagrams miner mentioned in sec. 1.2.1. can be considered, possibly based on ProM or any other tool. Now we have to construct a UML sequence diagram manually, and it would be a good challenge to provide ability for constructing such diagrams automatically, e.g. by a special ProM plug-in.

Then, it is rather desirable to obtain other kinds of models to provide more comprehensive analysis by using different mining algorithms. Nevertheless, it is still a problem to process large logs with a full range of academic tool basically implemented as ProM plug-ins. Also, we suppose that it is possible to use these models to detect/recognize other architectural patterns that can be used for improving systems in some ways. In order to do this one needs to consider also other methods for pattern recognition in a model like the one proposed in *compliance checking* research [20].

In this paper only few violations of architectural principles and patterns are concerned. One of our goals is to create a catalog of architectural patterns/architectural violations related to different kinds of systems.

Finally, there is also *Software Performance Analysis* which is a separate big problem we would like to investigate with the help of process mining techiques.

4. Acknowledgment

The authors would like to thank Fluxicon for powerfull process mining tool Disco provided.

References

[1] W. M. P. van der Aalst, *Process Mining — Discovery, Conformance and Enhancement of Business Processes*, Springer, 2011.

[2] IEEE Task Force on Process Mining, “Process Mining Manifesto”, *BPM 2011 Workshops, ser. Lecture Notes in Business Information Processing*, **99**, eds. F. Daniel, S. Dustdar, K. Barkaoui, Springer-Verlag, Berlin, 2011, 169–194.

[3] E. Kindler, V. Rubin, W. Schäfer, “Activity mining for discovering software process models”, *Software Engineering*, **79**, eds. B. Biel, M. Book, V. Gruhn, 2006, 175–180.

[4] V. Rubin, I. Lomazova, W. M. van der Aalst, “Agile development with software process mining”, *ICSSP 2014*, ACM, Nanjing Jiangsu, China, 2014, 70–74.

[5] V. Rubin, A. A. Mitsyuk, I. A. Lomazova, W. M. P. van der Aalst, “Process mining can be applied to software too!”, *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, NY: ACM, 2014.

[6] J. McGovern, O. Sims, A. Jain, M. Little*Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations*, Springer, 2006.

[7] A. Mitsyuk, A. Kalenkova, S. Shershakov, W. van der Aalst, “Using process mining for the analysis of an e-trade system: A case study”, *Software Engineering (in Russian)*, **3**, 2014, 15–27.

[8] H. Verbeek, J. Buijs, B. Dongen, W. Aalst, “ProM 6: The Process Mining Toolkit”, *Proc. of BPM Demonstration Track 2010, ser. CEUR Workshop Proceedings*, **615**, eds. M. L. Rosa, 2010, 34–39.

[9] [Online]. Available: http://www.fluxicon.com/disco.

[10] C. W. Günther, W. M. P. Van Der Aalst, “Fuzzy mining: Adaptive process simplification based on multi-perspective metrics”, *Proceedings of the 5th International Conference on Business Process Management, ser. BPM’07*, Springer-Verlag, Berlin, Heidelberg, 2007, 328–343.

[11] S. A. Shershakov, “VTMine framework as applied to process mining modeling”, *International Journal of Computer and Communication Engineering*, **4**:3 (2015), 166–179.

[12] S. Shershakov, “DPMine/P: modeling and process mining language and ProM plug-ins”, *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, eds. A. N. Terekhov, M. Tsepkov, ACM New York, NY, USA, 2013.

[13] S. A. Shershakov, “DPMine graphical language for automation of experiments in process mining [in russian]”, *Modeling and Analysis of Information Systems*, **21**:5 (2014), 102–115.

[14] K. Havelund, “Using runtime analysis to guide model checking of java programs”, *SPIN*, Lecture Notes in Computer Science, **1885**, eds. K. Havelund, J. Penix, W. Visser, Springer, 2000, 245–264.

[15] M. Fischer, J. Oberleitner, H. Gall, T. Gschwind, “System evolution tracking through execution trace analysis”, *IWPC*, IEEE Computer Society, 2005, 237–246.

[16] T. Ball, “The concept of dynamic analysis”, *ESEC / SIGSOFT FSE*, Lecture Notes in Computer Science, **1687**, eds. O. Nierstrasz, M. Lemoine, Springer, 1999, 216–234.

[17] A. Hamou-Lhadj, *Techniques to simplify the analysis of execution traces for program comprehension*, Ph.D. dissertation, Ottawa-Carleton Institute for Computer Science School of Information Technology and Engineering, University of Ottawa, 2005.

[18] W. Aalst, H. Verbeek, “Process Mining in Web Services: The WebSphere Case”, *IEEE Bulletin of the Technical Committee on Data Engineering*, **31**:3 (2008), 45–48.

[19] W. van der Aalst, “Service mining: Using process mining to discover, check, and improve service behavior”, *IEEE Transactions on Services Computing*, **99**:PrePrints (2012), 1.

[20] E. Ramezani, D. Fahland, B. F. van Dongen, W. M. P. van der Aalst, *Diagnostic information for compliance checking of temporal compliance requirements*, Tech. Rep., 2013., [Online]. Available: http://dblp.uni-trier.de/db/conf/caise/caise2013.html#TaghiabadiFDA13.

[21] M. Leemans, W. M. P. van der Aalst, “Process mining in software systems: Discovering real-life business transactions and process models from distributed systems”, *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, 2015, 44–53.

DOI: 2015\mais\22\1\1\

Анализ системных исполнений с помощью Process Mining

Шершаков С. А.[1], Рубин В. А.

получена 15 апреля 2013

Информационные системы (ИС) оставляют многочисленные следы и журналы событий своей работы. В контексте сервисно-ориентированной архитектуры (СОА) информационной системы такие журналы содержат детальную информацию о последовательностях вызовов процессов и сервисов. Современные инструменты мониторинга приложений и отслеживания ошибок их исполнения предоставляют довольно простые средства поиска и фильтрации журналов событий. Тем не менее, "интеллектуальный" анализ таких журналов событый является крайне полезным, так как может предоставить ценную информацию об архитектуре системы, взаимодействии между бизнес-доменами и сервисами. В работе рассматриваются журналы событий (представляющие данные о системных исполнениях) большой информационной системы поддержки бронирования, на основании данных которых производится обнаружение нарушений архитектурных принципов взаимодействия компонентов и общих антипаттернов СОА. Для анализа этих журналов применяются проверенные подходы дисциплины извлечения и анализа процессов (process mining). Process mining применяется для автоматического синтеза моделей процессов, анализа этих процессов и их улучшения на основе информации о поведении ИС, записанной в виде журналов событий. На базе нескольких конкретных примеров демонстрируется успешное применения подходов process mining для анализа системных исполнений и приводится обоснование необходимости дальнейших исследований в данной области.

Ключевые слова: извлечение и анализ процессов, программные процессы, анализ системных исполнений

Для цитирования: Шершаков С. А., Рубин В. А., "Анализ системных исполнений с помощью Process Mining", *Моделирование и анализ информационных систем*, **22**:2 (2015), 175–190.

Об авторах:
Шершаков Сергей Андреевич, orcid.org/0000-0001-8173-5970, научный сотрудник НУЛ ПОИС ФКН,
Национальный исследовательский университет Высшая школа экономики
101000 Россия, г. Москва, ул. Мясницкая, 20, e-mail: sshershakov@hse.ru

Рубин Владимир Александрович, orcid.org/0000-0001-8176-2426, PhD, CEO
Dr. Rubin IT Consulting,
60599, Frankfurt am Main, Germany, e-mail: vroubine@gmail.com

Благодарности:
[1]Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ в 2015 году.