Extraction of Operational Workflow-based User Behavior Profiles for Software Modernization

Gunnar Dittrich
b+m Informatik AG
24109 Melsdorf, Germany
gunnar.dittrich@bmiag.de

Christian Wulf
Software Engineering Group
Kiel University, Germany
chw@informatik.uni-kiel.de

Abstract

Static and dynamic analysis are the core parts in the software modernization process. They are required for the architecture reconstruction and the assessment of legacy software systems. One important use case is the extraction of user behavior profiles which can help in improving the system's frontend layer.

In this paper, we present our approach to extract and to visualize operational workflow-based user behavior profiles. Its implementation is based on two Java command line tools. The first tool extracts and anonymizes sessions from the records emitted by the monitoring framework Kieker. Based on these sessions, the second tool extracts a behavior model which is stored in several different graph formats on the file system. We evaluate our tools by instrumenting an industrial workflow-based Java web application for insurers. We show that our approach is able to automatically build and visualize a corresponding hierarchical behavior model. Such a model represents business processes as parents of workflows which in turn contain the visited views. Moreover, we show that this model can help in planning and prioritizing the software modernization process by identifying the most used and the least used views as well as the common screen- and workflow.

1 Introduction

Kieker [3] is a framework for application performance monitoring and dynamic software analysis. One of its main use cases is recording the runtime behavior of a software system in order to monitor the execution of operations in form of monitoring records. However, Kieker does not only allow to collect such monitoring records, but also to analyze and to visualize them.

In this paper, we use Kieker to monitor and to visualize the operational user behavior of an industrial real-world application called b+m bAV-manager developed by the b+m Informatik AG. This application serves as an administration software for customer and calculation data in the field of company pension schemes. In cooperation with a third-party calculation engine, it creates expert opinions for several valuation and accounting regulations. The usage of the b+m bAV-manager is thereby workflow-oriented.

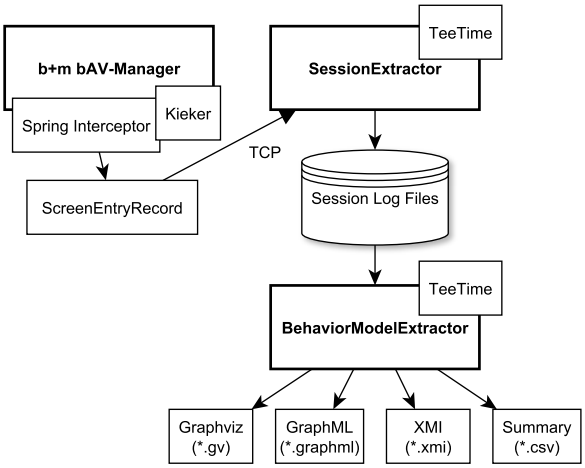


Figure 1: Overview of our approach to extract and to visualize operational user behavior profiles

Currently, we only know in an insufficient way how the users of the b+m bAV-manager work with the graphical user interface (GUI). By building a corresponding user behavior model, we can extract the actual behavior in terms of user behavior profiles in order to derive suggestions for the modernization process. For example, this approach allows us to identify screenflows which do not correspond to their intended workflow definitions. Moreover, it shows the most used screens and most complex processes. In this way, we are able to focus our (human) resources on improving the most crucial screens and screenflows.

2 Overview of our Approach

Figure 1 illustrates our approach to extract and to visualize operational user behavior profiles. The instrumented b+m bAV-Manager uses Kieker within a custom Spring interceptor to send monitored records necessary for the model extraction to the analysis node via TCP. The session extractor processes these records and outputs session logs. These logs are then used by the behavior model extractor to produce a behavior model in several file formats. We detail our approach in Section 3 and 4.

Our approach is inspired in parts by the DynaMod

project [2]. We also apply dynamic analysis to a legacy software to create models of the system. In our case, we build up user behavior models to support the modernization of the application's frontend layer. Moreover, we adapted the monitoring record and the session log extractor of the WESSBAS approach [5].

3 Monitoring

For monitoring the user activities, we use Kieker's monitoring component. We define custom probes which collect information about sessions, workflows, and think times as monitoring records. These records are then written to a monitoring log or stream by a monitoring writer. We choose the TCPWriter to reduce the load on the monitored application.

For recording the user behavior, we defined a custom monitoring record called `ScreenEntryRecord`. Listing 1 shows its structure in the syntax of the Kieker Instrumentation Record Language (IRL) [4]. The IRL is Kieker's DSL for defining monitoring records and is used to generate corresponding implementations in many different programming languages.

```
1  package de.bmiag.gear.util.monitoring.record
2  entity ScreenEntryRecord {
3    string userName
4    long   loginTime
5    string screenName
6    string subprocessName
7    string processName
8    string processExecutionId
9    long   entryTime
10   string eventName        }
```

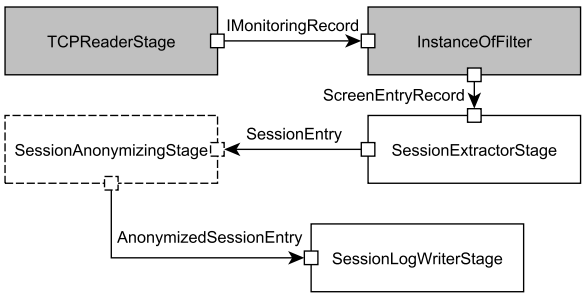
Listing 1: Our monitoring record defined with the IRL

Whenever a user enters a screen in the application, we collect the user's unique name (Line 3), the user's login time (Line 4), the name of the screen (Line 5), the name of the screen's subprocess (Line 6), the name of the process executing the subprocess[1] (Line 7), the process execution id (Line 8), the time when the user entered the screen (Line 9), and the name of the raised event to reach the screen (Line 10). Our approach can be applied to any application that is able to emit the `ScreenEntryRecord` with Kieker.

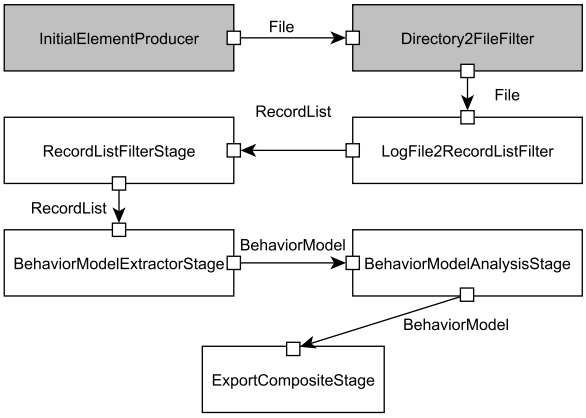
4 Model Extraction

Our approach includes two Java programs as shown in Figure 1. The session extractor receives the `ScreenEntryRecord`s from the Kieker monitoring component via TCP. Its Pipe-and-Filter (P&F) architecture is illustrated by Figure 2a and uses TeeTime [6, 8], a P&F framework for Java. We reuse two TeeTime-based stages (grey stages) from the Kieker analysis component for reading and selecting incoming records. The `SessionExtractorStage` then reads the session data from the incoming ScreenEntryRecords and converts them into comma-separated values. Finally, the

[1] Unlike a process, a subprocess cannot be executed directly. In return, it may be reused by other processes.



(a) Our P&F architecture of the session extractor.



(b) Our P&F architecture of the behavior model extractor.

Figure 2: The P&F architectures of our session extractor and behavior model extractor. The grey and white stages represent reused stages of the TeeTime distribution and, respectively, new stages.

`SessionLogWriterStage` writes these values as a session log file to the file system. An optional stage can be added to the execution, which anonymizes any user or time related information. In the future, this functionality could be moved to the monitoring node.

The behavior model extractor processes the log files produced by the session extractor. Figure 2b illustrates its P&F architecture. Similar to the session extractor, we also use TeeTime for the implementation. The first two stages are predefined TeeTime stages which in combination collect all log files from a directory. The entries of these log files are grouped and filtered in record lists. Afterwards, the `BehaviorModelExtractorStage` uses them to build up a corresponding behavior model consisting of states (screens) and transitions (user activities). We enriched this model by the following hierarchy concept to represent flows and business processes (in short: processes). Each screen belongs to a flow which in turn can be executed within a process or standalone.

Figure 3 shows the screen "Kontrolle festlegen" and its predecessors/successors—a very small part of an example behavior model. The screen belongs to the calculation process "Gutachten" (outer dark grey box) of the b+m bAV-Manager which we evaluate in Section 5. Usually, the screen is reached from the main

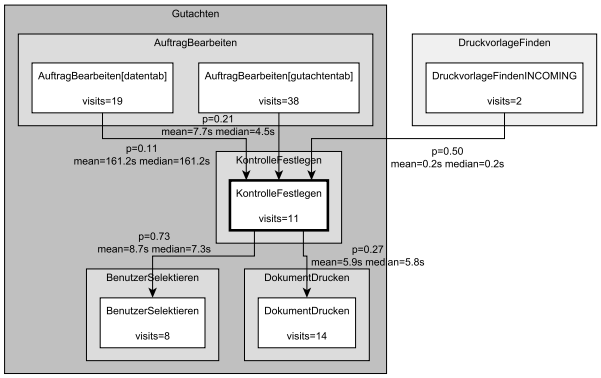


Figure 3: Excerpt of an example behavior model of the b+m bAV-Manager calculation process (dark grey) including its subprocesses (light grey) and screens (white).

administration flow "AuftragBearbeiten" and is left to the selection of a controller or to the print dialog. However, there is also an incoming transition from an external screen "DruckvorlageFinden" which does not belong to the process definition. This could be a point of interest in terms of software modernization.

The behavior model is described by GraphML, one of the export formats of the `ExportCompositeStage`. This stage also generates Graphviz files, serializes the behavior model, and aggregates a summary of all relevant statistics. Before exporting the results, the `BehaviorModelAnalysisStage` calculates statistics for the states and transitions, e.g., the number of visits per screen as well as the probability and the think time (mean, median, etc.) per user activity.

5 Evaluation

To evaluate our approach, we instrumented the b+m bAV-Manager introduced in Section 1 by a Spring method interceptor to collect `ScreenEntryRecord`s introduced in Section 3. We deployed it on a test server at the b+m Informatik AG and asked five b+m employees (developers, architects, and project managers) to execute 11 common business processes of the bAV-Manager via its GUI. Following the GQM approach [1], we define the goal of our experiment as "Identifying abnormal screenflows and workflows by users of the monitored application". Our research questions are as follows: (1) Which screenflows do significantly differ from the expectations of the professionals? (2) In which workflows does the usage significantly differ from the process definition? To answer these questions, we use the metrics "visits per screen", "think times", and "transition probabilities".

We collected 53 session log files with 2381 recorded user activities. We identified 23 of 109 screens which were not visited at all. Retrospectively, 8 of them were classified as obsolete by the professionals. We also recognized unusual workflows by a high number of visits on the error screen. This screen is displayed to the user when a system task fails, e.g., due to a wrong user configuration or a currently locked data entry. Two of these workflows were the calculation process and the administration process which also had the greatest number of unusual incoming and outgoing transitions. Such transitions are not part of the process definition and should therefore be reconsidered in the imminent modernization of the application. For more information on the evaluation, we refer to [7].

6 Conclusions

In this paper, we present our approach for monitoring and analyzing the operational user behavior in workflow-based applications including the design and the implementation of the corresponding components. Our evaluation with the industrial real-world application b+m bAV-Manager shows that our approach successfully builds up a corresponding behavior model. Moreover, it reveals unusual behavior in business processes and in the screen flow which will help us greatly during the imminent modernization of the application.

As future work, we plan to repeat the experiment on productive systems of the b+m bAV-Manager's customers. A high priority has the visualization of process information to improve the screen flow for the users since they often lack orientation.

References

[1] V. R. Basili and D. M. Weiss. "A Methodology for Collecting Valid Software Engineering Data". In: *Transactions on Software Engineering* 10 (1984).

[2] A. Van Hoorn et al. "DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization". In: *Joint Proc. of the Int. Workshops on Model-Driven Software Migration and on Software Quality and Maintainability*. Mar. 2011.

[3] A. Van Hoorn, J. Waller, and W. Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *Proceedings of the ICPE*. Apr. 2012.

[4] R. Jung. *An Instrumentation Record Language for Kieker*. Tech. rep. Kiel University, Aug. 2013.

[5] A. Van Hoorn et al. "Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-based Application Systems". In: *Proceedings of the VALUETOOLS*. 2014.

[6] C. Wulf, N. C. Ehmke, and W. Hasselbring. "Toward a Generic and Concurrency-Aware Pipes & Filters Framework". In: *Proceedings of the Symposium on Software Performance*. Nov. 2014.

[7] G. Dittrich. "Extraction of User Behavior Profiles for Software Modernization". MA thesis. Kiel University: Dept. of CS, May 2016.

[8] C. Wulf, C. C. Wiechmann, and W. Hasselbring. "Increasing the Throughput of Pipe-and-Filter Architectures by Integrating the Task Farm Parallelization Pattern". In: *Proc. of CBSE*. 2016.