A Journey to comprehensible User Behavior Models

Reiner Jung
reiner.jung@email.uni-kiel.de
Kiel University, Kiel, Germany

Lars Jürgensen
stu203350@mail.uni-kiel.de
Kiel University, Kiel, Germany

Abstract

Engineers use workload models to estimate the future utilization of services, understand usage profiles of services, and drive plans to evolve software systems. we usually separate workloads in intensities and user behavior models in context of performance evaluations and forecasting. Outside of software engineering user behaviors are often collected in clickstreams. They may represent the complete history of a user at one site. Ideally, a number of clickstreams can be transformed into a user behavior model containing multiple typical behavior pattern usable for a workload model.

While the collection of user behaviors is fairly simple, the extraction of behavioral patterns is complicated. Current approaches are tailored for specific domains, are only applicable for one purpose, and refuse to be understood and analyzed in a meaningful way. In this paper, we introduce our latest advances to identify a suitable identification approach and layout further obstacles which must overcome to have comprehensible user behavior models.

1 Introduction

In the second phase of the iObserve project [8], we required for our operator-in-the-loop approach comprehensible workload including behavior models, where each behavior covers a typical real life user and its variations. Such behaviors could then be added, removed and modified to customize workloads. This customizability is required, as not all changes in user behavior can be derived from observation data, e.g., behavioral changes based on sales events, weather conditions, and disasters. Furthermore, behavior models provide insight in the actual use of an service in contrast to the anticipated use during development. This can guide the software evolution and marketing of a service. Thus, it is important to have behavior models which correspond with real user behaviors and not only provide a purely descriptive incomprehensible representation of past behavior.

We experimented with existing workload characterization approaches. The WESSBAS approach [16] seemed to be a perfect solution for our needs allowing us to continue with our primary research goals regarding cloud-based applications. Unfortunately, the clustered behaviors did not match the behaviors we used to drive our case studies; and here our journey for comprehensible behavior models begun.

In this paper, Sections 2 and 3 summarize workload and behavior model concepts. Section 4 discusses our latest effort to identify behavior clusters and Section 5 reports on its evaluation. Section 6 summarizes our findings and identifies remaining challenges to comprehensible behavior models.

2 Workload Models

Early workload models were used to test service endpoints, like specific web pages, to estimate response times and system capacity. However, performance depends on functions, data and context. Thus, workload models had to start to emulate real users and combine these behaviors with intensities, i.e., the number of simultaneous users performing an operation.

Today, workload models are used during design time to predict and at runtime to test and forecast system qualities. Thus, workloads must be close to real world scenarios. In Palladio [4], workloads and behaviors are configured based on requirements engineering knowledge and design properties.

Observation based models decompose workloads in a seasonal, trend, burst and noise component to better understand their structure and improve forecasts regarding utilization and software performance [6, 7].

3 Behavior Model Concepts

Beside intensities, workloads contain user behaviors. A user behavior is a sequence of user requests to a system, also called clickstream [9]. Clickstreams may also contain wait times and can be clustered based on similarity graph [9]. As clickstreams can vary in length and may contain repetitive subsequences. Thus, the similarity graph can be based on these subsequences.

Alternatively, clickstreams can be converted to graphs. There are multiple ways to use graphs to model behaviors. One maps requests to nodes and transitions to edges. Another maps nodes to URLs and requests to edges to. Both can store parameter data in attributes (cf. [17]). These graphs can then be clustered to identify typical behaviors [16].

While, clickstream are able to retain most of the input information, they grow quickly, making comparisons of behaviors more complicated. In contrast graph-based representations are more concise, but lose information on the exact sequence of events.

4 Graph-based Clustering

Previous attempts to identify similar behaviors through clustering were based on the WESSBAS approach [16], just altering the clustering algorithm, with unsatisfactory results [10, 12, 14, 15]. Thus, we took a step back and searched for better suited approaches and found them in graph clustering.

For our graph-based approach, we chose a graph where nodes and edges represent pages and requests, respectively. Thus, the edges contain request information. In case an edge is followed multiple times, it contains a sequence of request information.

In general the process is as depicted in Fig. 1. Based on Kieker logs, we generate graphs for each found user behavior (currently based on sessions). These models are stored in an M-Tree utilizing the Graph Edit Distance (GED) metric. Subsequently, we use OPTICS to generate a reachability-plot and extract, based on the plot, clusters.

Graph Edit Distance is a metric to compute the distance between graphs [1]. This metric compares two graphs and counts the number of edit operations to convert one graph into the other.

M-Tree is a balanced tree that stores objects based on a distance function [2]. The advantage of M-Trees are, they do not need an absolute metric and can use a distance function d, like GED. The distance function must adhere the symmetry, positivity and triangle inequality, i.e., $d(A, B) = d(B, A)$, $d(A, B) > 0$ when $A \neq B$ and $D(A, B) = 0$ when $A = B$, and $d(A, B) \leq d(A, C) + d(C, B)$, respectively. Due to size constrains, a detailed description of the M-Trees can be found in [2, 17]. Essentially, the M-Tree provides fast access to objects based on the distance function.

OPTICS is a density-based algorithm to analyze cluster structures in data sets and generates a reachability-plot [3]. It is based on DBScan and inherits from it two configuration parameters: min. number of neighbors to be a dense area, and a maximum search radius for neighbors (ϵ). OPTICS calculates for each object the minimal ϵ for it to be in a cluster.

The reachability-plot in Fig. 2 depicts the ϵ values for all objects ordered by their reachability-distance with respect to all previously visited objects. Fig. 2 depicts various valleys indicating low ϵ values and, therefore, clusters. To extract clusters these valleys must be identified. The simplest way is to use a fixed ϵ value where a cluster begins/ends when the graph goes below/above the threshold. However, this may not successfully identify all clusters, as for example the separation between the two behaviors *Single Cat* and *Single Reptile* may not be detected correctly.

5 Evaluation

We evaluated our graph-based approach based on two monitoring datasets (a) derived from the JPetStore experiment suite [13] with a set of predefined user behaviors [11] from our previous cluster algorithm tests [10, 12, 14, 15], and (b) collected during seminars (Software Praktikum) where students used the ticket management system Jira to coordinate their development efforts [18, 19, 20, 21]. The general setup for the evaluation is as follows: We replayed monitoring events from Kieker logs. These were processed as explained above and used to generate clusters.

JPetStore Scenario We used predefined expected user behaviors which were used to create two sets of workload drivers: (a) a fixed user behavior and (b) randomized behaviors where, for example, the number of selected items change or the number of repetitions of a specific task. Based on these workloads and the JPetStore the Kieker logs were generated. These logs were both fed to our setup in separate experiment. Each time we compared the behaviors identified by the clustering with the predefined expected behavior. For the fixed user behavior, all seven behaviors where correctly detected and, beside some generated noise, each user was correctly assigned to the corresponding cluster which is an improvement over our previous approaches that were unable to identify them correctly.

In the randomized setup, we could also detect all clusters correctly. However, due to the variations some behaviors where not placed in a corresponding cluster. This is mainly to the current cluster detection method is sensitive to cluster densities and cause suboptimal cluster detection [17].

Jira Scenario Is a setup with real world users operating a real application to perform their work. We fed the complete Kieker log of a seminar (four weeks of user interactions) into our cluster detection software. The software identified a few behavior clusters. However, the associated user behaviors were unusual large. After investigating the behavior graphs, it became clear that users usually logged in once a day – keeping their session and perform multiple tasks. Thus, sessions are not a viable method to identify the beginning and end of an activity.

We tested timeouts as a potential approach. However, they do not work sufficiently. For example, a user creates a new ticket in Jira and starts adding a bug or features description, but before finally committing the ticket, there is a delay, e.g., a group discussion. This can create lengthly breaks between user requests while we are still within a typical behavior. Thus, timeouts are not suitable to split up sessions.

6 Conclusion

To generate user behavior models which are (a) a close representation of real users, (b) a good classification of the variety of behaviors, and (c) still usable to generate real workloads as well as models for workload forecasting, is still an open issue. However, based on our graph-based clustering approach with OPTICS, we are able to identify clusters in data from our example shop system even when user behavior is random-

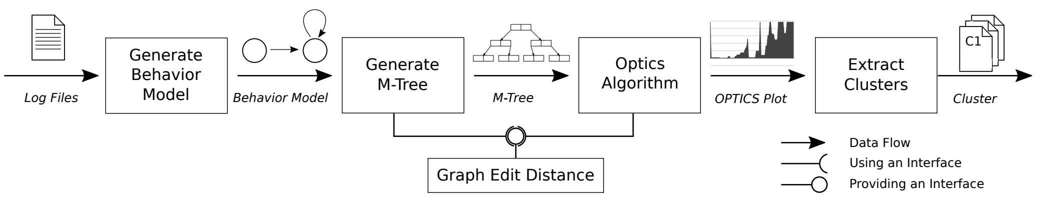

Figure 1: Conceptual representation of the workflow to generate user behavior clusters [17, p. 16]

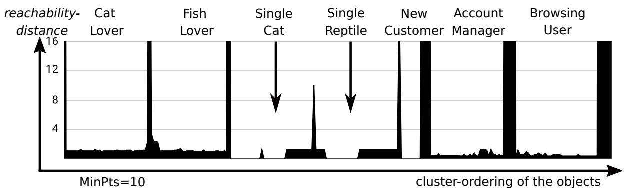

Figure 2: Reachability-plot for the randomized user behavior scenario (cf. Section 5, [17, p. 31])

ized to some extend. These are promising results.

Still we encountered new challenges in the detection of user behaviors, i.e., to identify the beginning and end of a behavior pattern. Currently, we looking examining approaches used to identify subgraphs in graph based on cluster detection in the graph, and an approach which defines certain action as end of a typical user behavior. Furthermore, there are other distance metrics which could be applied, and other approaches to consider request parameter values.

References

[1] A. Sanfeliu and K. Fu. “A distance measure between attributed relational graphs for pattern recognition”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3 (May 1983), pp. 353–362.

[2] P. Ciaccia, M. Patella, and P. Zezula. “M-tree: An Efficient Access Method for Similarity Search in Metric Spaces”. In: *Proc. of the 23rd International Conference on Very Large Data Bases.* 1997, pp. 426–435.

[3] M. Ankerst et al. “OPTICS: ordering points to identify the clustering structure”. In: *ACM Sigmod record.* Vol. 28. 2. ACM. 1999, pp. 49–60.

[4] S. Becker, H. Koziolek, and R. Reussner. “The Palladio component model for model-driven performance prediction”. In: *Journal of Systems and Software* 82.1 (2009), pp. 3–22.

[5] I. Assent. “Clustering high dimensional data”. In: *Wiley Interdis. Reviews: Data Mining and Knowledge Discovery* 2.4 (2012), pp. 340–350.

[6] N. R. Herbst et al. “Self-adaptive workload classification and forecasting for proactive resource provisioning”. In: *Concurrency and Computation: Practice and Experience* 26.12 (2014), pp. 2053–2078.

[7] J. v. Kistowski, N. R. Herbst, and S. Kounev. “Modeling Variations in Load Intensity over Time”. In: *Proc. of the 3rd International WS on Large Scale Testing.* ACM, 2014, pp. 1–4.

[8] R. Heinrich et al. “Architectural Run-Time Models for Operator-in-the-Loop Adaptation of Cloud Applications”. In: *MESOCA.* IEEE Computer Society, Sept. 2015, pp. 36–40.

[9] G. Wang et al. “Unsupervised Clickstream Clustering for User Behavior Analysis”. In: ACM, 2016, pp. 225–236.

[10] C. Dornieden. “Knowledge-Driven User Behavior Model Extraction for iObserve”. Masterarbeit. Kiel University, June 2017.

[11] R. Jung and M. Adolf. *JPetStore Workload Driver.* `https://github.com/research-iobserve/selenium-workloads`. 2017.

[12] R. Jung, M. Adolf, and C. Dornieden. “Towards Extracting Realistic User Behavior Models”. In: *STT* 37.3 (Nov. 2017), pp. 11–13.

[13] R. Jung. *JPetStore Experiment Suite.* `https://doi.org/10.5281/zenodo.1292788`. 2018.

[14] J. Kuckei. “Comparison Of User Behaviour Classification Methods”. Bachelorarbeit. Kiel University, März 2018.

[15] M. Lorenzen. “Classification of User Behavior through Connectivity-Based Clustering”. Bachelorarbeit. Kiel University, Apr. 2018.

[16] C. Vögele et al. “WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems”. In: *SoSyM* (May 2018), pp. 443–477.

[17] L. Jürgensen. “Clustering and Analysis of User Behaviors utilizing a Graph Edit Distance Metric”. Bachelorarbeit. Kiel University, Nov. 2019.

[18] H. Schnoor and W. Hasselbring. *Jira Monitoring Data February 2017.* Zenodo, Feb. 2020.

[19] H. Schnoor and W. Hasselbring. *Jira Monitoring Data February 2018.* Zenodo, Feb. 2020.

[20] H. Schnoor and W. Hasselbring. *Jira Monitoring Data September 2017.* Zenodo, Feb. 2020.

[21] H. Schnoor and W. Hasselbring. *Jira Monitoring Data September 2018.* Zenodo, Feb. 2020.