The JPetStore Suite: A concise Experiment Setup for Research

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

Marc Adolf
mad@informatik.uni-kiel.de
Kiel University, Kiel, Germany

Abstract

Viable experiment suites are in high demand by scientists in software engineering and especially for software quality. In particular, easy to understand yet feature rich case studies are needed to assess approaches, methods and tools for software qualities, like performance, usability, privacy and security.

Several attempts have been made to create reusable setups. Unfortunately, they focus on the software alone, without or limited documentation, workloads, and prepared deployments, harming reusability and repeatability. This hinders our research.

To circumvent these limitations, we created a JPetStore based experiment suite. Our suite includes distributed and single service variants with and without instrumentation, proper workload drivers, and experiment setups. All parts are documented to support scientists creating experiments. Using our suite fosters comparability and reproducibility of research making it more influential. Furthermore, our suite can be used as blue print for more complex suites.

1 Introduction

Scientists rely on case studies to evaluate methods and approaches addressing challenges in the software engineering domain. Especially, in the context of software quality, they require case studies close to and derived from real world scenarios, to gain meaningful results. In iObserve [6], case studies were in high demand to assess all aspects of our software quality and user behavior modeling experiments. We used different case studies, but setups from previous experiments where always hard to reuse and rejuvenate.

The software engineering community uses a wide range of case studies. For example Common Component Modeling Example (CoCoME) [2] and the Kieker Netflix setup [13] are complex distributed systems, while the JPetStore [14] is a simple single service application. Unfortunately, the complex systems are difficult to set up and experiment replication is error prone and cumbersome, as documentation, experiment setup, and workloads are no longer available. This renders gained results void, as reproducibility cannot be achieved.

That is why we need an experiment suite including a software system, workloads, experiments, and documentation. It is tempting to use larger case studies, like Netflix, in an experiment suite, as they resemble real systems. However, their complexity can cause more difficulties tainting results, development requires more resources, and we have only limited control over their evolution. Therefore, we selected the JPetStore as basis for our experiment suite, because (a) it is easy to understand and refactor into a distributed architecture, (b) it is widely used by other scientists in their research, and, (c) its simplicity allows to provide a modular and configurable workload model, which can be adjusted to all needs. Furthermore, our suite can serve as a blue print for larger experiment suites.

We present our **JPetStore suite**, comprising **single** service and **distributed** JPetStore variants, **preconfigured and instrumented** variants, a **configurable workload driver**, and experiment setups for **Docker** and **Kubernetes**.

In Section 2, we provide an brief overview of the JPetStore experiment suite. Section 3 discusses the different variants of the JPetStore and how we maintain and evolve them. Section 5 introduces our customizable workload driver. While Section 4 illustrates how to use our experiment set ups. Section 6 discusses related case studies, which might be good candidates to create experiment suites. Finally, we summarize our efforts and provide an outlook in Section 7.

2 Suite Overview

Our JPetStore suite comprises different JPetstore variants, a configurable workload driver, and experiment setups for Docker, Docker-Compose and Kubernetes. The variants include single service and distributed service architectures. We created three variants for each architecture: (a) without instrumentation, (b) with instrumentation for traces, and (c) with instrumentation for geolocation and entry level events.

Single Service Experiment Setup (SSES) The SSES, in Fig. 2, comprises an experiment host and a JPetStore container, which can be a Docker container, a Kubernetes pod or deployment. The experiment host contains the workload driver and the monitoring event collector service, which can be replaced by any Kieker-compatible [3] analysis, like, the iObserve behavior and privacy analysis services [7, 12].

Distributed Service Experiment Setup (DSES) The DSES (see Fig. 1) divides the JPetStore into

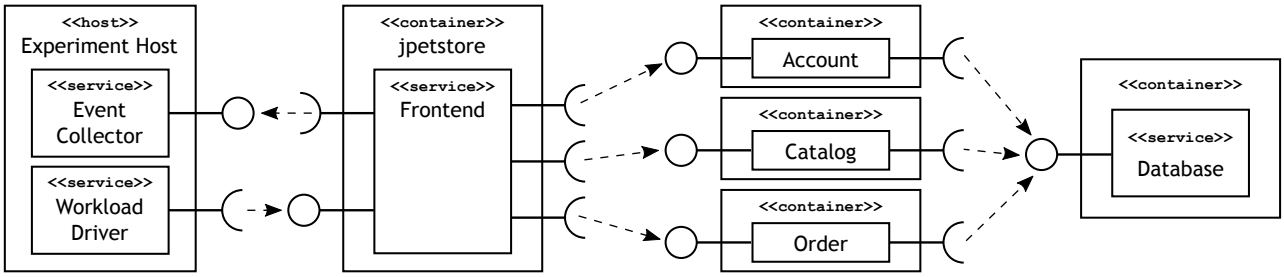

Figure 1: Experiment setup with distributed service variant of the JPetStore

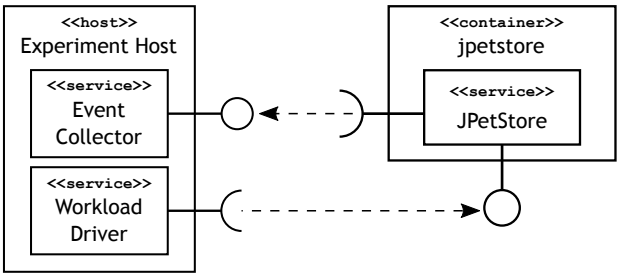

Figure 2: Single service variant for experiments

services for frontend, account management, catalog and orders including the shopping cart, each in its own docker container supporting distributed deployment. All services except the frontend have their own database, located on a shared database service.

The experiment control host comprises the same services as the SSES setup, but the collector must be able to accept multiple connections, as the distributed JPetStore consists of multiple containers. The iObserve project provides such collector.

3 Variants of the JPetStore

We forked the classic MyBatis JPetStore [15] and created one branch for a single service JPetStore without instrumentation and one distributed variant by placing the internal components in REST services. For both, we created two variants with instrumentation probes pre-configured to send monitoring data via TCP to an analysis or collector service. The first produces Kieker trace events, and the second adds entry level events with payload data (e.g., productId).

The variants are accompanied by a Palladio Component Model (PCM) [1]. We keep code and model in sync by following an evolution strategy. Changes are applied to the most common branch and then transferred to all derived branches. Each release is tagged and automatically published on the long time archive Zenodo [4]. This helps scientists to reproduce experiments and compare their results with others.

4 Experiment Setups

Executing experiments is often complicated and error prone. Scientists create scripts to put software deployments, workloads, and data collection together. Unfortunately, these scripts often run only on one machine and seize functioning after publication, as some details get lost. We mitigate this issue by providing documented scripts for experiment execution together with a set of pre-defined workload [10] and deployment configurations, which run out of the box. Therefore, scientists do no longer need to create their own scripts and setups and waste precious research time.

We provide two experiment repositories for single [11] and distributed [9] deployments. They contain scripts to execute the JPetStore with a workload (`execute-*-jpetstore.sh` *kube or docker) or run a complete experiment (`execute-observation.sh`). The latter starts an event collector and subsequently the JPetStore script. All scripts share a common configuration which reduces errors.

5 Workload Driver

The JPetStore comprises 5 servlets for login, catalog browsing, product and item selection, ordering, and account management (depicted in Fig. 3). We identified specific tasks in this graph, which can be combined to behaviors (cf. [12]).

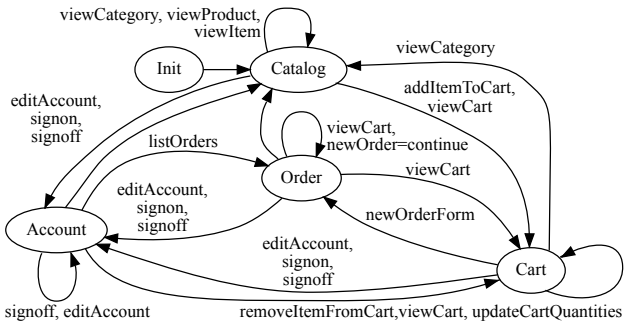

Figure 3: All servlets and transitions of the JPetStore

The workload driver uses Selenium and supports different web drivers, like `PhantomJS`. The behavior mix is specified in a YAML file with four sections for global parameters, webdriver configuration, workload intensities, and behaviors (cf. Listing 1).

Listing 1 depicts an excerpt with a constant intensity provider (`ConstantWorkloadIntensity`) together with a behavior (`AccountManager`), which is spawned every 10 s when active. The properties delays

and durations define when and how long a behavior is active. Here, it is first inactive for 50 s, then active 100 s, inactive for 40 s, and then active again for 70 s.

```
workloads:
- name: AccountManagerWL
  intensity:
    type: org.iobserve.selenium.configuration.ConstantWorkloadIntensity
    name: AccountManager
    spawnPerSecond: 0.1
    durations: [ 100, 70 ]
    delays: [ 50, 40 ]
behaviors:
  AccountManager:
    name: AccountManager
    subbehaviors:
    - name: LoginJPetStoreTask
      parameters:
        username: "j2ee"
        password: "j2ee"
    - name: ChangeAccountInformationTask
      repetition: { min: 1, max: 10 }
      parameters:
        attribute: ADDRESS2
        value: "Christian-Albrechts-Platz 4"
```

Listing 1: Excerpt of a workload configuration

Behaviors are composed from tasks. Each task has its own parameters and can be executed repeatedly. At runtime the repetition is computed within $[min : max]$. Listing 1 depicts a behavior with two tasks, where the latter can be executed multiple times.

6 Related Work

Scientists use a wide range of example applications, which differ in complexity and application domain, like shop systems and entertainment services. In recent years four case studies have become more prominent. Therefore, we use them here as examples.

The Common Component Modeling Example (CoCoME) [2] resembles a software system of a supermarket chain. It addresses both enterprise and embedded systems. While there exist PCM models for CoCoME, they do not map perfectly with the source code and there are no working workload drivers and experiment setups. The Kieker Netflix setup [13] is a well designed example derived from a real world software system. It includes an JMeter workload script with test plan, but it does not include a PCM model and setup scripts. SPECjbb [8] is a performance benchmark to test infrastructure and software components, like web servers and databases. It comes with a workload driver, but it is complicated to set up properly, lacks Kieker monitoring probes, and experiment setup scripts. Thus all of them involve complex tasks and require extensive knowledge to get them running. In contrast the JPetStore suite is simple to set up and execute in state of the art environments.

7 Conclusion

We presented our easy to setup and execute JPetStore suite comprising multiple JPetStore variants, workload drivers, and experiment setups ready to use in various research efforts in software quality.

In future, we will extend code and models to support different databases, as present code uses an internal HSQL database. We will add service effect specifications to the PCM model to reflect internal functionality and workload properties, we will additional workloads intensities, and we will switch to Jupyter notebooks [5] to ease result publication and enable us to provide interactive publications.

References

[1] 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.

[2] A. Rausch et al., eds. *The Common Component Modelling Example (CoCoME)*. Vol. 5153. LNCS. Springer, 2011.

[3] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *ICPE 2012*. Boston, USA: ACM, Apr. 2012, pp. 247–248.

[4] C. D. Centre. *Zenodo*. `https://www.zenodo.org`. 2013.

[5] Jupyter Project. *The Jupyter Notebook*. `http://jupyter.org/`. 2014.

[6] 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.

[7] E. Schmieders, A. Metzger, and K. Pohl. "Runtime Model-Based Privacy Checks of Big Data Cloud Services". In: *Service-Oriented Computing*. Springer, 2015, pp. 71–86.

[8] Standard Performance Evaluation Corporation. *SPECjbb benchmark*. 2015.

[9] R. Jung and M. Adolf. *Distributed JPetStore Experiments*. `https://github.com/research-iobserve/distributed-jpetstore-experiment`. 2017.

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

[11] R. Jung and M. Adolf. *Single Service JPetStore Experiments*. `https://github.com/research-iobserve/single-jpetstore-experiment`. 2017.

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

[13] A. van Hoorn and T. F. Duellmann. *Kieker Netflix Case Study*. `https://github.com/kieker-monitoring-docker`. 2018.

[14] R. Jung. *JPetStore Experiment Suite*. 2018.

[15] M. Spring. *MyBatis JPetStore*. `http://www.mybatis.org/spring/sample.html`.