Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

Generating Probabilistic and Intensity-Varying Workload for Web-Based Software Systems*

André van Hoorn, Matthias Rohr, and Wilhelm Hasselbring

Software Engineering Group, University of Oldenburg, Germany
E-Mail: {van.Hoorn,Rohr,Hasselbring}@Informatik.Uni-Oldenburg.DE

Abstract This paper presents an approach and a corresponding tool for generating probabilistic and intensity-varying workload for Web-based software systems. The workload to be generated is specified in two types of models. An application model specifies the possible interactions with the Web-based software system, as well as all required low-level protocol details by means of a hierarchical finite state machine. Based on the application model, the probabilistic usage is specified in corresponding user behavior models by means of Markov chains. Our tool Markov4JMeter implements our approach to probabilistic workload generation by extending the popular workload generation tool JMeter. A case study demonstrates how probabilistic workload for a sample Web application can be modeled and executed using Markov4JMeter.

1 Introduction

Web-based software systems, such as online shopping systems or auction sites, are large-scale software systems which users access through an interface provided by a Web server. These typically business-critical systems must satisfy contractually specified service level agreements, e.g., upper bounds on user-perceived response times with respect to certain load conditions. In order to systematically evaluate the performance, load tests are carried out: a software called *workload generator* mimics user behavior by submitting requests to the Web server; the performance of the software is monitored for later analysis [1]. Usually, such a workload generator either replays requests from recorded real-world workload or generates requests based on mathematical models [2]. In order to provide meaningful results, a key requirement for load tests is that the simulated user behavior is realistic, i.e., the *virtual users* behave like real users do.

The first part of this paper will present our approach for specifying and generating probabilistic workload for Web-based software systems based on mathematical models. The main elements of the workload specification are two types of models. An *application model* specifies the possible interactions with the Web-based software system, as well as all required low-level protocol details by means

* This work is supported by the German Research Foundation (DFG), grant GRK 1076/1.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

of a hierarchical finite state machine. By means of Markov chains, the probabilistic usage is specified in *user behavior models* corresponding to the application model. Moreover, our approach explicitly considers the specification of a varying workload intensity, i.e., the number of concurrent virtual users, within a single workload generation run. This allows to easily carry out long-term load tests with realistic workload intensity profiles. We will present the conceptual architecture of a workload generator which executes such specifications of probabilistic and intensity-varying workload. Based on our approach, we implemented the corresponding workload generation tool Markov4JMeter. Markov4JMeter extends the popular workload generator JMeter [3]. The resulting implementation and integration into JMeter are demonstrated in the second part of this paper. The case study of this paper illustrates how probabilistic workload for a sample Web application can be specified using our approach and how this specification can be executed with JMeter extended by Markov4JMeter.

The remainder of this paper starts with a summary of the background and related work in Section 2. A description of our workload generation approach including the workload specification and the conceptual workload generator is given in Section 3. Section 4 presents the implementation of Markov4JMeter and its integration into JMeter. As a case study, Section 5 demonstrates how Markov4JMeter is used to generate workload for a sample Web application. Our conclusions follow in Section 6.

2 Background and Related Work

Web-based software systems provide *services* through a Web interface using protocols like the Hypertext Transfer Protocol (HTTP) [4]. Each service can be considered a use case, e.g., signing on to the system or adding an item to the shopping cart. Invoking such a service requires submitting one or more parameterized lower-level protocol-specific *requests*. For example, in order to sign on, it is usually required to first request the corresponding HTML form and to submit the completed form including username and password in a second step. The HTTP request/response model is illustrated in Figure 1. A number of users concurrently accesses a Web-based system by submitting HTTP requests and waiting for the server response. Each user independently alternates between submitting a request and waiting for a time period called *think time* after it has received the server response. A *session* denotes the sequence of related request or service invocations issued by the same user [5].

In their workload generation approach, Barford and Crovella [2] introduced the ON/OFF model. Software processes called User Equivalents (UE) alternate between the two states ON (submit request and wait for the response) and OFF (think time period). We use the concept of UEs in our workload generation approach presented in Section 3.2. The UE concept is denoted as *user simulation thread* in this paper. A user simulation thread executes the workload model of a single virtual user.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

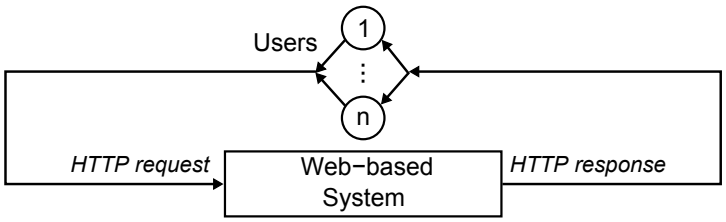


Figure 1. Typical HTTP request/response model of a Web-based system that is concurrently accessed by n users.

Markov chains are a common means for characterizing user behavior, e.g., for Web-based software systems [5] or in statistical software testing [6]. A Markov chain is a probabilistic finite state machine, i.e., each transition between two states is weighted with a probability. Menascé et al. [5] used Markov chains to model classes of user behavior within a session by so-called *Customer Behavior Model Graphs* (CBMG). The states of a CBMG represent service invocations. The CBMGs can be derived from Web server access logs using clustering algorithms [5]. Lee and Tian [7] showed that Markov chains provide fairly accurate models of Web usage. Ballocca et al. [8] derived user behavior in their workload generator from CBMGs. Based on the CBMGs by Menascé et al., Markov chains are the key elements of our user behavior models presented in Section 3.1.

According to Krishnamurthy et al. [9], we consider the class of *session-based systems*. In these systems, inter-requests dependencies exist, meaning that some requests within a session depend on requests submitted earlier during the same session. For example, a user must not submit an order without having added a single item to the shopping cart (and must not have removed all items from the cart later). Shams et al. [10] used so-called Extended Finite State Machines (EFSM) to model valid sequences of interactions with the application using conditional transitions between states and by explicitly considering the parameters to be passed with a submitted request. The application models defined in Section 3.1, specifying allowed sequences of service invocations within a session, were inspired by this work. However, they do differ from Shams et al.'s EFSMs in that the application model is separated into a logic session layer and an underlying technical protocol layer for abstraction purposes.

Peña-Ortiz et al. [11] provide an overview of outstanding and historical workload generators including an evaluation in terms of their features and capabilities. We explicitly modeled the workload generator on a conceptual level including the execution semantics and implemented the resulting tool Markov4JMeter as an extension for the popular workload generator JMeter [3].

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

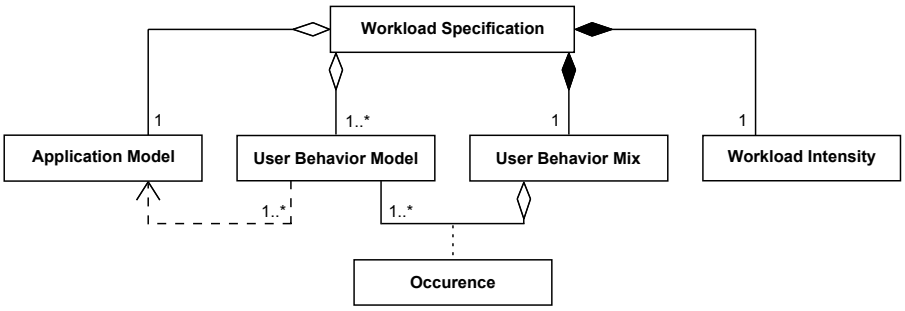


Figure 2. Class diagram of the workload specification elements and their relations.

3 Our Workload Generation Approach

Section 3.1 defines the workload specification including the probabilistic workload model. The conceptual architecture of the workload generation tool executing this workload specification is presented in Section 3.2.

3.1 Workload Specification

The workload specification for our probabilistic workload generation approach consists of the four elements listed below.

- An *application model*, specified as a hierarchical finite state machine.
- A number of corresponding *user behavior models*, each one specified as a Markov chain.
- A *user behavior mix*, specified as probabilities for the individual user behavior models to occur during workload generation.
- A definition of the *workload intensity*, specified as the (possibly varying) number of users to simulate during the experiment.

The application model defines the *allowed* sequences of service invocations submitted within a user session and contains all protocol-level details required to generate valid requests. The actual order of service invocations is derived from probabilistic *user behavior models* corresponding to the application model. The workload generator combines the application model and the user behavior models into *probabilistic session models* based on which the requests are executed for each virtual user. This is described in Section 3.2. The *user behavior mix* defines with which probability each user behavior model occurs during workload generation. The *workload intensity* is a specification of the number of users to simulate during the experiment, given as a mathematical formula of the elapsed experiment time.

These elements are described in detail in the remainder of this section. Figure 2 illustrates their multiplicities and relations among each other in a UML Class Diagram.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

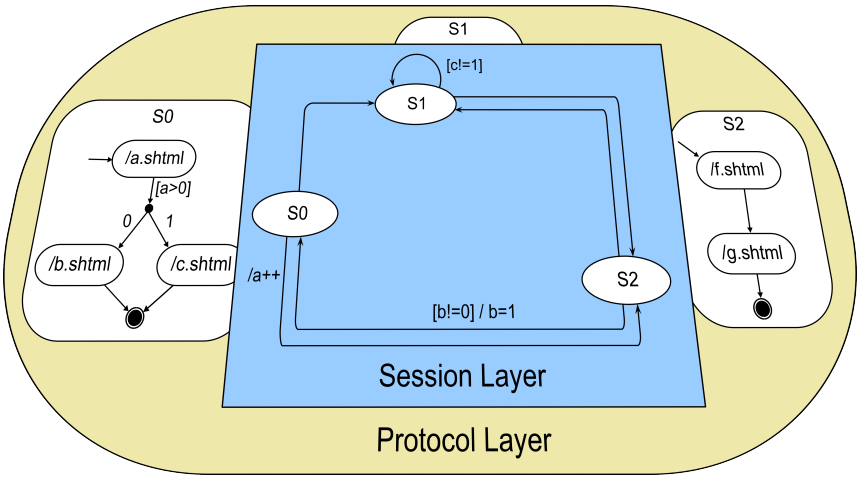


Figure 3. Sample application model illustrating the separation into session layer and protocol layer.

Application Model An application model is a two-layered hierarchical finite state machine. It consists of a *session layer* modeling the valid sequences of service invocations within a user session and a *protocol layer* specifying the related protocol details. Figure 3 displays the illustrating example used in this section.

Session Layer Each node on the session layer, called *application state*, corresponds to a service provided by the application. An edge between two states, called *application transition*, represents a valid sequence of service invocations within a session. Thus, our session layer corresponds to UML Protocol State Machines as they were introduced into version 2 of the UML standard [12].

Application transitions can be labeled with *guards* and *actions*. A guard is a boolean expression stating that a transition can only be taken if the expression evaluates to *true*. An action is a list of statements, such as variable assignments or function calls executed when a transition is taken.

The session layer in Figure 3 contains the states $S0$, $S1$, and $S2$ using the variables a, b, and c in the guards and actions. For example, a transition from state $S2$ to $S0$ is only possible if $b! = 0$ evaluates to true. When this transition fires, the variable b is assigned the value 1.

For the Web-based shopping system described in Section 5, we will demonstrate how variables, guards, and actions can be used in the application model to store additional state information during workload generation. For example, the session layer specifies that a customer must not submit a purchase request when no items are in the shopping cart. Whether an item has been added to the cart, is maintained in a dedicated variable.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

Protocol Layer Each application state has an associated finite state machine on the protocol layer. A state machine is executed when the related application state is entered. It models the sequence of protocol-level requests to be invoked. Analogous to the session layer, transitions may be labeled with guards and actions. Particularly, variables and functions can be used to assign request parameter values dynamically.

The state machine related to the application state $S0$ in Figure 3 contains the three protocol states *a.shtml*, *b.shtml*, and *c.shtml* which in this case correspond to URIs for HTTP requests. After the request for *a.shtml* has been submitted, the next state depends on the result of the evaluation of the expression $a > 0$ in the guard.

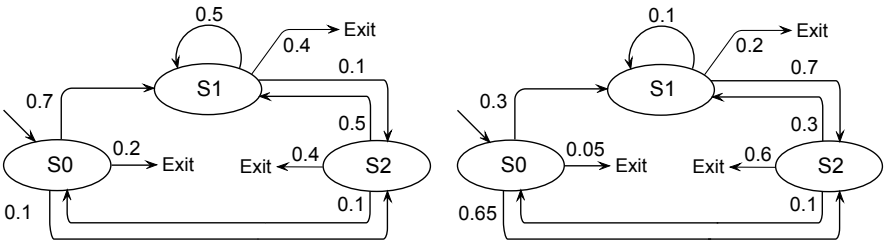
User Behavior Model In addition to an application model, our workload specification requires the definition of one or more corresponding user behavior models. A user behavior model constitutes a probabilistic model of service invocation sequences within simulated user sessions, i.e., given the last application service invoked by a user, what is the probability for each service to be invoked next by this user. A class of similarly behaving users can be represented by a single user behavior model. Additionally, such model contains a specification of the *think time*, i.e., the time period between two consecutive protocol layer requests of the same user. For each virtual user, the workload generator submits requests based on a probabilistic session model which is a composition of the application model and one corresponding user behavior model. Section 3.2 explains the semantics of this composition in detail.

The key element of a user behavior model is a Markov chain, which can be considered a probabilistic finite state machine with a dedicated entry and a dedicated exit state. Each transition between two states is weighted with a probability. The sum of probabilities associated with all outgoing transitions of each state must be 1. Aside from the additional exit state, each state in our user behavior model's Markov chain corresponds to one application state on the session layer of the application model.

Formally, we define a user behavior model $\mathcal{B}_{A,i}$ for an application model A as a tuple $(S \cup \{\mathsf{Exit}\}, P, z_0, f_{tt})$. S denotes the set of states contained in the Markov chain with entry state $z_0 \in S$. The state Exit is the dedicated exit state which has no corresponding application state. P denotes the matrix containing the transition probabilities. The transition matrix of a Markov chain with n states $s_0 \dots s_{n-1}$ is usually represented by an $n \times n$ matrix $P = [p_{i,j}]$. A value $p_{i,j}$ in the ith row and the jth column of the matrix P represents the transition probability from state s_i to s_j. The think time is specified as a probability distribution f_{tt}. For example when f_{tt} is assigned $N(300, 200^2)$, the think time is modeled according to the normal distribution $N(\mu, \sigma^2)$ with mean $\mu = 300\ ms$ and standard deviation $\sigma = 200\ ms$.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

Figure 4 shows the Markov chains of two possible user behavior models $\mathcal{B}_{A,0}$ and $\mathcal{B}_{A,1}$ corresponding to the application model with application states $S0 \ldots S2$ shown in Figure 3. Both user behavior models $\mathcal{B}_{A,0}$ and $\mathcal{B}_{A,1}$ solely differ in their transition probabilities.



(a) $\mathcal{B}_{A,0}$. **(b)** $\mathcal{B}_{A,1}$.

Figure 4. Markov chains of two user behavior models corresponding to the application model in Figure 3.

User Behavior Mix The user behavior mix specifies with which probability each user behavior model included in the workload specification occurs during workload generation. For example, let one user behavior model represent a class of users which mainly browse through the product catalog of an online shopping store without buying anything, and let a second user behavior model represent a class of users which actually buy products during their visit. These two classes of users do not necessarily occur with the same probability in real workloads.

Formally, a user behavior mix for an application A is a set $\{(\mathcal{B}_{A,0}, p_0), \ldots, (\mathcal{B}_{A,n-1}, p_{n-1})\}$ assigning probabilities p_i to user behavior models $\mathcal{B}_{A,i}$. A tuple $(\mathcal{B}_{A,i}, p_i)$ states that user sessions based on the user behavior model $\mathcal{B}_{A,i}$ occur with the probability $p_i \in [0, 1]$ during workload generation. The sum of probabilities must be 1.

Workload Intensity The workload intensity for an experiment is specified in terms of the number of active sessions, i.e., the number of virtual users being simulated concurrently. A generated session is considered *active* while the workload generator submits requests based on the corresponding probabilistic session model (the exit state of the Markov chain has not been reached). A function $n : \mathbb{R}_{\geq 0} \mapsto \mathbb{N}$ specifies this number $n(t)$ of active sessions relative to the elapsed experiment time t. Particularly, this allows for generating a varying workload intensity profile, e.g., based on measured workload data. Figure 5 shows the curve of a varying workload intensity specification for a workload generation experiment.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

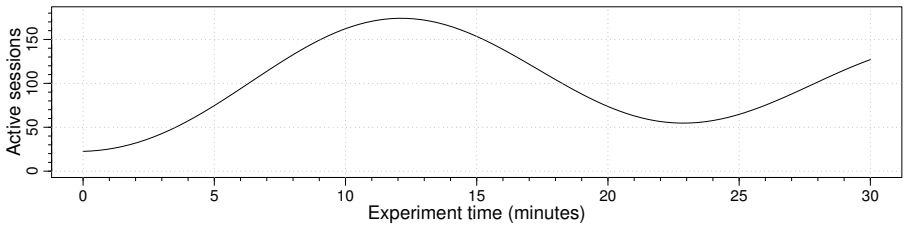


Figure 5. Curve of a varying workload intensity specification for a workload generation experiment.

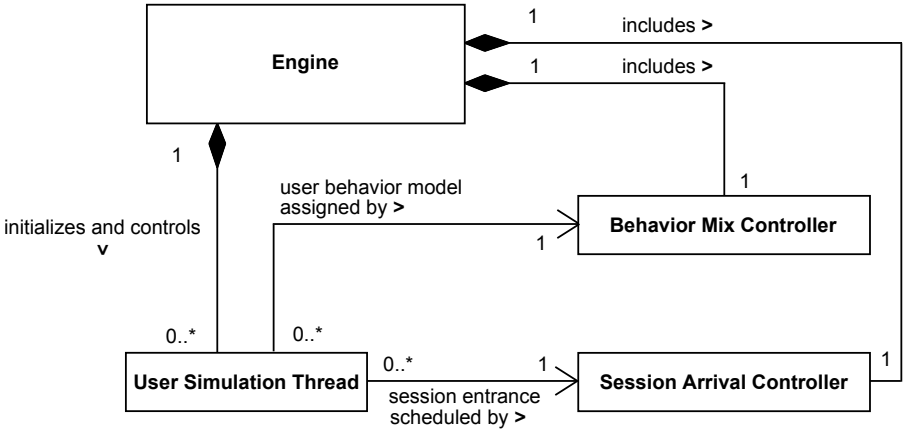


Figure 6. Architecture of the conceptual workload generator.

3.2 Workload Generation

This section describes the conceptual architecture of our workload generator. It consists of the following four components: a *workload generation engine*, a *behavior mix controller*, a *session arrival controller*, and a pool of *user simulation threads*. The workload generation engine initializes and controls the other components based on a workload specification as defined in the previous Section 3.1. Each user simulation thread periodically simulates a single user session based on probabilistic session models. The behavior mix controller assigns the user behavior models to the user simulation threads each time a new virtual user is to be simulated. The session arrival controller controls the number of active sessions according to the specified workload intensity. A more detailed description of the components, as well as the composition of the probabilistic session model and its execution, are given in the remainder of this section. Figure 6 shows the architecture including the four components and their relations as a UML Class Diagram.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

User Simulation Threads As described above, the workload generator contains a pool of user simulation threads, which are the executing entities during the workload generation. Each user simulation thread consecutively simulates users based on the specified application model and a corresponding user behavior model by executing the following steps in each iteration:

(1) Request a user behavior model from the behavior mix controller.
(2) Request the session arrival controller for a permission to execute a session.
(3) Execute the probabilistic session model which is a composition of the application model and the assigned user behavior model.

Behavior Mix Controller The behavior mix controller controls the assignment of user behavior models to user simulation threads. Before starting the simulation of a new session, in step (1) listed above, a user simulation thread is assigned the user behavior model based on which the user simulation thread generates the workload. The probability of assigning each of the user behavior models is based on the user behavior mix which is part of the workload specification.

Session Arrival Controller The session arrival controller controls the currently allowed number of active user sessions, i.e., the specified workload intensity, throughout the experiment. The controller provides a session entrance and exit protocol for the user simulation threads which is similar to the concept of synchronizing processes using semaphores [13].

- The blocking operation *enterSession()* must be called by a user simulation thread when starting the simulation of a session for a new virtual user, i.e., in the above-listed step (2). The operation returns immediately if the current number of active sessions is lower than the current maximum number of active sessions specified in the workload intensity function. Otherwise, the user simulation thread gets blocked in a waiting queue until the number of active sessions falls below the specified number.
- The non-blocking operation *exitSession()* must be called by a user simulation threads when the simulation of the probabilistic session model ends, i.e., after step (3). Thus, the number of active sessions is decremented by 1.

Probabilistic Session Model As explained in Section 3.1, the application model defines the *allowed* sequences of service invocations submitted within a user session and contains all protocol-level details required to generate valid requests; the actual order of service invocations and the think times between two consecutive requests are specified in the *user behavior models* corresponding to the application model. An application model and a corresponding user behavior model are directly related by the application states and the states of the Markov chain. We mentioned, that the actual requests to the Web-based software system are generated by the user simulation threads which periodically execute a

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

composition of the application model and a corresponding user behavior model – denoted a *probabilistic session model*. Now, we will define the semantics of this composition.

The composition of the application model and a user behavior model into a single probabilistic session model executed by a user simulation thread is performed straightforward by enriching the application transitions with the probabilities contained in the Markov chain of the user behavior model. Starting with the entry state z_0 defined in the user behavior model, a probabilistic session model is executed as follows. Given a current state, the next state is determined by first evaluating the guards of the outgoing transitions related to the current state. One of the transitions whose guards evaluate to *true* is randomly selected based on their assigned probabilities. The action of the selected transition is executed and the requests towards the application are submitted by traversing the deterministic state machine of the state within the protocol layer of the application model. A session ends when the determined transition leads to the Exit state of the user behavior model.

4 Tool for Generating Probabilistic and Intensity-Varying Workload

Based on the conceptual approach for generating probabilistic and intensity-varying workload presented in Section 3, we implemented a workload generation tool. Implementing such a tool from scratch would have required us to implement a bunch of low-level functionalities which do already exist in a number of workload generation tools (cf. [11] for an overview of existing tools). Instead, we integrated our approach into the popular open source workload generator Apache JMeter [3], and could thus focus on the implementation of those functionalities specific to our approach. Our extension, called Markov4JMeter, is freely available [14] under an open source license. The following Section 4.1 gives an overview of JMeter including relevant parts of its architecture. Section 4.2 describes the implementation of Markov4JMeter and the integration into JMeter. It is demonstrated how the sample workload specification used as the running example in Section 3 is defined in our workload generation tool.

4.1 Apache JMeter

Apache JMeter [3] is a Java-implemented workload generation tool for testing Web applications particularly in terms of performance. The workload is specified graphically in a so-called *Test Plan* which is a tree of *Test Elements*. The core Test Elements are *Logic Controllers* and *Samplers*. Logic Controllers, e.g., *If* and *While Controllers*, group Test Elements and define the control flow of a Test Plan when being executed. Samplers, such as *HTTP Request* or *FTP Request*, are located at the leafs of the tree and send the actual protocol-level requests. A test run can both be started by means of the graphical user interface (GUI) and from the command line using the non-GUI mode.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

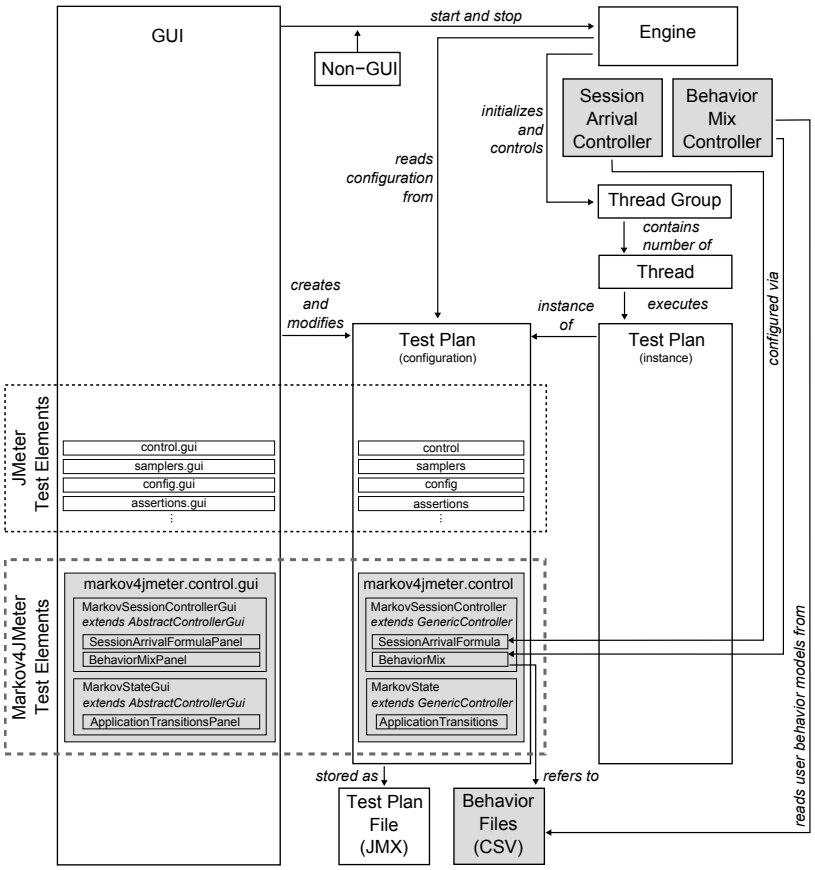


Figure 7. Integration of Markov4JMeter into the architecture of JMeter. The gray elements are Markov4JMeter components.

The internal architecture of JMeter including the core components and their relations is illustrated in Figure 7 (the non-gray elements). The *Engine* is responsible for controlling the workload generation run. It initializes the *Thread Group* including the specified number of *Threads* (Java threads). Each Thread, represents a virtual user and executes an instance of the Test Plan. A Test Plan is internally represented by a tree of Test Element classes (Java classes) corresponding to the respective Test Elements in the Test Plan. Each Test Element class contains the implementation of the Test Element's behavior. Also, it has a corresponding GUI class providing the configuration dialog for the Test Element. Moreover, the GUI class is responsible for creating and modifying the Test Element classes. Test Plans including the configuration of the Test Elements are stored in JMX files, a JMeter-specific XML format.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

4.2 Markov4JMeter

This section presents our JMeter extension called Markov4JMeter which allows for using JMeter to define and execute a workload specification according to the approach described in Section 3. A probabilistic workload specification as defined in Section 3.1 can be integrated into a JMeter Test Plan using the two additional Logic Controllers, *Markov Session Controller* and *Markov State*, added by Markov4JMeter. Moreover, Markov4JMeter includes a Session Arrival Controller and a Behavior Mix Controller corresponding to the components of the conceptual workload generator presented in Section 3.2. The remaining two components, workload generation engine and the pool of user simulation threads, could be mapped to the JMeter components Engine and Thread Group including the JMeter Threads. The Markov chains of the user behavior models are read from external comma-separated value (CSV) files. Figure 7 illustrates how the Markov4JMeter components are integrated into JMeter.

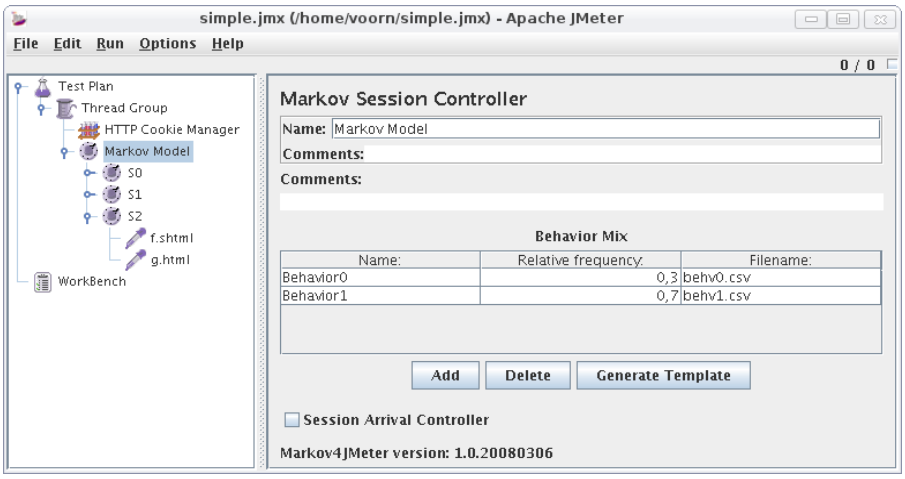
Session Controller This Logic Controller constitutes the root of a probabilistic session model within a Test Plan. According to the JMeter Test Elements, the Markov Session Controller is divided into a Test Element class and a GUI class including the configuration dialog.

The Test Element class contains the implementation of the session model composition and execution as described in Section 3.2. In each iteration, i.e., each time a new session is to be simulated, the Markov Session Controller requests a behavior from the Behavior Mix Controller and requests the Session Arrival Controller to start the execution of this session. An iteration ends when the exit state of the behavior model is reached. The configuration dialog allows the definition of the behavior mix and the configuration of the Session Arrival Controller. A screenshot is shown in Figure 8(a). The behavior mix is defined by selecting the respective behavior files and specifying the desired probabilities. The formula defining the number of allowed active sessions during the test execution must evaluate to a positive integer.

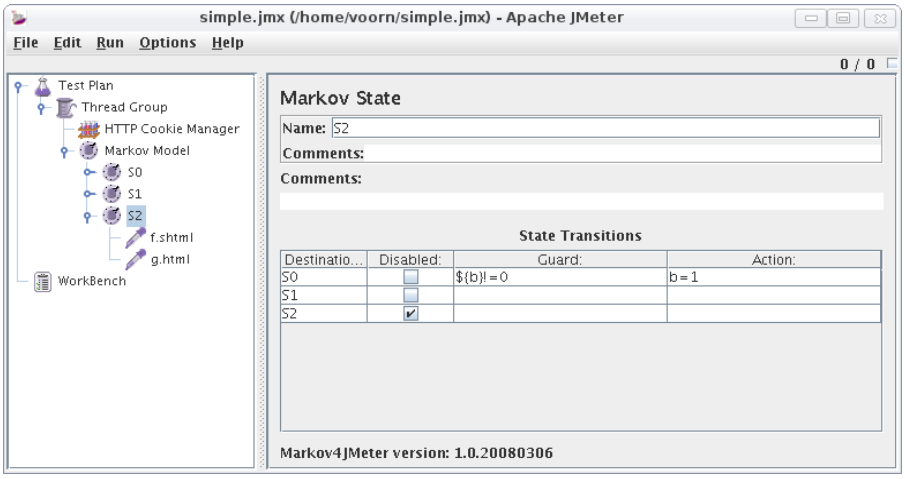
Markov State Markov State Test Elements are added directly underneath the Markov Session Controller. Each of these Logic Controllers represents an application state. Any subtree of JMeter Test Elements can be added to a Markov State representing the related deterministic state machine on the protocol layer of the application model. As the implementation of the Markov Session Controller, the Markov State is divided into a Test Element class and a GUI class.

The application transitions are configured within the configuration dialogs of the Markov States. Figure 8(b) shows the configuration of the application transitions starting in state $S2$ of the application model in Figure 3. The configuration dialog of the Test Element allows the definition of the state transitions with guards and actions using JMeter's variables and functions. The Markov State *S2* in Figure 8(b) contains the HTTP Samplers *f.shtml* and *g.shtml* which are executed in this order according to the application model in Figure 3.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com



(a) Probabilistic Test Plan and configuration dialog of the Markov Session Controller including the definition of the user behavior mix.



(b) Probabilistic Test Plan and configuration dialog of the Markov State $S2$. Disabling a transition is equivalent to a non-existing transition or to assigning a guard the value *false*.

Figure 8. Screenshots showing the probabilistic Test Plan and configuration dialogs of the Markov Session Controller and a Markov State. The Test Plan corresponds to the example from Section 3.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

Session Arrival Controller According to Section 3.2, the Session Arrival Controller provides the methods *enterSession()* and *exitSession()* which are called by the Markov Session Controller before starting to execute a new session. Depending on the current number of active sessions and the configured workload intensity, a thread might get blocked until the session entrance is granted. The active sessions function is specified as a Java expression (using BeanShell[1]) which evaluates to an integer value. Markov4JMeter provides a variable for the elapsed experiment time. BeanShell scripts in external files can be used as well.

Behavior Mix Controller As mentioned above, the Behavior Mix Controller assigns user behavior models to the Markov Session Controller based on the configured behavior mix. The models are read from the behavior files and converted into an internal representation which is passed to the Markov Session Controller. Figure 8(a) show a Behavior Mix Controller configuration with two user behavior models.

Behavior Files The Markov chain of each user behavior model is stored in a comma-separated value (CSV) file which can be edited with any spreadsheet application. It contains the names of all Markov States underneath a Markov Session Controller. The configuration dialog of the Markov Session Controller allows to generate valid behavior templates for the current Test Plan. Figure 9 shows the behavior file of the user behavior model in Figure 4(a). Valid behavior file templates can be generated through the Markov Session Controller configuration dialog (see Figure 8(a)).

```
    , S0   , S1   , S2   , $
S0* , 0.00 , 0.70 , 0.10 , 0.20
S1  , 0.00 , 0.50 , 0.10 , 0.40
S2  , 0.10 , 0.50 , 0.00 , 0.40
```

Figure 9. User behavior model of Figure 4(a) stored in CSV file format. The entry state of the model is marked with an asterisk (at most one). The column labeled with `$` represents the transition probability towards the exit state.

5 Case Study

This section demonstrates how probabilistic and intensity-varying workload for the iBATIS[2] JPetStore Web application can be specified using our approach and

[1] `http://www.beanshell.org/`
[2] `http://ibatis.apache.org/`

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

the corresponding tool Markov4JMeter, which have been presented in the previous Sections 3 and 4. Section 5.1 provides a basic overview of the JPetStore application. The workload specification following our approach presented in Section 3.1 is described in Section 5.2. Section 5.3 demonstrates how Markov4JMeter is used to create a JMeter Test Plan corresponding to this specification. Section 5.4 provides some interesting measurement results of workload generation runs which demonstrate the usefulness of our approach.

5.1 JPetStore

The iBATIS[3] JPetStore is a Java Web application which represents an online shopping store that offers pets. An HTML Web interface provides access to the application. The product catalog is hierarchically structured into *categories*, e.g., "Dogs" and "Cats". Categories contain *products* such as a "Bulldog". Products contain the actual *items*, e.g., "Male Adult Bulldog", which can be added to the virtual shopping cart, the content of which can later be ordered after having signed on to the application and having provided the required personal data, such as the shipping address and the credit card number.

5.2 Workload Specification

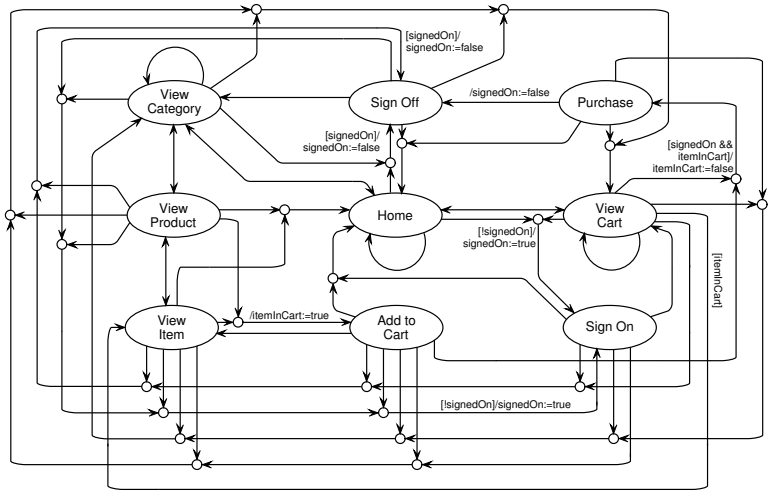
In order to define an application model including the session layer and the protocol layer underneath (cf. Section 3.1), we identified 29 protocol request types provided by JPetStore on the HTTP protocol level. These request types were categorized into 15 application services. We selected a subset of 9 services and the corresponding 13 request types considered part of a "typical" user session. The application transitions of the application model's session layer were defined based on the hyperlinks being present on the Web pages of the JPetStore. For example, by entering the application state *Home*, the server would return the JPetStore index page. This page provides hyperlinks to the product categories, to the shopping cart, to the index page itself, and allows to sign on or off.

Figure 10(a) shows the session layer of the application model which contains the 9 application states. The variables *signedOn* and *itemInCart* are used to store additional state information. A user can only sign on and sign off if the value of the variable *signedOn* is *false* or *true*, respectively. The variable *itemInCart* is assigned the value *true* when an item is added to the shopping cart. A transition to the state *Purchase* can only be selected when a user has signed on and has added at least one item in the shopping cart.

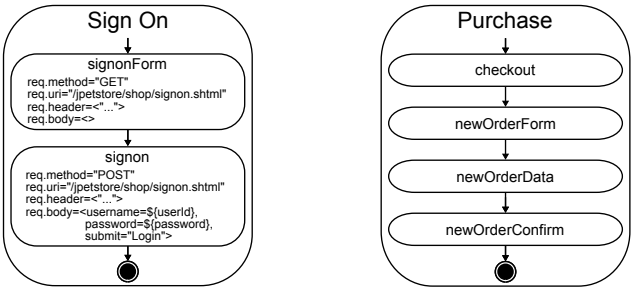
The protocol layer is specified based on the 13 considered HTTP request types. For each request type we determined its required HTTP request method, the URI, and parameters to be passed on an invocation. The protocol state machines corresponding to the application states *Sign On* and *Purchase* are shown in Figure 10(b). In order to sign on, a user first invokes an HTTP request of type *signonForm* using the HTTP protocol method GET. The server returns a

[3] `http://ibatis.apache.org/`

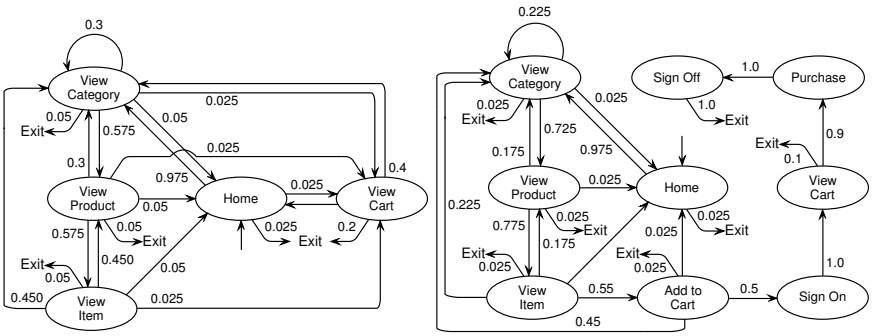
Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com



(a) Session layer of the application model. The junction connector ◯ is used to combine a set of transitions from multiple states to the same destination state (label considered label of all transitions in this set).



(b) Protocol state machines for two application states.



(c) Markov chains of the user behavior models *Browser* (left) and *Buyer*.

Figure 10. The application model (the session layer displayed in (a); two of the nine protocol-layer state machines displayed in (b)) and the two user behavior models (c) specified for the JPetStore.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

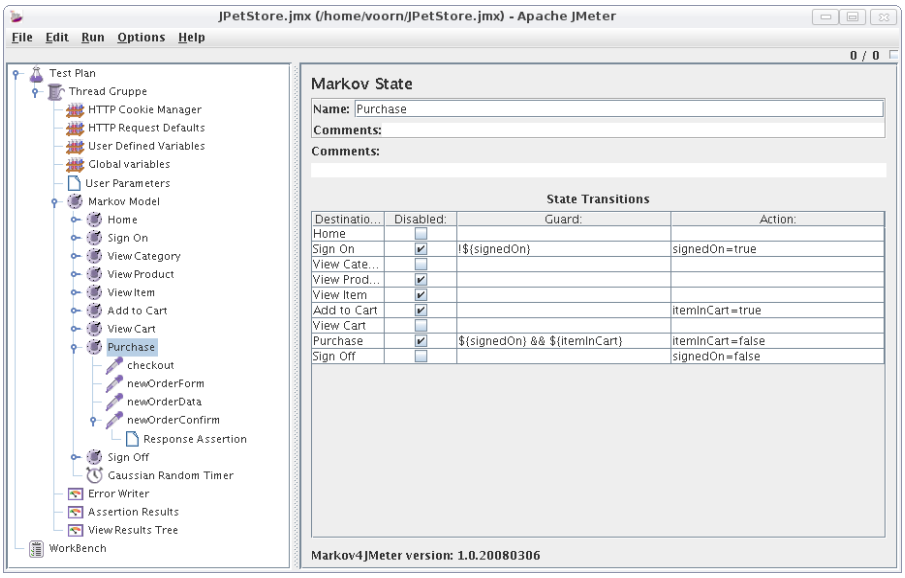


Figure 11. Probabilistic Test Plan for the JPetStore (corresponding to the underlying formal workload specification displayed in Figure 10) and the transition configuration of the Markov State *Purchase*.

form asking for a username and a password. In a subsequent invocation, the user passes the filled in data of the completed form by invoking the HTTP request type *signon*. The variables *userId* and *password* are used as placeholders for the username and password. The protocol state machine of the application state *Purchase* shows the sequence of HTTP requests to be executed when purchasing. We omitted the HTTP protocol details for this state.

We defined one user behavior model representing users solely browsing through the JPetStore and a second one where users tend to actually buy items from the store. The Markov chains of both models are displayed in Figure 10(c). For both models we specified a think time distribution $f_{tt} = N(300, 200^2)$ which is a parameterized normal distribution with mean $\mu = 300$ and standard deviation $\sigma = 200$, both values given in milliseconds.

5.3 Test Plan

As explained in Section 4.2, we created a probabilistic Test Plan for the JPetStore application model and the two user behavior models presented in the previous Section 5.2 using the additional Markov4JMeter Logic Controllers, Markov Session Controller and Session Arrival Controller. The Test Plan, as well as the configuration dialog of the Markov State *Purchase* including the definition of the application transitions, are shown in Figure 11. The active sessions function

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

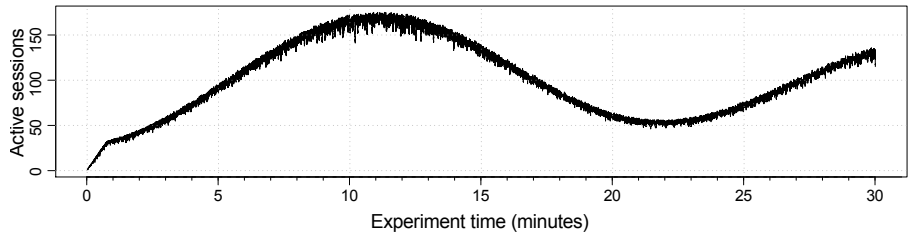


Figure 12. Measured number of active sessions during a probabilistic and intensity-varying workload generation run. The workload intensity was specified according to the curve shown in Figure 5.

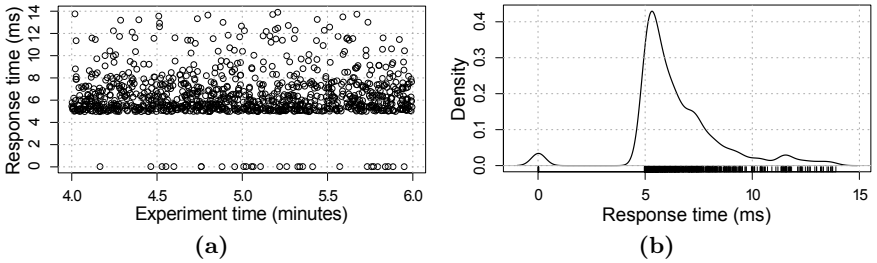


Figure 13. Scatter plot (a) and probability density plot (b) of method response times measured during a workload generation run with probabilistic workload and a constant workload intensity.

is configured to be read from an external BeanShell script. A Random Timer Test Element provides the think time.

Identifiers for categories, products, and items are randomly selected using a dedicated Markov4JMeter function before the respective request is submitted. Assertions are inserted to detect application errors which are not reflected in HTTP error codes. The server response of some requests is parsed for specific text strings in order to make sure that the requests have been processed correctly by the JPetStore. For example, after having signed on, the returned Web page must contain the string "Welcome" as well as a hyperlink labeled "Sign Out". "Thank you, your order has been submitted" must appear after having confirmed the order.

5.4 Measurement Results

Markov4JMeter has been used in a large number of workload generation experiments with the JPetStore and the workload specification described in the previous sections for the experimental evaluation of our research in the domains of performance evaluation [15], anomaly detection and automatic fault localization [16], as well as runtime reconfiguration of component-based software sys-

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

tems [17]. In this section we give two interesting measurement results of separate workload generation runs to demonstrate the usefulness of our approach.

Figure 12 displays a curve of the measured number of active sessions during a 30-minute workload generation run. The workload intensity was specified according to the curve shown in Figure 5. The number of active sessions was extracted from the Web server access logs. Markov4JMeter shows the expected behavior and varies the workload intensity following the input specification. The jitter is caused by the measurement granularity (1 ms) and the queueing implementation in the Session Arrival Controller.

For another experiment, Figure 13 shows the response time scatter plot and the corresponding probability density plots of the Java method *addItemToCart*. A constant workload intensity of 55 active sessions was specified for the entire run. As indicated by its name, the method *addItemToCart* is always executed when a users adds an item to the virtual shopping cart. The plots show that sporadically significantly lower response times for method executions occur. We found out that these low response times occur when a user adds an item with the same identifier to the cart more than once within the same session. This only requires a counter to be incremented. It is very likely that these low response times would not have been uncovered without our probabilistic workload and the random selection of item identifiers as described in Section 5.3.

6 Conclusions

This paper demonstrated our approach for specifying and generating probabilistic and intensity-varying workload for Web-based software systems. The workload specification provides a clean separation between application-specific details including the specification of allowed sequences of service invocations and all protocol-level details required to generate valid requests with the required technical details, as well as the corresponding models of probabilistic usage based on Markov chains. We presented a conceptual workload generator which generates workload based on the described specification. By including the specification of the (possibly varying) workload intensity, long-term load tests with realistic workload intensity profiles can be performed.

The corresponding workload generation tool Markov4JMeter has been implemented as an extension for the popular workload generator Apache JMeter. By being based on JMeter, probabilistic workload specifications for any protocol supported by JMeter can be executed. In a case study, we applied the workload generation technique to the JPetStore Web application by first specifying the underlying workload model and then creating the Test Plan executable by JMeter extended by Markov4JMeter.

Markov4JMeter is freely available at [14]. It is being used to generate probabilistic and intensity-varying workload for the evaluation of research in the domain of software timing behavior evaluation, anomaly detection and automatic fault localization, as well as runtime reconfiguration of component-based software systems.

Appeared in S. Kounev, I. Gorton, and K. Sachs (eds.), Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08), volume 5119 of Lecture Notes in Computer Science, pages 124–143, Heidelberg. Springer. June 2008.
The original publication is available at www.springerlink.com

References

1. Menascé, D.A.: Load testing of web sites. IEEE Internet Computing **6**(4) (2002) 70–74
2. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. In: Proceedings of the ACM SIGMETRICS, ACM (1998) 151–160
3. Apache Software Foundation: JMeter. `http://jakarta.apache.org/jmeter/`
4. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Request for comment (RFC) 2616: Hypertext Transfer Protocol – HTTP (1999)
5. Menascé, D.A., Almeida, V.A.F., Fonseca, R., Mendes, M.A.: A methodology for workload characterization of e-commerce sites. In: Proceedings of the ACM Conference on Electronic Commerce (EC '99), ACM (1999) 119–128
6. Whittaker, J.A., Thomason, M.G.: A markov chain model for statistical software testing. IEEE Transactions on Software Engineering **20**(10) (1994) 812–824
7. Li, Z., Tian, J.: Testing the suitability of markov chains as web usage models. In: Proceedings of the 27th International Conference on Computer Software and Applications (COMPSAC '03), IEEE (2003) 356–361
8. Ballocca, G., Politi, R., Ruffo, G., Russo, V.: Benchmarking a site with realistic workload. In: Proceedings of the 5th IEEE International Workshop on Workload Characterization (WWC-5), IEEE (2002) 14–22
9. Krishnamurthy, D., Rolia, J.A., Majumdar, S.: A synthetic workload generation technique for stress testing session-based systems. IEEE Transactions on Software Engineering **32**(11) (2006) 868–882
10. Shams, M., Krishnamurthy, D., Far, B.: A model-based approach for testing the performance of web applications. In: Proceedings of the International Workshop on Software Quality Assurance (SOQUA '06), ACM (2006) 54–61
11. Peña-Ortiz, R., Sahuquillo, J., Pont, A., Gil, J.A.: Modeling continuous changes of the user's dynamic behavior in the WWW. In: Proceedings of the 5th International Workshop on Software and Performance (WOSP '05), ACM (2005) 175–180
12. Arlow, J., Neustadt, I.: UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition). Addison-Wesley (2005)
13. Dijkstra, E.W.: Cooperating sequential processes. In Genuys, F., ed.: Programming Languages. Academic Press (1965)
14. van Hoorn, A.: Markov4JMeter. `http://markov4jmeter.sourceforge.net/`
15. van Hoorn, A.: Workload-sensitive timing behavior anomaly detection in large software systems (September 2007) Master's thesis (Diplomarbeit), Department of Computing Science, University of Oldenburg, Germany.
16. Rohr, M.: Workload-sensitive Timing Behavior Anomaly Detection for Automatic Software Fault Localization. PhD thesis, Department for Computing Science, University of Oldenburg, Oldenburg, Germany (2008) work in progress.
17. Matevska, J., Hasselbring, W.: A scenario-based approach to increasing service availability at runtime reconfiguration of component-based systems. In: Proceedings of the 33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE (2007) 137–144