A Model-Based Performance Testing Toolset for Web Applications

Diwakar Krishnamurthy, Mahnaz Shams, and Behrouz H. Far

Abstract - **Effective performance testing techniques are essential for understanding whether a Web-based application will meet its performance objectives when deployed in the real world. The workload of such an application has to be characterized in terms of sessions; a session being a sequence of inter-dependent requests submitted by a single user. Dependencies arise because some requests depend on the responses of earlier requests in a session. To exercise application functions in a representative manner, these dependencies should be reflected in the synthetic workloads used to test Web-based applications. The need to handle these dependencies makes performance testing a time consuming process. Specifically, since the nature of these dependencies varies across systems considerable effort needs to be dedicated towards tailoring a set of test tools that will work for a given system. Furthermore, the traditionally followed approach of manually developing test scripts that handle dependencies makes it difficult to achieve fine-grained control over a synthetic workload's characteristics. In this paper, we propose a new model-based approach to address these issues. Our approach uses an application model that captures the dependencies for a Web-based application system under study. The application model provides an indirection that allows a common set of workload generation tools to be used for testing different applications. Consequently, less effort is needed for developing and maintaining the testing tools and more effort can be dedicated towards the performance testing process. Furthermore, the application model can be used to obtain a large set of valid request sequences representing how users typically interact with the application. This feature facilitates techniques that can manipulate the set of sequences to achieve automated control over the characteristics of the synthetic workloads used in a testing exercise. The utility of the toolset is demonstrated by applying it to model two different Web applications and through a study that systematically investigates the impact of various workload characteristics on the performance of an e-commerce application.**

***Index Terms* - Synthetic Workload Generation, Software Performance Engineering, Finite State Machine Models**.

Manuscript received January 20, 2009. This work was financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of Calgary.

D. Krishnamurthy is with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB, T2N 1N4, Canada. (phone: 403-220-8129; fax: 403-282-6855;email: dkrishna@ucalgary.ca)

M. Shams is with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB, T2N 1N4, Canada. (email: mshams@ucalgary.ca)

B. H. Far is with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB, T2N 1N4, Canada. (email: far@ucalgary.ca)

I. INTRODUCTION

Enterprises are increasingly relying on Web-based systems to support critical business functions. Such systems often host complex, multi-tier applications such as e-commerce and business process management. With these systems, responses to user requests are typically generated dynamically by invoking the service of one or more server tiers. The system workload has to be characterized in terms of sessions; a session being a sequence of inter-dependent requests submitted by a single user. Dependencies arise because some requests depend on the responses of earlier requests in a session. For example, an order cannot be submitted to an e-commerce system unless the previous requests have resulted in an item being ordered. We define this phenomenon as *inter-request dependency* and refer to this class of systems as *session-based systems.*

Poor performance of session-based systems can adversely impact the profitability of an enterprise. As a result, effective performance testing techniques are essential for understanding whether a session-based system will meet its performance objective when deployed in the real world. *Performance testing* is a technique where synthetic workloads [7] are submitted to a system under study within a controlled environment. The *synthetic workload* used in a performance test serves to mimic the request patterns of real system users. Synthetic workloads are constructed from a workload model. A *workload model* specifies statistical characterizations for a set of workload attributes that are expected to affect performance the most. Measurements such as user response times and server utilizations are collected during the tests and used to support capacity planning and service level assessment exercises.

In general, a performance testing methodology must address several requirements with respect to synthetic workloads. Firstly, to reach reliable conclusions based on the results of a performance test, the synthetic workloads used must be representative of real workloads. A synthetic workload is said to be representative of a real workload if both workloads result in similar performance when submitted to a system. The representativeness of a synthetic workload is significantly influenced by the attributes in the workload model and the characterizations used for them [7]. Furthermore, for session-based systems requests within a session in the synthetic workload must reflect the correct inter-request dependencies to exercise application functions representatively. Krishnamurthy *et al.* experimentally showed that incorrect performance estimates can result when inter-request dependencies are ignored while constructing

(Advance online publication: 13 May 2010)

synthetic workloads [12]. Secondly, since it is very difficult to know precisely what a real workload's characteristics will be, a performance testing methodology must provide the flexibility to conduct a controlled sensitivity analysis on the characterizations of the workload model's attributes.

The need to satisfy inter-request dependencies makes it difficult to satisfy the representativeness and flexibility requirements. Typically, a small set of system-specific scripts is manually developed to create synthetic workloads. In a given test, each script has an *execution weight* that determines the number of times that script is executed and hence the overall characteristics of the synthetic workload generated. Different synthetic workload characteristics are achieved by manually selecting different sets of script execution weights. Such an approach can be time consuming especially when the characterizations of workload attributes need to be varied in a fine-grained and controlled manner. Furthermore, since the scripts developed are system-specific they need to be modified when changes are made to a system (e.g., changes in inter-request dependency, addition of new functionality). The manual selection of execution weights for the new set of modified scripts needs to be repeated again to construct the various desired synthetic workloads further increasing the performance testing effort. As a result, very often ad-hoc workloads are used and insufficient sensitivity analysis is carried out. Consequently, results from such tests are not likely to provide reliable insights into system performance.

In this paper, we propose a new model-based approach that addresses these limitations. Our approach uses an application model that captures the application logic of a session-based system under study. Essentially, the application model can be used to obtain a *large set* of user request sequences that satisfy the correct inter-request dependencies for the system under study in an *automated* manner. The ability to generate a large set of valid sequences, i.e., scripts, makes it possible to apply techniques [12] that can *automatically* determine the script execution weights needed to construct a synthetic workload with desired characteristics. Specifically, the advantages of the model-based approach over the traditional performance testing approach are as follows:

- The approach provides automated support for fine-grained control of workload characteristics. Many different synthetic workloads can be automatically constructed to support flexibility with respect to workload characteristics. For example, the approach would make it easy to create controlled workloads to study how varying the characteristics of a particular workload attribute impacts system performance.
- Lesser effort is needed to adapt the synthetic workload generation tools to handle changes made to a given system or to use them for testing other systems. This improved portability is a result of the indirection provided by the application model. The application model essentially makes the workload generation tools independent of the system under study. For example, by using different application models the same set of tools can be used to test an e-commerce system, a modified version of the e-commerce system and an online auction system.

To the best of our knowledge, we are not aware of existing work that addresses the research issues we have considered in this paper. As described in Section II, other model-based approaches do not support certain types of dependencies that occur commonly in Web applications. Furthermore, in contrast to our toolset described in Section III, tools that are frequently used to test Web applications do not support the ability to automatically control synthetic workload characteristics.

The rest of the paper is organized as follows. Section II discusses background and related work. An overview of the proposed approach is provided in Section III. Section IV discusses a methodology to construct application models for the purpose of performance testing. Section V briefly describes the implementation of a toolset embodying the proposed methodology. Section VI presents a case study that shows the ability of our approach to model two different Web applications. Section VII demonstrates the utility of the toolset through a performance evaluation case study of the Java Pet Store [9] e-commerce application. Section VIII provides conclusions and discusses future work.

II. BACKGROUND AND RELATED WORK

Knowledge of the characteristics of workloads observed at real systems can provide insights for creating representative synthetic workloads for performance testing. Menasce *et al.* characterized the workloads from an online bookstore and an online auction site [15]. Analysis of the bookstore revealed that over 70% of the total requests were for browsing books. Only 1.19% of the requests were associated with purchases. The auction site also displayed predominance of browse-related requests but not to the extent seen in the bookstore site. Furthermore, the authors found a significant percentage of sessions that had a very large number of requests. These were found to originate from automated robots that crawled the system endlessly to collect information such as the pricing of products.

Arlitt *et al*. characterized the workload of a large Web-based shopping system [3]. Over 95% of total requests in the workload they studied are for dynamically generated objects. An analysis of object popularity revealed that a large fraction of the total requests is for a small number of very popular objects. The study also reported a significant presence of robots, which is consistent with the results reported by Menasce *et al.* [12]. Finally, the study found that there is a large difference in the resource demands for the different types of requests (e.g., `Browse`, `Search`, `Purchase`) in the system. For example, a request to search a product placed 40 times more demand on the bottleneck resource of the system than a request to browse a product.

Kelly [11] characterized the workload at several real session-based systems and found a very strong correlation between the transaction mix (i.e., the proportions of different types of requests) and system performance. Specifically, the work found that the response times observed at the session-based systems over a given time interval could be predicted accurately based on the transaction mix they encountered in

that time interval. In a follow up work, Stewart *et al.* showed that transaction mixes observed at real session-based systems exhibit significant time varying behaviour or non-stationarity [22]. Specifically, the server was found to encounter different mixes over different time intervals.

The arrival of requests at many session-based systems was found to be self-similar [10] [15] [25]. *Self-similar* workloads exhibit significant request correlations or bursts over multiple timescales [6]. Researchers have argued that such a behaviour could possibly be due to the high variability in the number of requests in a session [15] and the high variability in request service times [25] observed in these systems.

The studies discussed so far identify several workload attributes that need to be considered while generating synthetic workloads for session-based systems. Specifically, techniques are required to control attributes such as the number of requests per session, transaction mix, and bursts in request arrivals. This would allow realistic workloads to be constructed and facilitate studies that help understand the sensitivity of system performance to these workload attributes. The traditional performance testing approach of manually creating scripts and manually determining script execution weights does not permit comprehensive sensitivity analyses exercises.

Generating synthetic Web workloads typically involves two steps: trace generation and request generation. The trace generation step handles the complexities of creating a synthetic trace of HTTP/HTTPS requests that adhere to a workload model. The request generation step submits the requests in the trace to the system under study. Pre-generating traces reduces overheads during request generation thereby ensuring that the achieved workload characteristics stay close to the specified characteristics. The `SURGE` [5] tool supports trace and request generation capabilities for testing Web servers. `S-Clients` [4] and `httperf` [17] are request generation tools that are capable of generating realistically heavy overloads during performance tests.

Requests for session-based systems can be split into two parts: 1) a request type; and 2) a name-value list. Typically a *request type* instructs an application server to execute a particular server-side script and the *name-value list*, which follows the request type, provides input data for the script. Each entry in the name-value list consists of a name of a parameter and the parameter's value. For example, the request `/Add?product_id=1&number=2` has `Add` as its request type and `product_id=1&number=2` as its name-value list. The two parameters for this request are `product_id` and `number`. For session-based systems the trace generation step must address the issue of handling inter-request dependencies. For example, considering an e-commerce system, a session can issue a `Purchase` request only after a product has been added to the shopping cart through a previous `Add` request. As described shortly, other researchers have considered the problem of inter-request dependencies.

For session-based systems, trace generation must also address data dependencies. *Data dependencies* govern the choice of values for parameters in the name-value lists. For example, a `Login` request type in an e-commerce system has to be submitted with valid user name-password combinations. To the best of our knowledge, we are not aware of other studies which have addressed this problem as it pertains to performance testing.

Finite State Machines (FSMs) have been extensively used to model application-specific dependencies. A FSM consists of sets of states, inputs, and outputs. Applying a set of inputs causes transition from one state to another state and produces a set of outputs. FSM models are widely used to test whether an implementation of a software application conforms to specifications. An FSM is used to describe the specifications. In essence, the transitions in the FSM capture the desired behaviour for the implementation. Testing typically involves obtaining sequences of input sets called test sequences from the FSM. Each test sequence is applied to the implementation and the resulting sets of outputs are observed. Decisions about the correctness of the implementation are made by comparing the observed behaviour with the desired behaviour as given by the FSM.

Andrews *et al.* proposed an FSM-based model to automate conformance testing of Web applications [1]. First, an FSM-based model is used to describe a Web application. A state in the FSM corresponds to what the authors define as a logical Web page (LWP). A LWP can be a physical Web page produced by the application or specific HTML links and forms used to interact with the application. Transitions occur between LWPs in response to user inputs (e.g., submitting a user name and password through a HTML form). Inter-request dependencies are enforced by controlling the transitions allowed between LWPs. The authors also annotate the transitions to handle several other types of dependencies. The annotations provide information on whether an input for a LWP is required or optional and the sequences in which a user can supply inputs to a LWP. The authors demonstrated the technique on a simple student information Web application.

Menasce *et al.* applied an FSM-based approach for the performance testing of Web applications [16]. The approach uses a probabilistic FSM called a *Customer Behaviour Model Graph* (CBMG) to model a user's session with a Web application. Similar to the FSM employed by Andrews *et al.* [1], the states of the CBMG represent the different request types supported by the Web application. The number of states equals the number of request types and transitions between states model the user behaviour of navigating from one request type to another within a session. In contrast to a non-probabilistic FSM, transitions are associated with probabilities with the sum of a state's outgoing transition probabilities being one. The transition probability gives the likelihood of a user choosing a particular transition from a state from among all the allowed transitions. By traversing the CBMG, a trace of synthetic sessions can be created for performance testing. CBMGs are also employed to specify and generate the synthetic workloads used in popular Web application benchmarks such as TPC-App [24].

Both the FSM-based approaches reviewed may not be expressive enough for modeling several types of dependencies commonly observed in session-based systems. Specifically, we discuss the following issues with these approaches:

1) Inability to fully support inter-request dependencies – Both approaches use a first-order FSM where a state is defined to be a request type. Such a definition implicitly introduces a first-order dependency between request types. With these models the next request type that needs to be generated within a session (i.e., the next state) depends solely on the current request type. However, such an assumption may not be valid in practice. For example, consider the following valid session of request types for an e-commerce system: [`Home, View, Add, View, Add, Delete, Purchase`]. This session describes a shopper who adds two products to the shopping cart, then deletes one of the products from the shopping cart before purchasing the other product. An FSM constructed based on this session and based on the first-order assumption would incorrectly deduce a dependency between `Delete` and `Purchase` without recognizing that the `Purchase` depended on one of the two previous `Add` states. As a result, the FSM could cause a sequence ([`Home, View, Add, Delete, Purchase`]) that invokes `Purchase` without any item in the shopping cart thereby violating inter-request dependencies for the system.

Higher-order dependencies can be captured, for example, by introducing states that represent sequences of request types. For example, creating a new state [`Add, View, Add, Delete`] can ensure that `Purchase` is invoked only when there is an item in the shopping cart. However, this may cause the well-known state explosion problem [14]. Another approach could be to build the FSM such that it only yields sequences with first-order request dependencies. Such an approach may not produce representative workloads since real workloads could contain several sequences with higher-order dependencies. Furthermore, our previous experience suggests that one can obtain more control over characteristics such as workload mix when there is more "variety" in the set of sequences a user can follow. We propose an alternate approach in Section IV.

2) Lack of support for data dependencies – The existing approaches have not focused on modeling data dependencies that are important for performance testing. Handling data dependencies is important since the correct choice of parameter values is essential for stressing the system under study in a representative manner. The application model must capture data dependencies so that they can be satisfied in an automated manner while creating workloads.

In this paper, we propose a new application modeling methodology to handle inter-request dependencies and data dependencies. The methodology is based on Extended Finite State Machines (EFSMs). EFSMs can model applications with higher-order request dependencies without encountering the state explosion problem [14]. Additionally, the modeling methodology layers additional functionality on top of EFSMs to allow different types of data dependencies to be captured. We present examples where our methodology is used to model two different e-commerce applications.

Our modeling methodology exploits a modified version of the Session-Based Web Application Tester (`SWAT`) tool developed by Krishnamurthy *et al.* to achieve flexible control over workload characteristics [12]. Our modifications allow `SWAT` to accept as input the application model for a system under study. This allows SWAT to obtain a large set of valid sequences that it needs to construct workloads. `SWAT` uses a workload model which includes attributes such as those discussed earlier in this section that can influence the performance of session-based systems. The characterizations for these attributes can either be based on those observed in real systems or perturbations for the purpose of a sensitivity analysis. The application and workload models are used by `SWAT`'s trace generation algorithm to create a synthetic workload that has the correct inter-request and data dependencies and that has the specified characteristics. The chief advantage of `SWAT` is the fine control it offers over workload characteristics. For example, it permits the characterizations of one or more attributes to be changed at a time while keeping those of others unchanged so that a system's sensitivity to those characteristics alone can be established.

III. OVERVIEW OF MODEL-BASED APPROACH

Figure 1 provides an overview of our proposed approach. An application model that captures the inter-request dependencies and data dependencies for the system under study is constructed based on inputs provided by the tester. The modeling methodology and the inputs needed to create a model are described in Section IV. The *sequence generator* uses the model to produce a large trace containing valid sequences of request types. We define each valid sequence of request types as a *sessionlet*. Each sessionlet in this trace satisfies the inter-request dependencies for the system under study. The trace of sessionlets is input to `SWAT` along with the workload model and the application model.

The workload model exposed by `SWAT` depends on the workload generation mode employed [12]. For the sake of brevity only the *session mode* of workload generation is discussed. In the session mode, new sessions are generated according to a session inter-arrival time distribution. *Session inter-arrival time* is defined as the time between successive arrivals of sessions at the system under study. Each generated session behaves as a user by issuing a request, waiting for the complete response from the system, and then waiting for an inactive period, defined as the *think time*, before issuing the next request. The think times are chosen according to a think time distribution while the number of requests per session is governed by a *session length* distribution. Finally, `SWAT` includes as attributes workload mix and the distributions for the values of parameters used within requests. *Workload mix* is defined as the overall proportions of the different request types in the workload. The parameter value distributions can be used to control the locality properties of name-value pairs (e.g., control the relative popularities of products in a bookstore).

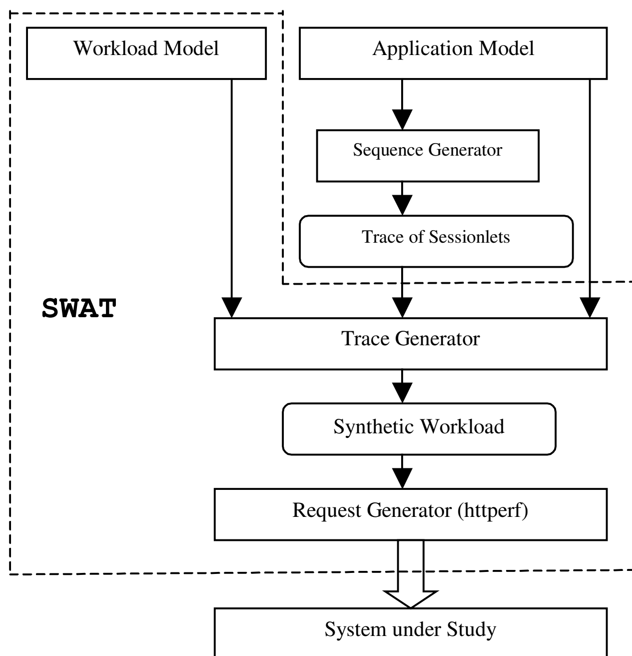

Figure 1. Model-based performance testing approach.

We note that the SWAT workload does not include requests for objects such as image and multimedia files that are embedded within the HTML responses of the system. This is because such content is often hosted on external "graphics" servers, or at content delivery networks. Krishnamurthy *et al.* provide a more detailed discussion on the rationale behind the workload model chosen for SWAT [12].

The objective of trace generation is to produce a trace of sessions that can be submitted to a system under study. Trace generation proceeds in two distinct steps. In the first step, the trace generator produces an intermediate trace of sessionlets. The intermediate trace is created by repeating selected sessionlets from the input trace of sessionlets. The trace generator determines the sessionlets selected and the number of times each sessionlet is repeated, i.e., the execution weights, so as to closely match the specified workload mix and session length distribution.

The next step in trace generation transforms the intermediate trace of sessionlets to a trace of sessions. This involves selecting name-value lists for request types to form URLs and inserting think times between successive URL requests in a session. The application model is consulted to handle data dependencies. The trace generator also ensures that the parameter value distributions specified in the workload model are achieved. With SWAT's trace generation approach, inter-request dependencies are satisfied since each session in the synthetic workload has the same sequence of request types as one of the sessionlets in the input trace.

The sessions produced by the trace generator and the specified session inter-arrival time distribution constitute the synthetic workload. A modified version of httperf, an open-source request generator, is used to submit the synthetic workload to the system under study. The modifications were required to support certain commonly occurring data dependencies (explained in Section IV) and to facilitate finer-grained reporting of performance metrics. We note that our approach is not limited to httperf. The trace generator can be easily modified to produce synthetic workloads in formats that conform to other request generators.

The trace generator and request generator shown in Figure 1 can be used to test different applications. The application model provides an indirection that allows these components to function independent of the system under study. The components can be used to study different systems by merely constructing different application models. As mentioned previously, this is in contrast to traditional trace generation methods that are typically system-specific. For a given application, characteristics of the synthetic workload can be varied by changing the characterizations for the attributes in the workload model.

IV. APPLICATION MODELING

As mentioned in Section II, the modeling methodology describes a Web application for the purpose of automating performance tests. The basic component of this methodology is an EFSM. Section IV.A describes an EFSM and introduces related terminology. We introduce additional modeling elements to address inter-request and data dependencies and to accommodate the trace generation process described in the previous section. Section IV.B presents examples to illustrate the modeling of inter-request dependencies. Modeling of data dependencies is discussed in Section IV.C.

A. Overview of Modeling Methodology

In this section we briefly describe an EFSM. For a more detailed discussion readers are referred to the survey paper by Lee and Yannakakis [14].

An EFSM is described as the following quintuple:

$$M = (I, O, S, \vec{x}, T)$$

I, O, S, $\vec{x}$, and T are finite sets of *input symbols*, *output symbols*, *states, variables*, and *transitions*, respectively. A transition t in the set T is defined by 6-tuple:

$$t = (s_t, q_t, a_t, o_t, \vec{P}_t, A_t)$$

where s_t, q_t, a_t, and o_t are the current state, next state, input, and output, respectively. $\vec{P}(\vec{x})$ is a predicate constructed from the current variable values, $A_t(\vec{x})$ defines an action on the variable values.

The operation of an EFSM can be described as follows. Let the machine's initial state be $s_{initial}$ where $s_{initial}$ belongs to S. Let the initial values of variables be given by $\vec{x}_{initial}$. Assume that the machine is currently at state s and that the current variable values are $\vec{x}$. On receiving an input a the machine makes a transition $t = (s, q, a, o, \vec{P}, A)$ if the predicate $\vec{P}(\vec{x})$ evaluates to true. If the predicate evaluates to true, then the machine produces the output o, the values of

the state variables are modified as per the function $A(\vec{x})$ and the machine moves to the state *q*.

For this work we use an EFSM as follows. We define as input the act of a user submitting a request to the system. Consequently, an input is associated with a request type and a name-value list. Predicates constructed from the state variables are used to capture inter-request and data dependencies. A transition from one state to another is allowed only if the predicate associated with that transition evaluates to true. A successful transition may result in modification of state variable values as per the transition's action function. The EFSM is *non-deterministic* since more than one transition can be followed from a given state. For example, in an e-commerce system users maybe able to both sign-in as well as view products from the homepage. Since our focus is on workload generation we ignore outputs in this work. However, they could be interpreted as the Web page resulting from the input.

To accommodate the approach described in Section III, we introduce several additional model elements to extend the basic EFSM functionalities. Firstly, to facilitate the generation of an input trace of sessionlets an EFSM always has a `Start` state and an `Exit` state. They model respectively, the starting and termination of interactions a user has with the Web-based application. Secondly, each transition has two distinct sets of predicates and actions. *Request dependency predicates* and *request dependency actions* are involved in enforcing correct inter-request dependencies. *Data dependency predicates* and *data dependency actions* are used to satisfy data dependencies. As described later, such a distinction is necessitated by the two step trace generation process described in Section III. Finally, data dependency actions may invoke `Select` functions. The *`Select`* function is used to choose a specific value for a given request parameter from among all the possible values for the parameter. The following sections provide a detailed description of these elements along with examples.

In general, an EFSM can support many different types of variables. However, for this work we use only the integer, string, Boolean, and array data types. These types of variables were found to be adequate for modeling the Web-based applications considered in Section IV.D. We believe other types of variables can be introduced in a straightforward manner to better support the needs of specific applications. Our approach also supports basic mathematical and logical operations within predicates and actions. In addition, it supports several functions that can be used to operate upon and manipulate variables. Such functions are provided to handle several common types of inter-request and data dependencies. A more detailed description of these functions is provided in Section IV.C.

B. Modeling Inter-Request Dependencies

As described in Section III, the sequence generator uses the application model to create a trace of input sessionlets. A sessionlet is generated as follows. The model is initialized by providing initial values to the state variables. The sequence generator causes a transition from the `Start` state by executing a randomly selected transition from among the set of allowed transitions from that state. Another transition is executed in a similar manner if the resulting state is not the `End` state. Sessionlet generation is complete if the `End` state is reached. The sequence generator outputs the sequence of inputs (i.e., request types) corresponding to the sequence of transitions executed. It then re-initializes the application model to generate more sessionlets. Valid sessionlets are produced as long as the application model enforces the correct inter-request dependencies.

We now present an e-commerce application example to illustrate modeling of inter-request dependencies. These examples also illustrate some of the limitations of the modeling approaches discussed in Section II. Figure 2 shows a simplified model for the application. In this application users execute the `Home` request type to request the homepage. The `Sign in` request type allows a user to login as a registered user. A user can view product information through the `Browse` request type. The `Add` and `Delete` request types allow a user to add and delete items from the shopping cart, respectively. The `Checkout` request type allows a user to initiate ordering of products in the shopping cart. A user submits the `Purchase` request type to provide payment details for finalizing the order.

Two request dependency state variables are used to enforce inter-request dependencies. The `items_in_cart` is an integer variable that indicates the number of items in the shopping cart. The `signed_on` Boolean variable states whether a user has signed on or not. The initial values of the `items_in_cart` and `signed_on` variables are 0 and FALSE, respectively. The values of these variables are changed by actions associated with several transitions. For example, from Figure 2, submitting the `Sign in` request type (transitions S_1 to S_7 and S_5 to S_7) changes the value of `signed_on` to TRUE. Similarly an `Add` request type (transition S_2 to S_3) increments `items_in_cart` variable by 1 while a `Delete` request type (transitions S_3 to S_4 and S_4 to S_4) decrements the variable by 1.

From Figure 2, certain transitions depend only on the current state of the EFSM. These first-order transitions are not associated with any predicates. For example, a user can submit a `Browse` request type after submitting a `Home` request type as indicated by the transition from S_1 to S_2. Similarly, a user can browse another product after browsing a particular product as indicated by the transition from S_2 to S_2.

Our application model also allows higher-order dependencies between request types to be captured. For example, consider the transition from S_4 to S_5 in Figure 2. In this transition, the user submits a checkout request after deleting an item from the shopping cart. This transition is allowed only when the previous sequences of requests have resulted in at least one item in the shopping cart. This dependency is enforced by the predicate associated with the transition which checks whether the `items_in_cart` variable is greater than 0. Consequently, the sequence `[Home, Browse, Add, Browse, Add, Delete, Checkout]` is allowed while the sequence `[Home, Browse, Add, Delete, Checkout]` is not.

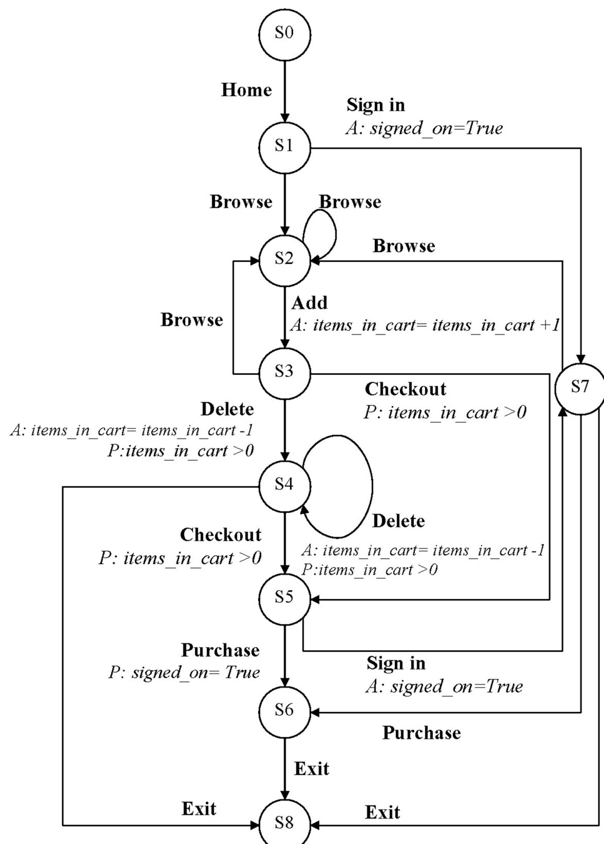

Figure 2. Modeling inter-request dependencies in an e-commerce system.

The EFSM can model different ways in which a user can complete a given task. Such a scenario is very common in Web-based applications. In the example considered a user can either sign-in just immediately before purchasing (transition S_5 to S_7 in Figure 2) or sign-in immediately after visiting the homepage (transition S_1 to S_7 in Figure 2). As a result, the sequences [Home, Browse, Add, Checkout, Sign in, Purchase] and [Home, Sign in, Browse, Add, Checkout, Purchase] represent two possible ways for a user to purchase an item. The EFSM handles the different scenarios through the predicate associated with the transition from S_5 to S_7 in Figure 2. The predicate uses the sign_in variable to determine whether or not a user has to sign-in before purchasing an item.

The simple example presented illustrates some advantages of the proposed approach when compared to the existing modeling approaches discussed in Section II. As mentioned previously, in our approach request types constitute the input. In contrast, with a CBMG-like approach request types constitute the states of the FSM. Furthermore, as described previously, due to the implicit first-order assumption the next state (request type) to be submitted in a session can be determined from the current state (request type) alone. However, the example presented shows that whether or not a certain request type can follow another request type can depend on certain complex preconditions being met or not. The approaches discussed in Section II are not expressive enough to capture such dependencies. For example, it is not possible to capture the conditional dependency between Delete and Checkout or Checkout and Purchase using the existing approaches. As a result, for systems characterized by complex inter-request dependencies only a limited number of unique sessionlets (i.e., those with only first-order transitions between request types) can be obtained from such models. Synthetic workloads constructed from such a limited number of sessionlets are not likely to be representative. Furthermore, our previous experience [13] suggests that having a limited number of sessionlets impacts the flexibility of the performance testing process. Specifically, with a limited number of sessionlets the trace generator does not have enough freedom to realize arbitrarily desired mixes and session length distributions.

We note that an FSM whose inputs represent request types could also be used instead of an EFSM. However, since a FSM does not support state variables, predicates, and actions the FSM equivalent of an EFSM typically has a larger number of states [14]. In particular, state explosion can occur when an FSM is used for complex systems characterized by a large number of state variables and large numbers of possible values for state variables. Consequently, an EFSM-based approach can model Web-based applications in a more succinct manner.

C. Modeling Data Dependencies

As discussed in Section III, the sessionlets generated with the help of the application model are used by the trace generator to create an intermediate trace of sessionlets exhibiting the desired workload mix and session length distribution. The intermediate trace has to be converted to a trace of sessions that can be submitted to the system under study. This is achieved by appending name-value lists for the request types in the intermediate trace.

We now describe how the application model is used to capture data dependencies. As mentioned in Section III, data dependencies govern the generation of parameter values. The set of state variables used by the model includes the parameters for all request types supported by the application. Each parameter is denoted using the notation "Request Type.Parameter Name". For example, Add.Item_ID refers to the Item_ID parameter of the Add request type. Depending on how their values are chosen, the modeling methodology classifies parameters into several categories.

Tester-specifiable parameters are provided by the tester as inputs to the model. Considering an e-commerce system example, user names and passwords as well as product identifiers belong to this category. Tester-specifiable parameters are further subdivided into independent parameters and inter-dependent parameters. The value chosen for an *independent parameter* does not have any dependency with the value chosen for another parameter. In contrast, the value chosen for an *inter-dependent* parameter is controlled by the value chosen for another independent parameter. For example, in an e-commerce system the password selected for a sign-in request type will depend on the user name chosen. Currently, the model only allows such one-to-one dependencies. We note that, where appropriate, the choice of values for tester-specifiable variables can be controlled through the parameter value

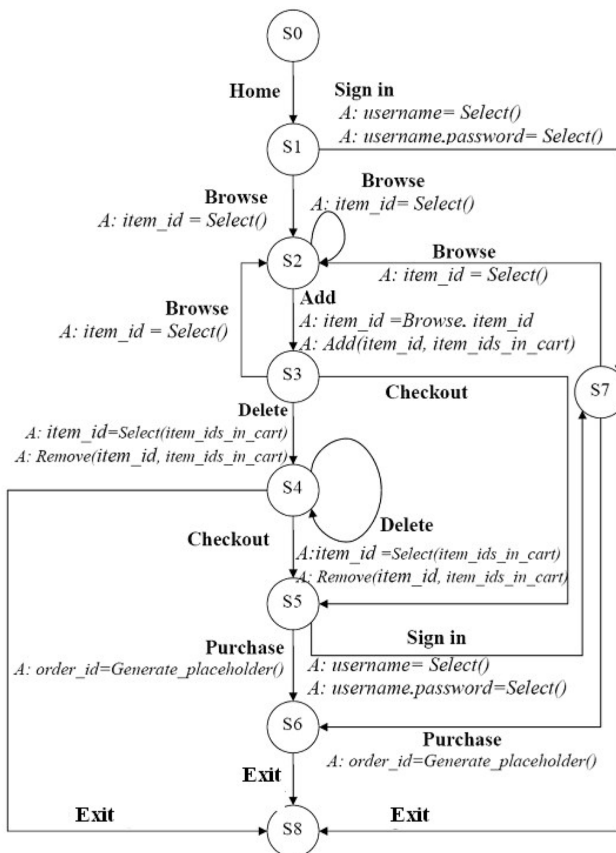

Figure 3. Data dependencies in an e-commerce system.

distributions specified to the workload model.

The values for *session-dependent* parameters depend on the sequence of URLs submitted in a session and hence cannot be specified explicitly by the tester. Considering an e-commerce system example again, the id of a product that needs to be deleted will depend on the products present in the shopping cart. These parameters are further classified into dynamically generated and non-dynamically generated parameters. The values of *dynamically generated parameters* are known only during request generation and hence cannot be resolved during trace generation. For example, the unique order identifier that is passed along with a purchase request type is typically assigned dynamically by the system only when the session is in progress. For such parameters, the trace generator uses placeholder values instead of the actual values. These placeholder values instruct the request generator that the actual values have to be obtained by parsing the responses of the Web pages returned by the system under study when the session is in progress. In contrast, values for *non-dynamically generated parameters* can be resolved during trace generation.

To generate name-value lists for a sessionlet in the intermediate trace, the sequence of states corresponding to the sessionlet are identified in the application model. As mentioned in Section IV.A, the state transitions have associated with them data dependency predicates and actions. The actions are used to choose parameter values for request types associated with the transitions. The actions use the `Select` function to choose parameter values. As described shortly, the behaviour of the `Select` function depends on the type of parameter being handled and the input arguments passed to the function. In addition to the `Select` functions, functions are also provided to manipulate array variables and to handle dynamically generated parameter values.

We present the following example to explain the process of handling data dependencies. Consider the state sequence [`Start`, S_1, S_7, S_2, S_3, S_2, S_3, S_4, S_5, S_6, `Exit`] generated from the EFSM shown in Figure 2. This sequence corresponds to the sessionlet [`Home, Sign in, Browse, Add, Browse, Add, Delete, Checkout, Purchase`]. Assume that `Sign in` takes two parameters `username` and `password`. `Browse`, `Add`, and `Delete` accept a parameter called `item_id` denoting the item to be browsed, added to the shopping cart, and deleted from shopping cart, respectively. `Purchase` requires an `order_id` parameter whose value is dynamically assigned by the system. A state variable called `item_ids_in_cart` maintains the `item_id` values of the products in the shopping cart. We now discuss cases involving the generation of values for tester-specifiable and session-dependent parameters. Figure 3 shows the model for handling data dependencies in the example e-commerce system. In Figure 3 the request type name is omitted when referring to a parameter for the sake of clarity.

Tester-specifiable parameters – Consider the action associated with the transition from S_1 to S_7 in Figure 3. The `Select` function first generates a value for the tester specifiable, independent parameter `Sign In.username`. The `Sign In.password` parameter is specified to be dependent on `Sign In.username`. Consequently, the second call to `Select` generates a value for password depending on the username selected in the first call. When many different values are possible for a tester-specifiable parameter, a value is either chosen randomly from among the possible values or as per a parameter value distribution, if such a distribution is specified in the workload model.

Session-dependent parameters – Consider the action associated with the transition from S_2 to S_3 in Figure 3. For this example, the `Add.item_ID` is session-dependent. As shown in Figure 3, the value of this parameter is the same as the value of the `Browse.item_ID` parameter chosen previously in the session. For certain session-dependent parameters there may be a choice between many possible values. In such cases, the `Select` function is used to choose a value from among the possible values. This is illustrated in the action associated with the transition from S_3 to S_4. The `Select` function takes as argument the `item_ids_in_cart` list variable. This variable is updated whenever an `Add` transition occurs (e.g., transition from S_2 to S_3 in Figure 3) and contains the ids of items added to the shopping cart. The function randomly selects a value from this list and assigns it to the `Delete.item_ID` parameter. The action also invokes the `remove` method on `item_ids_in_cart` to delete the item id from the list.

As mentioned previously, placeholder values are used for dynamically generated parameter values. Consider the action associated with the transition from S_5 to S_6. The call to the `Generate_placeholder` function inserts a placeholder value for the dynamically generated

`Purchase.order_ID` parameter. We note that the format of the placeholder value may differ for different request generators.

V. IMPLEMENTATION

We have developed a toolset to support the approach described in this paper. Testers use a text file to specify inputs related to the application and workload models. Details of the specification language have been omitted due to space constraints. A model verifier has been developed to check for consistency in the application model. For example, the program flags an error when no values are provided for a variable declared as tester-specifiable variable. The model verifier also checks for consistency between the application model and workload model. For example, an error is reported when a request type specified in the workload model does not appear in the application model and vice-versa. If the model consistency check is successful, the model verifier generates the application and workload models in a less verbose format suitable for the sequence and trace generator tools shown in Figure 1. A sequence generator that takes as input the less verbose application model to produce a trace of sessionlets has also been developed. Finally, we have also modified SWAT to accommodate the proposed methodology.

VI. MODELING CASE STUDY

We used the proposed modeling approach to develop models for two open-source session-based systems namely, the Java Pet Store [9] and the Rice University Bidding System (RUBiS) [20]. The main objective of this exercise was to verify whether our approach was robust enough to model different types of Web applications.

The Java Pet Store is essentially an e-commerce application that can be used to sell pets online. The application is built using the J2EE middleware. The model developed for this application is conceptually similar to the model presented for the e-commerce example in Sections IV.C and IV.D. The reader is referred to our technical report [21] for a detailed description. Section VII describes a study where this model was used to generate synthetic workloads for a Pet Store installation.

The RUBiS application has been modeled after eBay, the popular Internet auction site. Several implementations of RUBiS based on different middleware technologies such as Java Servlets and Enterprise Java Beans have been developed [Ref]. Many studies have used RUBiS to study performance issues in Web applications. Stewart *et al.* employed RUBiS to investigate the impact of non-stationarity of workload mix and its implications to prediction of server performance [22]. Similarly Parekh *et al.* tested their bottleneck detection approach on a RUBiS testbed [18].

The modeling methodology was found to be expressive enough to capture the dependencies of RUBiS. The model developed for RUBiS differed from the Pet Store model in many aspects. Specifically, more session-dependent parameters had to be used while handling data dependencies. For example, the `item id` of a product in RUBiS had to be treated as a dynamically generated, session-dependent parameter. The reason for this choice is because an item id is dynamically generated by the system when a seller initiates a new auction. Consequently an item's id has to be obtained during request generation by parsing the Web page response returned by the system. This is in contrast to the Pet Store system where item ids are known a priori and hence can be modeled as tester-specifiable parameters. A more detailed description of the RUBiS model can be found in our technical report [21].

The application models along with our toolset can be used to support more flexible performance tests for Web applications. For example, performance studies involving RUBiS typically make use of a workload generator that is bundled with the application. However, this workload generator does not allow a tester to specify arbitrarily desired characterizations for workload attributes. For example, a tester cannot vary the workload mix and session length distribution independently of one another [12]. Since both these attributes can impact performance, this limitation makes it difficult to isolate their individual impacts. The workload generators bundled with other benchmarks such as TPC-App also suffer from the same limitation. Consequently, these tools are not well suited for studies that require fine-grained, flexible control over workload characteristics. In section VII we present a sensitivity analysis case study to demonstrate how our toolset could be used for such studies.

VII. PERFORMANCE EVALUATION CASE STUDY

In this section we demonstrate the utility of the toolset by using it to test the performance of the Java Pet Store application. To the best of our knowledge currently there is no workload generator available for this application. Specifically, two sets of experiments are conducted. We note that we were unable to conduct a direct comparison of our approach with the CBMG-like approaches discussed in Section II. As discussed in Section IV, those approaches were not able to capture the types of dependencies present in the Pet Store application.

The first set provides results that establish the importance of preserving inter-request dependencies for the Pet Store application thereby motivating the need for our modeling approach. The second set demonstrates the flexibility of our toolset. It shows how our toolset can be exploited to automatically generate controlled synthetic workloads that can be used to study the sensitivity of system performance to various workload attributes.

This section is organized as follows. Section VII.A describes the experiment setup. Section VII.B studies the impact of inter-request dependencies on application performance. Sections VII.C and VII.D present experiments where synthetic workloads are used to explore the impact of workload mix and think time on application performance. Section VII.E discusses how the results of this case study vindicate the need for our toolset.

A. Experiment Setup

The hardware setup for the experiments was as follows.

Table I: Factors and levels used in the sensitivity analysis experiments

Factor	Levels
Session inter arrival time	Exponentially distributed. Different mean session inter-arrival times to generate HI-LOAD and LO-LOAD
Workload mix	Mix1 and Mix2
Tester specifiable parameters	Fixed probability destiny functions for parameter values
Think time	Exponentially distributed. Two different mean think time (Z) values (Z = 3 sec and Z = 45 sec)
Session length	Exponentially distributed. Mean session length = 10

The Pet Store application was installed on a Pentium III, 1 GHZ (dual), 1 GB RAM machine running under the Windows XP operating system. This node is referred to as the *server* node. httperf was executing on a Pentium III, 1 GHZ, 512 MB RAM machine running under the Linux operating system. This node is referred to as the *client* node. The server node and the client node were connected by Fast Ethernet switch. This switch provides dedicated 100 mbps full duplex bandwidth between the client node and the server node. The switch prevented the network from becoming the bottleneck during the tests. As a result, response time measured by httperf is a good indicator of server response time. For this study we define *response time* as the time between sending the first byte of a HTTP request and receiving the first byte of the corresponding HTTP response.

Performance metrics for the server node such as CPU and disk utilization were recorded by using the windows perfmon tool. We set the sampling interval for perfmon to be 2 minutes. In addition to the server node metrics, detailed traces were collected from httperf to characterize response times for individual requests submitted to the Pet Store. The primary performance metric used for this study is the 95th percentile of response times (R_{95}). We chose this metric since it is used frequently in service level assessment exercises. In addition, the mean response time (R_{mean}) is also reported for all experiments.

For all experiments in this study, the CPUs of the server node were found to be the bottleneck. On the other hand, very little disk activity was observed at the server node. Similarly, the server node did not display any memory bottlenecks or virtual memory activity. The network was also found to be very lightly utilized. The worst case peak network throughput was several orders of magnitude lower than the 100 mbps capacity of the fast Ethernet switch.

As mentioned previously, an application model was developed for the Java Pet Store (version 1.4). The application model developed was used by the sequence generator to create an input trace of 20,000 sessionlets for the Pet Store. SWAT used these input sessionlets to generate controlled synthetic workloads. The synthetic workloads were constructed in a manner that allowed us to evaluate the sensitivity of system performance to two workload attributes namely, workload mix and mean think time. Table I shows the experimental design followed for the sensitivity analysis.

From Table I, the exponential distribution is used for generating session inter-arrival times. This assumption is consistent with previous studies which have shown that arrivals of sessions at a Web server are uncorrelated with each other [19]. The value of the mean session inter-arrival time was manipulated to achieve two different mean request rates at the server namely, HI-LOAD and LO-LOAD. As the name implies, these mean request rates were chosen to place different loading levels on the server. With these mean request rates, the worst case mean server CPU utilization achieved over an experiment duration was found to be approximately 60%. Studies have shown that the mean utilizations of real servers over a timescale of a few hours rarely exceed this value [2]. Consequently, we did not conduct experiments that result in higher mean utilizations.

An exponential distribution with a mean of 10 requests per session was chosen as the session length distribution. This choice matches the behaviour observed by Arlitt *et al.* at a large Web-based shopping system [3]. As shown in Table I, fixed probability density functions (PDFs) were specified for tester specifiable variables in the Pet Store such as product and category ids. The same set of PDFs was used for all experiments. The PDFs were selected to cause a small set of products and categories in the system to be more popular than products and categories outside this set. Such a choice was motivated by similar behaviour observed at real Web applications [3] [15].

As shown in Table I, the exponential distribution was used for generating think time values. This distribution is widely used by practitioners as well as in popular benchmarks for Web applications such as TPC-W [23] and RUBiS [20]. Two different mean think time (Z) values were explored in the experiments. The lower Z value of 3.0 seconds closely mirrors the setting used by benchmarks such as TPC-W and RuBiS. Anecdotal evidence suggests that many practitioners use these benchmarks as guidelines for their own performance testing exercises. As a result, in practice performance tests very often employ such a low value of Z. The higher Z value of 45.0 seconds is consistent with observations recorded at a real Web-based application [1]. We used this value in some of our experiments to study the impact of using more realistic think times.

To study the impact of workload mix, workloads with two different mixes were created using SWAT. As shown in Table II, Mix2 does not contain any Checkout or Purchase request types. In contrast, Checkout and Purchase constitute about 4.75% of the requests in Mix1.

Table II: Request type distribution in Mix1 and Mix2

	Mix2	Mix1
Request Type	**Percentage of Request Types**	**Percentage of Request Types**
Shopping Cart	16.37	16.37
Browse	**46.30**	**41.46**
Sign out	0.20	0.30
Sign in	6.20	6.54
Add	5.96	5.95
Delete	0.61	0.59
Checkout	**0.00**	**3.57**
Purchase	**0.00**	**1.19**
Home	14.78	14.53
View item	9.53	9.45

Furthermore, Mix2 offsets the decrease in Checkout and Purchase request types through a corresponding increase in the number of Browse request types. Specifically, Mix2 contains about 4.80% more Browse request types than Mix1. The percentages of other request types are almost the same in both mixes. Mix2 is used to emulate a workload that has only "window" shoppers while Mix1 represents a workload with shoppers who buy occasionally.

As mentioned previously, the 20,000 sessionlets generated from the Pet Store application model were used to construct the synthetic workloads for this study. Each experiment evaluated the impact of a particular workload on the system. Each *workload* corresponds to a specific factor-level combination of Table I. Each experiment performed consisted of three different *runs*. SWAT was used to create three different statistically identical trace files for these three different runs. This was done by using different seeds for the random generators used by SWAT. We used the response time numbers from the three runs in an experiment to compute 95% confidence intervals for R_{mean} and R_{95}. Each run took around 3-4 hours to finish. httperf generated a log file for approximately 100,000 replies sent by server to the submitted requests. The server node was rebooted and the Pet Store application database was reinitialized between successive runs to ensure that the system was in the same initial state. For each experiment, chi-square tests [8] were conducted to ensure that the workload generated in an experiment conformed to the workload model specified to the toolset.

B. Impact of Inter-request Dependencies

Simple experiments were conducted on the Pet Store system to study the effect of inter-request dependencies. We selected two sessions from a trace of sessions created using our approach. The first session, named Browse_Session, contained requests to browse items in the pet store. It did not contain any purchase related request types such as Sign in, Shopping cart, Add, Delete, Checkout, and Purchase. The other session, named Purchase_Session, predominantly consisted of purchase related requests. For the Pet Store application, the browse related requests do not have any dependency with other requests and can occur at any point in a session. However, purchase related requests have more complex inter-request dependencies similar to those modeled in Figure 2. We also created scrambled versions of the two sessions called Browse-Scrambled_Session and Purchase-Scrambled_Session. The scrambled versions have requests occurring in a random order and hence ignore dependencies.

Since we were interested in characterizing the aggregate resource demand placed by a session on the system's resources, the number of concurrent sessions accessing the Pet Store was set to be 1. httperf was used to submit a session and measure the mean response time for requests in the session. Since there was no contention among sessions for system resources and since the network was lightly loaded, the mean response time reflects the end-to-end resource demands placed by the session across all resources in the Pet Store system. Multiple runs were carried out for each experiment to achieve statistical confidence in the results.

Table III shows the results from our experiments. There is no significant difference between the R_{mean} values for the Browse_Session and Browse-Scrambled_Session sessions. This is not surprising given that browse related requests can occur in any order within a session. However, the R_{mean} for the Purchase_Session session is almost 1.73 times that of the Purchase-Scrambled_Session session. The R_{95} values for these sessions were all statistically different since they had non-overlapping 95% confidence intervals. The Purchase-Scrambled_Session workload places less stress on the system since certain requests (e.g., Sign in, Shopping cart, Add, Delete, Checkout, Purchase) impose less demand on system resources if they occur at an incorrect point in a session. This result reinforces the importance of preserving correct inter-request dependencies in synthetic workloads for the system under study. When significant inter-request dependencies are present, ignoring them can yield incorrect performance estimates and can therefore cause incorrect conclusions to be drawn from performance tests. As a result, we believe our approach adds value to the performance testing process by providing a mechanism to capture and preserve complex dependencies in session-based systems.

Table III: Effect of inter-request dependencies

Workload	R_{mean} (ms)
Browse_Session	97.6
Browse-Scrambled_Session	101.4
Purchase_Session	390.6
Purchase-Scrambled_Session	226.3

Table IV: Effect of Workload Mix

Load (Requests/sec)	Mix1-Workload		Mix2-Workload	
	U_{CPU} (%)	R_{95} (ms)	U_{CPU} (%)	R_{95} (ms)
13.2 (HI-LOAD)	59.2	199.3	49.8	161.5
9.6 (LO-LOAD)	41.2	128.0	36.2	119.1

C. Impact of Workload Mix

To study the impact of workload mix, we created two different workloads namely, `Mix1-Workload` and `Mix2-Workload`. Both workloads are similar in all respects except their workload mixes. `Mix1-Workload` and `Mix2-Workload` display the `Mix1` and `Mix2` workload mixes of Table II, respectively. A mean think time of 3.0 seconds is used for these experiments. Experiments were performed for both the `HI-LOAD` and `LO-LOAD` cases of Table I. We note that ability to synthesize workloads that differ only with respect to their workload mixes is a unique aspect of our toolset. Other trace generation toolsets that we are aware of do not support the ability to control the workload mix independently of the session length distribution [13].

Table IV shows the results of this set of experiments. As the load in requests per second increases, the average utilization of the Pet Store system's processors over the experiment duration (U_{CPU}) increases for both workloads. Furthermore, R_{95} increases for increasing loads for both workloads. For a given load, the `Mix1-Workload` places more stress on the system than the `Mix2-Workload.` For the LO-LOAD case, U_{CPU} for the `Mix1-Workload` is approximately 14% higher that of the `Mix2-Workload`. Similarly, for this case the R_{95} value for `Mix1-Workload` is about 7.5% higher than that of the `Mix2-Workload`. The impact of workload mix becomes more significant at the higher load. For the `HI-LOAD` case the R_{95} value for `Mix1-Workload` is approximately 23% higher than that of the `Mix2-Workload`. We note that the differences observed in the U_{CPU} and R_{95} values are statistically significant since the values compared have non-overlapping 95% confidence intervals.

The main reason for the poorer performance with the `Mix1-Workload` is the presence of the resource intensive `Purchase` request type in the workload. Figure 4 provides the mean response time breakdown for the different request types in the workloads for the `HI-LOAD` case. From the figure, the mean response time of the `Purchase` request type is about 2.5 times that of the next most resource intensive request type (`Sign in`). Since `Mix2-Workload` has no `Purchase` request types, for a given load this workload imposes lesser stress on the system causing lower U_{CPU} and R_{95} values. The reason for the resource intensiveness of the `Purchase` request type is likely due to the fact that it involves updating the database with information such as the particulars of the order being placed. Interestingly a mere 1.19% increase in `Purchase` requests (Table II) resulted in a 23% increase in R_{95} for the `HI-LOAD` case (Table IV). This shows that system performance can be very sensitive to the workload mix.

D. Impact of Think Time

Two different workloads were created to investigate the impact of mean think time. The `HI-THINK` workload had a Z value of 45.0 seconds while the `LO-THINK` workload had a Z value of 3.0 seconds. The workloads were similar with respect to all other attributes. The experiments employed the `Mix1` workload mix and were conducted for the `LO-LOAD` loading level. The distributions for session length and the tester specifiable variables were as per Table I. Table V shows the results from this set of experiments.

From Table V, even though the mean request rate, workload mix, session length distribution, and distributions of tester specified variables are the same for both workloads, the `HI-THINK` workload places more stress on the system than the `LO-THINK` workload. The R_{95} with the `HI-THINK` workload is approximately 16% higher than that of the `LO-THINK` workload. Similarly, the U_{CPU} with the `HI-THINK` workload is about 6.5% higher than that with the `LO-THINK` workload. Since with both workloads the server encountered the same mean request rate, this implies that the per request service demand incurred at the server CPU was more for the `HI-THINK` workload. This increased demand is not due to changes in the workload mix since the same mix was used to construct both the workloads.

To understand this result, we consider the session durations with both workloads. The `HI-THINK` workload increases session durations. As a result, the server would encounter more number of concurrent sessions with the `HI-THINK` workload. Figure 5 plots the cumulative distribution function (CDF) of the number of concurrent sessions at the server for both workloads. From the figure, the median number of concurrent sessions for the `LO-THINK` workload is approximately 20 while it is approximately 475 for the `HI-THINK` workload. The mean numbers of concurrent sessions for the `HI-THINK` and `LO-THINK` workloads were found to be approximately 371 and 25, respectively. In other words, even though the mean request rates are the same with both workloads there is increased session concurrency with the `HI-THINK` workload. We believe that this can adversely impact performance in two different ways. Firstly, overheads (e.g., context switching) are introduced because more sessions have to be handled concurrently. This perhaps explains the higher per-request CPU demands observed with the `HI-THINK` workload. Secondly more number of concurrent sessions can also cause requests to arrive in a bursty manner. This can in turn cause more contention among requests for system resources leading to significant queuing delays.

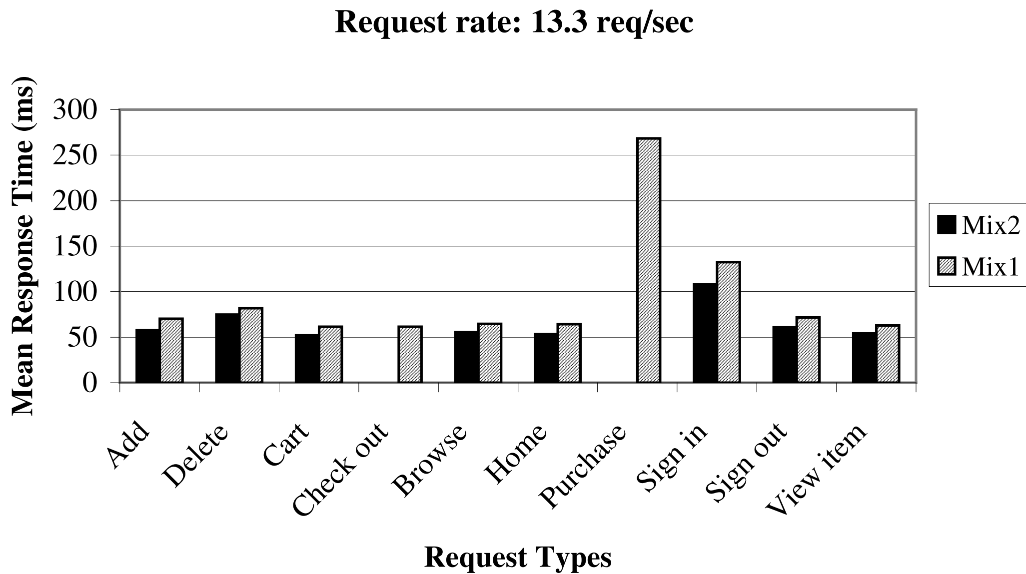

Figure 4. Mean response times of request types (HI-LOAD)

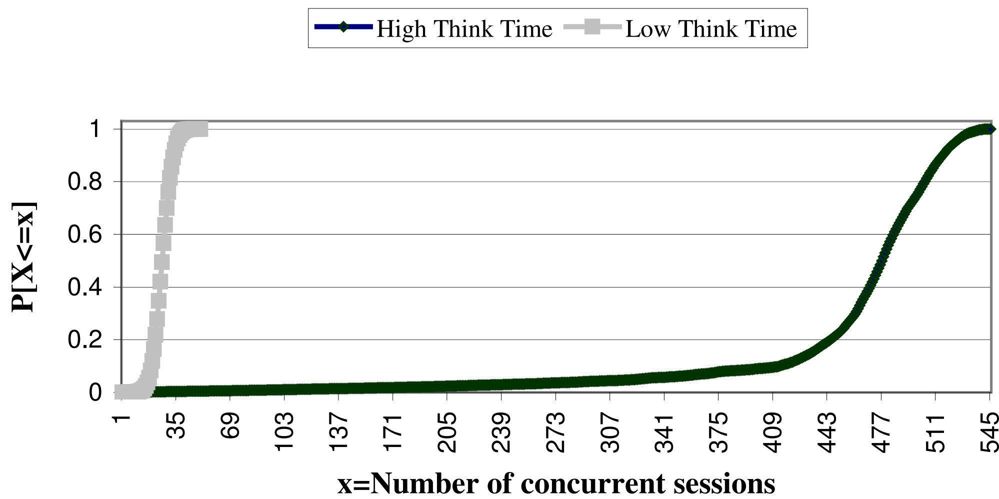

Figure 5. CDF of number of concurrent sessions

Table V: Effect of mean think time

Load (Requests/sec)	HI-THINK		LO-THINK	
	U_{CPU} %	R_{95} (ms)	U_{CPU} %	R_{95} (ms)
9.6 (LO-LOAD)	43.9	148.8	41.2	128.0

E. Discussion

The results presented in Sections VII.C and VII.D show that system performance can be very sensitive to attributes such as workload mix and mean think time. Specifically, a relatively modest increase in the number of resource intensive request types was found to cause significant degradation to the system's responsiveness. Furthermore, even under moderate loads a workload with longer think times imposed more demands on the processors and caused longer response times than a workload with short think times. Though memory bottleneck was not an issue in this case study, longer think times can cause such a bottleneck since a system has to support a large number of concurrent sessions. Apart from the attributes studied, the other attributes included in our workload model can also significantly influence system performance. For example, our previous study has shown that system performance can be very sensitive to the session length distribution [11].

These results reinforce the importance of careful selection of workload attribute characterizations for performance tests. For example, results of Section VII.D suggest that standard benchmark workloads that use short think times are likely to yield more optimistic estimates of system performance than a more realistic workload that uses longer think times. To obtain a more complete understanding of how workload characteristics impact performance, a performance testing toolset needs to be flexible enough to support sensitivity analysis studies such as that presented in this paper. As mentioned previously, existing performance testing methodologies do not permit fine-grained control over workload characteristics (e.g., perturbing mix while keeping other characteristics unchanged) for supporting such studies.

The toolset described in this work supports such flexibility by enabling the automated construction of controlled synthetic workloads. Specifically, the application model allows generation of a large trace of sessionlets that preserve the correct inter-request dependencies for a system under study. This trace of sessionlets permits automated construction of synthetic workloads with specified characteristics. Furthermore, the application model also allows correct data dependencies to be enforced in an automated manner. The ability to automate the construction of synthetic workloads and the ability to achieve arbitrarily desired workload characteristics can greatly improve the effectiveness of the performance testing process.

VIII. CONCLUSIONS AND FUTURE WORK

This paper developed a model-based toolset for testing the performance of Web applications. The toolset uses a formal model to capture an application's inter-request and data dependencies. The model can be used to obtain several sequences of requests representing how users typically interact with the Web-based application. The sequences can in turn be used to construct synthetic workloads with specified characteristics.

The proposed approach offers several advantages over a traditional performance testing approach. The application model provides an indirection which allows a common set of workload generation tools to be used for testing different applications. As an example, we found the modeling methodology underlying the toolset to be expressive enough to describe two different Web applications. Due to this model-based approach, less effort is needed for developing and maintaining the workload generation tools and more effort can be dedicated towards the performance testing process. Furthermore, the approach can improve the effectiveness of the performance testing process since it enables automated and flexible control over the characteristics of synthetic workloads. To the best of our knowledge, we are not aware of other model-based tools that provide support for the types of complex dependencies in Web applications discussed in this paper.

Future work will focus on reducing the effort needed to develop an application model. Specifically, we intend to explore ways in which application documentation (e.g., UML object diagrams, message sequence charts) can be exploited to generate application models. Our future work will also further refine the manner in which dynamically generated parameter values are handled. Currently in our toolset whenever the request generator (`httperf`) encounters a placeholder tag for a dynamically generated parameter it parses the HTTP response of the last submitted request in the session to obtain that parameter's value. This approach was found to be sufficient for the Pet Store application. In future, more complex operations will be supported by encoding a placeholder tag with a condition based on values obtained from the parsed response and an action that generates a value for the dynamic parameter based on the condition.

We intend to model other types of Web applications such as online banking and trading applications. Modeling of applications in enterprise environments such as SAP will also be explored. Insights gained from these studies would be used to enhance the modeling approach. We are currently developing an open-source implementation of our toolset. We also plan to make available models for popular Web applications such as the Java Pet Store, TPC-W, TPC-App, and RUBiS.

REFERENCES

[1] Andrews, A., Offutt, J., Alexander, R, 2005. Testing Web-Applications by Modeling with FSMs, Software and Systems Modeling 4(3), 326-345.

[2] Andrzejak, A., Arlitt, A., Rolia, J., 2002. Bounding the Resource Savings of Utility Computing Models, Hewlett-Packard Labs Technical Report (HPL-2002-339).

[3] Arlitt, M., Krishnamurthy, D., Rolia, J., 2001. Characterizing the Scalability of a Large Web-based Shopping System, ACM Transactions on Internet Technology 1(1), 44-69.

[4] Banga, G., Druschel, P., 1999. Measuring the Capacity of a Web Server under Realistic Loads, World Wide Web 2(1-2), 69-83.

[5] Barford, P., Crovella, M., 1998. Generating Representative Web Workloads for Network and Server Performance Evaluation, International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS), 151-160.

[6] Crovella, M., Bestavros, A., 1997. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes, IEEE/ACM Transactions on Networking 5(6), 835-846.

[7] Ferrari, D., 1984. On the Foundation of Artificial Workload Design, International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS), pp. 8-14.

[8] Jain, R., 1991. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley and Sons, New York.

[9] Java Pet Store, 2007. http://java.sun.com/developer/releases/petstore/

[10] Kant, K., Venkatachalam, M., 2002. Modeling Traffic Non-stationarity in E-commerce Servers, International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002), 949-956.

[11] Kelly, T, 2005. Detecting Performance Anomalies in Global Applications, Second Workshop on Real, Large Distributed Systems (WORLDS 2005).

[12] Krishnamurthy, D., Rolia, J., Majumdar, S., 2006. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems, IEEE Transactions on Software Engineering 32(11), 868-882.

[13] Krishnamurthy, D., 2004. Synthetic Workload Generation for Stress Testing Session-Based Systems, PhD. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada.

[14] Lee, D., Yannakakis, M., 1996. Principles and Methods of Testing Finite State Machines - A Survey, IEEE 84(8), 1090-1123.

[15] Menasce, D., Almeida, V., Reidi, R., Pelegrinelli. F., Fonesca, R., Meira Jr., W., 2000. In Search of Invariants in E-Business Workloads, ACM International Conference on Electronic Commerce, 56-65.

[16] Menasce, D., Almeida, V., Fonesca, R., Mendes, M., 1999. A Methodology for Workload Characterization of E-Commerce Sites, ACM Conference on Electronic Commerce, 119-128.

[17] Mosberger, D., Jin, T, 1998. httperf – A Tool for Measuring Web Server Performance, ACM SIGMETRICS Performance Evaluation Review 26(3), 31-37.

[18] Parekh, J., Jung, G., Swint, G., Pu, C., Sahai, A., 2006. Issues in Bottleneck Detection in Multi-Tier Enterprise Applications, International Workshop on Quality of Service, 302-303.

[19] Paxon, V., Floyd, S., 1995. Wide Area Traffic: The Failure of Poisson Modeling, IEEE/ACM Transactions on Networking 3(3), 226-244.

[20] RUBiS – Rice University Bidding System, 2007. http://rubis.objectweb.org/

[21] Shams, A., Krishnamurthy, D., Far, B., 2007. A Model-Based Approach for Testing the Performance of Web Applications, Technical Report, Software Engineering Research Group, University of Calgary, Calgary (SERG-2007-10).

[22] Stewart, C., Kelly, T., Zhang, A., 2007. Exploiting Non-stationarity for Performance Prediction, European Conference on Computer Systems (EuroSys 2007).

[23] TPC-W – Transactional Web E-Commerce Benchmark, 2007. http://www.tpc.org/tpc

[24] TPC-App – Application Server and Web Services Benchmark, 2007, http://www.tcp.org/tpc_app/

[25] Vallamsetty, U., Kant, K., Mohapatra, P., 2003. Characterization of E-Commerce Traffic, Electronic Commerce Research Journal 3(1-2)