Synthesizing Client Load Models for Performance Engineering via Web Crawling

Yuhong Cai
Dept of Computer Science
University of Auckland
Private Bag 92019, Auckland 1142
New Zealand
+64 9 3737599
ycai003@ cs.auckland.ac.nz

John Grundy
Depts of Computer Science and
Electrical and Computer Engineering
University of Auckland 1142
Private Bag 92019, Auckland
New Zealand +64 9 3737599
john-g@cs.auckland.ac.nz

John Hosking
Dept of Computer Science
University of Auckland
Private Bag 92019, Auckland 1142
New Zealand
+64 9 3737599
john@cs.auckland.ac.nz

ABSTRACT
Accurate web application performance testing relies on the use of loading tests based on a realistic client behaviour load model. Unfortunately developing such load models and associated test plans and scripts is tedious and error-prone with most existing web performance testing tools providing limited client load modelling capabilities. We describe a new approach and toolset that we have developed, MaramaMTE+, which improves the ability to model realistic web client load behaviour, automatically generates complex web application testing plans and scripts, and integrates load behaviour modelling with a generic performance engineering tool. MaramaMTE+ uses a stochastic form chart as its client loading model. A 3rd party web crawler application extracts structural information from a target web site, aggregating the collected data into a crawler database that is then used for form chart model generation. The performance engineer then augments this synthesized form chart with client loading probabilities. Realistic web loading tests for a 3rd party web load testing tool are then automatically generated from this resultant stochastic form chart client load model. We describe the development of our MaramaMTE+ environment, example usage of the tool, and compare and contrast the results obtained from our generated performance load tests against hand-built 3rd party tool load tests.

Categories and Subject Descriptors
D.2.8 [**Metrics**]: Performance measures; D.2.2 [**Design Tools and Techniques**] Computer-aided software engineering (CASE).

General Terms
Measurement, Performance, Model generation, Code generation, Web Load Testing, Web User Behavior.

Keywords
Form Chart, Web Load Testing, Web User Behavior Modeling

ASE'07, November 5-9, 2007, Atlanta, Georgia, USA.

1. INTRODUCTION
Web application load testing is an important part of web performance engineering. Load testing measures the response time, throughput, and availability of a target website from a client's perspective (usually a web browser). Load testing needs to be undertaken rigorously before a robust cost-effective website can be achieved [19]. A wide range of load testing tools [1] [4][8] [17][19][22] and more generic performance and reliability engineering tools [6][20][21] have been developed and are widely used for web application testing. Modelling client user behaviour and constructing load testing plans are core functionality of these tools. However, almost all such tools provide only a fairly basic model of client behaviour: a sequence of requests on the web server arranged into repeating groups, multiple threads (to mimic large numbers of client browsers), and some limited conditional behaviour depending on the web server response. Some tools support parameterisation of loading tests to allow configuration of different test cases and test data, but usually is limited to different example data set to the web application. Simulation-based performance engineering utilises formal load models e.g. queuing theory based [15] but these approaches do not run tests against real web applications, only limited models of them. Limited formal client loading models for web applications have been developed to date and only very limited support exists for using these behaviour models to generate web loading tests [7][8][22]. Building such client load models by hand is error-prone and time-consuming, especially for large web sites and evolving web sites.

In earlier work we developed MaramaMTE, an integrated performance engineering environment that supported the representation of client load models based on Draheim & Weber's Stochastic Form Charts [8]. A Form Chart model shows the structure of the target website and can be augmented with probabilities to capture client behavioural interaction with web forms. MaramaMTE supported simple test case generation from these form chart loading models and engineers could construct multiple testing plans by changing the values of model parameters. In MaramaMTE engineers must construct form chart loading models by hand and modify them incrementally to reflect changing web site structure and user behaviour. Only a simple Java client load test application was generated to run the tests, providing very limited stress testing and analysis capabilities.

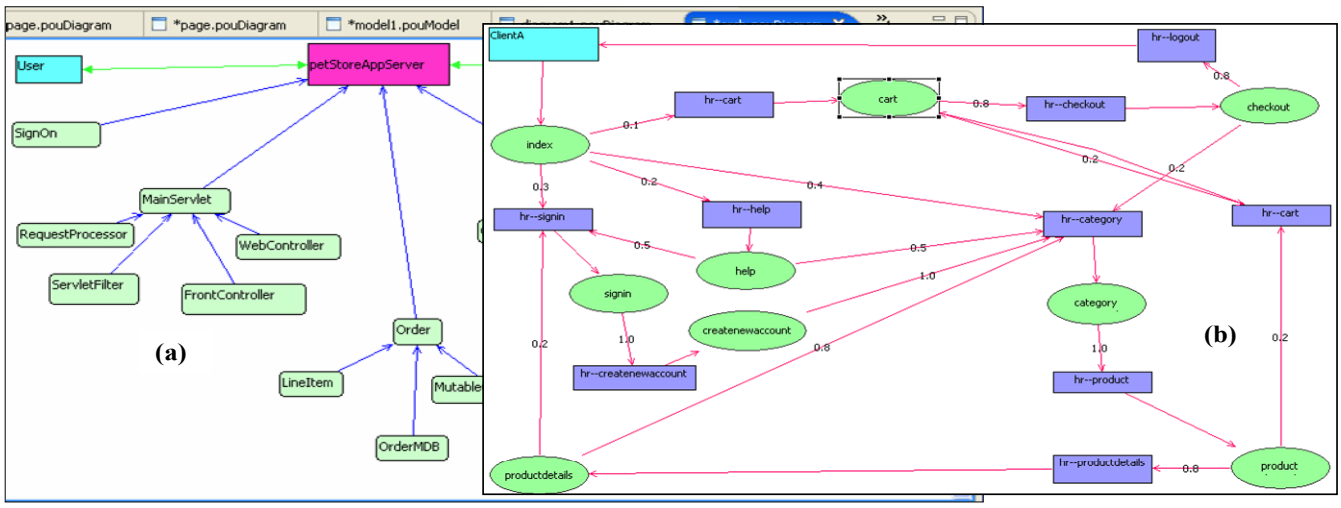


Figure 1. (a) High-level view of Java PetStore software architecture; and (b) sample Stochastic Form Chart loading model.

MaramaMTE+ extends MaramaMTE to automate the tedious and error-prone process of client load model structuring and testing plan and script generation. MaramaMTE+ uses a web crawler to extract structural information from a target web site, aggregating collected data for use in form chart model generation. After manual probability augmentation to produce an enriched stochastic form chart model, this model is then used to automatically generate a testing plan for a specific third party testing tool e.g. Apache JMeter. Testers can both design testing plans in the model from scratch, when empirical user behaviour data is unavailable, and also use the model to analyze collected empirical user behavior data and construct more realistic testing plans. MaramaMTE+ integrates form chart modeling with architecture design models, making it a generic software architecture performance engineering tool.

We first present a motivation for our work, the analysis of the Java PetStore J2EE reference application including required support in MaramaMTE+ and related research to date. We then overview the MaramaMTE approach to web application performance engineering and illustrate an example usage of the toolset applied to the Java PetStore web application. We describe the design and implementation of our MaramaMTE+ toolset and then compare and contrast the results of using MaramaMTE+ to performance test the PetStore vs. using hand-built load testing models in Apache JMeter and Microsoft Web Application Stress tool. We conclude by summarising the main contributions of our research and key areas for future research.

2. MOTIVATION AND RELATED WORK

We developed the MaramaMTE performance engineering tool to allow us to explore new ways to model software architectures and client load models for performance engineering complex systems. MaramaMTE [11] is a generic software architecture design and performance engineering tool. It supports a client-server (with tiered-server-side) architectural style. Users can model software architecture, generate a test bed (fully functional client-server application with embedded performance evaluation parameters), execute the test bed, and collect performance test results. MaramaMTE and its predecessors [4] demonstrate the merit of allowing software engineers to efficiently optimize candidate architecture design via early-design phase performance engineering before system implementation.

Our motivation for the work described here was a strong desire for MaramaMTE to provide support during *reengineering* of existing software systems, particularly of *existing* web applications. Here there is a need to compare an existing software system's architecture and performance with that of new candidate designs.To achieve this, we needed to solve a series of problems, particularly around the methods needed to recover legacy system architectural components into a MaramaMTE architecture model and to compare the performance of legacy system components with that of the proposed new design. A rich, realistic client behavior model is required in MaramaMTE before reliable performance comparisons can be made.

However, like most other generic performance evaluation tools, MaramaMTE provides good capabilities for modeling server-side parts of an application but relatively weak support for modelling realistic client behaviours. Figure 1 (a) is a simplified MaramaMTE architecture model of the standard java PetStore application [14]. This architecture design, at a high level of abstraction, shows the main components of the PetStore system, including client-side components and the main server-side components (including remote objects "SignOn", "MainServlet", and "RequestProcessor", etc; application server "PetStoreAppServer", etc). An engineer uses MaramaMTE to vastly enrich this simplified server side model by providing operations for each functional component and to set up a wide variety of component properties used to generate performance loading information. However, the client side model (the "user" component in Figure 1 (a)) contains little more than a sequence of remote server component requests (not shown). This is quite insufficient for modelling realistic web user behaviour [11].

A *form chart* model is a technology-independent bipartite state diagram used to simulate user behaviour [7][8]. The model describes at a high level what the user sees as system output, and what he or she provides as input to the system. Form chart models consist of pages, actions, and transitions. Pages represent possible states of a website; actions represent website server side components, their behavior, and their response to requests; transitions represent association between pages and actions. Form charts have limited intention, and their semantics are also limited.

MaramaMTE supports a semantically-improved version of form chart model, the *stochastic form chart* model, to provide a formal model of realistic user behavior. In order to simulate realistic users, the basic model is extended with stochastic functions to describe navigation, time delays and user input [8]. Figure 1 (b) shows a sample MaramaMTE page flow model diagram representing user behavioural interaction with part of the PetStore. This shows a starting state (top left rectangle: "ClientA"), various web pages (ovals: "index", "cart", signin", "productdetails", etc), and various actions allowing movement between web forms (rectangles: "hr-signin", "hr-cart", etc). The engineer captures probabilities of moving from a given web form via actions e.g. from "index" to "signin" 0.3; from "index" to "help" 0.2. For each web page, various properties of the page are captured, such as the URL. Integrating such page flow models with MaramaMTE provides an effective and realistic user behavior model to augment the simple MaramaMTE performance model [11].

The construction of a user behavior model i.e. the MaramaMTE form chart model is done manually which is both error-prone and time consuming, especially when large websites are being re-engineered. In addition, generation of only a simple Java application from this model to load test the web application is insufficient. To reduce the bottleneck around user behavior model specification and improve load test generation we developed the following requirements for an improved tool:

- an analysis model is needed to relate actual web site structure (from a user perspective) to possible form-level interactions (via e.g. a form chart) to capture a realistic model of web application usage for test generation
- the structure of the analysis model needs to be automatically generated from the target web site instead of manually built
- tool support must allow users to easily change client load model testing parameter values then generate multiple testing plans and scripts automatically
- a 3rd party load testing tool should be used leveraging its capabilities for large web application stress testing but generating its limited client loading models from the more abstract stochastic form chart load model
- the tool should be well-integrated within a generic performance engineering environment allowing the realistic client behavior model to influence the design of other parts of software system e.g. the software architecture

A wide range of performance engineering tools have been developed for both web applications and other more general software applications [19]. Some use simulation-based modelling to provide approximations of application performance [15]. This is useful for applications with extremely large clients and difficult-to-obtain hardware resources e.g. supercomputers where empirical testing is very difficult. Unfortunately the performance estimations obtained may be very different to that obtained by running on actual hardware and networks.

A number of practical load testing tools have been developed for stress-testing web and other software applications. One example is Apache JMeter [1] which offers both a textual and a GUI environment for users to construct testing plans and scripts. Unlike many such tools JMeter testing plans can be reasonably complex and can simulate and analyze a variety of load scenarios to obtain a quantitative insight into a web site's loading characteristics. However, JMeter does not support formal behavior analysis modeling so users must construct their testing plans based on ad-hoc testing goals. Microsoft's Web Application Stress Tool (WAS) [17] is a simple load and stress testing tool that can be closely integrated with Visual Studio or other development environments. It does not support as complex a testing plan as JMeter. It supports simple load modeling capability via a test setting wizard resulting in a testing plan with limited flexibility. Most other commonly-used web testing tools provide similarly limited client load model specification. For example, Selenium [13] replays pre-recorded tests; while the combination of e.g. FitNesse and Selenium can replay pre-packaged black-box tests. Tool users need to construct interactions to mimic user behavior based on ad-hoc analysis of the web application and this is normally informal, non tuneable and problematic for automated information exchange [8].

Several experimental web application testing tools have been developed that try to encapsulate load models in various ways [2][6][9][20]. These require users to construct models of the system under test, usually UML or similar architectural models or black-box services e.g. URLs, and generate loading tests from them. Many, including our own ArgoUML[4], focus on server-side modelling with limited client load modelling support. Some, like Surge, scale up to large web site analysis [2]. Others provide proof-of-concept approaches to modelling client and server properties for synthesising loading models [6][20][22]. Some web analysis tools try to obtain realistic testing plans by recording user behavior then replaying or analysing it [21]. The recorded user behavior may be close to reality, but the raw behavior data needs to be abstracted to a suitable client loading model for the purpose of user behavior analysis and data exchange. No existing web application testing tool provides the functionality demanded by our requirements.

3. OUR APPROACH

Figure 2 shows how MaramaMTE+ supports automatic extraction of a user client load behavioural model and generation of load testing plans and scripts via a web crawler and a stochastic form chart model. Initially the target web site is crawled (1) by a 3rd party web crawler we have adapted for the purpose. The web crawler generates http request data from the target website (2), and this web structural data is stored in a database of possible requests to the target web site (3). The extracted http requests are used to synthesize an initial form chart model which is then imported into MaramaMTE+ (4) and a default layout applied to generate one or more form chart diagrams. This initial form chart model is then manually augmented by the performance engineer using the MaramaMTE+ form chart diagramming tools to specify probabilities and other stochastic parameters. The model may be versioned to allow variations of the parameters and user behaviour to be modelled for comparison (5). Load testing plans are then generated for third party web application stress-testing tools (e.g. Apache JMeter or Microsoft Web Stress Tool) (6). In addition, MaramaMTE+ can generate server-side applications from its architectural modelling views to allow load testing of proposed web applications as well as the existing application from which the form chart was synthesized. Stress-tests are run against the

target web application and results are collected by the stress-testing tool (7). These test results are shown either inside MaramaMTE+ or via a third party visualization tool (8) and may be stored for future reference and comparison (9).

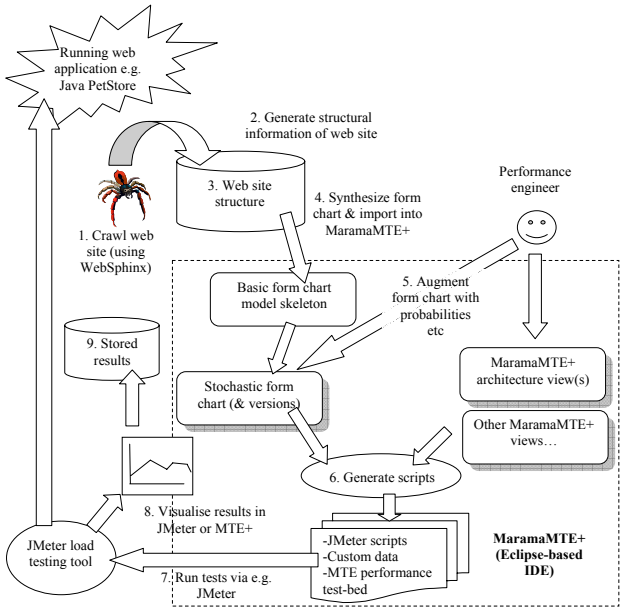


Figure 2. Crawling web sites to extract form charts and generating stress-tests with MaramaMTE+

Web Crawlers have long been used to explore the structure of web sites from a user's perspective. We use WebSphinx [18] to extract the main screens, screen content, hyper links, and http requests plus parameters and values from a web application. The extracted information is collected into a crawler result database which makes the web site structural data available for further use. The MaramaMTE+ model and view generator then retrieves this web site structural data from the database and generates a basic form chart model, an instance of one of the MaramaMTE+ view types. The generated form chart needs to be enhanced and validated manually by adding additional data, such as action flow transition probabilities and MaramaMTE-specific code generation data (parameters and values). The enriched page flow model is a stochastic form chart designed to contain sufficient information to generate specific load testing plans for the web site, including a JMeter testing plan, Microsoft Web Stress Test or a MaramaMTE+ thick client testing plan (a Java client-side application). Most parts of the toolset execute as components inside MaramaMTE+, itself a set of plug-ins to the Eclipse IDE. MaramaMTE+ provides the superstructure to start up the web crawler, extract target web site structural data, manage the crawler database, generate the initial form chart model, support the modification of the model, generate testing script/plans and test beds, initiate third party tools (e.g. JMeter) or MaramaMTE+-specific test beds and show evaluation results.

4. EXAMPLE USAGE

We use the Java Pet Store reference application [14] as an example to illustrate our approach. This is a de facto benchmark application for performance evaluation technologies. Its architecture has been well documented, with the main functional modules represented in Figure 1 (a). From an end user (shopper) perspective, basic interactions with the web site include: users sign on; they browse the catalog; they buy pet(s) by putting them into a shopping cart; they check out; and receive purchase confirmation. It is obvious that the distribution of the types of user-website interactions is not linear. For example, "browse catalog" will be the most frequent interaction as, just as in a real shop, there will always be more browsers than buyers. The "buy pet" interaction is likely to be more frequent than "check out" because a buyer may buy more than one pet before checking out. Our stochastic form chart model(s) for the PetStore application must capture these nuances. In addition, we may want multiple models for different kinds of users e.g. business vs. personal shoppers or situations, eg Christmas vs February, providing even more fine-grained client load modelling for the web application.

4.1 HTTP Request Extraction

MaramaMTE+ uses WebSphinx to extract the pet store structural information into an http requests database. As shown in Figure 3 (a), users run WebSphinx as a java application invoked from MaramaMTE+ running as a set of plug-ins in an Eclipse workbench. The user supplies the target web site information to the crawler (Figure 3 (b)). The crawler explores the main screens, hyper links among screens, and http requests, parameters and values for the target web site. MaramaMTE+ collects data into a purpose-built crawler/result/ http request database. The crawler database contains 7 tables to offer easy data access when the database is needed to generate form chart model. In Figure 3 (c) shows the "http_request" table that holds http requests and associated pages, and Figure 3 (d) shows the "page" table that holds information about page ids and names.

4.2 Form Chart Extraction

MaramaMTE+ retrieves data from the database and uses it to generate an initial form chart model. Figure 4 shows the main processes of form chart generation. Tool users start the process by opening a wizard "import pages", as shown in Figure 4 (a).

Tool users choose the database for the target website they are interested in and import available pages. Tool users choose the pages they require, go to a form chart model, push an "AddTo Diagram" button, and corresponding form chart Page elements are added to the diagram/model. The wizard then allows tool users to add "Action" components to the model (Figure 4 (b)). Tool users are given a list of available actions (based on the previously chosen pages), and similarly add "Action" components to the form chart model. The wizard then allows tool users to add transitions between "Page" and "Action" components. The generated form chart model (Figure 4 (c)) is a perfectly correct form chart model but is far from ideal aesthetically and requires rearranging by the performance engineer to improve its readability (Figure 4 (d)). Each oval component represents a web page users may request, and each rectangle component models actions directed at the server application. For large web sites, MaramaMTE allows engineers to view and edit partial form charts in multiple diagrams that share a single model. MaramaMTE's diagram and model versioning mechanism [16] allows performance engineers to create alternate versions of the form charts for a given web application and compare, difference and merge them as required.

4.3 Form Chart Augmentation

MaramaMTE+ uses the generated form chart in two ways: 1) using it as an independent model to generate testing plans and scripts for third party load testing tools (e.g. Apache JMeter); or 2) using it together with other MaramaMTE+ models (e.g. the architecture design models, business process models and service composition models) to generate a performance evaluation test bed [11]. For either of these purposes a generated form chart model normally needs to be augmented by appropriate properties that contain essential code generation information. For example, MaramaMTE+ stochastic form chart transitions use a property "Probability" to model the chance of users requesting a web page. This property may be computed in various ways e.g. randomly within a specified range, fixed

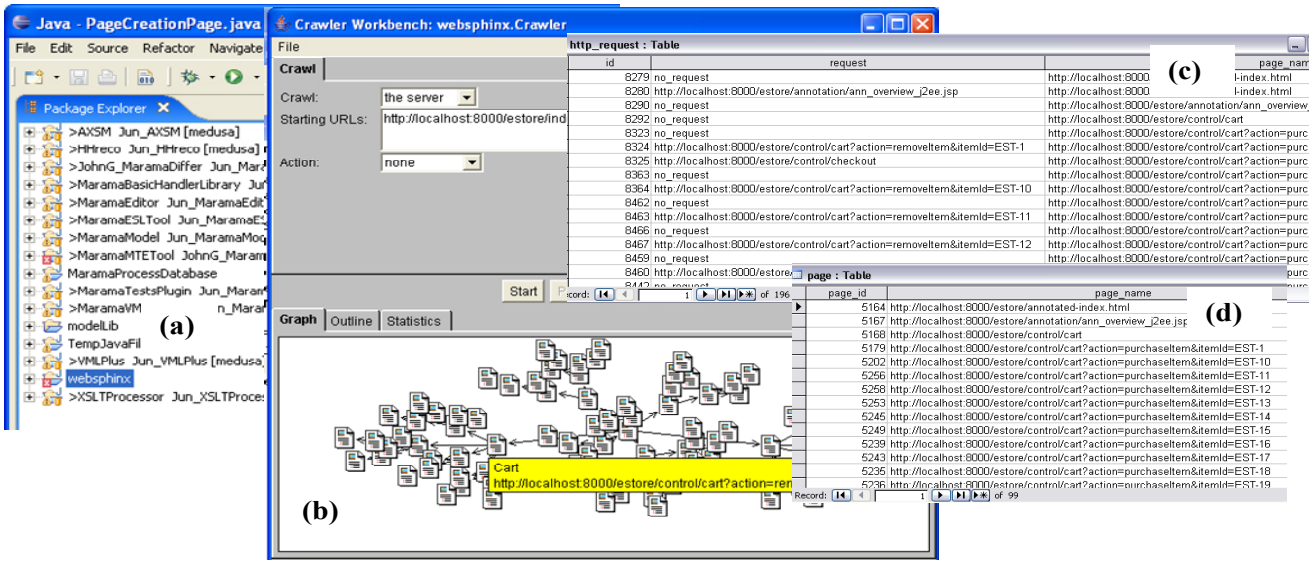


Figure 3. MaramaMTE+ using WebSphinx to extract structural information from PetStore web application.

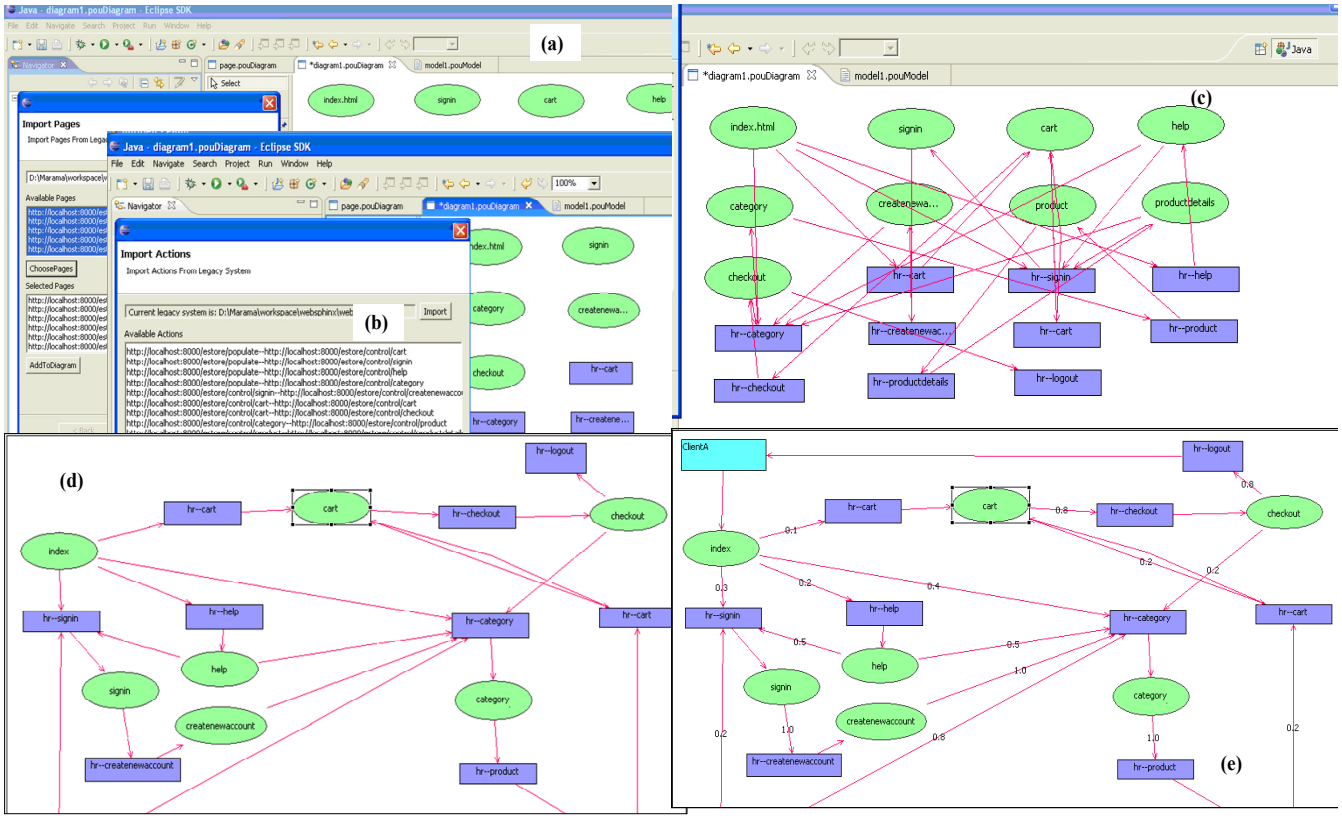


Figure 4. MaramaMTE+ automatically generating a form chart model from the web crawler http request database.

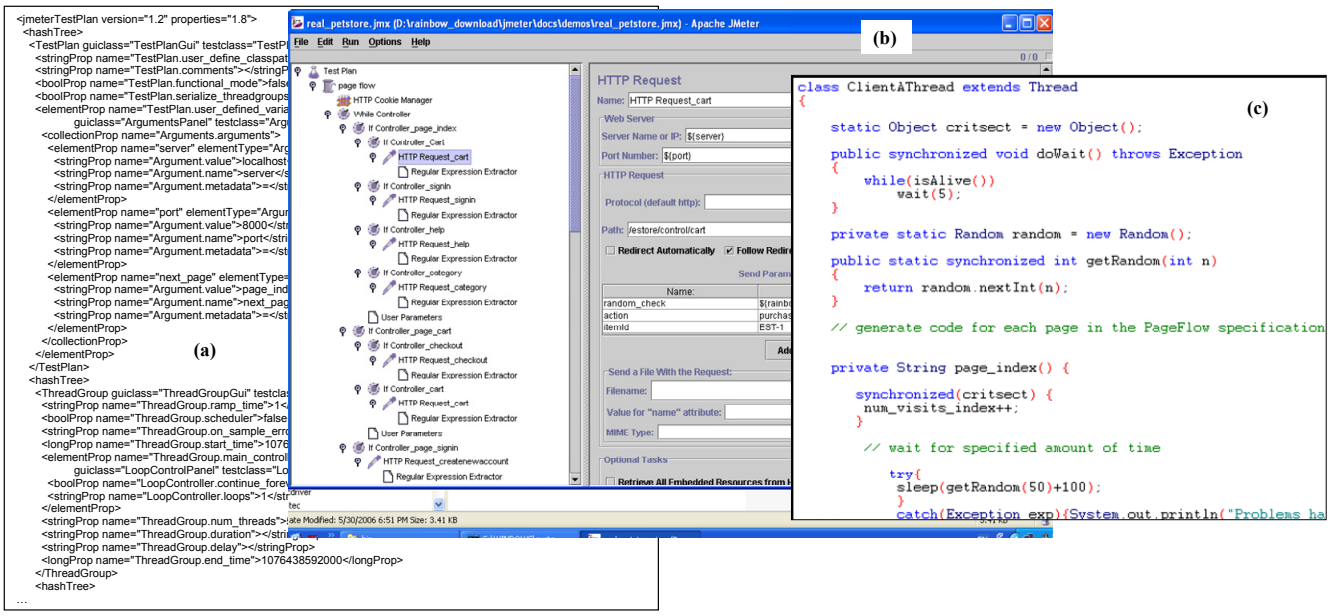


Figure 5. JMeter test plan, JMeter, and test bed client application

value, normal curve distribution or from monitored web site usage [8]. Form chart pages require properties to specify such stochastic information including "delayKind", "delayMax", "delayMin", and "delayStdDev". The performance engineer thus needs to flesh out the initially generated form chart model with suitable probabilities on all transition links. A generated model may also be augmented by adding MaramaMTE+-specific modeling components. For example, the "ClientA" component in Figure 4 (e) does not belong to a generic form chart model. It represents a client-side start-up component for use in loading test code generation, because all testing plans need an entry point and various configuration properties. Figure 4 (e) shows the augmented stochastic form chart for the PetStore application in MaramaMTE.

4.4 Loading Test Generation

A stochastic form chart model can generate a range of target load testing plans and associated scripts or programs to implement the testing plan. Here we describe the generation of an Apache JMeter testing plan and associated scripts. A JMeter test plan consists of one or more thread Groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements [1].

When generating a JMeter test plan, each MaramaMTE+ form chart "Page" component represents the state of the website, each "Action" component represents the JMeter http requests to obtain certain web pages, and each "Transition" link models transitions between web pages via "Actions". Element properties such as "Probability" and "URL" are used to generate the logic controllers of the JMeter testing plan. MaramaMTE+ uses the Eclipse JET template-based code generator to translate a form chart model into a JMeter testing plan, as illustrated in Figure 5 (a). In the generated JMeter test plan the form chart page components and properties have been translated into http requests on the target web application server. The actions and transitions have been translated into control logic, essentially implementing a state machine. The client component has been translated into the root JMeter test plan configuration and properties. This generated test plan can be loaded into the JMeter test tool's GUI environment, as shown in Figure 5 (b). A JMeter plan does not need to be loaded into JMeter's GUI interface but can instead be executed directly through a command line in an automatic tool suite environment or directly by MaramaMTE+. MaramaMTE+ can also generate a test-bed-specific client side Java application using the PetStore form chart model, as illustrated in Figure 5 (c).

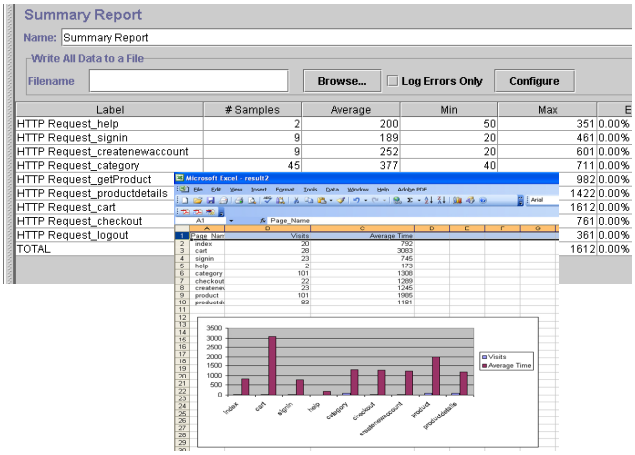


Figure 6. (top) sample load testing evaluation result and report in JMeter; (bottom) sample performance evaluation result of MaramaMTE+ testbed

4.5 Running Generated Load Tests

To complete our case study, we have undertaken initial load testing using generated test plans both with JMeter and MaramaMTE+. We ran our generated JMeter testing plan against the real running Java PetStore application. The evaluation results are displayed using JMeter's summary reporting component, as shown in Figure 6 (a). This summary is generated as JMeter runs the http requests

against the PetStore server and reports visited web pages, sample response time, throughput, etc. The MaramaMTE+ generated test plans can be configured to allow small numbers of requests to be sent to the PetStore application server or very large numbers of concurrent requests. JMeter supports both multi-threaded test execution and distributed test execution using multiple hosts. MaramaMTE+ allows the performance engineer to specify these properties for the generated JMeter test plan, supporting large-scale stress-testing. Multiple test runs are stored by JMeter and results from multiple test runs can be displayed simultaneously to compare performance of the target web application. Figure 6 (b) shows example results (in Excel) from our generated MaramaMTE+ client-side Java application running against a generated server-side test-bed. This supports exploratory performance engineering for web applications in early-phase design or reengineering.

5. DESIGN AND IMPLEMENTATION

MaramaMTE+ is a set of Eclipse IDE plug-ins that we developed using the Marama meta-tool development framework [12]. A Marama tool includes a shared data model, multiple diagram editors, and a set of event handlers providing constraint handling and code generation support. The high level architecture of MaramaMTE+ is illustrated in Figure 7. A set of editors support diagrammatic modelling (1). Two key diagram types are used – the form chart model and the architecture model (2). Diagrammatic editors are instantiated to edit these models using the Eclipse Graphical Editing Framework.

We used the WebSphinx crawler tool [18] to extract target web application structure. WebSphinx is designed as an independent Eclipse plug-in and MaramaMTE+ starts up WebSphinx through a "WebSphinx coordinator" component (3). This component also coordinates with a "DatabaseProcessor" component to collect crawled target web application structural information and store it in the crawler database (4). The "DatabaseProcessor" component in "MaramaModelGenerator" accepts all crawled data from WebSphinx and saves it in the crawler database (4). The database processor manages database connections, retrieves website data and transforms it into data suitable for initial form chart extraction by the "DiagramGenerationManager" component.

MaramaMTE+ manages a custom web crawler result database which is designed to provide website information for the synthesis of MaramaMTE+ form chart models. The database can hold data for multiple target websites and tool users can version the database. The crawler database consists of tables "http_request", "page", "parameter", "path", "port", "server", and "target system". Table "target system" contains data from target websites (such as PetStore) that have been explored by the web crawler. Tables "server" and "port" respectively record the server and port a target website is running, for example, we run PetStore website on port 8000 of the "localhost" server. Table "path" is a "convenient table that records a website's main http request paths by combining tables "port", "server", and "target_system". Table "parameter" records parameter and value for each http request path. Table "page" gives every main web page a sensible name and table "http request" is another "convenient table" that records fully-formatted http requests by joining tables "page", "parameter", and "path".

The "DiagramGenerationManager" component is the key element of the "MarmaModelGenerator" subsystem. This retrieves website information from the web crawler database (5) and communicates numbers, average response time, min and max with the MaramaModel to generate form chart model entities, associations and their visual icons. The "Algorithm" component arranges the generated visual icons and connectors into a basic form chart diagram layout. We currently use a simple layout algorithm to arrange the pages and actions one after another as illustrated previously. We have also experitmented with force-directed layout algorithms as provided by the CCVisu 3rd party package [3]. "DiagramGenerationManager" then instantiates the synthesized form chart diagram using MaramaEditor ready for performance engineer augmentation.

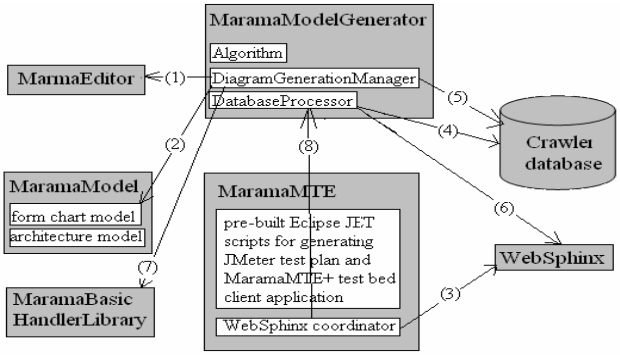


Figure 7. High-level architecture of MaramaMTE+

MaramaMTE+ uses Eclipse Java Emitter Templates (JET) scripts to generate form chart-based test plans and scripts and server-side implementations from the form chart and architecture MaramaModel components. JET uses a subset of the Java Server Pages (JSP) syntax making it easy to write the required code generation templates with embedded Java code extracting values from MaramaModel components. MaramaMTE+ traverses the form chart model and transforms each element into a set of target load testing tool abstractions. For example, in Figure 8 we see a JET template generating a root JMeter test plan from the client form chart component and its properties. This includes the name of the test plan, host(s) for tests, threads per test host, etc. JMeter initialisation components and scripting are also generated e.g. to set up and initialise timing monitors (WhileController). The first page in the form chart model is then transformed into an initial http request on the target web application (IfController). This includes the target url and any parameters and example values encoded in the form chart model. Transitions to Actions in the form chart model generate decision logic in the JMeter test script (through JMeter's RegexExtractor, UserParameters, etc), which implements a state machine model of user behaviour. Probabilities may be simple random, fixed times, or complex stochastic probability models as specified by the form chart model (through JMeter's BeanShellTimer, Gaussian Random Timer, etc).

6. DISCUSSION

We have evaluated MaramaMTE+'s effectiveness for supporting realistic client load test generation by using it to synthesize a formal model of client loading for several web-based systems, including the Java PetStore, a micro-payment system we have previously analysed extensively [5], and our own Departmental web site. This involved for each target system: (1) extracting a form chart from the target system using MaramaMTE+ and WebSphynx; (2) augmenting the form chart with stochastic

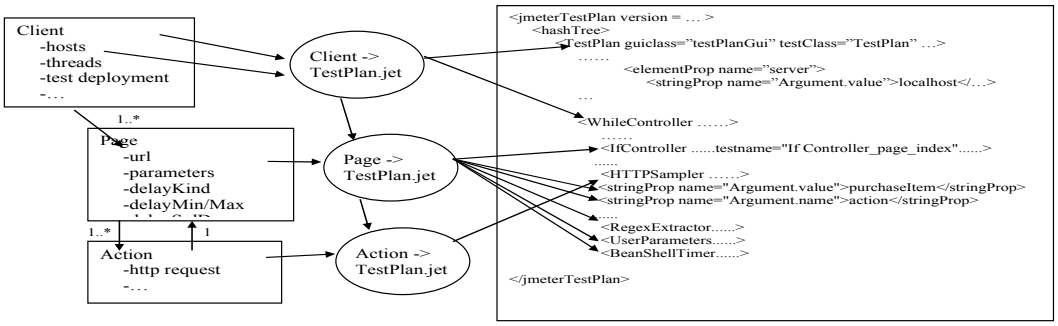


Figure 8. JMeter test plan generation from MaramaMTE+ form chart model.

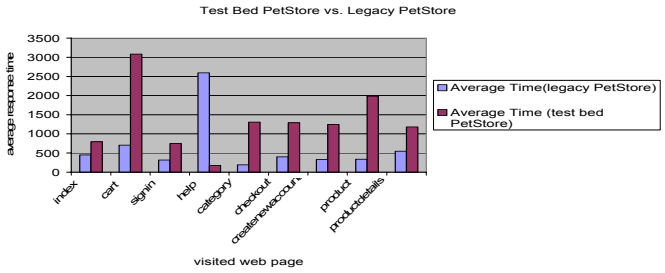
System	Work Efficiency		Performance of Test Bed PetStore vs. that of Legacy PetStore
	Effort to manually build test plan and scripts and run	Effort to augment extracted form chart and run tests	
Java PetStore	18 hours	3 hours	
NetPay micro-payment system	25 hours	3 hours	
Computer Science web site	15 hours	2.5 hours	

Table 1. Empirical comparison results

probability information, informed by the requirements of the system concerning expected user loading; (3) generating JMeter client load tests from MaramaMTE+; (4) running these tests and comparing results against hand-built JMeter system load tests.

Table 1 presents some initial empirical evaluation results. *Work Efficiency* compares the manual effort to construct JMeter test plans with the JMeter IDE tool for the legacy websites (PetStore, NetPay, and our department website) with the effort needed to augment an automatically extracted form chart and generate comparable JMeter loading test plans. The load testing work was undertaken by an experienced software engineer who knew each system well. The manually created JMeter test plans were done using the JMeter GUI editor rather than replay/capture tool. This was because of the conditional logic and parameterised inputs for several pages that had to be used in order to make the tests as flexible as the MaramamMTE+ generated tests. An additional one-off overhead of approximately 4 hours was incurred for the engineer to become familiar with MaramaMTE+. Efficiency gains of between 5-8 times were demonstrated. We are currently undertaking a more extensive user evaluation to draw more reliable results.

The right hand column shows another important result. We ran the same PetStore testing plan on the generated MaramaMTE test bed and the legacy PetStore to compare average response times. Here we can see different response time values and distributions in the two systems. This comparison of performance can be used to guide the abstraction process from a legacy system to an abstracted architecture design in Marama MTE. More specifically, the comparison of performance can tell 1) if the abstraction/refinement has influenced the performance distribution; 2) if the performance differs between the low and high level models and permit investigation as to which part of the system has caused the difference. We are developing a mechanism to support refining legacy web systems to Marama MTE-styled architecture design, and will use such performance comparisons to guide users in their abstractions and refinements.

Most current web performance engineering tools are designed for evaluating the performance of existing web sites under stress loading [6][21]. Most require considerable knowledge of the system under test to formulate and build appropriate loading test plans and scripts. Almost all current web performance engineering tools have limited formal models to capture client load modelling, usually limited to server request sequences and decision logic on server responses. These require much effort to build and maintain, especially for systems under change, large systems or systems the performance engineer is unfamiliar with. In contrast MaramaMTE+ allows a target web application to be crawled and a high-level, formal client load model to be extracted and developed quickly by the performance engineer. As shown by preliminary results with MaramaMTE+ in Table 1, effort is much lower for generating a client load model with it than using JMeter's GUI test plan designer directly.

MaramaMTE+ also supports early design phase exploratory performance engineering of yet-to-be-built systems. This is done by generating prototypical server-side components from a MaramaMTE+ architectural model and running the generated client load model against this rapid prototype web application server. Unlike our previous work where a simple client load model was used [4], MaramaMTE+ allows a richer client load model to be used for the prototypical web server evaluation. Unlike simulation-based modelling for such analysis [15], real server code is generated and run, resulting in more accurate estimation of eventual web application performance.

Key advantages to our new approach are its use of a formal model for client load behaviour modelling; the ability to extract

model structure from a web application via web crawling; model-based generation of 3rd party stress testing tool test plans and scripts; and ability to run and compare web application performance under numerous different loading models accurately and efficiently. Unlike most web performance engineering tools, MaramaMTE+ extracts most information needed for the client load testing model from the web application directly. This greatly reduces errors and time taken to develop the model used to represent client behaviour. The stochastic form chart model we use for this provides a formal model which can be reasoned about in addition to its use to generate test plans and scripts. It allows performance engineers to model client behaviour in terms of probabilistic interactions and server responses, and we have shown this to provide a reasonably accurate model of user behaviour [8]. Engineers can even build different versions of stochastic form charts for all or part of a web application to compare and contrast performance under different client behaviour models. Generating test plans and scripts for 3rd party stress testing tools allows Marama MTE+ to leverage their advanced features for load testing. For example, JMeter provides sophisticated measurement, reporting, distributed test execution and test scheduling support features that MaramaMTE+ is able to reuse directly with little effort. However we must also live with 3rd party tool limitations. Most web application stress testing tools have less rich client behavioural models than MaramaMTE+ form charts. Thus we need to simplify the model when test scripts are generated or extend the testing tool (if possible). Sometimes implementing form chart-specified behaviour is quite complex in the 3rd party testing tool. For example, to implement a probabilistic state machine in JMeter proved to be quite challenging. Use of 3rd party monitoring and measurement tools can also be problematic as these may perturb the results obtained. It is not always easy to control such tools in the way we are able to when generating our own client load test implementation.

The extracted form chart structure can be very large for large web sites. We mitigate this issue in MaramaMTE+ by allowing any number of partial form charts to be rendered in diagrams, managing cognitive complexity of the models. Automatic layout of the extracted form chart diagrams is currently rudimentary but MaramaMTE+ allows the engineer to modify layout and diagram elements. This supports tailoring of the form chart-based client behavioural models by the engineer to best suit their cognitive modelling of the target application's client behaviour.

The use of 3rd party testing tools to carry out the stress testing means we can leverage their advantages e.g. JMeter's support for data capture, presentation and distributed stress testing.

The fundamental issue at the heart of our approach is the adequacy of the stochastic form chart model to capture "realistic" client behaviour for web application stress testing. MaramaMTE+'s stochastic form charts rely on the performance engineer specifying probabilities of different client interactions with web pages and example data parameters for invoked web server pages. These are thus as sensitive to erroneous data as in any other stress testing tool. Our experiments to date have shown the client load modelling and performance results obtained from generated JMeter test plans to compare favourably with observed user behaviour. However, a client load test model is always going to be a model, even if based on empirically measured or benchmarked results. One mitigating approach we have adopted for MaramaMTE+ is to allow multiple form chart models to be defined for the same target web application. This allows the performance engineer to experiment with a variety of different client load mixes and compare web application performance under these different conditions.

Augmenting extracted form charts with probabilistic information about user behaviour can be a complex process for large web sites. A key area of future work is to infer such stochastic form chart parameters from observed target web application behaviour. We plan to monitor the actual usage of web sites using tools similar to those used for logging and fault analysis [21] to provide real user session histories with large numbers of http requests. We will then analyse these logs to infer transition probabilities improving accuracy of the client behaviour model. We will also use example data in these user session http requests to provide realistic sequences of parameter values to invoke web server pages. This will still allow performance engineers to change these probabilities or to specify alternative versions of the form chart for the same application to investigate impact of differing client load models on the web application performance.

Targeting our test plan and script generation to other popular web stress loading tools e.g. Microsoft's Web Stress Testing tool [17] is also desirable. Some applications may require tuning of parameters better supported by stress loading tools other than JMeter e.g. database and web server thread pools and memory management. Combining our stress testing tools with server management tools would also allow us to monitor other system resource usage in addition to response time e.g. CPU, disk and memory. Currently all testing is done by invoking the web server from a "browser proxy", with no rendering of returned data or execution of client-side JavaScript. With the increasing use of AJAX and similar client-side scripting technologies, plug-ins and rich content client side code may impact greatly on client request/response behaviour. We plan to experiment with generating Selenium Remote Control unit tests which allow a browser proxy to communicate with the target web application server, and render/execute returned content/scripts.

We need to improve the automatic layout of synthesized form charts along with support for visualising test performance measures. In addition, support for semi-automatic grouping of large web site structures into multiple form chart diagrams is needed to manage large web site evaluation. Side-by-side comparison of performance measures resulting from tests generated from different form chart models for the same application could be improved to show different performance under different client behaviour models.

Ultimately we would like to support reverse-engineering of architecture designs into MaramaMTE+, reverse engineering of client behaviour models, and forward engineering of parts of the system with alternative architectural and client behaviour models. This would allow an engineer to reverse engineer an existing web application structure and behaviour model; modify parts of these models; and then generate not only new client load tests but prototypical new server components and stress-test these rapid prototypes. This would provide a performance engineering environment allowing exploratory analysis of the performance effects of web application architecture changes.

7. SUMMARY

In this paper, we presented MaramaMTE+, which uses an innovative approach to automate the process of retrieving website structural data from a web user's perspective and using it to generate a form chart model and load testing plans. We demonstrated the effectiveness of the approach through a case study, where a running PetStore site was crawled, structural data extracted, a form chart model automatically generated and manually augmented, JMeter testing plans generated and executed, and load testing results collected. Future directions for MaramaMTE+'s development were also described, including combining the generated form chart with a generated design level model of a legacy system which will make it possible for ordinary tool users to make rigorous comparisons between different products (e.g. Java PetStore and .NET PetShop).

8. ACKNOWLEDGMENTS

Parts of this research were supported by a grant from the New Zealand Foundation for Research, Science and Technology Research for Industry programme. The authors would like to thank Christof Lutteroth for discussions around Form Chart reverse engineering approaches.

9. REFERENCES

[1] Apache JMeter: http://jakarta.apache.org/jmeter/index.html

[2] Barford, P. and Crovella, M. Generating representative Web workloads for network and server performance evaluation, Proc 1998 ACM SIGMETRICS Joint Intnl Conference on Measurement and Modeling of Computer Systems, Madison, Wisconsin, 1998, pp. 151-160.

[3] Beyer, D. CCVisu - A Tool for General Force-Directed Graph Layout and Co-Change Visualization, See http://directory.fsf.org/ccvisu.html

[4] Cai, Y., Grundy, J.C. and Hosking, J.G. Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool, In Proc 2004 IEEE Intnl Conference on Automated Software Engineering, Linz, Austria, September 20-24, IEEE CS Press, pp. 36-45.

[5] Dai, X., Grundy, J. and Lo, B.: Comparing and contrasting micro-payment models for Ecommerce systems, International Conferences of Info-tech and Info-net (ICII), China (2001).

[6] Denaro, G., Polini, A., Emmerich, W. Early performance testing of distributed software applications, In Proceedings of the 4th Intnl Workshop on Software and Performance, Jan 14-18 2004, Redwood City, California, pp. 94-103.

[7] Draheim, D. and Weber, G., Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints, LNCS Volume 2889/2003, Springer.

[8] Draheim, D., Grundy, J.C., Hosking, J.G., Lutteroth, C. and Weber, G. Realistic Load Testing of Web Applications, In Proc 10th European Conference on Software Maintenance and Re-engineering, Berlin, 22-24 March 2006.

[9] Elbaum, S., Karre, S., Rothermel, G. Improving Web Application Testing with User Session Data, In Proceedings of the 2003 International Conference on Software Engineering, IEEE CS Press, 2003.

[10] Foy, C. Fitnesse Selenium Wrapper, Saturday, September 16, 2006, http://www.cornetdesign.com/2006/09/fitnesse-selenium-wrapper.html

[11] Grundy, J.C., Hosking, J.G., Li, L. And Liu, N. Performance engineering of service compositions, ICSE 2006 Workshop on Service-oriented Software Engineering, Shanghai, May 2006.

[12] Grundy, J.C., Hosking, J.G., Zhu, N. and Liu, N. Generating Domain-Specific Visual Language Editors from High-level Tool Specifications, In Proceedings of the 2006 IEEE/ACM International Conference on Automated Software Engineering, Tokyo, 24-28 Sept 2006, IEEE CS Press.

[13] Hellsten, C., Automate acceptance tests with Selenium, IBM Developerworks, 20 Dec 2005, See: http://www-128.ibm.com/developerworks/web/library/wa-selenium-ajax/#N100A2

[14] Java.net, Java Pet Store 2.0 Reference Application, https://blueprints.dev.java.net/petstore/

[15] Liu, Y., Fekete, A., Gorton, I., Design-Level Performance Prediction of Component-Based Applications, IEEE Trans Software Eng, vol. 31, no.11, pp. 928-941, November, 2005.

[16] Mehra, A., Grundy, J.C. and Hosking, J.G. A generic approach to supporting diagram differencing and merging for collaborative design, In Proceedings of the 2005 ACM/IEEE Automated Software Engineering, Long Beach, CA, Nov 7-11 2005, IEEE Press, pp. 204-213.

[17] Microsoft Download Center, Web Application Stress Tool, See:http://www.microsoft.com/downloads/details.aspx?familyid=e2c0585a-062a-439ea67d75a89aa36495&displaylang=en

[18] Miller, R.C. and Bharat, K. SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. In Proceedings of WWW7, Brisbane Australia, April 1998, See: http://www.cs.cmu.edu/~rcm/websphinx/

[19] Menasce, D.A., Load testing of web sites, IEEE Internet Computing, Jul/Aug 2002, vol. 6, no. 4, pp. 70-74.

[20] Smith, C.U., Lladó, C.M., Cortellessa, V., Di Marco, A., Williams, L.G. From UML models to software performance results: an SPE process based on XML interchange formats, In Proceedings of the Fifth International Workshop on Software and Performance, Palma, Spain, July 12-14, 2005, ACM Press, pp. 87-98.

[21] Sprenkle, S., Gibson, E., Sampath, S., Pollock, L. Automated Replay and Failure Detection in Web Applications, In Proceedings of the 2005 International Conference on Automated Software Engineering, Long Beach, California, November 7-11, IEEE CS Press, 2005.

[22] Subraya, B.M., Subrahmanya, S.V., Object Driven Performance Testing in Web Applications, In Proceedings of the 1st Asia-Pacific Conference on Quality Software (APAQS'00), IEEE CS Press, 2000.