MODELOM RIADENÉ TESTOVANIE VÝKONNOSTI WEBOVÝCH APLIKÁCIÍ

Spoľahlivosť webového systému je možné zaručiť iba pre tie podmienky, pre ktoré bol úspešne otestovaný.

Zoltán Harsányi

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
xharsanyiz[zavináč]is[.]stuba[.]sk

Abstrakt. *V dnešnej dobe elektronického obchodovania môže nedostatočná kvalita webových systémov nepriaznivo ovplyvniť ziskovosť podnikov, ktoré sú závislé od týchto systémov. Preto efektívne techniky testovania výkonnosti webových systémov sú nevyhnutné pre zistenie, či daný systém spĺňa výkonnostné požiadavky pre jeho nasadenie do komerčnej prevádzky. Výkonnosť webových systémov závisí od rozličných parametrov, a tak každý jeden parameter musí byť otestovaný s rôznou úrovňou zaťaženia. Kvôli komplexnosti systémov nie je možné tieto parametre všeobecne použiť na ich otestovanie. Samotný systém sa musí rozdeliť na mnoho častí, ktoré reprezentujú rôzne biznis komponenty, a tie sa potom testujú pre niektoré parametre. Samotné pracovné zaťaženie systému musí byť charakterizované ako sekvencia medzi sebou závislých požiadaviek od jedného používateľa. Závislosti vzniknú tým, že niektoré požiadavky závisia od odpovedí na predchádzajúce požiadavky v jednej relácii. Pri testovaní systémov sa tieto závislosti musia zohľadniť. Zaoberám sa riešením tejto problematiky, a to tvorbu aplikačných modelov, pomocou ktorých je možné skúmať a modelovať závislosti jednotlivých komponentov systémov.*

Kľúčové slová: *testovanie, web aplikácie, výkonnosť, modely*

Úvod

Webové systémy čím ďalej, tým viac podnikom zabezpečujú kľúčové biznis funkcie, ako napr. elektronické obchody, manažment vnútrofiremných biznis procesov, atď. Rýchly nárast webových aplikácií pre elektronické obchodovanie bol spôsobený tým, že firmy chceli svoje produkty čo najlacnejšie, najrýchlejšie predávať, a tiež chceli, aby ich služby alebo výrobky boli všade dostupné. Web aplikácie sa stali ideálnym nástrojom na tvorbu elektronických obchodov.

Webovú aplikáciu treba chápať ako množinu služieb dostupných cez webový prehliadač. Pri takejto aplikácií sú odpovede na požiadavky používateľa dynamicky generované na základe toho, akú službu daný používateľ vyvolal. Pracovnú záťaž systému treba spojiť s pojmom relácie. Relácia je množina medzi sebou závislých požiadaviek, prijatých od jedného používateľa. Závislosti vzniknú tým, že jednotlivé požiadavky potrebujú odpovede predošlých požiadaviek v relácií. Napr. objednávka v elektronickom obchode môže byť poslaná až vtedy, ak predošlé požiadavky skončili tak, že nejaký tovar bol objednaný.

Návrh a implementácia sú iba dve fázy vývoja takýchto systémov. Testovanie je kritická tretia fáza, ktorá hrá dominantnú rolu v životnom cykle webového systému. Pre úspešné testovanie si treba pripraviť obsiahly testovací plán, scenár, rôzne nástroje na automatizáciu testovania jednotlivých častí.

Prvý krok testovania web aplikácií spočíva v overení aplikačnej vrstvy. Aplikačná vrstva sa testuje pomerne skoro v životnom cykle vývoja, mnohokrát ešte predtým, ako by vôbec existovalo grafické používateľské rozhranie. Následne sa overí používateľské rozhranie, či spĺňa rôzne ergonomické požiadavky. Na záver sa overí najdôležitejšia vlastnosť webových systémov, výkonnosť.

Pri klasických aplikáciách klient - server architektúry sa dá pomerne jednoducho predvídať zaťaženie systému. Pri webových aplikáciách sa však počet používateľov a pracovná záťaž môže meniť behom pár sekúnd a nepredvídateľne.

Testovanie webových aplikácií

Testovanie tradičných softvérových aplikácií je pomerne dobre definované rôznymi metódami, ktoré sa zrodili počas rokov. Každá metóda bola vytvorené s istou stratégiou. Väčšinou každá metóda slúži na otestovanie určitej časti aplikácie. Tradičné metódy však nemusia fungovať pri testovaní webových systémov. Tieto systémy sa líšia totiž vo funkcionalite, v prezentácií a v cieľovej skupine. Testovanie webových systémov vyžaduje veľmi silnú metodológiu, keďže majú dynamickú povahu.

Ako zabezpečiť kvalitu testovania?

Ak sa webová aplikácia stane kľúčovým alebo centrálnym komponentom biznisu, kvalita aplikáciou poskytnutých služieb sa stane prioritnou záležitosťou. Používateľov väčšinou netrápia javy, ako výpadky stránky, zahltenie siete, šírka pásma alebo iné druhy výpadkov systému. Pre používateľa webového systému kvalita znamená rýchlu a predvídateľnú odpoveď na jeho požiadavku v reálnom čase. Používateľ meria kvalitu v čase odozvy, dostupnosti a spoľahlivosti služby a samozrejme v cene. Nedostatočná kvalita totiž má za

následok, že zákazník v budúcnosti nenavštívi web stránku daného elektronického obchodu a firma tím stráca na obchode.

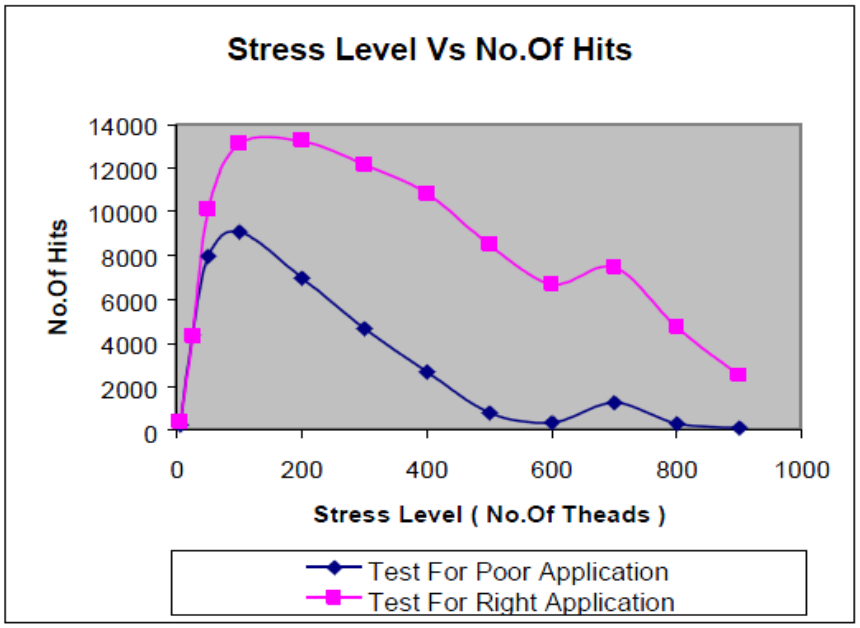
Testovanie je jeden zo spôsobov, ako zabezpečiť kvalitu webových systémov. Kvalita webovej stránky sa meria nasledovnými parametrami: čas, štruktúra, obsah, presnosť, konzistentnosť, čas odozvy a oneskorenie, výkonnosť. Na to, aby sa kvalita mohla zabezpečiť dlhodobo, treba tieto parametre neustále monitorovať a vyhodnocovať. V tomto príspevku sa bližšie zaoberám s jedným z parametrov kvality, a to výkonnosťou webových systémov.

Čo je testovanie výkonnosti?

Testovanie výkonnosti priamo odráža správanie sa celého systému. Návštevníci očakávajú čo najrýchlejšie odozvy na ich požiadavky. Pre každú stránku webovej aplikácie preto musí byť vykonaný dôkladný test výkonnosti. Testovanie výkonnosti patrí do kategórie funkčného testovania (metóda čiernej skrinky), čiže systém sa testuje bez toho, aby bol známy zdrojový kód systému. Hlavné ciele tohto testovania sú:

- maximálny počet návštevníkov pracujúcich súčasne s „dostatočnou výkonnosťou“
- maximálny počet návštevníkov pracujúcich súčasne bez toho, aby nastal výpadok systému
- detekcia spomalení v rámci architektúry aplikácie
- vplyvy softvérových a hardvérových zmien na výkonnosť celkového systému
- škálovateľnosť systému

Testovanie výkonnosti sa vykonáva tak, že sa nahrajú jednotlivé akcie, ktoré reálni používatelia vykonávajú a tieto nahrávky alebo skripty sa znova spúšťajú v automatizovanom a riadenom prostredí. Výsledky testov sa zaznamenajú a vytvoria sa rôzne druhy grafov. Na obrázku nižšie (**Obr. 1**) je graf výsledkov testov, ktorý ukazuje závislosť medzi počtom vlákien a počtom spracovaných požiadaviek.



Obr. 1.: Graf výkonnosti

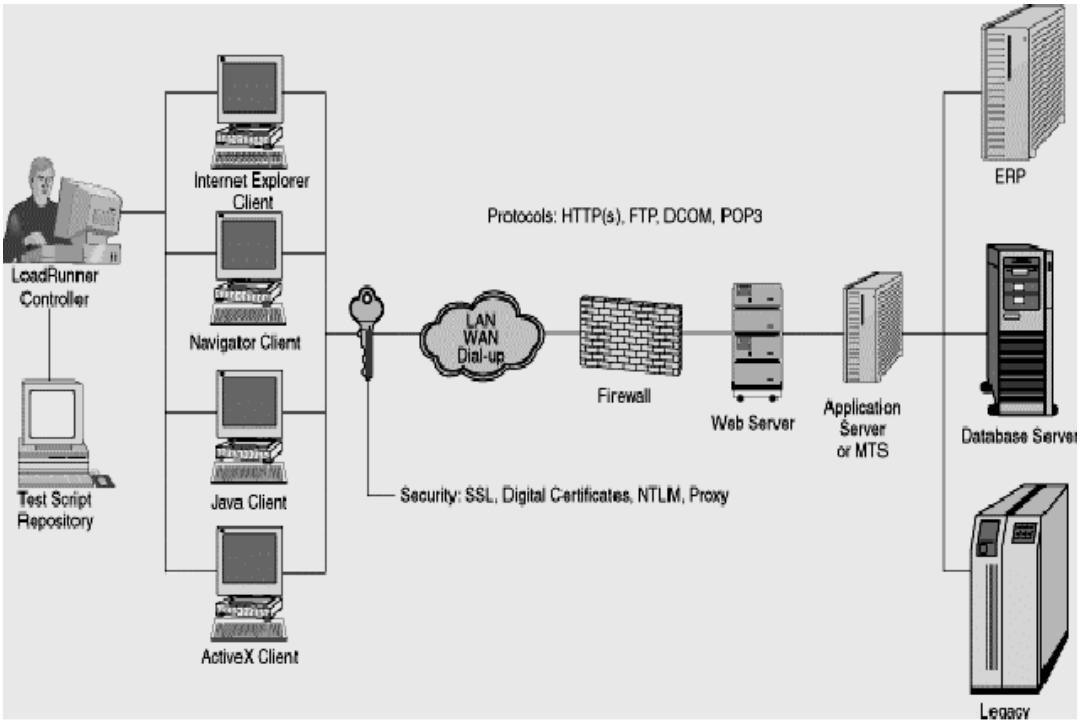
Typy výkonnostných testov

Tabuľka nižšie (**Tab. 1**) uvádza jednotlivé typy výkonnostných testov, ďalej vysvetľuje cieľ konkrétneho testu, čiže ktorú vlastnosť daný test overuje.

Tab. 1. Typy výkonnostných testov

Typ testu	Cieľ	Popis
Záťažový test (load test)	Overenie správanie sa systému pod bežnou záťažou	Test slúži na overenie, či aplikácia spĺňa základné výkonnostné požiadavky. Tento test umožňuje merať čas odoziev, priepustnosť aplikácie
Test silnej záťaže (stress test)	Zistenie a validovanie správanie sa systému, keď je záťaž oveľa väčšia, ako za bežných okolností	Tento test slúži na odhalenie tých chýb aplikácie, ktoré sa objavujú iba pod extrémnou záťažou, ako napr. synchronizačné problémy, slabé miesta aplikácie, úniky pamäte
Test kapacity (capacity test)	Overenie, koľko používateľov dokáže systém obsluhovať s akceptovateľným výkonom	Test slúži na to, aby uľahčil naplánovanie možných kapacít, ktoré sa využívajú najmä pri plánovaní možných rozšírení systému v budúcnosti, ako napr. rozšírenie používateľskej základne, dát, atď.

Na obrázku nižšie (**Obr. 2**) je znázornená typická architektúra, ktorá sa používa na testovanie výkonnosti webových aplikácií.



Obr. 2.: Typická architektúra pre testovanie výkonnosti

Rôzne pohľady na výkonnostné testovanie webových aplikácií

Existuje niekoľko základných pohľadov, akým spôsobom testovať výkonnosť webových aplikácií. V krátkosti by som ich zhrnul:

- objektom riadené testovanie výkonnosti
- modelom riadené testovanie výkonnosti
- akciami riadené testovanie výkonnosti
- agentom riadené testovanie výkonnosti
- integrovaný pohľad testovanie výkonnosti

Dôvod, prečo som si vybral práve modelom riadený prístup je ten, že podľa mojej mienky tento prístup reflektuje najreálnejšie testovaný systém, keďže sa vytvárajú modely daného systému, tým pádom môžeme navrhnúť presnejšie výkonnostné testy, taktiež modely nám umožnia identifikovať kľúčové časti nášho systému a klásť väčší dôraz na overovanie kvality týchto modulov. V ďalších kapitolách sa zaoberám detailnejšie týmto pohľadom na výkonnostné testovanie.

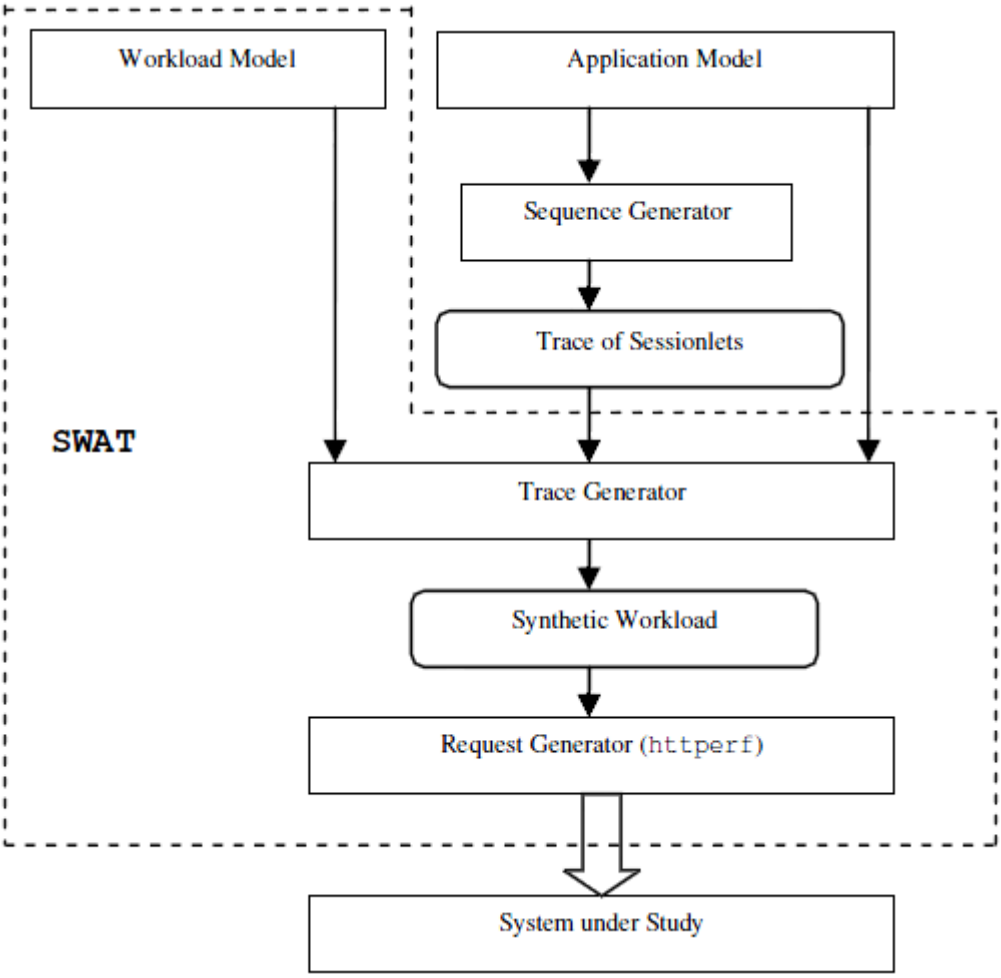
Prehľad modelom riadeného testovania výkonnosti

Pri tomto testovaní tester najprv vytvára aplikačné a záťažové modely študovaného systému, ktoré zachytávajú jednotlivé závislosti medzi požiadavkami, závislosti dát. Metodológia modelovania a samotný proces vytvárania je vysvetlené nižšie (v kapitole Modelovanie aplikácie). Na generovanie rôznych druhov korektných požiadaviek od používateľov slúži tzv. generátor požiadaviek. Každú korektnú sekvenciu požiadaviek

nazývame *sessionlet*. Každý sessionlet spĺňa kritériá závislosti medzi požiadavkami a slúži ako vstup spolu so záťažovým a aplikačným modelom pre testovací mechanizmus. Tento testovací mechanizmus sa v skratke volá SWAT (Session-Based Web Application Tester). Výhoda tohto prístupu spočíva hlavne v tom, že testovací mechanizmus umožňuje za chodu meniť rôzne testovanie parametre a ostatné ponechať v pôvodnom stave. Samotné testovanie prebieha nasledovne:

1. SWAT vygeneruje novú reláciu, pričom každá relácia sa správa ako reálny používateľ, čiže pošle požiadavku, čaká, kým sa neprijme celá odpoveď od systému
2. nastavia sa parametre jednotlivých požiadaviek v relácií a pošle sa
3. po prijatí odpovede sa čaká podľa nastaveného času nečinnosti (tzv. think time)
4. vygeneruje sa ďalšia relácia a proces sa začne odznova

Záťaž, ktorú vygeneruje SWAT nezahŕňa požiadavky, ktoré vyžadujú multimediálny obsah, ako fotky, zvuky, videá, ktoré sú často zahrnuté v HTML odpovedi systému. Dôvodom je to, že multimediálny obsah je často uložený na externom, tzv. multimediálnom serveri. Na obrázku nižšie (**Obr. 3**) je znázornená schéma testovania pomocou modelov a SWAT-u.



Obr. 3.: Modelom riadené testovanie

Modelovanie aplikácie

V tejto kapitole popíšem novú metodológiu modelovania aplikácií na spracovanie závislostí medzi požiadavkami a závislostí dát. Táto metodológia je založená na konečných automatoch, tzv. Extended Finite State Machines (EFSM). Pomocou konečných automatov dokážeme pomerne jednoduchým spôsobom modelovať poradie a závislosti požiadaviek a tým pádom automatizovať testovanie. Automat umožňuje modelovanie závislostí vďaka tomu, že z jedného stavu umožňuje prechod iba do určitého stavu, alebo do určitých stavov až vtedy, keď sú zadané podmienky splnené.

EFSM môžeme vyjadriť nasledovným spôsobom:

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

kde I, O, S, $\vec{x}$ a T sú konečné množiny vstupných, výstupných symbolov, stavov, premenných a prechodov. Fungovanie tohto automatu by som mohol zhrnúť v týchto krokoch:

1. automat má vždy začiatočný a koncový stav
2. automat má definovaný iniciálny stav, v ktorom sa nachádza
3. premenné automatu majú iniciálnu hodnotu, ktoré sú popísané v $\vec{x}$
4. po prijatí vstupov automat spraví prechod do ďalšieho stavu vtedy, ak predikát, ktorý je definovaný v T sa vyhodnotí ako pravdivý
5. automat vygeneruje výstupy a uloží ich do výstupnej premennej
6. premenné sú modifikované podľa aktuálneho stavu

Ako vstup si môžeme predstaviť formulár údajov, ktorý používateľ vyplní a následne odošle systému spolu s typom požiadavky a takisto množinou dvojíc názov - hodnota formulárových prvkov. Prechod do ďalšieho stavu je možný, ak predikát pre konkrétny vstup sa vyhodnotí ako pravdivý. Tento automat je zároveň nedeterministický, keďže umožňuje viac prechodov z jedného stavu. Ako príklad by som uviedol používateľa elektronického obchodu, ktorý sa môže prihlásiť z istého stavu alebo si pozrieť produkty firmy na domovskej stránke. Každý prechod definuje 2 druhy predikátov: predikát závislostí požiadaviek a predikát závislostí dát.

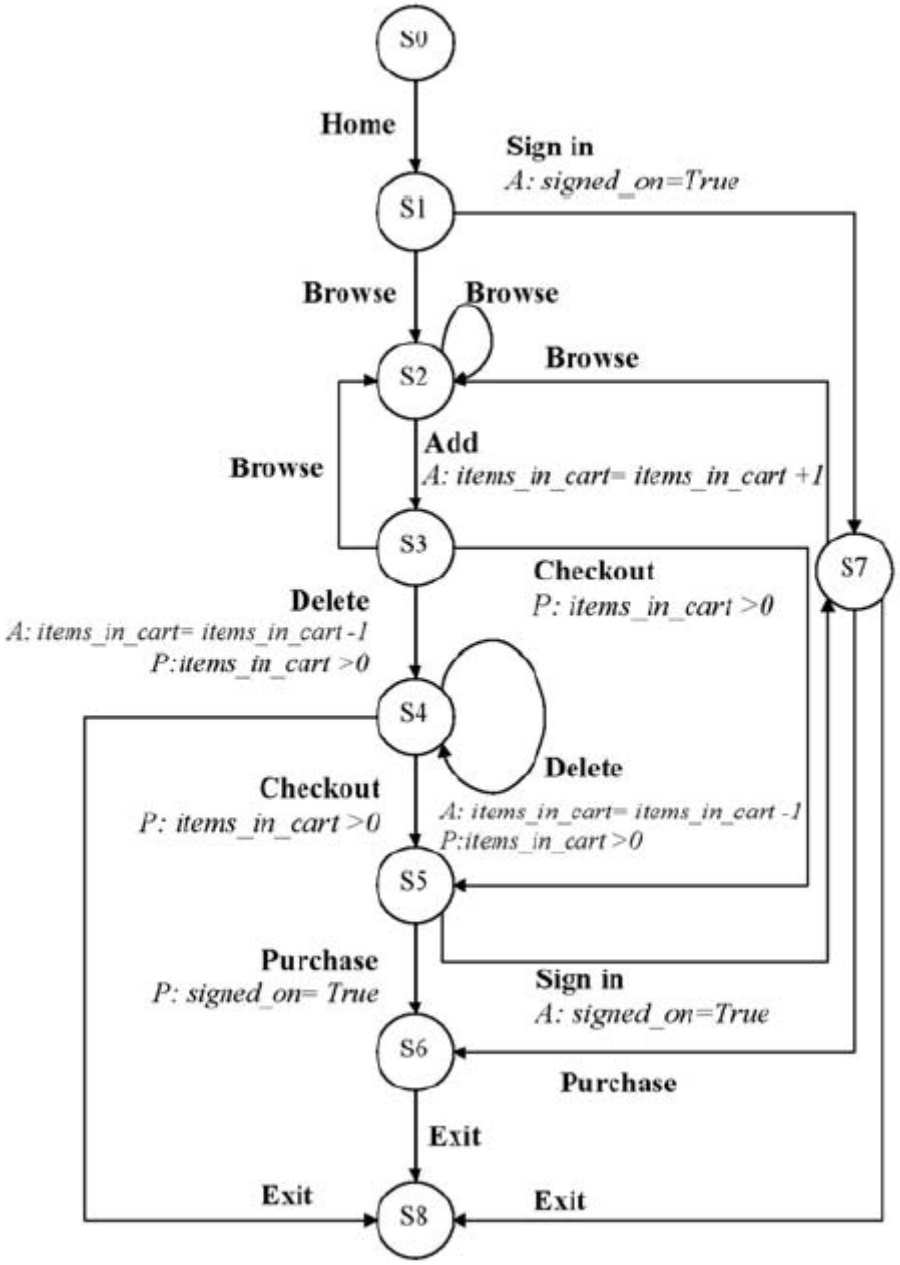
Modelovanie závislostí medzi požiadavkami

Ako príklad modelovania závislostí medzi požiadavkami uvediem aplikáciu internetového obchodu. Na obrázku nižšie (**Obr. 4**) je znázornený jednoduchý model takéhoto systému. Používateľ pošle na začiatku požiadavku typu Home pre získanie domovskej stránky. Ak sa chce prihlásiť do systému, môže to urobiť napríklad v stave S1 zavolaním služby Sign-in. Informácie o rôznych produktoch môže pozerať pomocou požiadavky Browse. Požiadavky Add, Delete dovoľujú pridávať, resp. odoberať produkty z košíka. Keď si chce vytvoriť objednávku tovarov, ktoré má v košíku, pošle požiadavku Checkout. Na zaplatenie nákupu a dokončenie objednávky musí odoslať požiadavku typu Purchase.

Ako vidieť na obrázku, tento model používa 2 premenné na stanovenie závislostí medzi jednotlivými požiadavkami, konkrétne:

- item_in_cart: táto premenná typu celé číslo udržiava počet tovarov v košíku
- signed_on: táto premenná boolovského typu označuje, či je daný používateľ prihlásený v systéme

Iniciálne hodnoty týchto premenných sú 0 a FALSE. Ich hodnota sa mení podľa jednotlivých požiadaviek používateľa (napr. ak sa prihlási, hodnota premennej signed_on sa zmení na TRUE, po pridaní tovaru do koša sa inkrementuje hodnota premennej item_in_cart, atď.). Závislosti medzi jednotlivými požiadavkami sú vynútené tým, že prechody do ďalšieho stavu sú podmienené predikátmi, ktoré rozhodujú na základe hodnôt lokálnych premenných automatu (jednotlivé predikáty sú na obrázku znázornené medzi dvoma stavmi).



Obr. 4.: EFSM pre elektronický obchod

EFSM takisto umožňuje rovnaký scenár namodelovať rôznymi spôsobmi, čo je pri webových aplikáciách bežný jav, keďže používateľ môže ten istý cieľ dosiahnuť rôznymi spôsobmi. Na druhej strane však tento automat má aj isté nevýhody. Pri modelovaní zložitých systémov, kde existuje obrovské množstvo možných stavov a scenárov, teda prechodov, vzniknú buď zložité modely, ktoré sú priam neprehľadné, alebo sa nenamodelujú všetky možné prechody a stavy. To však môže viesť k prípadu, keď sa neodhalia isté chyby, ktoré sa vyskytujú iba pri určitých scenároch.

Modelovanie závislostí dát

V tejto sekcií v krátkosti popíšem, ako sa dá aplikačný model využiť na modelovanie závislostí dát v systéme. Závislosti dát v systéme spravujú generovanie hodnôt parametrov. Modelom použitá množina stavových premenných zahrňujú parametre všetkých typov požiadaviek, ktoré aplikácia podporuje. Každý parameter používa nasledujúcu notáciu: *„Typ požiadavky.Meno parametra“*. Napríklad ADD.ITEM_ID znázorňuje požiadavku ADD a parameter ITEM_ID. Modelovacia metodológia rozlišuje rôzne typy parametrov, ako napr. tester - špecifické, parametre závislé od relácie, atď.

Závislosti dát môžeme chápať ako množinu vstupov, ktorú jednotlivé služby vyžadujú. Ako príklad môžem uviesť prihlasovanie používateľa, čiže volanie služby Sign_in na obrázku **Obr. 4**. Táto požiadavka potrebuje ako vstupné parametre prihlasovacie meno a heslo. Pokiaľ ich používateľ nezadá, prechod do stavu, v ktorom je používateľ prihlásený sa neuskutoční. Iný príklad by som mohol uviesť pri pridávaní alebo odoberaní tovarov z košíka. Každý tovar má svoj jedinečný identifikátor, ID. Keď sa volajú služby Add alebo Delete, ako parameter musia dostať tento identifikátor, na základe čoho dokážu identifikovať, ktorý tovar používateľ pridal alebo odstránil.

Implementácia a vyhodnotenie výsledkov testovania

Existujú rôzne nástroje, ktoré podporujú mnou popísaný pohľad na testovanie výkonnosti. Testeri používajú textové súbory na špecifikáciu aplikačného modelu. Kvôli zdĺhavosti špecifikačného jazyka neuvediem príklad daného jazyka. Testovací nástroj obsahuje modul, ktorý slúži na overenie daného textového súboru, presnejšie na overenie konzistencie aplikačného modelu a takisto konzistencie medzi aplikačným modelom a modelom pracovnej záťaže. Ak overovač modelu nenájde chybu v špecifikácií modelu, vygeneruje aplikačný model a model záťaže vo formáte, ktorý je jednoduchšie a rýchlejšie spracovateľný testovacím nástrojom. Po spustení testu, testovací nástroj prechádza definovanými stavmi a zaznamenáva čas, ktorý bol potrebný systému na spracovanie požiadavky a vygenerovanie odpovede. Jednotlivé testy je samozrejme možné zopakovať a z výsledných hodnôt spraviť priemerné výsledky.

Najznámejší testovací nástroj, ktorý podporuje tento prístup testovania výkonnosti webových aplikácií, je softvér s menom *httperf* (http://www.hpl.hp.com/research/linux/httperf/), ktorý je voľne dostupný, má otvorený zdrojový kód a je pomerne zrozumiteľne dokumentovaný. Po kliknutí na daný odkaz sa dostanete na domovskú stránku projektu *httperf*, odkiaľ si môžete jednak stiahnuť daný nástroj, a takisto si niečo o ňom prečítať.

Záver

Webové aplikácie sú značne komplexnejšie v porovnaní s tradičnými aplikáciami kvôli veľkému množstvu neznámych a nejasných faktorov. Testovanie výkonnosti webových aplikácií je jedna z najzložitejších fáz životného cyklu vývoja takýchto aplikácií.

Testovanie výkonnosti má zabezpečiť spokojnosť zákazníka s danou aplikáciou. Takéto testovanie je založené na rôznych druhov stratégií a prístupov. Jednou z takýchto prístupov je modelom riadené testovanie výkonnosti. Pri tomto prístupe sa vytvoria

aplikačné modely na definovanie sekvencie akcií, ktoré sú identické akciám, ktoré bežní používatelia vykonávajú v systéme. Vďaka týmto modelom je potrebné vynaložiť oveľa menej úsilia na generovanie pracovnej záťaže na systém, tým pádom sa úsilie môže sústrediť na samotný proces testovania. Tento pohľad však vyžaduje od testera, aby mal schopnosti zjednodušiť aplikačnú logiku systému a tiež vytvoriť aplikačný model.

Myslím si, že tento postup testovania výkonnosti má svoje miesto aj v praxi a nie iba vo výskume, keďže umožňuje testerom definovať a spúšťať sekvencie akcií, tým pádom simulovať reálneho používateľa systému. Čím reálnejšie budú výkonnostné testy, tým väčšiu šancu majú na odhalenie chýb, ktoré sa môžu vyskytnúť až v ostrej prevádzke. Ako ďalší a určite dôležitý vývoj v tejto oblasti by som si vedel predstaviť hlavne v rozšírení aplikačných modelov, aby bolo možné vytvárať a generovať aplikačné modely pomocou jazyka UML.

Použitá literatúra

1. Andrej van der Zee, Alexandre Courbot, Tatsuo Nakajima.: *Action-based Performance Monitoring of Multi-Tier Web Applications*. In: 2009 International Conference on Computational Science and Engineering, 978-0-7695-3823-5/09 © 2009 IEEE
2. B.M. Subraya and S.V. Subrahmanya.: Object driven Performance Testing of Web Applications. In: *Proceedings of the First Asia-Pacific Conference on Quality Software, 0-7695-0825-1/00 2000 IEE*.
3. Liming Zhu, Ian Gorton, Yan Liu: *Model Driven Benchmark Generation for Web Services*. In: IW-SOSE'06, May 27–28, 2006, Shanghai, China
4. Mahnaz Shams, Diwakar Krishnamurthy, Behrouz Far.: A Model-Based Approach for Testing the Performance of Web Applications. In: *Proceedings of the Third International Workshop on Software Quality Assurance (SOQUA'06)*, November 6, 2006, Portland, OR, USA.

Annotation

Model driven performance testing in web applications

Nowadays in the world of e-commerce poor performance of Web-based systems can adversely impact the profits of enterprises that rely on these systems. That's the reason why effective performance testing techniques are inevitable for understanding whether the Web-based system will satisfy performance requirements, when deployed in the real world. The performance of Web-based system depends on many parameters. Each one must be tested under different stress levels. Due to the complexity of web-based systems it's not possible to use these parameters generally for testing them. It's necessary to decompose the system into smaller components, which represents various business components and these components will later be tested for some parameters. The workload of system is characterized as a sequence of inter-dependent requests submitted by a single user. Dependencies arise because some user requests depend on the user responses of earlier requests in a session. My approach solves this problem by using an application model, which can be used for studying and modeling the dependencies of the components of system.